C++ - Como Programar

D325c Deitel, H.M.
C++: como programar/H.M. Deitel e P.J. Deitel trad. Carlos
Arthur Lang Lisbôa e Maria Lúcia Lang Lisbôa. - 3.ed. - Porto
Alegre : Bookman, 2001.
1. Computação - Programação de Computadores - C++. I.Deitel, P.J. II. Título.
CDU 681.3(C+÷)
Catalogação na publicação: Mônica Baliejo Canto - CRB 10/1023
ISBN 85-7307-740-9

1

II. M. DEITEL
Deitel & Associates, mc.
P. J. DEITEL
Deitel & Associates, mc.
C++ Como Programar

Tradução
Carlos Arthur Lang Lisbôa e Maria Lúcia Blanck Lisbôa
Professores do Instituto de Informática da UFRGS
Reimpressão 2004 ;1

LWJ
Bnokm3n;0] PORTO ALEGRE, 2001

Obra originalmente publicada sob o título
C++ How to program: Third edition
© 2001
Tradução autorizada do original em idioma inglês pela Prentice-Hali, mc.
ISBN 0-13-089571-7
Capa: Mário Rõhnelt
Preparação do original: Daniel Grassi
Supervisão editorial: Arvsinha Jacques Affonso
Editoração eletrônica: Laser House - m.q.o.f
Reservados todos os direitos de publicação em língua portuguesa à
ARTMED EDITORA LTDA.
(BOOKMAN COMPANHIA EDITORA é uma divisão da Artmed Editora Ltda)
Av. Jerônimo de Ornellas, 670 - Fone (51) 330-3444 Fax (51) 330-2378
90040-340 - Porto Alegre, RS, Brasil
SÃO PAULO
Rua Francisco Leitão, 146 - Pinheiros
Fone (11) 3085-7270/3062-9544 Fax (11) 3083-6160
054 14-020 - São Paulo, SP, Brasil

IMPRESSO NO BRASIL
PRINTED IN BRAZIL

Para
Don Kostuch:
Por seu firme compromisso com a excelência em ensinar e escrever sobre C++ e a tecnologia de objetos.
Obrigado por ser nosso mentor, nosso colega e nosso amigo.
Obrigado por ter sido, durante uma década, nosso maior crítico, e ainda assim mais
construtivo, revisor.
Obrigado por ter altruisticamente sacrificado seu tempo pessoal para ajudar-nos a cumprir nossos prazos de publicação.
É um previlégio para nós sermos seus alunos.
Esperamos contar com você como co-autor de Advanced C++ How to Program.
Harvey e Paul Deitei

Prefácio

Bem-vindo à linguagem C++ padrão ANSI/ISO Este livro foi escrito por um cara velho e um cara jovem. O cara velho (HMD; Massachusetts Institute of Technology 1967) vem programando e/ou ensinando programação nos últimos 39 anos, O cara jovem (PJD; MIT, 1991) tem programado durante 18 anos e pegou gosto por ensinar e escrever, O cara velho programa e ensina usando sua experiência; o cara jovem faz isso com uma reserva inesgotável de energia. O cara velho quer clareza; o cara jovem quer desempenho. O cara velho aprecia a elegância e a beleza; o cara jovem quer resultados. Reunimo-nos para produzir um livro que esperamos que você achará informativo, interessante e divertido.

Esta é uma época empolgante para a comunidade C++, com a aprovação do padrão ANSIIISO para C++. O

ANSI (o Instituto Americano de Padrões Nacionais) e o ISO (a Organização de Padrões Internacionais) cooperaram para desenvolver o que se tornou um dos padrões mundiais mais importantes para a comunidade de computação.

Quando escrevemos a segunda edição de C++ How to Program, direcionamos o livro para cursos de nível acadêmico, nos quais então eram ensinados principalmente Pascal ou C, enfatizando o paradigma de programação procedural. Escrever um livro de ensino de C÷+ para o público dos cursos de Ciência da Computação 1 e II apresentava-nos um desafio difícil. Necessitávamos descrever dois paradigmas de programação, tanto a programação procedural (porque C++ ainda inclui C) e a programação orientada a objetos. Isto praticamente dobrou a quantidade de material que precisaria ser apresentado no nível introdutório. Escolhemos uma estratégia de apresentar o material ao estilo de C sobre tipos de dados primitivos, estruturas de controle, funções, arrays, ponteiros, strings e estruturas nos primeiros cinco capítulos do livro. Então apresentamos a programação orientada a objetos nos Capítulos 6 a 15.

C++ How to Program se tornou o livro de ensino de C++ mais amplamente usado no mundo no ambiente

acadêmico. Atrasamos a redação desta nova edição por duas razões:

1. C++ estava em desenvolvimento ativo durante este tempo, com novas minutas do documento de padronização surgindo regularmente, mas sem um sinal claro do comitê de padronização de que a minuta do

padrão seria aceita “como está” dentro de pouco tempo.

2. Estávamos esperando por um sinal claro de que já era tempo de lançar uma nova edição de C++ How to Program. Esse sinal chegou em julho de 1997 com a publicação da terceira edição do livro de Bjarne Stroustrup, A Linguagem de Programação C++. Stroustrup criou C++, e seus livros são os trabalhos definitivos sobre a linguagem. Nesse momento, sentimos que a “nova definição” de C++ estava suficientemente estável para publicarmos C++ How to Program - Second Edition.

Desviamos nossa atenção por um tempo para produzir cinco publicações sobre Java. Mas a excitação da aceitação iminente da minuta do padrão ANSI/ISO para C++ trouxe nossa atenção de volta para C++.

C++ Como Programar - Terceira Edição
Nesta Terceira Edição, executamos um processo extensivo de revisão, que levou a milhares de aperfeiçoamentos. Também atualizamos completamente os programas no texto, para ficarem de acordo com o uso de ambientes de nomes em C÷+ padrão.
_1

VIII PREFÁCIO
A principal novidade desta Terceira Edição é um estudo de caso completo, totalmente implementado, sobre o projeto orientado a objetos usando a Unified Modeling LanguageTM (UML). Sentimos que um comprometimento com projetos orientados a objetos de grande porte é algo que está faltando em livros-texto introdutórios. Este estudo de caso opcional é altamente recomendado porque vai melhorar consideravelmente a experiência do estudante em uma seqüência de programação de primeiro ano na universidade. Este estudo de caso oferece aos estudantes uma
oportunidade de mergulhar em um programa em C++ com mais de 1000 linhas, que foi cuidadosamente examinado por uma equipe de revisores de empresas e acadêmicos destacados.
Na edição anterior deste livro, incluímos seções especiais, denominadas “Pensando em objetos”, no final dos Capítulos 1 a 7. Estas seções conduziram o estudante através das etapas necessárias para projetar o simulador em software de um sistema de elevador. Pedimos ao estudante para completar estas etapas e implementar seu projeto em C++. Para C++ Como Programar - Terceira Edição, remodelamos completamente este estudo de caso. Nos finais dos Capítulos 1 a 7 e no final do Capítulo 9, usamos a seção “Pensando em objetos” para
apresentar uma introdução cuidadosamente cadenciada ao projeto orientado a objetos usando a UML. A UML é, agora, o esquema de representação gráfica mais amplamente usado para a modelagem de sistemas orientados a
objetos. A UML é uma linguagem gráfica complexa, rica em recursos. Em nossas seções “Pensando em objetos”, apresentamos um subconjunto conciso e simplificado destes recursos. Usamos, então, este subconjunto para
guiar o leitor através de uma primeira experiência de projeto com a UML voltada ao programador/projetista
] orientado a objetos iniciante. Apresentamos este estudo de caso de forma totalmente resolvida. Isto não é um exercício; em vez disso, é uma experiência de aprendizado de ponta a ponta que termina com um walkthrough detalhado do código em C++.
Em cada um dos cinco primeiros capítulos, concentramo-nos na metodologia “convencional” de programação estruturada, pois os objetos que iremos construir serão compostos, em parte, por pedaços de programas estruturados. Então, concluímos cada capítulo com uma seção “Pensando em objetos”, na qual apresentamos uma introdução à orientação a objetos utilizando a Unified Modeling Language (UML). Nosso objetivo, nestas seções “Pensando em objetos”, é ajudar os estudantes a desenvolver uma forma de pensar orientada a

objetos, de modo que possam imediatamente colocar em uso os conceitos de programação orientada a objetos que eles começam a aprender no Capítulo 6. Na primeira destas seções, no fim do Capítulo 1, introduzimos conceitos básicos (i.e., “pense em objetos”) e terminologia (i.e., “fale em objetos”). Nas seções “Pensando em objetos” opcionais, no fim dos Capítulos 2 a 5, consideramos tópicos mais substanciais, à medida em que atacamos um problema desafiador com as técnicas de projeto orientado a objetos (OOD). Analisamos uma definição de problema típica, que requer que um sistema seja construído, determinamos os objetos necessários para implementar aquele sistema, determinamos os atributos que os objetos precisarão ter, determinamos os comportamentos que estes objetos precisarão exibir e especificamos como os objetos precisarão interagir uns com os outros para atender aos requisitos do sistema. Fazemos tudo isto mesmo antes de discutir como escrever programas C++ orientados a objetos. Nas seções “Pensando em objetos” no fim dos Capítulos 6, 7 e 9, discutimos uma implementação em C++ do sistema orientado a objetos que projetamos nos capítulos anteriores.

‘1 Este estudo de caso é significativamente maior do que qualquer outro projeto tentado no livro. Sentimos que o estudante adquire experiência significativa seguindo este processo completo de projeto e implementação. Este projeto nos forçou a incorporar tópicos que não discutimos em nenhuma outra seção do livro, incluindo interação entre objetos, uma discussão aprofundada de handies, a filosofia de uso de referências versus ponteiros e o uso de declarações antecipadas para evitar o problema de referências circulares em inclusões. Este estudo de caso vai ajudar a preparar os estudantes para os tipos de projetos de grande porte encontrados nas empresas.

Seções “Pensando em objetos”

No Capítulo 2, começamos a primeira fase de um projeto orientado a objetos (OOD) para o simulador de elevador - identificar as classes necessárias para implementar o simulador. Também introduzimos o caso de uso de UML, diagramas de classes e objetos e os conceitos de associações, multiplicidade, composição, papéis e vínculos.

No Capítulo 3, determinamos muitos dos atributos de classes necessários para implementar o simulador de elevador. Também introduzimos o diagrama de estados e diagramas de atividades da UML e os conceitos de eventos e ações e como eles se relacionam com estes diagramas.

No Capítulo 4, determinamos muitas das operações (comportamentos) das classes na simulação de elevador. Também introduzimos o diagrama de seqüência da UML e o conceito de mensagens enviadas entre objetos.

No Capítulo 5, determinamos muitas das colaborações (interações entre objetos do sistema) necessárias para implementar o sistema de elevador e representamos tais colaborações usando o diagrama de colaboração da UML. Além disso, incluímos uma bibliografia e uma lista de recursos da Internet e da

World Wide Web que contêm as especificações da UML 1.3 e outros materiais de referência, recursos gerais, tutoriais, FAQs, artigos, publicações e software.
No Capftulo 6, usamos o diagrama de classes da UML desenvolvido em seções anteriores para esboçar os
arquivos de cabeçalho C++ que definem nossas classes. Também introduzimos o conceito de handies para objetos do sistema e começamos a estudar como implementar handies em C++.
No Capítulo 7, apresentamos um programa simulador de elevador completo (aproximadamente 1000 linhas de código) e um walkthrough detalhado do código. O código é derivado diretamente do projeto baseado em UML criado em seções anteriores e emprega nossas boas práticas de programação, incluindo o uso de membros de dados e funções static e const. Também discutimos alocação dinâmica de memória, composição e interação entre objetos através de handies e como usar declarações antecipadas para evitar o problema de referências circulares em inclusões.
No Capítulo 9, atualizamos o projeto e implementação da simulação do elevador para incorporar herança.
Também sugerimos modificações adicionais, de modo que o estudante possa então projetar e implementar usando as ferramentas apresentadas nas seções anteriores.
Sinceramente, esperamos que este recém-atualizado estudo de caso de simulação de elevador ofereça uma experiência desafiadora e significativa tanto para estudantes quanto para instrutores. Empregamos um processo incremental orientado a objetos cuidadosamente desenvolvido para produzir um projeto baseado em UML para nosso simulador de elevador. A partir deste projeto, produzimos uma implementação em C++ substancial que funciona, usando conceitos-chave de programação, incluindo classes, objetos, encapsulamento, visibilidade, composição e herança. Agradeceríamos muito se você dedicasse um momento para nos enviar seus comentários, críticas e sugestões, a fim de aprimorar este estudo de caso, para: deitei@deitei com.
Material auxiliar para C++: Como Programar - Terceira Edição
Trabalhamos arduamente para produzir um livro-texto e material auxiliar que, esperamos, você e seus estudantes vão considerar valiosos. Os seguintes recursos auxiliares estão disponíveis:
Os 268 exemplos de programas de C++: Como Programar - Terceira Edição estão incluídos no CDROM na contracapa final do livro-texto. Isto ajuda os instrutores a preparar aulas mais rapidamente e ajuda os estudantes a dominar C++. Os exemplos também estão disponíveis para download em www. deitei com. Quando extrair o código fonte do arquivo ZIP, você deve usar um leitor de arquivos ZIP tal como WinZip (http: //www winzip comi) ou PKZIP (http: //www pkware com/), que entenda diretórios. O arquivo deve ser extraído para um diretório separado (por exemplo, cpphtp3e_exempios).
• O sofíware Microsofi Visual C++ Introductory Edition é fornecido no CD-ROM do livro-texto. Este software permite aos estudantes editar, compilar e depurar programas C++. Tornamos disponível, sem custo adicional, um breve tutorial de Visual C++ 6 (no formato PDF da Adobe) em nosso site da Web

(www.deitei.com).
• Um site da Web relacionado (www prenhaii com/deitei) oferece recursos para instrutores e estudantes. Os recursos para instrutores incluem apêndices do livro-texto (por exemplo, Apêndice D, “Recursos sobre C++ na Internet”) e um gerenciador de sumários, para planejamento de aula. Os recursos para estudantes incluem objetivos dos capítulos, perguntas do tipo verdadeiro/falso, destaques dos capítulos, materiais de referência e um quadro de avisos.
• PowerPoint(I?) Instructor Lecture Notes customizáveis, com muitos recursos completos, incluindo código fonte e tópicos para discussão para cada programa e ilustração importantes. Estas notas de aula estão
disponíveis sem custo para instrutores e estudantes no site www. deitei com.
• Lab Manual (disponível na primavera americana de 2001) - um item para venda contendo sessões fechadas para laboratório.

Uma revolução no desenvolvimento de software
Durante anos, o hardware vem melhorando drasticamente. Mas o software, por alguma razão, parecia resistir a quase todas as tentativas para construí-lo de forma mais rápida e melhor. Hoje em dia, estamos em meio a uma revolução na maneira como o software está sendo projetado e escrito. Essa revolução está baseada na noção de bom senso, herdada do hardware, de usar componentes padronizados e intercambiáveis, exatamente como feito por Henry Ford nos dias do Ford Modelo T. Estes componentes de software são chamados “objetos” - mais corretamente, “classes,” que são as “fôrmas” com as quais os objetos são produzidos.
A mais madura e bem-conhecida das linguagens orientadas a objetos é a Smalltalk, desenvolvida no início dos anos 70 no Palo Alto Research Center da Xerox. Mas a linguagem orientada a objetos mais amplamente usada - por um fator de 10 vezes a mais que a Smalltalk - é a linguagem C++ desenvolvida por Bjarne Stroustrup e outros no início dos anos 80 na AT&T. No tempo decorrido entre a publicação da primeira e segunda edições deste livro, outro competidor apareceu em cena - a linguagem de programação orientada a objetos Java, desenvolvida no início dos anos 90 por James Gosling e outros na Sun Microsystems.
Por que uma nova linguagem de programação orientada a objetos importante a cada 10 anos? Na verdade, Smalltalk estava à frente de seu tempo, como uma experiência de pesquisa. C++ estava adequada à sua época e às necessidades de programação dos sistemas de alta performance e do desenvolvimento de aplicativos de hoje em dia. JavaTM ofereceu aos desenvolvedores a possibilidade de criar aplicativos altamente portáveis, com uso intensivo de multimídia, e aplicativos com uso intensivo de redes baseados na InternetlWorld Wide Web.
Programação procedural, programação baseada em objetos,
programa ção orientada a objetos e programação genérica
Neste livro, você dominará os cinco componentes-chave de C++, bem como quatro paradigmas de programação contemporâneos:

• Programa ção procedural em C - Capítulos 1-5 e l6-ll os tópicos-chave incluem tipos de dados, estruturas de controle, funções, arrays, ponteiros, strings, estruturas, manipulação de bits, manipulação de caracteres, pré-processamento e outros.
2. Melhorias introduzidas por C++ em relação à programação procedural em C - Seções 3.15-3.21; os tópicos- chave incluem funções mime, referências, argumentos default, sobrecarga de funções e funções gabarito.
3. Programação baseada em objetos em C++ - Capítulos 6-8; os tópicos-chave incluem tipos de dados abstratos, classes, objetos, encapsulamento, ocultamento de informações, controle de acesso a membros, construtores, destruidores, reusabilidade de software, objetos e funções membro constantes, composição, o conceito de friend, alocação dinâmica de memória, membros static, o ponteiro this e outros.
4. Programação orientada a objetos em C++ - Capítulos 9-15, 19 e 21; os tópicos-chave incluem classes base, herança simples, classes derivadas, herança múltipla, funções virtual, vinculação dinâmica, polimorfismo, funções virtual puras, classes abstratas, classes concretas, entrada/saída com streams, classes gabarito, tratamento de exceções, processamento de arquivos, estruturas de dados, strings como objetos no pleno sentido, tipo de dados bool, operadores de coerção, ambientes de nomes, informações sobre tipo durante a execução (RTTI, run-time type information), construtores explicit e membros muta- bie.
5. Programação genérica em C++ - Capítulo 20 - o maior capítulo do livro; os tópicos-chave incluem a biblioteca padrão de gabaritos (STL), contêineres genéricos, contêineres seqüenciais, contêineres associativos, adaptadores de contêineres, iteradores que percorrem contêineres genéricos e algoritmos que processam os elementos de contêineres genéricos.

Evoluindo de Pascal e C para C.+ e JavaTM
C++ substituiu C como a linguagem de implementação de sistemas preferida na indústria. Mas a programação em C continuará a ser uma habilidade importante e valiosa na próxima década por causa da quantidade enorme de código legado em C que deve ser mantido. O Dr. Harvey M. Deitei vem ministrando cursos de programação introdutórios em ambientes acadêmicos por duas décadas, com ênfase no desenvolvimento de programas claramente escritos e bem-estruturados. Muito do que é ensinado nestes cursos são os princípios básicos de programação com ênfase no uso efetivo de estruturas de controle e funções. Apresentamos este material exatamente do modo feito por HMD em seus cursos acadêmicos. Existem algumas armadilhas, mas, onde aparecem, nós as apontamos e explicamos procedimentos para lidar com elas eficazmente. Nossa experiência foi que os estudantes encaram o curso aproximadamente da mesma maneira que encaram cursos introdutórios de Pascal ou C. Existe uma diferença notável, no entanto: os estudantes estão altamente motivados pelo fato que eles estão aprendendo uma linguagem de ponta (C++) e um paradigma de programação de ponta (programação orientada a objetos) que serão

imediatamente úteis para eles assim que deixarem o ambiente acadêmico. Isto aumenta seu entusiasmo em relação ao material - uma grande ajuda quando você pensar que C÷+ é mais difícil de se aprender que Pascal ou C.
Nossa meta era clara: produzir um livro de ensino de programação C++ para cursos introdutórios de programação de computadores, de nível universitário, para estudantes com pouca ou nenhuma experiência em programação e, ainda assim, oferecer a profundidade e o tratamento rigoroso de teoria e prática exigidos por cursos tradicionais de C++ de nível mais avançado. Para atingir estas metas, produzimos um livro maior que outros textos sobre C++ - isto ocorre porque nosso texto também ensina pacientemente os princípios da programação procedural, da programação baseada em objetos, da programação orientada a objetos e da programação genérica. Centenas de milhares de pessoas estudaram este material em cursos acadêmicos e seminários profissionais a nível mundial.
Até o início da década de 90, cursos de ciência da computação estavam focalizados na programação estruturada em Pascal e C. Desde então, estes cursos voltaram-se amplamente para programação orientada a objetos em C++ e Java. Na Deitei & Associates mc., estamos focados na produção de materiais educacionais de qualidade para as linguagens de programação de ponta atuais. Enquanto C+ + Como Programar - Terceira Edição vai para impressão, estamos trabalhando em fava: How to Program - Fourth Edition, Advanced C++ How to Program e Advanced lava How to Program.
Introdução da orientação a objetos desde o Capítulo 1!
Enfrentamos um desalio ditícli ao projetar este livro. O livro deveria apresentar uma abordagem orientada a objetos pura? Ou deveria apresentar uma abordagem híbrida, balanceando programação procedural com programação orientada a objetos?
Muitos instrutores que vão ensinar a partir deste texto têm ensinado programação procedural (provavelmente
em C ou Pascal). C++ em si não é uma linguagem puramente orientada a objetos. Em vez disso, é uma linguagem híbrida que possibilita tanto a programação procedural como a programação orientada a objetos.
Assim, escolhemos a seguinte abordagem. Os primeiros cinco capítulos do livro introduzem a programação procedural em C++. Apresentam conceitos de computadores, estruturas de controle, funções, arrays, ponteiros e strings. Estes capítulos cobrem a “parte C” de C++ e as “melhorias na programação procedural” de C++ em relação aC.
Fizemos algo para tornar estes primeiros cinco capítulos realmente únicos no gênero. No fim de cada um destes capítulos, incluímos uma seção especial, intitulada “Pensando em objetos”. Estas seções introduzem os conceitos e a terminologia da orientação a objetos para ajudar os estudantes a começar a se familiarizar com o que são objetos e como se comportam.
A seção “Pensando em objetos” do Capítulo 1 introduz os conceitos e a terminologia da orientação a objetos. As seções nos Capítulos 2 a 5 apresentam uma especificação de requisitos para o projeto de um sistema significativo orientado a objetos, ou seja, construir um simulador de elevador, e guia

cuidadosamente o estudante através das fases típicas do processo de projeto orientado a objetos. Estas seções discutem como identificar os objetos em um problema, como especificar os atributos e comportamentos dos objetos e como especificar as

interações entre objetos. Quando o estudante tiver terminado o Capítulo 5, terá completado um cuidadoso projeto orientado a objetos do simulador de elevador e estará pronto - se não ansioso - para começar a programação do elevador em C++. Os Capítulos 6 e 7 cobrem a abstração de dados e classes. Estes capítulos também contêm seções “Pensando em objetos” que ajudam os estudantes através das várias fases da programação de seus simula- dores de elevador em C++. A seção “Pensando em objetos” do Capítulo 9 aplica os conceitos que herdou de C++ ao simulador de elevador.

Sobre este livro

C++ Como Programar contém uma rica relação de exemplos, exercícios e projetos retirados de muitos campos para oferecer ao estudante uma oportunidade de resolver problemas interessantes do mundo real. O livro se concentra nos princípios da boa engenharia de software e enfatiza a importância da clareza nos programas. Evitamos uma terminologia obscura e as especificações de sintaxe, preferindo o ensino por exemplos.

Este livro foi escrito por educadores que passam a maioria de seu tempo ensinando e escrevendo sobre linguagens de programação na vanguarda do “estado da prática”.

O texto coloca uma ênfase forte em pedagogia. Por exemplo, virtualmente todo conceito novo, tanto de C++ como de programação orientada a objetos, é apresentado no contexto de um programa em C++ completo, que funciona, imediatamente seguido por uma janela mostrando a saída do programa. Ler estes programas é muito semelhante a digitá-los e executá-los em um computador. Chamamos esta nossa abordagem de “código ativo”.

Entre as outras técnicas pedagógicas usadas no texto estão um conjunto de Objetivos e uma Visão Geral no início de cada capítulo; Erros Comuns de Programação, Boas Práticas de Programação, Dicas de Desempenho, Dicas de Portabilidade, Observações de Engenharia de Sojtware e Dicas de Teste e Depura ção, enumerados em cada capítulo e resumidos no fim dos mesmos; um Resumo abrangente em forma de lista de tópicos e uma seção de Terminologia em ordem alfabética em cada capítulo; Exercícios de Auto-Revisão e Respostas em cada capítulo; e a coleção mais rica de Exercícios disponível em qualquer livro sobre C++.

Os exercícios variam de perguntas simples de recordação até problemas de programação longos e projetos de porte. Os instrutores que necessitam de projetos significativos de conclusão de curso encontrarão muitos problemas apropriados listados nos exercícios para os Capítulos 3 a 21. Colocamos muito esforço nos exercícios, para aumentar a utilidade deste curso para o estudante.

Ao escrever este livro, usamos diversos compiladores C++. Na sua maioria, os programas do texto funcionarão em todos os compiladores ANSI/ISO.

Este texto está baseado na linguagem de programação C++ tal como desenvolvida pelo Accredited Standards Committee X3, Information Technology e seu Technical Committee X3J]6, Programming Lctnguage C++÷, respectivamente. Esta linguagem foi aprovada pela International Standards Organization (ISO). Para detalhes adicionais, entre em contato com:
X3 Secretariat
1250 Eye Street NW
Washington DC 20005, EUA
Um programador sério deveria ler estes documentos cuidadosamente e usá-los como referência regularmente. Estes documentos não são tutoriais. Em vez disso, definem C++ e C com o nível de precisão extraordinário que os implementadores de compiladores e desenvolvedores "industriais" exigem. Auditamos cuidadosamente nossa apresentação contra estes documentos. Nosso livro foi planejado para ser
usado nos níveis introdutórios e intermediários. Não tentamos cobrir todas as características discutidas nestes documentos abrangentes.
Objetivos
Cada capítulo começa com uma exposição de objetivos. Esta diz ao estudante o que esperar e dá ao mesmo uma oportunidade, depois de ler o capítulo, de determinar se atingiu estes objetivos. E um construtor de alta confiança e uma fonte de estímulo positivo.

Citações
Os objetivos do aprendizado são seguidos por uma série de citações. Algumas são humorísticas, algumas são filosóficas e algumas propiciam a percepção de conceitos interessantes. Nossos estudantes apreciam relacionar as citações ao material do capítulo. Você pode apreciar melhor algumas das citações depois de ler os capítulos.
Visão geral
A visão geral do capítulo ajuda o estudante a abordar o material de "cima para baixo (top-down)". Isto também auxilia os estudantes a antecipar o que está por vir e estabelecer um ritmo confortável e eficiente de aprendizado.
Seções
Cada capítulo é organizado em pequenas seções que abordam tópicos-chave de C++.
13.741 linhas de código em 268 programas exemplo (com as saídas dos programas)
Os recursos de C++ são apresentados no contexto de programas completos em C++ que funcionam. Cada programa é imediatamente seguido por uma janela contendo a saída produzida quando o programa é executado - chamamos isso de nossa "abordagem com código ativo". Isto possibilita ao estudante confirmar que os programas são executados conforme esperado. Relacionar a saída aos comandos do programa que produzem as saídas é uma maneira excelente de aprender e reforçar conceitos. Nossos programas exercitam os diversos recursos de C++. Ler o livro cuidadosamente se assemelha muito a digitar e

executar estes programas em um computador.
469 Ilustrações/figuras
Foram incluídos quadros e desenhos em abundância. A discussão de estruturas de controle no Capítulo 2 apresenta fluxogramas cuidadosamente desenhados. (Nota: não ensinamos o uso de fluxogramas como uma ferramenta de desenvolvimento de programas, mas usamos breves apresentações apoiadas em fluxogramas para especificar a operação precisa das estruturas de controle de C++). O Capítulo 15, “Estruturas de dados”, utiliza desenhos para ilustrar a criação e a manutenção de listas encadeadas, filas, pilhas e árvores binárias. O resto do livro é fartamente ilustrado.
625 Dicas de programação
Incluímos seis elementos de projeto para ajudar os estudantes a enfocar aspectos importantes do desenvolvimento de programas, teste e depuração, desempenho e portabilidade. Destacamos centenas destas dicas na forma de Boas práticas de programação, Erros comuns de programação, Dicas de desempenho, Dicas de portabilidade, Observações de engenharia de software e Dicas de teste e depura ção. Estas dicas e práticas representam as melhores que pudemos compilar em quase seis décadas (combinadas) de experiência de programação e ensino. Um de nossos estudantes - uma especialista em matemática - disse-nos recentemente que ela acha que esta abordagem é algo semelhante a enfatizar axiomas, teoremas e corolários em livros de matemática; fornece uma base sobre a qual se pode construir software de qualidade.
115 Boas práticas de programação
Boas práticas de programação são destacadas no texto. Elas chamam a atenção do estudante para técnicas que ajudam a produzir programas melhores. Quando damos cursos introdutórios a não-programadores, afirmamos que o “lema” de cada curso é “clareza” e dizemos aos estudantes que destacaremos (nestas Boas práticas de programação) as técnicas para escrever programas que sejam mais claros, mais compreensíveis e de manutenção mais fácil.
216 Erros comuns de programação
Os estudantes que estão aprendendo uma linguagem - especialmente em seu primeiro curso de programação - tendem a cometer certos tipos de erros freqüentemente. Chamar a atenção dos estudantes para estes Erros comuns de programação ajuda os estudantes a evitar cometer os mesmos erros. Também ajuda reduzir as longas filas do lado de fora dos salas dos instrutores durante seu horário de trabalho!
87Dicas de desempenho
f Em nossa experiência, ensinar os estudantes a escrever programas claros e compreensíveis é sem dúvida a
meta mais importante de um primeiro curso de programação. Mas os estudantes querem escrever progra

mas que sejam executados o mais rápido possível, usem menos memória, necessitem do mínimo de digitação para escrevê-los ou impressionem as pessoas de várias outras maneiras. Os estudantes realmente se importam com

o desempenho. Eles querem saber o que podem fazer para “turbinar” seus programas. Assim, incluímos Dicas de desempenho para destacar as oportunidades de melhoria do desempenho de um programa.
37 Dicas de portabilidade
O desenvolvimento de software é uma atividade complexa e cara. As organizações que desenvolvem software necessitam freqüentemente produzir versões customizadas para uma diversidade de computadores e sistemas operacionais. Assim, existe hoje em dia uma forte ênfase na portabilidade, i.e., na produção de software executável em diversos sistemas de computador com pouca, ou nenhuma, alteração. Muitas pessoas aclamam C÷+ como uma linguagem apropriada para desenvolver software portável, especialmente por causa do relacionamento estreito de C++ com ANSI/ISO C e pelo fato de que o ANSI/ISO C++ é o padrão global para C++. Algumas pessoas supõem que, se elas implementarem um aplicativo em C++. o aplicativo será automaticamente portável. Este simplesmente não é o caso. A obtenção de portabilidade exige um projeto cuidadoso e cauteloso. Existem muitas armadilhas. Incluímos numerosas Dicas de portabilidade para ajudar os estudantes a escrever código portável.
146 Observações de engenharia de software
O paradigma de programação orientada a objetos exige um completo repensar do modo como construímos sistemas de software. C++ é uma linguagem adequada para praticar a boa engenharia de software. As Observações de engenharia de software destacam técnicas, assuntos relativos a arquitetura e assuntos relativos ao projeto, etc., que afetam a arquitetura e a construção de sistemas de software, especialmente grandes sistemas. Muito do que o estudante aprender aqui será útil em cursos de nível superior e na indústria, à medida que o estudante começar a trabalhar com sistemas grandes e complexos do mundo real.
27 Dicas de teste e depura ção
Este “tipo de dica” pode estar mal-nomeado. Quando decidimos incorporar as Dicas de teste e depura ção
a esta nova edição, pensávamos que estas dicas seriam sugestões para testar programas e expor erros (bugs)
e sugestões para remover aqueles erros. De fato, a maioria destas dicas tendem a ser observações sobre as
capacidades e recursos de C++ que, antes de mais nada, evitam a introdução de erros nos programas.
Resumo
Cada capítulo termina com recursos pedagógicos adicionais. Apresentamos um Resumo extenso do capítulo, no
estilo de uma lista de tópicos, em todos os capítulos. Isto ajuda o estudante a revisar e reforçar conceitos-chave.
Existem em média 37 tópicos no resumo de cada capítulo.
Terminologia
Incluímos uma seção de Terminologia com uma lista dos termos importantes definidos no capítulo em ordem alfabética - também neste caso trata-se de reforço adicional. Existem em média 72 termos por capítulo.

Resumo das dicas, práticas e erros
Coletamos e listamos as Boas práticas de programação, os Erros comuns de programação, as Dicas de desempenho, as Dicas de portabilidade, as Observações de engenharia de soflware e as Dicas de teste e depura ção do capítulo.
554 Exercícios de auto-revisão e respostas (a contagem inclui partes separadas)

Numerosos Exercícios de auto-revisão e Respostas aos exercícios de auto-revisão são incluídos para estudo individual. Isto dá ao estudante uma oportunidade para ganhar confiança com o material e se preparar para tentar os exercícios regulares.
877 Exercícios (a contagem inclui partes separadas; 1431 exercícios no total)
Cada capítulo termina com um conjunto significativo de exercícios, inclusive uma recordação simples da terminologia e conceitos importantes; escrever comandos de C++ individuais; escrever partes pequenas de funções em C++ e

ligita- classes; escrever funções, classes e programas completos em C++; e desenvolver projetos de conclusão de curso de nte se porte significativo. O grande número de exercícios possibilita aos instrutores adaptar seus cursos às necessidades amas. peculiares de cada um de seus públicos e variar as tarefas dos cursos a cada semestre. Os instrutores podem usar de um estes exercícios para elaborar lições de casa, pequenos testes e exames finais.
Manual do instrutor de 550 páginas com soluções para os exercícios
As soluções para os exercícios estão incluídas no CD do Instrutor. [NOTA: por favor, não nos escreva pedindo o
CD do instrutor. A distribuição deste CD é estritamente limitada a professores do meio acadêmico que ensi ve nam usando o livro. Os instrutores podem obter o manual de soluções somente através de seus representantes
puta- Bookman.1 As soluções para aproximadamente metade dos exercícios estão incluídas 110 CD C & C+ + Multimedia Cvber Classroom: Third Edition (Prentice Hall dos EUA).
Mui ecial
4523 entradas de índice (total de 7653 contando referências múltiplas)
++ Incluímos um extenso índice no final do livro. Ajuda o estudante a encontrar qualquer termo ou conceito por palaldd vra-chave. O Índice é útil para as pessoas que estão lendo o livro pela primeira vez e é especialmente útil para i a e programadores praticantes que usam o livro como referência. A maioria dos termos nas seções de Terminologia
7orta- aparecem no Índice (junto com muitos outros itens de cada capítulo). Assim, o estudante pode usar o Índice junto com as seções de Terminologia para certificar-se de que cobriu o material-chave de cada capítulo.
- Um passeio pelo livro
Lumos
re. As O livro está dividido em várias partes principais. A primeira parte, os Capítulos 1 a 5, apresenta um tratamento

untos completo da programação procedural em C++, incluindo tipos de dados, entrada/saída, estruturas de controle, fun ment ções, arrays, ponteiros e strings.
A seção “Pensando em objetos” nos finais dos Capítulos 1 a 5 introduz a tecnologia
ndús- de objetos e apresenta um caso de estudo opcional interessante e desafiador para projetar e implementar um sistema al. orientado a objetos de porte substancial.
A segunda parte, os Capítulos 6 a 8, apresenta um tratamento substancial da abstração de dados com classes,
objetos e sobrecarga de operadores. Esta seção poderia ser efetivamente chamada “Programando com objetos”. As seções “Pensando em objetos” nos finais dos Capítulos 6 e 7 desenvolvem e apresentam um programa C++ com raçao mais de 1000 linhas que implementa o projeto apresentado nos Capítulos 2 a 5.
bugs) A terceira parte, os Capítulos 9 e 10, apresenta herança, as funções virtuais e o polimorfismo - as tecnologias
bre as básicas da verdadeira programação orientada a objetos.
5. A seção “Pensando em objetos” no final do Capítulo 9 incorpora herança no projeto e na implementação do
simulador de elevador.
A quarta parte, os Capítulos 11 a 14, apresenta o estilo C++ de entrada/saída orientada a streams - incluindo
lo, no o uso de EIS em stream pelo teclado, pela tela do monitor de vídeo, com arquivos e com arrays de caracteres; são have. discutidos tanto o processamento de arquivos seqüenciais como de acesso direto (i.e., acesso aleatório).
A quinta parte, os Capítulos 12 e 13, discute duas das mais recentes adições principais a C++, quais sejam,
gabaritos e o tratamento de exceções. Os gabaritos, também chamados tipos parametrizados, estimulam a reusabilidade de software. Algumas exceções ajudam os programadores a desenvolver sistemas mais robustos, resistentes a Ifabe- falhas e para negócios e missões críticas.
A sexta parte, o Capítulo 15, apresenta um tratamento completo de estruturas de dados dinâmicas, tais
como listas encadeadas, filas, pilhas e árvores. Este capítulo, quando suplementado com o estudo da biblioteca padrão de gabaritos (STL) no Capítulo 20, cria um rico tratamento de estruturas de dados que compõe um
enho, agradável suplemento em C++ aos cursos tradicionais de estruturas de dados e de algoritmos do currículo de ilo. Ciência da Computação.
A sétima parte, os Capítulos 16 a 18, discute uma variedade de tópicos incluindo a manipulação de bits,
caracteres e strings, o pré-processador e uma miscelânea de “outros tópicos”.
ndivi- A última parte do texto principal, os Capítulos 19 a 21, é dedicada às melhorias mais recentes de C++ e da tar biblioteca padrão de C++ que foram incluídos no padrão ANSIJISO C++. Estão incluídas discussões da classe string, do processamento de strings em srreams, a biblioteca padrão de

gabaritos e uma apresentação variada de outras adições recentes a C++.
O assunto final do livro consiste em materiais de referência que suportam o texto principal, incluindo Apêndiinolo- ces sobre precedência de operadores, o conjunto de caracteres ASCII, sistemas de numeração (binário, decimal, ,++ e

octal e hexadecimal) e recursos para C++ disponíveis na InternetlWorld Wide Web. Uma bibliografia extensa é incluída para encorajar uma leitura adicional dos temas. O texto termina com um índice detalhado que ajuda o leitor a localizar quaisquer termos no texto por palavra-chave. Agora vamos olhar cada um dos capítulos em detalhe.
Capítulo 1 - Introdução aos computadores e à programação em C++ - discute o que são computadores, como funcionam e como são programados. Introduz a noção de programação estruturada e explica por que este conjunto de técnicas conduziu a uma revolução no modo como são escritos os programas. O capítulo apresenta uma história breve do desenvolvimento de linguagens de programação, das linguagens de máquina às linguagem de montagem, até as linguagens de alto nível. A origem da linguagem de programação C++ é discutida. O capítulo inclui uma introdução a um ambiente de programação C++ típico e fornece uma introdução concisa sobre como escrever programas em C++. E apresentado um tratamento detalhado de tomada de decisões e operações aritméticas em C++. Depois de estudar este capítulo, o estudante compreenderá como escrever programas simples, mas completos, em C+÷. Discutimos a explosão do interesse pela Internet que aconteceu com o advento da World Wide Web e a linguagem de programação Java. Discutimos nainespace e o comando using para o benefício dos leitores com acesso a compiladores compatíveis com o padrão. Usamos os arquivos de cabeçalho no novo estilo. Levará alguns anos para “limpar” os compiladores mais velhos que ainda estão sendo amplamente usados. Os leitores mergulham direto na orientação a objetos na seção “Pensando em objetos”, que introduz a terminologia básica da tecnologia de objetos.
Capítulo 2 - Estruturas de controle - introduz a noção de algoritmos (procedimentos) para resolver problemas. Explica a importância de utilizar estruturas de controle eficazmente para produzir programas que sejam compreensíveis, depuráveis, de manutenção mais fácil e mais prováveis de funcionarem corretamente na primeira tentativa. Introduz a estrutura de seqüência, as estruturas de seleção (if, iflelse e switch) e as estruturas de repetição (while, do/while e for). Examina a repetição em detalhes e compara as alternativas de laços (loops) controlados por contadores e por sentinelas. Explica a técnica de refinamento passo a passo, de cima para baixo, que é fundamental para a produção de programas corretamente estruturados, e apresenta um auxflio popular ao projeto de programas, o pseudo código. Os métodos e abordagens usados no Capítulo 2 são aplicáveis para o uso efetivo de estruturas de controle em qualquer linguagem de programação, não apenas em C++. Este capítulo ajuda o estudante a desenvolver bons hábitos de programação, como

preparação para enfrentar as tarefas de programação mais substanciais no restante do texto. O capítulo termina com uma discussão de operadores lógicos - && (e), 1 1 (ou) e (negação). O quadro de palavras-chave foi aumentado com as novas palavras-chave de C++ introduzidas em C++ padrão ANSI/ISO. Introduzimos o novo estilo do operador static cast. Este é mais seguro que o antigo estilo de coerção de C++ herdado de C. Acrescentamos o exercício sobre o problema de “Peter Minuit”, de maneira que os estudantes possam ver as maravilhas do juro composto - com o computador fazendo a maior parte do trabalho! Discutimos as novas regras de escopo para contadores de laço em laços for. Na seção “Pensando em objetos”, começamos a primeira fase de um projeto orientado a objetos (OOD, object-oriented design) para o simulador de elevador - identificando as classes necessárias para implementar o simulador. Também introduzimos o caso de uso da UML, diagramas de classes e objetos e os conceitos de associações, multiplicidade, composição, papéis e vínculos (links).
Capítulo 3 - Funções - discute o projeto e construção de módulos de programas. Os recursos de C++ relacionados com funções incluem funções da biblioteca padrão, funções definidas pelo programador, recursão e os recursos de chamadas por valor e chamadas por referência. As técnicas apresentadas no Capítulo 3 são essenciais para a produção de programas estruturados adequadamente, em especial os tipos de programas e software maiores que os programadores de sistema e os programadores de aplicativos provavelmente desenvolverão em aplicações do mundo real. A estratégia “dividir para conquistar” é apresentada como um meio efetivo de resolver problemas complexos dividindo-os em componentes mais simples que interagem entre si. Os estudantes apreciam o tratamento de números aleatórios e de simulação e apreciam a discussão do jogo de dados craps que faz um uso elegante das estruturas de controle. O capítulo oferece uma introdução sólida à recursão e inclui uma tabela resumindo as dezenas de exemplos e exercícios de recursão distribuídos ao longo do restante do livro. Alguns textos deixam a recursão para um capítulo mais à frente no livro; acreditamos que este tópico seja mais bem coberto gradualmente ao longo do texto. A relação extensa de 60 exercícios no fim do capítulo inclui vários problemas clássicos de recursão, tal como o das Torres de Hanoi. O capítulo discute as chamadas “melhorias de C++ em relação a C”, incluindo funções mime, parâmetros por referência, argumentos default, o operador unário de resolução de escopo, sobrecarga de funções e gabaritos de funções. O quadro de arquivos de cabeçalho foi modificado para incluir muitos dos novos arquivos de cabeçalho que

recompensá-lo com uma profunda compreensão do complexo tópico de ponteiros. Salientamos, mais uma vez, que cobrimos arrays e strings objetos no pleno sentido mais adiante no livro. No Capítulo 8, usamos a sobrecarga de operadores para elaborar classes personalizadas Array e String. No Capítulo 19, discutimos a classe string da biblioteca padrão e mostramos como manipular objetos string. No Capítulo 20, discutimos a classe vector para implementar

arrays como objetos. O Capítulo 5 está repleto de exercícios desafiadores. Não deixe de ler a Seção especial: construindo seu próprio computador Na seção “Pensando em objetos”, determinamos muitas das colaborações (interações entre objetos no sistema) necessárias para implementar o sistema do elevador e representamos estas colaborações usando o diagrama de colaboração da UML. Também incluímos uma bibliografia e uma lista de recursos da Internet e da World Wide Web que contém as especificações da UML 1.3 e outros materiais de referência de UML, recursos genéricos, tutoriais, FAQs, artigos, publicações e software.

Capítulo 6- Classes e abstração de dados - inicia nossa discussão da programação baseada em objetos. O capítulo representa uma oportunidade maravilhosa para ensinar a abstração de dados da “maneira certa” - através de uma linguagem (C+÷) expressamente dedicada a implementar tipos de dados abstratos (ADTs, abstract data types). Em anos recentes, a abstração de dados se tornou um dos tópicos principais nos cursos introdutórios de computação. Os Capítulos 6 a 8 incluem um tratamento sólido da abstração de dados. O Capítulo 6 discute a implementação de ADTs como structs, a implementação de ADTs como classes no estilo de C++ - e por que esta abordagem é superior a usar structs - o acesso a membros de class. a separação da interface da implementação, o uso de funções de acesso e funções utilitárias, a inicialização de objetos com construtores, a destruição de objetos com destruidores, a atribuição por cópia membro a membro default e a reusabilidade de software. Os exercícios do capítulo desafiam o estudante a desenvolver classes para números complexos, números racionais, horas, datas, retângulos, inteiros enormes e para jogar “jogo da velha”. Os estudantes geralmente apreciam programas de jogos. A seção “Pensando em objetos” lhe pede para escrever um arquivo de cabeçalho de classe para cada uma das classes em seu simulador de elevador. O leitor mais inclinado à matemática apreciará os exercícios sobre a criação da classe Complex (para números complexos), da classe Rational (para números racionais) e da classe HugeInteger (para inteiros arbitrariamente grandes). Na seção “Pensando em objetos”, usamos o diagrama de classes da UML desenvolvido nas seções anteriores para esboçar os arquivos de cabeçalho de C++ que definem nossas classes. Também introduzimos o conceito de handles para objetos no sistema e começamos a estudar como implementar handles em C++.

Capítulo 7- Classes: parte II - continua o estudo de classes e abstração de dados. O capítulo discute a declaração e uso de objetos constantes, funções membro constantes, composição - o processo de construir classes que têm objetos de outras classes como membros, funções friend e classes friend que têm direitos de acesso especiais a membros private e protected de classes, o ponteiro this. que possibilita a um objeto saber seu próprio endereço, a alocação dinâmica de memória, membros de classe static para armazenar e manipular dados usados em toda a classe, exemplos de tipos de dados abstratos populares (arrays, strings e filas), classes contêineres e iteradores. Os exercícios do capítulo pedem ao estudante para desenvolver uma classe conta de poupança e uma classe para armazenar conjuntos de inteiros. Em nossa

discussão de objetos const. mencionamos brevemente a nova palavra-chave mutable que, como veremos no Capítulo 21, é usada de uma maneira sutil para possibilitar a modificação de implementação “não-visível” em objetos const. Discutimos a alocação dinâmica de memória com new e delete. Quando new falha, retorna um ponteiro O no estilo de C++ antes da padronização. Usamos este estilo anterior ao padrão nos Capítulos 7 a 12. Adiamos para o Capítulo 13 a discussão do novo estilo de falha de new, em que new agora “dispara uma exceção”. Motivamos a discussão de membros de classe static com um exemplo baseado em videogame. Enfatizamos ao longo do livro e em nossos seminários profissionais como é importante esconder detalhes da implementação dos clientes de uma classe. Então, mostramos dados private em nossos cabeçalhos de classe, que certamente revelam a implementação. Introduzimos uma nova seção sobre classes proxy, um meio agradável de ocultar até mesmo dados private dos clientes de uma classe. A seção “Pensando em objetos” lhe pede para incorporar administração dinâmica de memória e composição ao seu simulador de elevador. Os estudantes apreciarão o exercício de criar a classe Integerset. Esta serve como uma excelente motivação para o tratamento da sobrecarga de operadores no Capítulo 8. Na seção “Pensando em objetos”, apresentamos um programa completo de simulador de elevador em C++ (aproximadamente 1000 linhas de código) e um walkthrough detalhado do código. O código é diretamente derivado do projeto baseado na UML criado em seções anteriores e emprega nossas boas práticas de programação, inclusive o uso de membros de dados e funções static e const. Também discutimos alocação dinâmica de memória, composição e interação entre objetos através de handles e como usar declarações antecipadas para evitar o problema de referências circulares em inclusões.

desenhados. Cada objeto sabe como desenhar a si próprio. Um novo objeto pode ser acrescentado ao programa sem modificar aquele programa, desde que aquele novo objeto também saiba desenhar a si próprio. Este estilo de programação é usado tipicamente para implementar as interfaces de usuário gráficas (GUIs, graphical user interfaces), hoje em dia populares. O capítulo discute a mecânica de obtenção do comportamento polimórfico através do uso de funções virtual. Distingue entre classes abstratas (das quais não podem ser instanciados objetos) e classes concretas (das quais podem ser instanciados objetos). Classes abstratas são úteis para fornecer uma interface que possa ser transmitida por herança às classes ao longo da hierarquia. Um destaque do capítulo são seus dois estudos de caso principais de polimorfismo - um sistema de folha de pagamento e outra versão da hierarquia de formas ponto, círculo e cilindro discutida no Capítulo 9. Os exercícios do capítulo pedem ao estudante para discutir vários assuntos e abordagens conceituais, acrescentar classes abstratas à hierarquia de formas, desenvolver um pacote de gráfico básico, modificar a classe empregado do capítulo - e desenvolver todos estes projetos com funções virtual e programação polimórfica. Os dois estudos de caso de polimorfismo do capítulo mostram um contraste nos estilos de herança. O

primeiro exemplo (de um sistema de folha de pagamento) é um uso claro, “sensato”, da herança. O segundo, que se baseia na hierarquia ponto, círculo e cilindro desenvolvida no Capítulo 9, é um exemplo do que alguns profissionais chamam de “herança estrutural” - não tão natural e sensata quanto a primeira - mas, “mecanicamente correta”. Usamos este segundo exemplo por causa da seção intitulada Polimorfismo, funções virtuais e vincula ção dinâmica “vistos por dentro “. Damos nossos seminários profissionais de C++ para engenheiros de software seniores. Estas pessoas apreciaram os dois exemplos de polimorfismo na primeira edição, mas sentiram que estava faltando algo em nossas apresentações. Sim, disseram, mostramos a eles como programar com polimorfismo em C++. Mas eles queriam mais. Eles nos disseram que estavam preocupados com a sobrecarga operacional resultante de reprogramar polimorficamente. E um recurso desejável, disseram, mas é claro que ele tem um custo. Assim, nosso público profissional insistiu que fornecêssemos uma explicação mais profunda que mostrasse precisamente como o polimorfismo é implementado em C++ e, como consequência, precisamente que “custos”, em termos de tempo de execução e de memória, devemos pagar quando programarmos com este poderoso recurso. Respondemos a eles desenvolvendo uma ilustração que mostra as vtables (tabelas das funções virtual) que o compilador de C++ constrói automaticamente para suportar o estilo de programação polimórfico. Desenhamos estas tabelas em nossas classes em que discutimos a hierarquia de formas ponto, círculo e cilindro. Nosso público nos disse que isto realmente lhes deu as informações para decidir se o polimorfismo é um estilo de programação apropriado a cada novo projeto que viessem apegar. Incluímos esta apresentação na Seção 10.10 e a ilustração da vtable na Fig. 10.2. Estude esta apresentação cuidadosamente. Ela lhe dará uma compreensão muito mais profunda do que realmente está acontecendo no computador quando você programa com herança e polimorfismo.
Capítulo 11 - Entrada/saída com streams em C++ - contém um tratamento abrangente do novo estilo de entrada! saída orientado a objetos introduzido em C++. O capítulo discute os vários recursos de E/S de C++, incluindo a saída com o operador de inserção em stream, a entrada com o operador de extração de stream, E/S segura quanto a tipos (uma agradável melhoria em relação a C), E/S formatada, E/S não-formatada (para melhor desempenho), manipuladores de stream para controlar a base do stream (decimal, octal ou hexadecimal), números de ponto flutuante, o controle dos comprimentos de campos, manipuladores definidos pelo usuário, estados de formato do stream, estados de erro do stream, E/S de objetos de tipos definidos pelo usuário e a vinculação de streams de saída a streams de entrada (para assegurar que os prompts realmente apareçam antes do momento em que se espera que o usuário digite respostas). O conjunto extenso de exercícios pede ao estudante para escrever vários programas que testam a maioria dos recursos de EIS discutidos no texto.
Capítulo 12 - Gabaritos - discute uma das mais recentes adições à linguagem C ++. Gabaritos de funções foram introduzidos no Capítulo 3. O Capítulo 12 apresenta um exemplo adicional de gabarito de função. Os gabaritos de classes possibilitam ao programador capturar a essência de um tipo de dados abstrato -

ADT - (tal como uma pilha, um array ou uma fila) e então criar - com um mínimo de código adicional - versões daquele ADT para tipos particulares (tal como uma fila de ints, uma fila de floats, uma fila de strings, etc.). Por essa razão, classes gabarito são freqüentemente chamadas de tipos parametrizados. O capítulo discute o uso de parâmetros de tipo e parâmetros não de tipo e considera a interação entre gabaritos e outros conceitos de C++, tais como herança, friends e membros static. Os exercícios desafiam o estudante a escrever uma variedade de gabaritos de funções e gabaritos de classes e a empregar estes em programas completos. Aumentamos muito o tratamento de gabaritos com a discussão dos contêineres, iteradores e algoritmos da biblioteca padrão de gabaritos (STL) no Capítulo 20.

PREFÁCIO XXI

Capítulo 13- Tratamento de exceções - discute uma das mais recentes melhorias da linguagem C++. O tratamento de exceções possibilita ao programador escrever programas que são mais robustos, mais tolerantes a falhas e mais apropriados para ambientes de negócios críticos e missões críticas. O capítulo discute quando o tratamento de exceções é apropriado; introduz os fundamentos do tratamento de exceções com blocos try, comandos throw e blocos catch; indica como e quando “disparar novamente” uma exceção; explica como escrever uma especificação de exceção e processar exceções inesperadas; e discute os importantes vínculos entre exceções e construtores, destruidores e herança. Um destaque do capítulo são seus 43 exercícios que orientam o estudante na implementação de programas que ilustram a diversidade e o poder dos recursos de tratamento de exceções de C++. Discutimos disparar novamente uma exceção e ilustramos os dois modos em que new pode falhar quando a memória se esgota. Antes do novo padrão para C ++. new falhava retornando O, de forma semelhante à que nialloc falha em C retornando um valor de ponteiro NULL. Mostramos o novo estilo de new de falhar, disparando uma exceção badalloc (alocação ruim). Ilustramos como usar set newhandler para especificar uma função customizada que deve ser chamada para lidar com situações de esgotamento de memória. Discutimos o gabarito de classe autoytr para garantir que a memória dinamicamente alocada será corretamente deletada, para evitar perdas de memória. Apresentamos a nova hierarquia de exceções da biblioteca padrão.

Capítulo 14- Processamento de arquivos - discute as técnicas usadas para processar arquivos de texto com acesso seqüencial e acesso aleatório, O capítulo começa com uma introdução à hierarquia de dados de bits, bytes, campos, registros e arquivos. Em seguida, é apresentada a visão simples de arquivos e streams de C++. Os arquivos de acesso seqüencial são discutidos usando-se programas que mostram como abrir e fechar arquivos, como armazenar dados seqüencialmente em um arquivo e como ler dados seqüencialmente de um arquivo. Arquivos de acesso aleatório são discutidos usando-se programas que mostram como criar seqüencialmente um arquivo para acesso aleatório, como ler e escrever dados em um arquivo com acesso aleatório e como ler dados seqüencialmente de um arquivo acessado

aleatoriamente. O quarto programa de acesso aleatório combina muitas das técnicas de acesso a arquivos, tanto seqüencial como aleatoriamente, em um programa de processamento de transações completo. Estudantes em nossos seminários em empresas nos disseram que, após estudar o material sobre processamento de arquivos, eles podiam produzir programas de processamento de arquivos significativos, que eram imediatamente úteis em suas organizações. Os exercícios pedem ao estudante para implementar diversos programas que constroem e processam tanto arquivos de acesso seqüencial como arquivos de acesso aleatório. O material relacionado ao processamento de strings em streams foi posicionado no fim do Capítulo 19.

Capítulo 15 - Estruturas de dados - discute as técnicas usadas para criar e manipular estruturas de dados dinâmicas. O capítulo começa com discussões de classes com auto-referência e alocação dinâmica de memória e prossegue com uma discussão sobre como criar e manter várias estruturas de dados dinâmicas, incluindo listas encadeadas, filas (ou linhas de espera), pilhas e árvores. Para cada tipo de estrutura de dados, apresentamos programas completos, que funcionam, e mostramos amostras de suas saídas. O capítulo realmente ajuda o estudante a dominar ponteiros. O capítulo inclui exemplos abundantes, usando indireção (acesso indireto) e dupla indireção - um conceito particularmente difícil. Um problema que ocorre quando se trabalha com ponteiros é que os estudantes têm dificuldade de visualizar as estruturas de dados e como seus nodos são interligados. Assim, incluímos ilustrações que mostram os links e a seqüência em que são criados. O exemplo de árvore binária é um belo ponto de fechamento para o estudo de ponteiros e estruturas dinâmicas de dados. Este exemplo cria uma árvore binária; garante a eliminação de duplicatas; e introduz percursos recursivos da árvore em pré-ordem, em ordem e pós-ordem. Os estudantes têm uma genuína sensação de realização quando estudam e implementam este exemplo. Particularmente, gostam de ver que o percurso em ordem imprime os valores dos nodos em uma ordem classificada. O capítulo inclui uma coleção substancial de exercícios. Um destaque dos exercícios é a seção especial “Construindo seu próprio compilador”. Os exercícios encaminham o estudante por todo o desenvolvimento de um programa de conversão de notação in-fixa para pós-fixa e um programa de avaliação de expressões pós-fixas. Então, modificamos o algoritmo de avaliação de expressões pós-fixas para gerar código em linguagem de máquina. O compilador coloca este código em um arquivo (usando as técnicas do Capítulo 14). Os estudantes então executam a linguagem de máquina produzida por seus compiladores nos simuladores de software que eles construíram nos exercícios do Capítulo 5! Os 67 exercícios incluem uma simulação de supermercado usando filas, uma busca recursiva em uma lista, uma impresão recursiva de uma lista de trás para diante, a exclusão de um nodo de uma árvore binária, um percurso em ordem de nível de uma árvore binária, impressão de árvores, escrever uma parte de um compilador otimizador, escrever um

interpretador, inserção/deleção em qualquer lugar em uma lista encadeada,

implementação de listas e filas sem ponteiros de cauda, análise do desempenho da busca e classificação de uma árvore binária e implementação de uma classe lista indexada. Depois de estudar o Capítulo 15, o leitor está preparado para o tratamento de contêineres, iteradores e algoritmos da STL, no Capítulo 20. Os contêineres da STL são estruturas de dados parametrizadas pré- empacotadas, que a maioria dos programadores achará suficientes para a maioria dos aplicativos que necessitarão implementar. A STL é um salto gigante para se visualisar a abordagem de reusar, reusar, reusar.
Capítulo 16- Bits, caracteres, strings e estruturas - apresenta diversos recursos importantes. Os poderosos recursos de manipulação de bits de C++ possibilitam aos programadores escrever programas que utilizam recursos de hardware em nível mais baixo. Isto ajuda os programadores a processar strings de bits, ligar ou desligar bits individuais e armazenar informações mais compactamente. Tais recursos, freqüentemente encontrados apenas em linguagem de montagem de baixo nível, são valiosos para os programadores que estão escrevendo software de sistema, tais como sistemas operacionais e software de rede. Como você se lembra, introduzimos a manipulação de strings char* ao estilo de C no Capítulo 5 e apresentamos as funções de manipulação de strings mais populares. No Capítulo 16, continuamos nossa apresentação de strings de caracteres e char* ao estilo de C. Apresentamos os vários recursos de manipulação de caracteres da biblioteca <cctype> - estes incluem a possibilidade de testar um caractere para ver se ele é um dígito, um caractere alfabético, um caractere alfanumérico. um dígito hexadecimal, uma letra minúscula, uma letra maiúscula, etc. Apresentamos as demais funções de manipulação de strings das várias bibliotecas relacionadas com strings; como sempre, toda função é apresentada no contexto de um programa em C++ completo e que funciona. Estruturas são como registros em Pascal e outras linguagens - elas agregam itens de dados de vários tipos. São usadas no Capítulo 14 para formar arquivos que consistem de registros de informações. São usadas em conjunto com ponteiros e a alocação dinâmica de memória, no Capítulo 15 para formar estruturas de dados dinâmicas, tais como listas encadeadas, filas, pilhas e árvores. Um destaque do capítulo é sua simulação revisada, de alta performance, do embaralhamento e distribuição de cartas. Esta é uma oportunidade excelente para o instrutor enfatizar a qualidade dos algoritmos. Os 38 exercícios incentivam o estudante a pôr à prova a maioria dos recursos discutidos no capítulo. O exercício especial conduz o estudante através do desenvolvimento de um programa de revisão ortográfica. Os Capítulos 1 a 5 e 16 a 18 tratam principalmente da parte de C÷+ “herdada de C”. Em particular, este capítulo apresenta um tratamento mais profundo de strings char* ao estilo de C, para benefício dos programadores de C ++ que provavelmente trabalharão com código legado em C. Lembramos ainda uma vez que o Capítulo 19 discute a classe string e a manipulação de strings como objetos no pleno sentido da orientação a objetos.
Capítulo 17 - O pré-processador - fornece discussões detalhadas das diretivas do pré-processador. O capítulo inclui informações mais completas sobre a diretiva #include. que faz com que uma cópia de um arquivo especificado seja incluída em lugar da diretiva antes de o arquivo ser compilado, e a diretiva

#define, que cria constantes e macros simbólicas. O capítulo explica a compilação condicional para possibilitar ao programador controlar a execução de diretivas do pré-processador e a compilação do código do programa. São discutidos o operador #, que converte seu operando em um string, e o operador ##. que concatena duas “unidades léxicas”. São apresentadas as várias constantes simbólicas pré-definidas do pré-processador LLINE , FILE, DATE e TIMEJ. Finalmente, a macro assert do arquivo de cabeçalho assert. h é discutida; assert é de grande valor em testes, depuração, verificação e validação de programas. Usamos assert em muitos exemplos, mas o leitor é fortemente aconselhado a começar a usar o tratamento de exceções em seu lugar, como discutimos no Capítulo 13.
Capítulo 18-Tópicos sobre código legado em C - apresenta tópicos adicionais, incluindo vários tópicos avançados normalmente não cobertos em cursos introdutórios. Mostramos como redirecionar a entrada de um programa para vir de um arquivo, redirecionar a saída de um programa para ser colocada em um arquivo, redirecionar a saída de um programa para ser fornecida como entrada para outro programa (piping), anexar a saída de um programa a um arquivo existente, desenvolver funções que usam listas de argumentos de comprimento variável, passar argumentos através da linha de comando para a função main e usá-los em um programa, compilar programas cujos componentes estão espalhados em diversos arquivos, registrar funções com atexit para serem executadas no término do programa, terminar a execução do programa com a função exit, usar os qualificadores de tipo const e volatile, especificar o tipo de uma constante numérica usando os sufixos de inteiro e ponto flutuante, usar a biblioteca de manipulação de sinalizadores para capturar eventos inesperados, criar e usar arrays dinâmicos com calloc e realloc, usar unions como uma técnica de economizar espaço e usar especificações de “ligação” quando pro

através de conversões implícitas. Discutimos a palavra-chave mutable, que permite que um membro de um objeto const seja alterado. Anteriormente, isto era realizado fazendo-se uma coerção para “retirar a característica de const”, uma prática perigosa. Também discutimos alguns recursos que não são novos, mas que optamos por não incluir na parte principal do livro porque são relativamente obscuros, quais sejam: operadores ponteiros para membros * e -> e o uso de classes base virtual com herança múltipla.
Apêndice A - Tabela de precedência de operadores - reformatamos a tabela para ser mais útil. Cada operador está agora em uma linha própria com o símbolo do operador, seu nome e sua associatividade.
Apêndice B - Conjunto de caracteres ASCII - resistimos à tentação de expandir este apêndice substancialmente para incluir o relativamente novo conjunto internacional de caracteres Unicode. Na próxima edição, esperamos discutir o Unicode em detalhes.
Apêndice C - Sistemas de numeração - discute os sistemas de numeração binário, octal, decimal e hexadecimal. Examina como converter números entre

bases e explica as representações binárias em complemento de um e de dois.
Apêndice D - Recursos sobre C++ na Internet e na Web - contém uma listagem enorme de recursos de C++ valiosos, tais como demonstrações, informações sobre compiladores populares (incluindo gratuitos), livros, artigos, conferências, bancos de ofertas de emprego, diários, revistas, ajudas, tutoriais, FAQs (perguntas feitas freqüentemente), grupos de notícias, cópias do documento padrão C÷+ ANSI/ISO, cursos baseados na Web, notícias sobre produtos e ferramentas de desenvolvimento em C++.
Bibliografia - lista 125 livros e artigos - alguns de interesse histórico e a maioria bastante recente - para incentivar o estudante a fazer leituras adicionais sobre C ++ e OOP.
Índice - o livro contém um índice abrangente para possibilitar ao leitor localizar por palavra-chave qualquer termo ou conceito no texto.
Agradecimentos
Um dos grandes prazeres de escrever um livro de ensino é agradecer os esforços de muitas pessoas cujos nomes não podem aparecer na capa, mas cujo trabalho duro, cooperação, amizade e compreensão foram cruciais para a produção do livro.
Muitas outras pessoas na Deitel & Associates, Inc. dedicaram longas horas a este projeto.
Tem Nieto, um diplomado do Massachusetts Institute ofTechnology, é um de nossos colegas em tempo integral na Deitei & Associates, mc. e recentemente foi promovido a Director of Product Development. Ele ministra seminários de C+ +, C e Java e trabalha conosco na redação de livros-texto e nos esforços de desenvolvimento de cursos e criação de material multimídia. Tem foi co-autor do Capítulo 19, do Capítulo 21 e da seção especial intitulada “Construindo seu próprio compilador” no Capítulo 15. Ele também fez contribuições para o Manual do Instrutor e para a C+ + Multimedia Cyber Classroom:
Third Edition.
• Barbara Deitei administrou a preparação do manuscrito e coordenou junto à Prentice Hail todos os esforços relacionados à produção do livro. Os esforços de Barbara são sem dúvida os mais esmerados dos que fazemos para desenvolver livros. Ela tem uma paciência infinita. Manipulou os infindáveis detalhes relativos à publicação de um livro de 1.100 páginas, um manual do instrutor de 550 páginas e o CD de 650 megabytes da C++ Multimedia Cyber Classroom. Passou longas horas pesquisando as citações no princípio de cada capítulo. E fez tudo isso em paralelo com suas vastas responsabilidades financeiras e administrativas na Deitel & Associates. Inc.
• Abbey Deitei, uma diplomada do programa de administração industrial da Carnegie Melion University e agora Presidente e Diretora de Marketing Internacional na Deitel & Associates, Inc., escreveu o Apêndice D e sugeriu o título para o livro. Pedimos que Abbey navegasse a World Wide Web e procurasse os melhores sites de C++. Utilizou todos os principais mecanismos de busca da Web e reuniu estas informações

para você no Apêndice D. Para cada recurso e demonstração, Abbey forneceu uma breve explicação. Rejeitou centenas de sites e listou para você os melhores que ela pôde encontrar. Abbey estará mantendo a listagem destes recursos e demonstrações em nosso site da Web www. deitei com. Envie a ela as URLs para seus sites favoritos, por e-mau para o endereço deitei@deitei com, e ela colocará links para estes sites em nosso próprio site.

O grupo de estudantes estagiários na Deitel & Associates, mc. que trabalhou neste livro inclui:

• Ben Wiedermann - um estudante de Ciência da Computação da Boston University - foi o desenvolvedor, programador e escritor líder, trabalhando com o Dr. Harvey M. Deitei no estudo de caso da UML. Desejamos reconhecer o extraordinário comprometimento e contribuições de Ben para este projeto.

• Sean Santry - um graduado em Ciência da Computação e filosofia pelo Boston Coilege - trabalhou na codificação e nos walkthroughs de código do estudo de caso da UML. Sean juntou-se à Deitei & Associates, lnc. em tempo integral e está trabalhando como líder de desenvolvimento com Paul Deitei em nosso futuro livro, Advanced Java How to Program.

• Biake Perdue - um estudante de Ciência da Computação da Vanderbilt University - ajudou a desenvolver o estudo de caso da UML.

• Kalid Azad - um estudante de Ciência da Computação da Princeton University - trabalhou extensamente no material auxiliar do livro, incluindo as notas de aula para o instrutor em PowerPoint e o banco de testes.

• Aftab Bukhari -um estudante de Ciência da Computação da Boston University - executou verificação e testes extensos dos programas e trabalhou no material auxiliar do livro, incluindo as Notas de Aula para o Instrutor em PowerPoint e o Manual do Instrutor.

• Jason Rosenfeld - um estudante de Ciência da Computação da Northwestern University - trabalhou no material auxiliar do livro, incluindo o Manual do Instrutor.

• Melissa Jordan - uma estudante de projeto gráfico da Boston University - coloriu a arte final de todo o livro e criou diversas ilustrações originais.

• Rudolf Faust - um calouro da Stanford University - ajudou a criar o banco de testes.

Nós temos sorte de ter podido trabalhar neste projeto com um time talentoso e dedicado de profissionais de publicação na Prentice Hall. Este livro foi realizado por causa do encorajamento, entusiasmo e persistência de nossa editora de Ciência da Computação, Petra Recter, e sua chefe - a melhor amiga que tivemos em 25 anos de publicação - Marcia Horton, Editora Chefe da Divisão de Engenharia e Ciência da Computação da Prentice Hall. Camille Trentacoste fez um trabalho maravilhoso como gerente de produção. Sarah Burrows fez um trabalho maravilhoso com sua atuação tanto no processo de revisão quanto nos suplementos do livro.

A C++ Multimedia Cyber Classroom: Third Edition foi desenvolvida em paralelo com C++ Como Programar - Terceira Edição. Apreciamos sinceramente a perspicácia, compreensão e perícia técnica nas “novas mídias” de nosso editor Mark Taub e sua colega Karen McLean. Mark e Karen fizeram um trabalho

notável conseguindo publicar a C++ Multimedia Cyber Classroom: Third Edition dentro de um cronograma apertado. Eles estão, seguramente, entre os líderes mundiais em publicacação nas novas mídias.
Devemos um agradecimento especial à criatividade de Tamara Newnam Cavallo (smart- art@earthiink net), que fez o trabalho artístico para nossos ícones de dicas de programação e para a capa. Eia criou a criatura deliciosa que compartilha com você as dicas de programação do livro. Ajude-nos a dar um nome a este amável bichinho. Algumas primeiras sugestões: D. Bug, InterGnat, Ms. Kito, DeetleBug (um apelido infeliz dado ao “cara velho” no segundo grau) e Feature (“não é um bug, é uma feature”).
Queremos reconhecer os esforços dos revisores de nossa Terceira Edição e deixar uma nota especial de agradecimento a Crissy Statuto, da Prentice Hall, que administrou este extraordinário trabalho de revisão.
Revisores do material de C++
• Tamer Nassif (Motorola)
• Christophe Dinechin (Hewlett-Packard)
• Thomas Kiesler (Montgomery College)
• Mary Astone (Troy State University)
• Simon North (Synopsis)

1

• Harold Howe (Inprise)
• William Hasserman (University ofWisconsin)
• Phillip Wasserman (Chabot Coliege)
• Richard Albright (University of Delaware)
• Mahe Velauthapilla (Georgetown University)
• Chris Uzdavinis (Automated Trading Desk)
• Stephen Clamage (Chairman do Comitê de Padrões ANSI C++)
• Ram Choppa (Akili Systems; University of Houston)
• Wolfgang Peiz (University ofAkron)
Revisores do estudo de caso da I]ML
• Spencer Roberts (Titus Corporation)
• Don Kostuch (You Can C Clearly Now)
• Kendall Scott (Consultor independente; autor de UML)
• Grant Larsen (Blueprint Technologies)
• Brian Cook (Technical Resource Connection; OMG)
• Michael Chonoles (Chefe de Metodologia, Lockheed Martin Advanced Concepts; OMG)
• Stephen Tockey (Construx Software; OMG)
• Cameron Skinner (Advanced Software Technologies; OMG)
• Rick Cassidy (Advanced Concepts Center)
• Mark Contois (NetBeans)

• David Papurt (Consultor independente; professor e autor de C++)
• Chris Norton (AD2IT; consultor independente)
Desejamos reconhecer, novamente, os esforços de nossos revisores de edições anteriores (alguns da primeira edição, alguns da segunda edição e alguns de ambas):
• Richard Albright (University of Delaware)
• Ken Arnold (Sun Microsystems)
• lan Baker (Microsoft)
• Pete Becker (Membro do Comitê ANSI/ISO C++; Dinkumware LTD.)
• Timothy D. Bom (Delta C-Fax)
• John Carson (George Washington University)
• Steve Clamage (Chairman do Comitê de Padrões ANSJJISO C++; Sunsoft)
• Marian Corcoran (Membro do Comitê de Padrões ANSI/ISO C++)
• Edgar Crisostomo (Siemens/Roim)
• David Finkel (Worcester Polytechnic Institute)
• Rex Jaeschke (Chairman do Comitê ANSI/ISO)
• Frank Kelbe (Naval Postgraduate School)
• Chris Kelsey (Kelsey Associates)
• Don Kostuch (You Can C Clearly Now)
• Meng Lee (Co-criador da STL; Hewlett-Packard)
• Barbara Moo (AT&T Bell Labs)
• David Papurt (Consultor)
• Wolfgang Pelz (University ofAkron)
• Jandelyn Plane (University of Maryland Coilege Park)
• Paul Power (Borland)
• Kenneth Reek (Rochester Institute of Technology)
• Larry Rosler (Hewlett-Packard)
• Robin Rowe (HalyconlNaval Postgraduate School)
• Brett Schuchert (ObjectSpace; Co-Autor de STL Primer)
• Alexander Stepanov (Co-criador da STL; Silicon Graphics)
• William Tepfenhart (AT&T; Autor de UML and C++. A Practical Guide to Object-Oriented Development)

• David Vandevoorde (Membro do Comitê ANSI/ISO C++ Hewlett-Packarcl)
• Terry Wagner (University of Texas)
Dentro de prazos finais apertados, eles vasculharam todos os aspectos do texto e fizeram incontáveis sugestões para melhorar a precisão e perfeição da apresentação.
Apreciaríamos sinceramente seus comentários, críticas, correções e sugestões para melhorar o texto. Enderece toda correspondência para:
deitel@deitel com
Responderemos imediatamente. Bem, por agora é só. Bem-vindo ao mundo excitante de C+÷, da programação orientada a objetos, da UML e da programação genérica com a STL. Esperamos que você aprecie esta olhada na programação de computadores contemporânea. Boa sorte!

Dr. Harvey M. Deitei
Paul J. Deitei
Sobre os autores
Dr. Harvey M. Deitei, CEO de Deitei & Associates, mc., tem uma experiência de 39 anos no campo da computação, incluindo extensa experiência acadêmica e na indústria. Ele é um dos principais instrutores de Ciência da Computação e apresentadores de seminários do mundo. Dr. Deitei recebeu os graus de B.S. e M.S. do Massachusetts Institute of Technology e um Ph.D. da Boston University. Trabalhou em projetos pioneiros de sistemas operacionais de memória virtual na IBM e no MIT, que desenvolveram técnicas amplamente implementadas hoje em dia em sistemas como UNIX, Windows NTTM e 0S12. Tem 20 anos de experiência de ensino acadêmico, incluindo o de professor assalariado e de Chairman do Departamento de Ciência da Computação no Boston Coilege antes de fundar a Deitei & Associates, mc. com Paul J. Deitei. E autor ou co-autor de dezenas de livros e pacotes de multimídia e está escrevendo atualmente mais cinco. Com traduções publicadas em japonês, russo, espanhol, chinês elementar, chinês avançado, coreano, francês, português, polonês e italiano, os textos dos Deitei alcançaram um reconhecimento internacional.
Paul J. Deitei, Vice-Presidente Executivo da Deitei & Associates, mc., é diplomado pela Sloan School of Management do Massachusetts Institute ofTechnology, onde estudou Tecnologia de Informação. Através da Deitei & Associates, mc., deu cursos de Java, C, C++, Internet e World Wide Web para clientes da indústria, incluindo Compaq, Sun Microsystems, White Sands Missile Range, Rogue Wave Software, Computervision, Stratus, Fidelity, Cambridge Technology Partners, Open Environment Corporation, One Wave, Hyperion Software, Lucent Technologies, Adra Systems, Entergy, CableData Systems, NASA no Kennedy Space Center, National Severe Storm Center, IBM e muitas outras organizações. Tem lecionado C++ e Java para o Boston Chapter da Association for Computing Machinery e ministrou cursos de Java baseados em satélite atavés de uma colaboração entre a Deitei & Associates, mc., a Prentice Hali e a Tecnology Education Network.
Os Deitei são co-autores dos livros de ensino de Ciência da Computação a nível acadêmico introdutório mais vendidos, C How to Program. Third Edition, Java How to Program: Third Edition, Visual Basic 6 How lo Program (em co-autoria com Tem R. Nieto) e Internet and World Wide Web How to Program (em co-autoria com Tem R. Nieto). Os Deitei são também co-autores do C++ Multimedia Cyber Classroom: Third Edition (cuja primeira edição foi o primeiro livro baseado em muitimídia da Prentice Hali), do Java 2 Multimedia Cyber Classroom: Third Edition, da Visual Basic 6 Multimedia Cyber Classroom e do Internet and World Wide Web Programming Multimedia Cvber Classroom. Os Deitei são também co-autores do The Complete C+ + Training Course. Third Edition, The Complete Visual Basic 6 Training Course. The Complete Java 2 Training Course: Third Edition e do The Complete Internel and World Wide Web Programming Training Course - cada um destes produtos contém o livro correspondente da série Como Programar e a Multimedia Cyber Classroom correspondente.

Sobre a Deitei & Associates, mc.

A Deitei & Associates, mc. é uma organização em rápido crescimento, internacionalmente reconhecida no treinamento corporativo e publicações, especializada na educação em linguagens de programação, Internet, World Wide Web e tecnologia de objetos. A empresa oferece cursos sobre programação em C++, Java, C, Visual Basic, Internet e World Wide Web e tecnologia de objetos. Os principais dirigentes da Deitei & Associates, mc. são Dr. Harvey M. Deitei e Paul J. Deitei. Entre os clientes da empresa incluem-se algumas das maiores empresas fabricantes de computadores do mundo, agências governamentais e organizações comerciais. Através de sua parceria para publicações com a Prentice Hail, Deitei & Associates, mc. publica livros de ponta e livros profissionais, Cyber Classrooms interativas em multimídia baseadas em CD-ROM e cursos baseados na World Wide Web. A Deitei & Associates, lnc. e os autores podem ser contactados via e-mau em deitei@deitei com

Para conhecer mais sobre Deitei & Associates, mc., suas publicações e sobre seus currículos de cursos no local, visite:

www. deitei, com

Para conhecer mais sobre as publicações Deitei/Prentice Hail, visite:

www prenhali .com/deitei

Sumário

Capítulo 1 Introdução aos computadores e à programação em C++ 49
1.1 Introdução 50
1.2 O que é um computador? 52
1.3 Organização de computadores 53
1.4 Evolução dos sistemas operacionais 53
1.5 Computação pessoal, computação distribuída e computação cliente/servidor 54
1.6 Linguagens de máquina, linguagens simbólicas e linguagens de alto nível 55
1.7 AhistóriadeC/C++ 56
1.8 A biblioteca padrão de C++ 57
1.9 Java e Java Como Programar 58
1.10 Outras linguagens de alto nível 58
1.11 Programação estruturada 58
1.12 A tendência-chave em software: tecnologia de objetos 59
1.13 Fundamentos de um ambiente típico de C++ 61
1.14 Tendências de hardware 63
1.15 História da Internet 63
1.16 História da World Wide Web 65
1.17 Notas gerais sobre C++ e este livro 65
1.18 Introdução à programação de C++ 66
1.19 Um programa simples: imprimindo uma linha de texto 66
1.20 Outro programa simples: somando dois inteiros 70
1.21 Conceitos de memória 73
1.22 Aritmética 74
1.23 Tomada de decisões: operadores relacionais e de igualdade 77
1.24 Pensando em objetos: introdução à tecnologia de objetos e à Unified Modeling LanguageTM 82
Capítulo 2 Estruturas de controle 99
2.1 Introdução 100
2.2 Algoritmos 101
2.3 Pseudocódigo 101
2.4 Estruturas de controle 101
2.5 A estrutura de seleção if 104
2.6 A estrutura de seleção if/else 105
2.7 A estrutura de repetição while 109
2.8 Formulando algoritmos: estudo de caso 1 (repetição controlada por contador) 110
2.9 Formulando algoritmos com refinamento top-down, passo a passo: estudo de caso 2 (repetição controlada por sentinela) 113

2.10 Formulando algoritmos com refinamento top-down, passo a passo: estudo de caso 3
(estruturas de controle aninhadas) 120
2.11 Operadores de atribuição 124

2.12 Operadores de incremento e decremento 125
2.13 Aspectos essenciais da repetição controlada por contador 127
2.14 A estrutura de repetição for 129
2.15 Exemplos usando a estrutura for 133
2.16 A estrutura de seleção múltipla switch 137
2.17 A estrutura de repetição do/while 143
2.18 Os comandos break e continue 144
2.19 Operadores lógicos 147
2.20 Confundindo os operadores de igualdade (==) e atribuição (=) 149
2.21 Resumo de programação estruturada 151
2.22 (Estudo de caso opcional) Pensando em objetos:
identificando as classes em um problema 155
Capítulo 3 Funções 189
3.1 Introdução 190
3.2 Componentes de programas em C++ 190
3.3 Funções da biblioteca matemática 191
3.4 Funções 193
3.5 Definições de funções 193
3.6 Protótipos de funções 197
3.7 Arquivos de cabeçalho 199
3.8 Geração de números aleatórios 201
3.9 Exemplo: um jogo de azar e apresentando enum 206
3.10 Classes de armazenamento 209
3.11 Regras de escopo 211
3.12 Recursão 214
3.13 Exemplo usando recursão: a série de Fibonacci 217
3.14 Recursão versus iteração 220
3.15 Funções com listas de parâmetros vazias 222
3.16 Funções mime 223
3.17 Referências e parâmetros por referência 224
3.18 Argumentos default 228
3.19 Operador unário de resolução de escopo 229
3.20 Sobrecarga de funções 230
3.21 Gabaritos de funções 232
3.22 (Estudo de caso opcional) Pensando em objetos:
identificando os atributos de uma classe 234
Capítulo 4 Arrays 261
4.1 Introdução 262
4.2 Arrays 262
4.3 Declarando arrays 264
4.4 Exemplos usando arrays 264
4.5 Passando arrays a funções 278
4.6 Ordenando arrays 283
4.7 Estudo de caso: calculando média, mediana e moda usando arrays 284
4.8 Pesquisando arrays: pesquisa linear e pesquisa binária 288
4.9 Arrays multidimensionais 293

4.10 (Estudo de caso opcional) Pensando em objetos:
identificando as operações de uma classe 298

SUMÁRIO 31

Capítulo 5 Ponteiros e strings 319
5.1 Introdução 320
5.2 Declarações e inicialização de variáveis ponteiro 320
5.3 Operadores sobre ponteiros 322
5.4 Chamando funções por referência 324
5.5 Usando o qualificador const com ponteiros 328
5.6 Bubble sort usando chamada por referência 334
5.7 Expressões com ponteiros e aritmética de ponteiros 339
5.8 A relação entre ponteiros e arrays 341
5.9 Arrays de ponteiros 345
5.10 Estudo de caso: uma simulação de embaralhamento e distribuição de cartas 346
5.11 Ponteiros de função 350
5.12 Introdução ao processamento de caracteres e strings 354
5.12.1 Fundamentos de caracteres e strings 355
5.12.2 Funções de manipulação de strings da biblioteca de tratamento de strings 357
5.13 (Estudo de caso opcional) Pensando em objetos: colaborações entre objetos 363
Capítulo 6 Classes e abstração de dados 395
6.1 Introdução 396
6.2 Definições de estruturas 397
6.3 Acessando membros de estrutura 398
6.4 Implementando um tipo Time definido pelo usuário com uma struct 399
6.5 Implementando um tipo de dado abstrato Time com uma class 401
6.6 Escopo de classe e acesso a membros de classes 407
6.7 Separando a interface da implementação 408
6.8 Controlando o acesso a membros 411
6.9 Funções de acesso e funções utilitárias 414
6.10 Inicializando objetos de classes: construtores 417
6.11 Usando argumentos default com construtores 417
6.12 Usando destruidores 421
6.13 Quando construtores e destruidores são chamados 421
6.14 Usando membros de dados e funções membro 424
6.15 Uma armadilha sutil: retornando uma referência a
um membro de dados private 428
6.16 Atribuição usando cópia membro a membro default 431
6.17 Reutilização de software 432
6.18 (Estudo de caso opcional) Pensando em objetos: começando a programar as classes para o simulador de elevador 432
Capítulo 7 Classes: parte II 451
7.1 Introdução 452

7.2 Objetos const (constantes) e funções membro const 452
7.3 Composição: objetos como membros de classes 460
7.4 Funções friend e classes friend 465
7.5 Usando o ponteiro this 468
7.6 Alocação dinâmica de memória com os operadores new e delete 473
7.7 Membros de classe static 474
7.8 Abstração de dados e ocultação de informações 479
7.8.1 Exemplo: tipo de dado abstrato array 481
7.8.2 Exemplo: tipo de dado abstrato string 481
7.8.3 Exemplo: tipo de dado abstrato fila 482
7.9 Classes contêiner e iteradores 482
7.10 Classes proxy 482
7.11 (Estudo de caso opcional) Pensando em objetos: programando as classes para
o simulador de elevador 484

Capítulo 8 Sobrecarga de operadores 515
8.1 Introdução 516
8.2 Fundamentos da sobrecarga de operadores 516
8.3 Restrições sobre a sobrecarga de operadores 518
8.4 Funções operador como membros de classe versus como funções friend 519

8.5 Sobrecarregando os operadores de inserção em stream e extração de stream 520
8.6 Sobrecarregando operadores unários 523
8.7 Sobrecarregando operadores binários 524
8.8 Estudo de caso: uma classe Array 524
8.9 Convertendo entre tipos 535
8.10 Estudo de caso: uma classe String 536
8.11 Sobrecarregando ++ e -- 546
8.12 Estudo de caso: uma classe Date 548
Capítulo 9 Herança 563
9.1 Introdução 564
9.2 Herança: classes base e classes derivadas 565
9.3 Membros protected 567
9.4 Fazendo coerção de ponteiros de classe base para ponteiros de classe derivada 568
9.5 Usando funções membro 573
9.6 Sobrescrevendo membros da classe base em uma classe derivada 573
9.7 Herança public, protected e private 577
9.8 Classes base diretas e classes base indiretas 577
9.9 Usando construtores e destruidores em classes derivadas 578
9.10 Conversão implícita de objeto de classe derivada para objeto de classe base 582
9.11 Engenharia de software com herança 583

9.12 Composição versus herança 584
9.13 Relacionamentos “usa um” e “conhece um” 585
9.14 Estudo de caso: ponto, círculo e cilindro 585
9.15 Herança múltipla 592
9.16 (Estudo de caso opcional) Pensando em objetos: incorporando herança à simulação do elevador 595
Capítulo 10 Funções virtuais e polimorfismo 607
10.1 Introdução 608
10.2 Campos de tipo e comandos switch 608
10.3 Funções virtual 609
10.4 Classes base abstratas e classes concretas 609
10.5 Polimorfismo 610
10.6 Estudo de caso: um sistema de folha de pagamento usando polimorfismo 612
10.7 Novas classes e vinculação dinâmica 621
10.8 Destruidores virtual 622
10.9 Estudo de caso: herdando a interface e herdando a implementação 622
10.10 Polimorfismo, funções virtual e vinculação dinâmica “vistos por dentro” 630

Capítulo 11 Entrada/saída com streams em C++ 637
11.1 Introdução 639
11.2 Streams 639
11.2.1 Arquivos de cabeçalho da biblioteca iostream 640
11.2.2 Classes e objetos de entrada/saída com streams 640
11.3 Saída com streams 641
11.3.1 Operador de inserção em stream 642
11.3.2 Encadeando operadores de inserção/extração do stream 643
11.3.3 Saída de variáveis char* 644

11.3.4 Saída de caracteres com a função membro put; encadeando puts 645
11.4 Entrada com streams 645
11.4.1 Operador de extração do stream 646
11.4.2 Funções membro get e getline 647
11.4.3 Funções membro peek. putback e ignore de istream 650
11.4.4 E/S segura quanto ao tipo 651
11.5 E/S não-formatada com read. gcount e write 651
11.6 Manipuladores de streams 652
11.6.1 Base do stream de inteiros: dec, oct, hex e setbase 652
11.6.2 Precisão em ponto flutuante (precision, setprecision) 653
11.6.3 Largura de campo (setw. width) 654
11.6.4 Manipuladores definidos pelo usuário 656
11.7 Estados de formato do stream 656
11.7.1 Indicadores de estado de formato 657
11.7.2 Zeros à direita e pontos decimais (ios: : e showpoint) 658
11.7.3 Alinhamento (ios: : left, ios: : right, ios: : internal) 659

) 11.7.4 Preenchimento (f iii, setfill) 660
11.7.5 Basedostreamdeinteiros(ios: :dec,ios: :oct.ios: :hex,
i8 ios: :showbase) 661
11.7.6 Números em ponto flutuante; notação científica (ios: : scientific.
73 ios: :fixed) 662
11.7.7 Controle de maiúsculas/minúsculas (ios: :uppercase) 663
11.7.8 Inicializando e reinicializando os indicadores de formato (flags,
setiosflags, resetiosflags) 664
11.8 Estados de erro do stream 665
11.9 Vinculando um stream de saída a um stream de entrada 667
Capítulo 12 Gabaritos 679
12.1 Introdução 680
12.2 Gabaritos de função 681
12.3 Sobrecarregando funções gabarito 683
12.4 Gabaritos de classe 684
12.5 Gabaritos de classe e parâmetros não-tipo 689
12.6 Gabaritos e herança 690
12.7 Gabaritos e friends 690
12.8 Gabaritos e membros static 691

Capítulo 13 Tratamento de exceções 697
13.1 Introdução 697
13.2 Quando o tratamento de exceções deve ser usado 699
13.3 Outras técnicas de tratamento de erros 700
13.4 Fundamentos do tratamento de exceções em C++: try, throw, catch 700
13.5 Um exemplo simples de tratamento de exceção: divisão por zero 701
13.6 Disparando uma exceção 703
13.7 Capturando uma exceção 704
13.8 Disparando novamente uma exceção 707
13.9 Especificações de exceção 708
13.10 Processando exceções inesperadas 709
13.11 Desempilhando a pilha 709
13.12 Construtores, destruidores e o tratamento de exceções 710
13.13 Exceções e herança 711
13.14 Processando falhas de new 711
13.15 A classe autoptr e a alocação dinâmica de memória 715
13.16 Hierarquia de exceções da biblioteca padrão 716

Capítulo 14 Processamento de arquivos 726
14.1 Introdução 727
14.2 A hierarquia de dados 727
14.3 Arquivos e streams 729
14.4 Criando um arquivo de acesso seqüencial 729
14.5 Lendo dados de um arquivo de acesso seqüencial 733
14.6 Atualizando arquivos de acesso seqüencial 739

14.7 Arquivos de acesso aleatório 739
14.8 Criando um arquivo de acesso aleatório 740
14.9 Gravando dados aleatoriamente em um arquivo de acesso aleatório 742
14.10 Lendo dados seqüencialmente de um arquivo de acesso aleatório 744
14.11 Exemplo: um programa de processamento de transações 746
14.12 Entrada/saída de objetos 751
Capítulo 15 Estruturas de dados 759
15.1 Introdução 760
15.2 Classes auto-referentes 761
15.3 Alocação dinâmica de memória 761
15.4 Listas encadeadas 763
15.5 Pilhas 774
15.6 Filas 778
15.7 Árvores 781
Capítulo 16 Bits, caracteres, strings e estruturas 807
16.1 Introdução 808
16.2 Definições de estrutura 808
16.3 Inicializando estruturas 810
16.4 Usando estruturas com funções 810
16.5 typedef 811
16.6 Exemplo: uma simulação de alto desempenho do embaralhamento e distribuição de cartas 811
16.7 Operadores sobre bits 814
16.8 Campos de bits 822
16.9 A biblioteca de manipulação de caracteres 825
16.10 Funções de conversão de strings 830
16.11 Funções de pesquisa da biblioteca de manipulação de strings 834
16.12 Funções de memória da biblioteca de manipulação de strings 839
16.13 Uma outra função da biblioteca de manipulação de strings 843
Capítulo 17 O pré-processador 855
17.1 Introdução 856
17.2 A diretiva #include do pré-processador 856
17.3 A diretiva #define do pré-processador: constantes simbólicas 857
17.4 A diretiva #define do pré-processador: macros 857
17.5 Compilação condicional 859
17.6 As diretivas #error e #pragma do pré-processador 860
17.7 Os operadores # e 861
17.8 Números de linhas 861
17.9 Constantes simbólicas predefinidas 862
17.10 Asserções 862
Capítulo 18 Tópicos sobre código legado em C 867
18.1 Introdução 868
18.2 Redirecionando entrada/saída nos sistemas UNIX e DOS 868

18.3 Lista de argumentos com tamanho variável 869

18.4 Usando argumentos na linha de comando 871
18.5 Notas sobre compilação de programas de múltiplos arquivos-fonte 872
18.6 Terminando um programa com exit e atexit 874
18.7 O qualificador de tipo volatile 875
18.8 Sufixos para constantes inteiras e de ponto flutuante 875
18.9 Tratamento de sinais 876
18.10 Alocação dinâmica de memória com calloc e realloc 878
18.11 Desvio incondicional: goto 878
18.12 Uniões 880
18.13 Especificações de ligação 883
Capítulo 19 A classe string e o processamento em stream de strings 889
19.1 Introdução 890
19.2 Atribuição e concatenação de strings 891
19.3 Comparando strings 894
19.4 Substrings 896
19.5 Intercambiando strings 896
19.6 Características de string 897
19.7 Encontrando caracteres em um string 899
19.8 Substituindo caracteres em um string 901
19.9 Inserindo caracteres em um string 903
19.10 Conversões para strings char* no estilo da linguagem C 904
19.11 Iteradores 906
19.12 Processamento de strings em streams 907
Capítulo 20 A biblioteca padrão de gabaritos (STL) 916
20.1 Introdução à biblioteca padrão de gabaritos (STL) 918
20.1.1 Introdução a contêineres 919
20.1.2 Introdução a iteradores 923
20.1.3 Introdução a algoritmos 928
20.2 Contêineres seqüenciais 930
20.2.1 O contêiner seqüencial vector 931
20.2.2 O contêiner seqüencial list 937
20.2.3 O contêiner seqüencial deque 941
20.3 Contêineres associativos 943
20.3.1 O contêiner associativo multiset 943
20.3.2 O contêiner associativo set 946
20.3.3 O contêiner associativo multimap 947
20.3.4 O contêiner associativo map 949
20.4 Adaptadores de contêineres 951
20.4.1 O adaptador stack 951
20.4.2 O adaptador queue 953
20.4.3 O adaptador priorityqueue 954
20.5 Algoritmos 955
20.5.1 f iii, filin. generate e generate_n 956
20.5.2 equal, mismatch e lexicographical compare 958
20.5.3 remove, remove if. remove copy e remove_copy_if 960
20.5.4 replace, replace if, replace copy e replacecopyif 963

20.5.5 Algoritmos matemáticos 965
20.5.6 Algoritmos básicos de pesquisa e classificação 968
20.5.7 swap, iter_swape swap ranges 971
20.5.8 copy_backward, merge. unique e reverse 972
20.5.9 inplacemerge, unique copy e reverse_copy 975

20.5.10 Operações sobre conjuntos . 976
20.5.11 lowerbound, upper_bound e equal range 979
20.5.12 Heapsort 981
20.5.13 minemax 984
20.5.14 Algoritmos não-cobertos neste capítulo 985
20.6 A classe bitset 986
20.7 Objetos função 990

Capítulo 21 Acréscimo a Linguagem Padrão C++
21.1 Acréscimos à linguagem padrão C++ 1003
21.2 Introdução 1004
21.3 O tipo de dados bool 1004
21.4 O operador staticcast 1006
21.5 O operador constcast 1008
21.6 O operador reinterpretcast 1009
21.7 Ambientes de nomes 1010
21.8 Informação sobre tipo durante a execução - RTTI 1013
21.9 Palavras-chave operadores 1017
21.10 Construtores explicit 1018
21.11 Membros de classe mutable 1023
21.12 Ponteiros para membros de classes (. * e _>*) 1024
21.13 Herança múltipla e classes base virtual 1026

Observações finais 1030

Apêndice A Tabela de precedência de operadores 1035
1037

Apêndice C
C. 1 Sistemas de numeração 1038
C.2 Introdução 1039
C.3 Abreviando números binários como números octais e hexadecimais 1041
C.4 Convertendo números octais e hexadecimais em números binários 1042
C.5 Convertendo do sistema binário, octal ou hexadecimal para o sistema decimal 1043
C.6 Convertendo do sistema decimal para o sistema binário, octal ou hexadecimal 1044
Números binários negativos: notação em complemento de dois 1045

Recursos sobre C++ na Internet e na Web 1050
Recursos 1050
Tutoriais 1051
FAQs 1051
Visual C++ 1052
comp.lang.c++ 1052
Compiladores 1054
Ferramentas de desenvolvimento 1055
Biblioteca padrão de gabaritos 1055
Bibliografia 1057

Apêndice B Conjunto de caracteres ASCII

Apêndice D
D. 1
D.2
D.3
D.4
D.5
D.6
D.7
D.8

Índice 1062

Sumário das ilustrações

Capítulo 1 Introdução aos computadores e à programação em C++
1.1 Um ambiente típico de C++ 62
1.2 Programa de impressão de texto 67
1.3 Algumas seqüências comuns de escape 68
1.4 Impressão em uma linha com comandos separados usando cout 69
1.5 Impressão em múltiplas linhas com um único comando usando cout 69
1.6 Um programa de adição 70
1.7 Posição de memória mostrando o nome e valor de uma variável 74
1.8 Posições de memória depois de os valores para as duas variáveis terem sido

fornecidos como entrada 74
1.9 Posições de memória após um cálculo 74
1.10 Operadores aritméticos 75
1.11 Precedência dos operadores aritméticos 76
1.12 Ordem em que um polinômio de segundo grau é calculado 78
1.13 Operadores relacionais e de igualdade 78
1.14 Utilizando operadores relacionais e de igualdade 79

1.15 Precedência e associatividade dos operadores discutidos até agora 81
Capítulo 2 Estruturas de controle
2.1 Colocando em um fluxograma a estrutura de seqüência de C++ 102
2.2 Palavras-chave de C++ 103
2.3 Representando em fluxograma a estrutura de seleção única if 105
2.4 Representando em fluxograma a estrutura de seleção dupla if/else 106
2.5 Representando em fluxograma a estrutura de repetição while 110
2.6 Algoritmo em pseudocódigo que usa repetição controlada por contador para resolver o problema da média da turma 111
2.7 Programa em C++ e exemplo de execução para o problema de cálculo da média
da turma com repetição controlada por contador 111
2.8 Algoritmo em pseudocódigo que usa repetição controlada por sentinela para resolver o problema da média da turma 116
2.9 Programa em C++ e exemplo de execução para o problema de cálculo da média
da turma com repetição controlada por sentinela 117
2.10 Pseudocódigo para o problema dos resultados do teste 122
2.11 Programa em C++ e exemplos de execuções para o problema dos resultados do teste 122
2.12 Operadores aritméticos de atribuição 125
2.13 Os operadores de incremento e decremento 125
2.14 A diferença entre pré-incrementar e pós-incrementar 126
2.15 Precedência dos operadores encontrados até agora no texto 127
2.16 Repetição controlada por contador 128
2.17 Repetição controlada por contador com a estrutura for 130

2.18 Componentes de um cabeçalho for típico 130
2.19 Fluxograma de uma estrutura de repetição for típica 133
2.20 Somatório com for 134
2.21 Calculando juros compostos com for 136
2.22 Um exemplo usando switch 138
2.23 A estrutura de seleção múltipla switch com breaks 141
2.24 Usando a estrutura do/while 144
2.25 O fluxograma da estrutura de repetição do/while 145
2.26 Usando o comando break em uma estrutura for 145
2.27 Usando o comando continue em uma estrutura for 146
2.28 Tabela verdade para o operador && (E lógico) 147
2.29 Tabela verdade para o operador || (OU lógico) 148
2.30 Tabela verdade para o operador ! (negação lógica) 149
2.31 Precedência e associatividade de operadores 149
2.32 As estruturas de repetição de seqüência, seleção e repetição com uma única
entrada/única saída em C++ 151
2.33 Regras para formar programas estruturados 152

2.34 O fluxograma mais simples 152
2.35 Aplicando repetidamente a regra 2 da Fig. 2.33 ao fluxograma mais simples 152
2.36 Aplicando a regra 3 da Fig. 2.33 ao fluxograma mais simples 153
2.37 Blocos de construção empilhados, aninhados e sobrepostos 154
2.38 Um fluxograma não-estruturado 154
2.39 Diagrama de caso de uso para o sistema do elevador 160
2.40 Lista de substantivos na definição do problema 160
2.41 Representando uma classe na UML 162
2.42 Associações entre classes em um diagrama de classes 162
2.43 Tabela de multiplicidade 163
2.44 Diagrama completo de classes para a simulação do elevador 163
2.45 Diagrama de objetos do edifício vazio 165
Capítulo 3 Funções
3.1 Relacionamento hierárquico função chefe/função trabalhadora 191
3.2 Funções comumente usadas da biblioteca de matemática 192
3.3 Criando e usando uma função definida pelo programador 193
3.4 Função rnaximum definida pelo programador 196
3.5 Hierarquia de promoção para os tipos de dados primitivos 199
3.6 Arquivos de cabeçalho da biblioteca padrão 200
3.7 Inteiros em uma escala ajustada e deslocada produzidos por 1 + rand ( ) % 6 202
3.8 Lançando um dado de seis faces 6.000 vezes 202
3.9 Randomizando o programa de lançamento de um dado 204
3.10 Programa para simular o jogo de craps 206
3.11 Exemplos de resultados do jogo de craps 208
3.12 Um exemplo de escopos 212
3.13 Cálculo recursivo de 5! 215
3.14 Cálculo de fatoriais com uma função recursiva 216
3.15 Gerando os números de Fibonacci recursivamente 217
3.16 Conjunto de chamadas recursivas à função fibonacci 219
3.17 Resumo dos exemplos e exercícios sobre recursão no livro 221
3.18 Duas maneiras de declarar e usar funções que não recebem argumentos 222
3.19 Usando uma função mime para calcular o volume de um cubo 223
3.20 Um exemplo de chamada por referência 225
3.21 Usando uma referência não-inicializada 226
3.22 Tentando usar uma referência não-inicializada 227
3.23 Usando argumentos default 228

3.24 Usando o operador unário de resolução de escopo 229
3.25 Usando funções sobrecarregadas 231
3.26 Alterando o nome para possibilitar a ligação segura quanto aos tipos 231
3.27 Usando um gabarito de função 233
3.28 Palavras e frases descritivas na definição do problema 235

3.29 Diagrama de classes mostrando os atributos 235
3.30 Diagrama de estados para as classes FloorButton e ElevatorButton 236
3.31 Diagrama de estados para a classe Elevator 236
3.32 Diagrama de atividades modelando a lógica do elevador para responder a pressionamentos de botões 238
3.33 O problema das Torres de Hanói para o caso com quatro discos 257
Capítulo 4 Arrays
4.1 Um array com 12 elementos 263
4.2 Precedência e associatividade dos operadores 264
4.3 Inicializando os elementos de um array com zeros 265
4.4 Inicializando os elementos de um array com uma declaração 265
4.5 Gerando os valores para serem colocados nos elementos de um array 267
4.6 Inicializando e usando corretamente uma variável constante 268
4.7 Um objeto const deve ser inicializado 268
4.8 Calculando a soma dos elementos de um array 270
4.9 Um programa de análise de uma votação de estudantes 270
4.10 Um programa que imprime histogramas 272
4.11 Programa de lançamento de dados usando arrays em vez de switch 274
4.12 Tratando arrays de caracteres como strings 275
4.13 Comparando a inicialização de arrays static com a inicialização automática 277
4.14 Passando arrays e elementos individuais de arrays para funções 280
4.15 Demonstrando o qualificador de tipo const 281
4.16 Ordenando um array com o bubble sort 283
4.17 Programa de análise de uma pesquisa de opinião 285
4.18 Exemplo de execução do programa de análise de dados de uma pesquisa de opinião 287
4.19 Pesquisa linear de um array 289
4.20 Pesquisa binária em um array ordenado 290
4.21 Um array bidimensional com três linhas e quatro colunas 293
4.22 Inicializando arrays multidimensionais 294
4.23 Exemplo de uso de arrays bidimensionais 296
4.24 Frases com verbos para cada classe do simulador 299
4.25 Diagrama de classes com atributos e operações 300
4.26 Diagrama de seqüência modelando um laço de simulação 302
4.27 Diagrama de seqüência para o processo de agendamento 303
4.28 As 36 combinações possíveis lançando-se dois dados 311
4.29 Os 8 movimentos possíveis do cavalo 314
4.30 As 22 casas eliminadas ao se colocar uma rainha no canto superior esquerdo do tabuleiro 316
Capítulo 5 Ponteiros e strings
5.1 Referenciando direta e indiretamente uma variável 321
5.2 Representação gráfica de um ponteiro que aponta
para uma variável inteira na memória 322
5.3 Representação de y e yptr na memória 322
5.4 Os operadores de ponteiro & e * 323

5.5 Precedência e associatividade de operadores 324
5.6 Elevando uma variável ao cubo usando uma chamada por valor 325
5.7 Elevando uma variável ao cubo usando chamada por referência
com um ponteiro como argumento 326

5.8 Análise de uma chamada por valor típica 327
5.9 Análise de uma chamada por referência típica com um ponteiro como
argumento 328
5.10 Convertendo um string para maiúsculas 330
5.11 Imprimindo um string, um caractere de cada vez, usando um ponteiro
não-constante para dados constantes 330
5.12 Tentando modificar dados através de um ponteiro não-constante para
dados
constantes 331
5.13 Tentando modificar um ponteiro constante para dados não-constantes 332
5.14 Tentando modificar um ponteiro constante para dados constantes 333
5.15 Bubble sort com chamada por referência 334
5.16 O operador sizeof, quando aplicado a um nome de array, retorna o número
de
bytes no array 337
5.17 Usando o operador sizeof para determinar os tamanhos de tipos de dados
padrão 338
5.18 O array v e uma variável ponteiro vPtr que aponta para v 339
5.19 O ponteiro vPtr após operação de aritmética de ponteiros 340
5.20 Usando quatro métodos de fazer referência a elementos de array 342
5.21 Copiando um string usando a notação de array e a notação de ponteiro 344

5.22 Representação gráfica do array suit 345
5.23 Representação de um baralho com um array bidimensional 346
5.24 Programa de embaralhamento e distribuição de cartas 348
5.25 Exemplo de execução do programa de embaralhamento e distribuição de
cartas 350
5.26 Programa de classificação de finalidade múltipla usando ponteiros para
função 350
5.27 Saídas do programa bubble sort da Fig. 5.26 352
5.28 Demonstrando um array de ponteiros para funções 353
5.29 Funções de manipulação de strings da biblioteca de tratamento de strings
357
5.30 Usando strcpy e strncpy 358
5.31 Usando strcat e strncat 359
5.32 Usando strcmp e strncmp 360
5.33 Usando strtok 362
5.34 Usando strlen 362
5.35 Lista modificada de frases com verbos para as classes no sistema 364
5.36 Colaborações no sistema do elevador 364

5.37 Diagrama de colaborações para entrada e saída de passageiros 365
5.38 Array deck não-embaralhado 379
5.39 Exemplo de embaralhamento do array deck 380
5.40 Códigos de operação da Simpletron Machine Language (SML) 381
5.41 Exemplo de dump 384
5.42 As letras do alfabeto expressas em código Morse internacional 393
Capítulo 6 Classes e abstração de dados
6.1 Criando uma estrutura. inicializando seus membros e imprimindo a estrutura 399
6.2 Uma definição simples da classe Time 401
6.3 Implementação do tipo de dado abstrato Time como uma classe 402
6.4 Acessando os membros de dados e as funções membro de um objeto através de cada tipo de handie de objeto - através do nome do objeto, através de uma referência e através de um ponteiro para o objeto 407
6.5 Separando a interface da classe Time da implementação 409
6.6 Tentativa errônea de acessar membros private de uma classe 412
6.7 Usando uma função utilitária 414
6.8 Usando um construtor com argumentos default 418
6.9 Demonstrando a ordem em que construtores e destruidores são chamados 422
6.10 Usando funções sete get 425
6.11 Retornando uma referência a um membro de dados private 429
6.12 Atribuindo um objeto a um outro por cópia membro a membro default 431

SUMÁRIO DAS ILUSTRAÇÕES 41
6.13 Diagrama completo de classes com notações de visibilidade 433
6.14 Lista de handies para cada classe 434
6.15 Arquivo de cabeçalho da classe Beli 434
6.16 Arquivo de cabeçalho da classe Clock 435
6.17 Arquivo de cabeçalho da classe Person 435
6.18 Arquivo de cabeçalho da classe Door 436
6.19 Arquivo de cabeçalho da classe Light 436
6.20 Arquivo de cabeçalho da classe Building 437
6.21 Arquivo de cabeçalho da classe ElevatorButton 438
6.22 Arquivo de cabeçalho da classe FloorButton 438
6.23 Arquivo de cabeçalho da classe Scheduler 439
6.24 Arquivo de cabeçalho da classe Floor 440
6.25 Arquivo de cabeçalho da classe Elevator 441
Capítulo 7 Classes: parte II
7.1 Usando uma classe Time com objetos const e funções membro const 454
7.2 Usando um inicializador de membro para inicializar uma constante de um tipo de dado primitivo 457
7.3 Tentativa errônea de inicializar uma constante de um tipo de dado primitivo por atribuição 458
7.4 Usando inicializadores para objetos membro 461
7.5 friends podem acessar membros private de uma classe 466

7.6 Funções não-friend / não-membro não podem acessar membros de classe private 467
7.7 Usando o ponteiro this 469
7.8 Encadeando chamadas a funções membro 470
7.9 Usando um membro de dados static para manter uma contagem do número de objetos de uma classe 475
7.10 Implementando uma classe proxy 483
7.11 Acionador para a simulação do elevador 486
7.12 Arquivo de cabeçalho da classe Building 486
7.13 Arquivo de implementação da classe Building 487
7.14 Arquivo de cabeçalho da classe Clock 488
7.15 Arquivo de implementação da classe Clock 488
7.16 Arquivo de cabeçalho da classe Scheduler 489
7.17 Arquivo de implementação da classe Scheduler 490
7.18 Arquivo de cabeçalho da classe Beil 492
7.19 Arquivo de implementação da classe Beli 492
7.20 Arquivo de cabeçalho da classe Light 493
7.21 Arquivo de implementação da classe Light 493
7.22 Arquivo de cabeçalho da classe Door 494
7.23 Arquivo de implementação da classe Door 495
7.24 Arquivo de cabeçalho da classe ElevatorButton 496
7.25 Arquivo de implementação da classe ElevatorButton 496
7.26 Arquivo de cabeçalho da classe FloorButton 497
7.27 Arquivo de implementação da classe FloorButton 498
7.28 Arquivo de cabeçalho da classe Elevator 499
7.29 Arquivo de implementação da classe Elevator 500
7.30 Arquivo de cabeçalho da classe Floor 504
7.31 Arquivo de implementação da classe Floor 505
7.32 Arquivo de cabeçalho da classe Person 506
7.33 Arquivo de implementação da classe Person 507

42 SUMÁRIO DAS ILUSTRAÇÕES

Capítulo 8 Sobrecarga de operadores
8.1 Operadores que podem ser sobrecarregados 518
8.2 Operadores que não podem ser sobrecarregados 518
8.3 Operadores de inserção em stream e extração de stream definidos pelo usuário 521
8.4 Uma classe Array com sobrecarga de operadores 525
8.5 Uma classe String com sobrecarga de operadores 536
8.6 Date com operadores de incremento sobrecarregados 548
8.7 Uma classe de números complexos 557
8.8 Uma classe de inteiros enormes 559
Capítulo 9 Herança
9.1 Alguns exemplos simples de herança 566
9.2 Uma hierarquia de herança para membros da comunidade universitária 566
9.3 Uma parte de uma hierarquia de classes Shape 567

9.4 Coerção de ponteiros da classe base para ponteiros da
classe derivada 568
9.5 Sobrescrevendo membros de classe base em uma classe derivada 574
9.6 Resumo da acessibilidade de membros da classe base em uma classe
derivada 578
9.7 Ordem na qual construtores e destruidores de classes bases e classes
derivadas
são chamados 579
9.8 Demonstrando a classe Point 585
9.9 Demonstrando a classe Circle 587
9.10 Demonstrando a classe Cylinder 589
9.11 Demonstrando a herança múltipla 592
9.12 Atributos e operações das classes ElevatorButton e FloorButton 596
9.13 Diagrama completo de classes do simulador de elevador indicando herança
a
partir da classe Button 596
9.14 Arquivo de cabeçalho da classe Button 597
9.15 Arquivo de implementação da classe Button 598
9.16 Arquivo de cabeçalho da classe ElevatorButton 598
9.17 Arquivo de implementação da classe ElevatorButton 599
9.18 Arquivo de cabeçalho da classe FloorButton 600
9.19 Arquivo de implementação da classe FloorButton 600
Capítulo 10 Funções virtuais e polimorfismo
10.1 Demonstrando o polimorfismo com a hierarquia de classes Employee 612
10.2 Demonstrando a herança de interface com a hierarquia de classes Shape
623
10.3 Fluxo de controle de uma chamada de função virtual 631
Capítulo 11 Entrada/saída com streams em C++
11.1 Parte da hierarquia de classes de EIS com streams 640
11.2 Parte da hierarquia de classes de E/S com streams com as
classes essenciais para processamento de arquivo 641
11.3 Enviando um string para a saída usando inserção no stream 642
11.4 Enviando um string para a saída usando duas inserções no stream 642
11.5 Usando o manipulador de stream endl 643
11.6 Enviando valores de expressões para a saída 643
11.7 Encadeando o operator « sobrecarregado 644
11.8 Imprimindo o endereço armazenado em uma variável char* 645
11.9 Calculando a soma de dois inteiros lidos do teclado com cm e o
operador de extração de stream 646
11.10 Evitando um problema de precedência entre o operador de inserção em
stream e o operador condicional 646
11.11 Operador de extração de stream retornando false quando encontra
fim de arquivo 647

SUMÁRIO DAS ILUSTRAÇÕES 43
11.12 Usando funções membro get, put e eof 648

11.13 Comparando a leitura de um string usando cm com extração do stream e a leitura com cm . get 649
11.14 Entrada de caracteres com a função membro getline 650
11.15 E/S não-formatada com as funções membro read, gcount e write 651
11.16 Usando os manipuladores de stream hex, oct, dec e setbase 652
11.17 Controlando a precisão de valores em ponto flutuante 653
11.18 Demonstrando a função membro width 655
11 . 19 Criando e testando manipuladores de stream não-parametrizados definidos pelo usuário 656
11.20 Indicadores de estado de formato 657
11.21 Controlando a impressão de zeros à direita e de pontos decimais com valores float 658
11.22 Alinhamento à esquerda e alinhamento à direita 659
11.23 Imprimindo um inteiro com espaçamento interno e forçando o sinal de mais 660
11.24 Usando a função membro f iii e o manipulador setfill
para mudar o caractere de preenchimento para campos maiores
do que os valores que estão sendo impressos 660
11.25 Usando o indicador ios: : showbase 662
11.26 Exibindo valores em ponto flutuante nos formatos default,
científico e fixo do sistema 663
11.27 Usando o indicador ios: : uppercase 663
11.28 Demonstrando a função membro flags 664
11.29 Testando estados de erro 666
Capítulo 12 Gabaritos
12.1 Um gabarito de função 681
12.2 Usando funções gabarito 682
12.3 Demonstrando o gabarito de classe Stack 685
12.4 Passando um objeto gabarito Stack para um gabarito de função 688
Capítulo 13 Tratamento de exceções
13.1 Um exemplo simples de tratamento de exceção com divisão por zero 701
13.2 Disparando novamente uma exceção 707
13.3 Demonstração do desempilhamento da pilha 710
13.4 Demonstrando new retornando 0 em caso de falha 712
13.5 Demonstrando new disparando bad alloc em caso de falha 713
13.6 Demonstrando setnewhandler 714
13.7 Demonstrando autoytr 715
Capítulo 14 Processamento de arquivos
14.1 A hierarquia de dados 728
14.2 Como a linguagem C++ visualiza um arquivo de n bytes 729
14.3 Parte da hierarquia de classes de E/S com streams 730
14.4 Criando um arquivo seqüencial 730
14.5 Modos de abertura de arquivos 731
14.6 Combinações de teclas que indicam fim de arquivo para vários sistemas computacionais populares 733
14.7 Lendo e imprimindo um arquivo seqüencial 734

14.8 Programa de consulta de crédito 736
14.9 Exemplo de saída do programa da Fig. 14.8 738
14.10 A visão de C++ de um arquivo de acesso aleatório 740
14.11 Arquivo de cabeçalho clntdata . h 741
14.12 Criando um arquivo de acesso aleatório seqüencialmente 741
14.13 Exemplo de execução do programa da Fig. 14.12 743

14.14 Lendo seqüencialmente um arquivo de acesso aleatório 744
14.15 Programa de contas bancárias 747
Capítulo 15 Estruturas de dados
15.1 Dois objetos vinculados de uma classe auto-referente 761
15.2 Uma representação gráfica de uma lista 764
15.3 Manipulando uma lista encadeada 764
15.4 Exemplo de saída para o programa da Fig. 15.3 769
15.5 A operação insertAtFront 771
15.6 Uma representação gráfica da operação insertAtBack 772
15.7 Uma representação gráfica da operação removeFromFront 772
15.8 Uma representação gráfica da operação removeFromBack 773
15.9 Um programa simples de pilha 775
15.10 Exemplo de saída do programa da Fig. 15.9 776
15.11 Um programa simples com pilha usando composição 777
15.12 Processando uma fila 778
15.13 Exemplo de saída do programa da Fig. 15.12 780
15.14 Uma representação gráfica de uma árvore binária 781
15.15 Uma árvore de pesquisa binária 781
15.16 Criando e percorrendo uma árvore binária 782
15.17 Exemplo de saída do programa da Fig. 15.16 785
15.18 Uma árvore de pesquisa binária 787
15.19 Uma árvore de pesquisa binária de 15 nodos 791
15.20 Comandos de Simple 796
15.21 Programa em Simple que determina a soma de dois inteiros 797
15.22 Programa em Simple que encontra o maior de dois inteiros 797
15.23 Calcula o quadrado de vários inteiros 797
15.24 Escrevendo, compilando e executando um programa na linguagem Simple 798
15.25 Instruções SML produzidas após o primeiro passo do compilador 800
15.26 Tabela de símbolos para o programa da Fig. 15.25 800
15.27 Código não-otimizado do programa da Fig. 15.25 804
15.28 Código otimizado para o programa da Fig. 15.25 804
Capítulo 16 Bits, caracteres, strings e estruturas
16.1 Um alinhamento de memória possível para uma variável do tipo Example mostrando uma área indefinida na memória 810
16.2 Simulação de alto desempenho de embaralhamento e distribuição de cartas 812
16.3 Saída da simulação de alto desempenho de embaralhamento e distribuição

de cartas 813
16.4 Os operadores sobre bits 814
16.5 Imprimindo um inteiro sem sinal em binário 815
16.6 Resultados de combinar dois bits com o operador AND sobre bits (&) 817
16.7 Usando os operadores sobre bits AND, OR inclusivo, OR exclusivo e complemento 817
16.8 Saída do programa da Fig. 16.7 818
16.9 Resultados de combinar dois bits com o operador sobre bits OR inclusivo (1) 819
16.10 Resultados de combinar dois bits com o operador sobre bits OR exclusivo (A) 819
16.11 Usando os operadores de deslocamento sobre bits 819
16.12 Os operadores de atribuição sobre bits 821
16.13 Precedência e associatividade de operadores 821
16.14 Usando campos de bits para armazenar um baralho 823
16.15 Saída do programa da Fig. 16.14 823
16.16 Resumo das funções da biblioteca de manipulação de caracteres 825
16.17 Usando isdigit, isalpha, isainume isxdigit 826

SUMÁRIO DAS ILUSTRAÇÕES 45
16.18 Usando islower. isupper, tolower e toupper 827
16.19 Usando isspace. iscntrl, ispunct. isprint e isgraph 828
16.20 Resumo das funções de conversão de strings da biblioteca de utilitários gerais 830
16.21 Usando atof 830
16.22 Usando atoi 831
16.23 Usando atol 831
16.24 Usando strtod 832
16.25 Usando strtol 833
16.26 Usando strtoul 834
16.27 Funções de pesquisa da biblioteca de manipulação de strings 834
16.28 Usando strchr 835
16.29 Usando strcspn 836
16.30 Usando strpbrk 836
16.3 1 Usando strrchr 837
16.32 Usando strspn 838
16.33 Usando strstr 838
16.34 Funções de memória da biblioteca de manipulação de strings 839
16.35 Usando memcpy 840
16.36 Usando memmove 840
16.37 Usando mexncmp 841
16.38 Usandomemchr 842
16.39 Usandomemset 842
16.40 Uma outra função de manipulação de strings da biblioteca de manipulação

de strings 843

16.41 Usando strerror 843
Capítulo 17 O pré-processador
17.1 As constantes simbólicas predefinidas 862
Capítulo 18 Tópicos sobre código legado em C
18.1 O tipo e as macros definidas no cabeçalho cstdarg 869
18.2 Usando listas de argumentos de tamanho variável 870
18.3 Usando argumentos na linha de comando 871
18.4 Usando as funções exit e atexit 874
18.5 Os sinais definidos no cabeçalho csignal 876
18.6 Utilizando tratamento de sinais 876
18.7 Usando goto 879
18.8 Imprimindo o valor de uma union com os tipos de dados dos dois membros 881
18.9 Usando uma union anônima 882

Capítulo 19 A classe string e o processamento em stream de strings
19.1 Demonstrando a atribuição e concatenação de string 891
19.2 Comparando strings 894
19.3 Demonstrando a função substr 896
19.4 Usando a função swap para intercambiar dois strings 896
19.5 Imprimindo características de um string 897

19.6 Demonstrando as funções de procura em um string/find 899
19.7 Demonstrando as funções erase e replace 901
19.8 Demonstrando as funções insert de string 903
19.9 Convertendo strings para strings e arrays de caracteres no estilo de C 904
19.10 Usando um iterador para exibir um string 906
19.11 Usando um objeto ostringstream alocado dinamicamente 908
19.12 Demonstrando a entrada de dados a partir de um objeto istringstream 910

46 SUMÁRIO DAS ILUSTRAÇÕES
Capítulo 20 A biblioteca padrão de gabaritos (STL)
20.1 Classes contêiner da biblioteca padrão 920
20.2 Funções comuns a todos os contêineres da STL 920
20.3 Arquivos de cabeçalho dos contêineres da biblioteca padrão 922
20.4 typedefs comuns encontrados em contêineres de primeira classe 922
20.5 Demonstrando iteradores de entrada e saída com streams 924
20.6 Categorias de iteradores 925
20.7 Hierarquia das categorias de iteradores 926
20.8 Tipos de iteradores suportados em cada contêiner da biblioteca padrão 926

20.9 typedefs de iteradores predefinidos 927
20.10 Algumas operações de iteradores para cada tipo de iterador 927
20.11 Algoritmos seqüenciais mutantes 929
20.12 Algoritmos seqüenciais não-mutantes 929
20.13 Algoritmos numéricos do arquivo de cabeçalho <numeric> 930

20.14 Demonstrando o gabarito de classe vector da biblioteca padrão 931
20.15 Demonstrando as funções de manipulação de elementos do
gabarito de classe vector da biblioteca padrão 934
20.16 Tipos de exceções da STL 936
20.17 Demonstrando o gabarito de classe list da biblioteca padrão 937
20.18 Demonstrando o gabarito de classe deque da biblioteca padrão 941
20.19 Demonstrando o gabarito de classe multiset da biblioteca padrão 943
20.20 Demonstrando o gabarito de classe set da biblioteca padrão 946
20.21 Demonstrando o gabarito de classe multimap da biblioteca padrão 948
20.22 Demonstrando o gabarito de classe map da biblioteca padrão 950
20.23 Demonstrando a classe adaptadora stack da biblioteca padrão 951
20.24 Demonstrando os gabaritos de classe adaptadora queue da biblioteca
padrão 953
20.25 Demonstrando a classe adaptadora priority queue da biblioteca padrão
955
20.26 Demonstrando as funções f iii, fi].ln, generate
e generate_n da biblioteca padrão 956
20.27 Demonstrando as funções equa]., mismatch e
lexicographical compare da biblioteca padrão 958
20.28 Demonstrando as funções remove, remove if, remove copy e
remove copyi f da biblioteca padrão 960
20.29 Demonstrando as funções replace. replace if. replace copy e
replacecopyif da biblioteca padrão 963
20.30 Demonstrando alguns algoritmos matemáticos da biblioteca padrão 965
20.31 Algoritmos básicos de pesquisa e classificação da biblioteca padrão 969
20.32 Demonstrando swap, iter_swap, e swap ranges 971
20.33 Demonstrando copy_backward. merge. unique e reverse 972
20.34 Demonstrando inplace merge, unique copy e reversecopy 975
20.35 Demonstrando as operações set da biblioteca padrão 977
20.36 Demonstrando lower bound, upper bound e equa]. range 979
20.37 Usando funções da biblioteca padrão para executar um heapsort 981
20.3 8 Demonstrando os algoritmos mm e max 984
20.39 Algoritmos não-cobertos neste capítulo 985
20.40 Demonstrando a classe bitset e o Crivo de Eratóstenes 988
20.41 Objetos função na biblioteca padrão 990
20.42 Demonstrando um objeto função binária 990
Capítulo 21 Acréscimos à linguagem padrão C++
21.1 Demonstrando o tipo de dados primitivo bool 1004
21.2 Demonstrando o operador static_cast 1006
21.3 Demonstrando o operador const_cast 1008
21.4 Demonstrando o operador reinterpret cast 1009

SUMÁRIO DAS ILUSTRAÇÕES 47
21.5 Demonstrando o uso de namespaces 1010
21.6 Demonstrando typeid 1013
21.7 Demonstrando dynamiccast 1014

21.8 Palavras-chave operadores como alternativa para símbolos de operadores 1017
21.9 Demonstrando as palavras-chave operadores 1017
21.10 Construtores de um único argumento e conversões implícitas 1018
21.11 Demonstrando um construtor explicit 1021
21.12 Demonstrando um membro de dados mutable 1024
21.13 Demonstrando os operadores * e -> 1025
21.14 Herança múltipla para formar a classe iostreain 1026
21.15 Tentando chamar polimorficamente uma função herdada de multiplicação 1027
21.16 Usando classes base virtual 1028

Apêndice A Tabela de precedência de operadores
A.1 Tabela de precedência de operadores 1035
Apêndice B Conjunto de caracteres ASCII
B.1 O conjunto de caracteres ASCII 1037

Apêndice C Sistemas de numeração
C.1 Dígitos dos sistemas de numeração binário, octal, decimal e hexadecimal 1039
C.2 Comparação entre os sistemas de numeração binário, octal, decimal e hexadecimal 1040
C.3 Valores posicionais no sistema de numeração decimal 1040
C.4 Valores posicionais no sistema de numeração binário 1041
C.5 Valores posicionais no sistema de numeração octal 1041
C.6 Valores posicionais no sistema de numeração hexadecimal 1041
C.7 Equivalentes decimais, binários, octais e hexadecimais 1042
C.8 Convertendo um número binário em decimal 1043
C.9 Convertendo um número octal em decimal 1043
C.10 Convertendo um número hexadecimal em decimal 1043

Introdução aos computadores e à programação C++
Objetivos

• Entender conceitos básicos de Ciência da Computação.
• Familiarizar-se com tipos diferentes de linguagens de programação.
• Entender um ambiente de desenvolvimento de programas C++ típico.
• Ser capaz de escrever programas simples de computador em C++.
• Ser capaz de usar comandos de entrada e saída simples.
• Familiarizar-se com tipos de dados fundamentais.
• Ser capaz de usar operadores aritméticos.
• Entender a precedência de operadores aritméticos.
• Ser capaz de escrever comandos de tomada de decisão simples.

Pensamentos elevados devem ter uma linguagem elevada.
Aristófanes

Nossa vida é desperdiçada em detalhes... Simplifique, simplifi que.
Henry Thoureau

My object all sublime
1 shall achieve in time.
W. 5. Gilbert

Visão Geral
1.1 Introdução
1.2 O que é um computador?
1.3 Organização de computadores
1.4 Evolução dos sistemas operacionais
1.5 Computação pessoal, computação distribuída e computação cliente/servidor
1.6 Linguagens de máquina, linguagens simbólicas e linguagens de alto nível
1.7 A história de C e C++
1.8 A biblioteca padrão de C++
1.9 Java e Java Como Programar
1.10 Outras linguagens de alto nível
1.11 Programação estruturada
1.12 A tendência-chave em software: tecnologia de objetos
1.13 Fundamentos de um ambiente típico de C++
1.14 Tendências de hardware
1.15 História da Internet
1.16 História daWorldWideWeb
1.17 Notas gerais sobre C++ e este livro
1.18 Introdução à programação de C++
1.19 Um programa simples: imprimindo uma linha de texto
1.20 Outro programa simples: somando dois inteiros

1.21 Conceitos de memória
1.22 Aritmética
1.23 Tomada de decisões: os operadores relacionais e de igualdade
1.24 Pensando em objetos: introdução à tecnologia de objetos e à Unified Modeling LanguageTM
Resumo • Terminologia Erros comuns de programação • Boas práticas de programação Dicas de desempenho Dicas de portabilidade• Observações de engenharia de software Exercícios de auto-revisão• Respostas aos exercícios de auto-revisão • Exercícios
1.1 Introdução
Bem-vindo a C++! Trabalhamos bastante para criar o que esperamos que seja uma experiência de aprendizado informativa, divertida e desafiadora para você. C++ é uma linguagem difícil, normalmente só é ensinada a programadores experientes. Assim, este livro é único entre os livros de ensino de C++ porque:
• É apropriado para pessoas tecnicamente orientadas, com pouca ou nenhuma experiência de programação.
• É apropriado para programadores experientes que querem um tratamento mais profundo da linguagem.
Como um único livro pode atrair ambos os grupos? A resposta é que o núcleo comum do livro enfatiza a obtenção da clareza nos programas através das técnicas comprovadas da programação estruturada e da programação orientada a objetos. Os não-programadores aprendem programação do jeito certo desde o início. Tentamos escrever de uma maneira clara e direta. O livro é abundantemente ilustrado. Talvez o mais importante: o livro apresenta um número enorme de programas em C++ que funcionam e mostra as saídas produzidas quando aqueles programas são execu

tados em um computador. Chamamos isto de “abordagem através de código ativo”. Todos esses programas de exemplo são fornecidos no CD-ROM que acompanha este livro. Você pode fazer o download de todos estes exempios de nosso site na Web: www. deitei . com. Os exemplos também estão disponíveis em nosso produto em CDROM interativo, a C+ + Multimedia Cyber Classroom: Third Edition. A Cyber Classroom também contém muitos hyperlinks, walkthroughs com áudio dos programas de exemplo do livro e respostas a aproximadamente metade dos exercícios deste livro (inclusive respostas curtas, pequenos programas e muitos projetos completos). As características da C’yber Classroom e as informações sobre como pedir o produto estão no final deste livro.
Os primeiros cinco capítulos introduzem os fundamentos sobre computadores, programação de computadores e a linguagem de programação de computador C ++. Os principiantes que assistiram a nossos cursos nos disseram que o material nos Capítulos 1 até 5 apresenta uma fundação sólida para o tratamento mais profundo de C÷÷ nos capítulos restantes. Os programadores experientes em geral lêem rapidamente os primeiros cinco capítulos e depois acham o

tratamento de C++ no restante do livro ao mesmo tempo rigoroso e desafiador. Muitos programadores experientes nos disseram que apreciaram o nosso tratamento da programação estruturada. Freqüentemente, já programaram em linguagens estruturadas como C ou Pascal, mas como nunca tiveram uma introdução formal à programação estruturada, não estavam escrevendo o melhor código possível nestas linguagens. A medida que revisavam a programação estruturada nos capítulos do início deste livro, eles puderam melhorar seus estilos de programação em C e também em Pascal. Então, seja você um novato ou um programador experiente, existe muito neste livro para informá-lo, entretê-lo e desafiá-lo.

A maioria das pessoas está pelo menos um pouco familiarizada com as coisas excitantes que os computadores fazem. Usando este livro, você aprenderá como instruir os computadores para fazer essas coisas. E o software (ou seja, as instruções que você escreve para instruir o computador sobre como executar ações e tomar decisões) que controla os computadores (freqüentemente chamados de hardware). C++ é uma das linguagens de desenvolvimento de software mais populares hoje em dia. Este texto fornece uma introdução à programação na versão de C++ padronizada nos Estados Unidos através do American National Standards Institute (ANSI) e, no mundo, através dos esforços da International Standards Organization (150).

O uso de computadores está aumentando em quase todos os campos de atividade. Em uma era de custos continuamente crescentes, os custos de computação têm diminuído drasticamente por causa dos desenvolvimentos rápidos, tanto na tecnologia de hardware quanto na de software. Computadores que ocupavam grandes salas e custavam milhões de dólares 25 a 30 anos atrás podem agora ser criados nas superfícies de chips de silício menores que uma unha e custando, talvez, apenas alguns dólares cada. Ironicamente, o silício é um dos materiais mais abundantes na Terra - é um dos ingredientes da areia comum. A tecnologia de circuitos integrados de silício tornou a computação tão barata que centenas de milhões de computadores de uso geral estão em utilização no mundo, ajudando pessoas nos negócios, na indústria, no governo e em suas vidas pessoais. Esse número pode facilmente dobrar dentro de alguns anos.

Este livro o desafiará por várias razões. Seus colegas durante os últimos anos provavelmente aprenderam C ou Pascal como sua primeira linguagem de programação. Você, na verdade, aprenderá tanto C como C++! Por quê? Simplesmente porque C++ inclui C, acrescentando muito mais.

Seus colegas provavelmente também aprenderam a metodologia de programação conhecida como programação estruturada. Você aprenderá tanto a programação estruturada como a mais nova e excitante metodologia, a programação orientada a objetos. Por que ensinamos ambas?A orientação a objetos certamente será a metodologia-chave de programação para a próxima década. Você criará e trabalhará com muitos objetos neste curso. Mas descobrirá que a estrutura interna daqueles objetos é freqüentemente mais bem construída usando-se as técnicas da programação mais e estruturada. Além disso, a lógica de manipulação de objetos é ocasionalmente mais bem expressa

com programação estruturada.
Outra razão pela qual apresentamos ambas as metodologias é que atualmente está acontecendo uma migração de grande vulto de sistemas baseados em C para sistemas baseados em C++. Existe um volume enorme do chamado "código legado em C" em uso. C foi amplamente usada por mais ou menos um quarto de século e seu uso em anos recentes tem aumentado drasticamente. Uma vez que as pessoas apreendem C++, elas acham esta mais poderosa que C e freqüentemente escolhem migrar para C++. Assim, começam a converter seus sistemas "legados" para C++. Começam a usar os vários recursos que C++ apresenta, geralmente chamados de "melhorias de C++ em relação a C", para melhorar seu estilo de codificar programas "à maneira de C". Finalmente, começam a empregar os recursos de programação orientada a objetos de C++ para conseguir os benefícios completos do uso da linguagem.
N. de R.T.: Termo usado para descrever uma das atividades de desenvolvimento de software, que consiste em reunir a equipe de desenvolvimento para analisar o código de um programa e discutir a implementação e as decisões tomadas pelos programadores; usada como ferramenta de treinamento e como parte de um processo de qualidade no desenvolvimento de software.

Um fenômeno interessante em linguagens de programação é que a maioria dos fornecedores simplesmente comercializa um produto de C/C++ combinado, em lugar de oferecer produtos separados. Isto dá aos usuários a possibilidade de continuar programando em C, se assim desejarem, e então migrar gradualmente para C++ quando acharem conveniente.
c++ se tornou a linguagem de implementação preferencial para construir sistemas de computação de alto desempenho. Mas pode ela ser ensinada em um primeiro curso de programação, que é o público pretendido para este livro? Acreditamos que sim. Nove anos atrás, enfrentamos um desafio semelhante quando Pascal era a linguagem firmemente estabelecida nos cursos de graduação em Ciência da Computação. Escrevemos C How to Program. Centenas de universidades pelo mundo agora usam a terceira edição de C How to Program. Os cursos baseados naquele livro têm demonstrado ser igualmente tão adequados quanto seus predecessores baseados em Pascal. Nenhuma diferença significativa foi observada, a não ser que agora os estudantes mais motivados porque sabem que é mais provável que usem C, em seus cursos de nível mais avançado e em suas carreiras, do que Pascal. Os estudantes, ao aprender em C, sabem também que estarão mais bem preparados para aprender C++ e a nova linguagem baseada em C++, com recursos para o ambiente Internet, Java.
Nos primeiros cinco capítulos do livro, você aprenderá programação estruturada em C++, a "parte C" de C++ e as "melhorias de C++ em relação a C". No conjunto do livro, você aprenderá programação orientada a objetos em C++. Contudo, não queremos fazê-lo esperar até o Capítulo 6 para começar a apreciar a orientação a objetos. Assim, cada um dos cinco primeiros capítulos

conclui com uma seção intitulada “Pensando em objetos”. Estas seções introduzem conceitos e terminologia básicos sobre programação orientada a objetos. Quando atingirmos o Capítulo 6, “Classes e Abstração de Dados”, você estará preparado para começar a ultilizar C++ para criar objetos e escrever programas orientados a objetos.
Este primeiro capítulo tem três partes. A primeira parte introduz os fundamentos de computadores e sua
programação. A segunda parte faz com que você comece a escrever imediatamente alguns programas simples em C++. A terceira parte o ajuda a começar a “pensar em objetos”.
Bem, é isso aí! Você está para começar um caminho desafiante e recompensador. A medida que avançar, caso
queira se comunicar conosco, envie-nos um e-mau para
deitei@deitei.com
ou visite nosso site na World Wide Web em
http: //www . deitei, com
Nós lhe responderemos imediatamente. Esperamos que você aprecie aprender com a ajuda de C++ Como Programar
Você pode querer usar a versão interativa do livro em CD-ROM, a C++ Multimedia Cyber Classroom: Third Edition.
1.2 O que é um computador?
Um computador é um dispositivo capaz de executar cálculos e tomar decisões lógicas em velocidades milhões, e até bilhões, de vezes mais rápidas do que podem os seres humanos. Por exemplo, muitos dos computadores pessoais de hoje podem executar centenas de milhões de adições por segundo. Uma pessoa usando uma calculadora de mesa poderia necessitar de décadas para completar o mesmo número de cálculos que um computador pessoal poderoso pode executar em um segundo. (Pontos para pensar: como você saberia se a pessoa somou os números corretamente? Como você saberia se o computador somou os números corretamente?) Os supercomputadores mais rápidos de hoje podem executar centenas de bilhões de adições por segundo - aproximadamente o número de cálculos que centenas de milhares de pessoas podem executar em um ano! E computadores capazes de executar trilhões de instruções por segundo já estão funcionando em laboratórios de pesquisas!
Computadores processam dados sob o controle de conjuntos de instruções chamados de programas de computador Estes programas de computador guiam o computador através de conjuntos ordenados de ações especificados por pessoas chamadas de programadores de computador
Um computador é composto por vários dispositivos (como o teclado, tela, mouse, discos, memória, CD-ROM
e unidades de processamento) que são chamados de hardware. Os programas de computador que são executados em um computador são chamados de software. Os custos de hardware têm declinado drasticamente em anos recentes,

ao ponto de os computadores pessoais se tornarem um artigo comum. Infelizmente, os custos de desenvolvimento de software têm subido continuamente, à medida que os programadores desenvolvem aplicativos sempre cada vez mais poderosos e complexos, sem que tenha surgido uma tecnologia de desenvolvimento de software com melhoria significativa. Neste livro, você aprenderá métodos de desenvolvimento de software comprovados que podem reduzir os custos do desenvolvimento de software - programação estruturada, refinamento top-down passo a passo, divisão do programa em funções, programação baseada em objetos, programação orientada a objetos, projeto orientado a objetos e programação genérica.

1.3 Organização de computadores

Não importando as diferenças em aparência física, virtualmente todo computador pode ser visto como sendo dividido em seis unidades lógicas ou seções. São as seguintes:

1. Unidade de entrada. Esta é a “seção receptora” do computador. Ela obtém informações (dados e programas de computador) de vários dispositivos de entrada e coloca estas informações à disposição das outras unidades, de forma que as informações possam ser processadas. A maioria das informações, hoje, é inserida nos computadores através de dispositivos como teclados e mouses. As informações também podem ser inseridas falando-se com o computador e por varredura de imagens.
2. Unidade de saída. Esta é a “seção de expedição” do computador. Ela pega as informações que foram processadas pelo computador e as coloca em vários dispositivos de saída para tornar as informações disponíveis para uso fora do computador. A maioria das saídas de informações de computadores hoje é exibida em telas, impressa em papel ou usada para controlar outros dispositivos.

3. Unidade de memória. Esta é a seção de “armazenamento” de acesso rápido, de capacidade relativamente baixa, do computador. Retém informações que foram fornecidas através da unidade de entrada, de forma que as informações possam ser tornadas imediatamente disponíveis para processamento quando forem necessárias. A unidade de memória retém também informações processadas até que aquelas informações possam ser colocadas em dispositivos de saída pela unidade de saída. A unidade de memória é freqüentemente chamada de memória ou memória primária.
4. Unidade de aritmética e lógica (UAL). Esta é a “seção industrial” do computador. Ela é responsável por executar cálculos como adição, subtração, multiplicação e divisão. Contém os mecanismos de decisão que permitem ao computador, por exemplo, comparar dois itens na unidade de memória para determinar se eles são ou não iguais.
5. Unidade central de processamento (CPU, central processing unit). Esta é a “seção administrativa” do computador. E a coordenadora do computador e é responsável por supervisionar a operação das outras seções. A CPU diz à unidade de entrada quando as informações devem ser lidas para a unidade de

memória, diz à UAL quando informações da unidade de memória devem ser utilizadas em cálculos e diz à unidade de saída quando enviar as informações da unidade de memória para certos dispositivos de saída.
6. Unidade de armazenamento secundário. Esta é a seção de “armazenamento” de grande capacidade e longo prazo do computador. Os programas ou dados que não estão sendo ativamente usados pelas outras unidades são normalmente colocados em dispositivos de armazenamento secundário (como discos) até que sejam novamente necessários, possivelmente horas, dias, meses ou até anos mais tarde. As informações no armazenamento secundário levam muito mais tempo para serem acessadas do que as informações na memória primária. O custo por unidade de armazenamento secundário é muito menor que o custo por unidade de memória primária.

1.4 Evolução dos sistemas operacionais

Os primeiros computadores eram capazes de executar só um trabalho ou tarefa de cada vez. Esta forma de operação de computador é freqüentemente chamada de processamento em lotes com usuário único. O computador executa um único programa de cada vez, enquanto processa os dados em grupos ou lotes. Nestes primeiros sistemas, os usuários

expressão computação cliente/servidor. C e C++ se tomaram as linguagens de programação preferidas para escrever software para sistemas operacionais, para computação em redes e para aplicativos distribuídos para ambientes cliente! servidor. Os sistemas operacionais mais populares hoje em dia, como o UNIX, Linux e os sistemas baseados no Windows da Microsoft, oferecem os tipos de recursos discutidos nesta seção.

1.6 Linguagens de máquina, linguagens simbólicas
e linguagens de alto nível

Os programadores escrevem instruções em várias linguagens de programação, algumas diretamente compreensíveis pelo computador e outras que exigem passos de tradução intermediária. Centenas de linguagens de computador estão em uso hoje em dia. Elas podem ser divididas em três tipos gerais:
1. Linguagens de máquina,
2. Linguagens simbólicas,
3. Linguagens de alto nível.

Qualquer computador pode entender diretamente apenas sua própria linguagem de máquina. A linguagem de máquina é a “linguagem natural” de um computador em particular. Ela é definida pelo projeto de hardware daquele computador. As linguagens de máquina consistem geralmente em seqüências de números (em última instância reduzidos a ls e Os) que instruem os computadores a executar suas operações mais elementares, uma de cada vez. As linguagens de máquina são dependentes da máquina, isto é, uma linguagem de máquina em particular pode ser usada em só um tipo de computador. As linguagens de máquina são incômodas para as pessoas, como pode ser visto

pela seguinte seção de um programa em linguagem de máquina que soma as horas extras a pagar ao salário base e armazena o resultado em pagamento bruto.

```
+130 0042 77 4
+14 005 934 19
+12 002 74 027
```

À medida que os computadores se tornaram mais populares, tornou-se evidente que a programação em linguagem de máquina era muito lenta, tediosa e sujeita a erros. Em vez de usar as seqüências de números que os computadores podiam entender diretamente, os programadores começaram a usar abreviações semelhantes às das palavras inglesas para representar as operações elementares do computador. Estas abreviações* formaram a base das linguagens simbólicas. Programas tradutores chamados de assemblers (montadores) foram desenvolvidos para converter, à velocidade do computador, os programas em linguagem simbólica para a linguagem de máquina. A seção de um programa em linguagem simbólica mostrada a seguir também soma horas extras a pagar ao salário base e armazena o resultado em pagamento bruto, mas com maior clareza que seu equivalente em linguagem de máquina:

```
LOAD BASEPAY
ADD OVERPAY
STORE GROSSPAY
```

Embora tal código seja mais claro para as pessoas, ele é incompreensível para os computadores até ser traduzido para a linguagem de máquina.

O uso de computadores aumentou rapidamente com o advento das linguagens simbólicas, mas estas ainda exigiam muitas instruções para realizar até as tarefas mais simples. Para acelerar o processo de programação, foram desenvolvidas as linguagens de alto nível, nas quais uma única instrução realiza tarefas significativas. Programas tradutores chamados compiladores convertem os programas em linguagem de alto nível para linguagem de máquina. As linguagens de alto nível permitem que os programadores escrevam instruções que parecem quase com o

*N. de R.T. Denominadas, também, símbolos mnemônicos.

inglês comum e contêm notações matemáticas comumente usadas. Um programa de folha de pagamento escrito em uma linguagem de alto nível pode conter um comando como:

```
grossPay = basePay + overTimePay
```

Obviamente, as linguagens de alto nível são muito mais desejáveis do ponto de vista do programador que as linguagens de máquina ou as linguagens simbólicas. C e C++ estão entre as mais poderosas e amplamente utilizadas linguagens de alto nível.

O processo de compilar um programa em linguagem de alto nível para linguagem de máquina pode tomar um tempo considerável do computador. Por isso, foram desenvolvidos programas interpretadores, os quais podem executar diretamente programas em linguagem alto de nível sem a necessidade de

compilar aqueles programas para linguagem de máquina. Embora programas compilados executem mais rápido que programas interpretados, os interpretadores são populares em ambientes de desenvolvimento de programas em que os mesmos são modificados freqüentemente, à medida que novas características são acrescentadas aos programas e seus erros são corrigidos. Uma vez que o desenvolvimento de um programa tenha terminado, uma versão compilada do mesmo pode ser produzida para ser executada de forma mais eficaz.

1.7 A história de C e C++

C++ uma evolução de C, que evoluiu de duas linguagens de programação anteriores, BCPL e B. BCPL foi desenvolvida em 1967 por Martin Richards, como uma linguagem para escrever software de sistemas operacionais e compiladores. Ken Thompson modelou muitas características de sua linguagem B inspirando-se em suas correspondentes em BCPL e usou B para criar as primeiras versões do sistema operacional UNIX no Bell Laboratories, em 1970, em um computador DEC PDP-7. Tanto BCPL como B eram linguagens typeless, ou seja, sem definição de tipos de dados - todo item de dados ocupava uma "palavra" na memória e o trabalho de tratar um item de dados como um número inteiro ou um número real, por exemplo, era de responsabilidade do programador.

A linguagem C foi derivada de B por Dennis Ritchie no Bell Laboratories e foi originalmente implementada em um computador DEC PDP-l 1 em 1972. C usa muitos conceitos importantes de BCPL e B, ao mesmo tempo que acrescenta tipos de dados e outras características. C se tornou inicialmente conhecida como a linguagem de desenvolvimento do sistema operacional UNIX. Hoje em dia, a maioria dos sistemas operacionais são escritos em C ei ou C++. C está atualmente disponível para a maioria dos computadores. C é independente de hardware. Com um projeto cuidadoso, é possível se escrever programas em C que são portáveis para a maioria dos computadores.

No final dos anos 70, C evoluiu para o que agora é chamado de "C tradicional", "C clássico" ou "C de
Kernighan e Ritchie". A publicação pela Prentice Hall, em 1978, do livro de Kemighan e Ritchie, The C Programming Language, chamou muita atenção para a linguagem.

O uso difundido de C com vários tipos de computadores (às vezes chamados de plataformas de hardware) infelizmente levou a muitas variações da linguagem. Estas eram semelhantes, mas freqüentemente incompatíveis. Isto era um problema sério para desenvolvedores de programas que precisavam escrever programas portáveis que seriam executados em várias plataformas. Tornou-se claro que uma versão padrão de Cera necessária. Em 1983, foi criado o comitê técnico X3J 11 do American National Standards Committee on Computers and Information Processing (X3) para "produzir uma definição não-ambígua e independente de máquina da linguagem". Em 1989, o padrão foi aprovado. O ANSI cooperou com a International Standards Organization (ISO) para padronizar C a nível mundial; o documento de padronização conjunta foi publicado em 1990 e é chamado de ANSI/ISO 9899: 1990. Cópias deste

documento podem ser pedidas ao ANSI. A segunda edição de Kernighan e Ritchie, publicada em 1988, reflete esta versão chamada ANSI C, uma versão da linguagem agora utilizada mundialmente.
Dica de portabilidade 1.1
Como C é uma linguagem padronizada, independente de hardware e amplamente disponível, aplicativos escritos em C podem ser freqüentemente executados com pouca ou nenhuma modificação em uma ampla variedade de sistemas de computação diferentes.
C++, uma extensão de C, foi desenvolvida por Bjame Stroustrup no início dos anos 80 no Beil Laboratories. C++
apresenta várias características que melhoram a linguagem C, mas o mais importante é que fornece recursos para a
programação orientada a objetos.

CAPÍTULO 1 - INTRODUÇÃO AOS COMPUTADORES E À PROGRAMAÇÃO C ++ 57
Existe uma revolução em andamento na comunidade de software. Construir software rápida, correta e economicamente permanece um objetivo difícil de se atingir, e isto em uma época em que a demanda por softwares novos e mais poderosos está aumentando rapidamente. Objetos são, essencialmente, componentes de software reutilizáveis que modelam coisas do mundo real. Os desenvolvedores de software estão descobrindo que usar uma abordagem de implementação e projeto modulares, orientada a objetos, pode tornar os grupos de desenvolvimento de software muito mais produtivos do que é possível usando-se técnicas de programação anteriormente populares, como a programação estruturada. Os programas orientados a objetos são mais fáceis de entender, corrigir e modificar.
Muitas outras linguagens orientadas a objetos foram desenvolvidas, incluindo Smalltalk, desenvolvida no Palo Alto Research Center da Xerox (PARC). Smalltalk é uma linguagem orientada a objetos pura - literalmente, tudo nela é um objeto. C++ é uma linguagem híbrida - é possível programar em C++ tanto em um estilo semelhante ao de C, como em um estilo orientado a objetos, ou ambos. Na Seção 1.9, discutimos a nova e excitante linguagem Java - baseada em C e em C++.
1
1.8 A biblioteca padrão de C++
Os programas em C÷+ consistem em peças chamadas classes e funções. Você pode programar cada peça que possa precisar para criar um programa em C++. Mas a maioria dos programadores de C++ aproveitam as ricas coleções de classes e funções existentes na biblioteca padrão de C+÷. Deste modo, existem realmente duas fases para se aprender o “mundo” C++. A primeira é aprender a linguagem C++ propriamente dita e a segunda é aprender como utilizá-las as classes e funções da biblioteca padrão de C++. Por todo o livro, discutimos muitas destas classes e funções. O livro de Plauger é leitura obrigatória para programadores que precisam de uma compreensão profunda das funções da biblioteca de ANSI C que estão incluídas em C++, como implementá-las e como

utilizá-las para escrever código portável. As bibliotecas padrão de classes são geralmente fornecidas pelos fornecedores de compiladores. Muitas bibliotecas de classes para finalidades especiais são oferecidas por fornecedores independentes de software.

JObservação de engenharia de software 1.1

Use uma “abordagem de blocos de constru ção” para criar programas. Evite reinventar a roda. Use pedaços existentes onde for possível - isto é chamado de “reutilização de software” e é um aspecto central da programação orientada a objetos.

Observação de engenharia de software 1.2

_____ Quando estiver programando em C++, você usará tipicamente os seguintes blocos de construção: classes e funções da biblioteca padrão de C+ +, classes e funções que você mesmo cria e classes e funções de várias bibliotecas populares não-padrão.

A vantagem de criar suas próprias funções e classes é que você saberá exatamente como funcionam. Você poderá examinar o código em C++. A desvantagem é o tempo consumido e o esforço complexo para projetar, desenvolver
c e manter novas funções e classes que sejam corretas e que operem de forma eficaz.

Dica de desempenho 1.1

Usarfunções e classes de bibliotecas padrão, em vez de escrever suas próprias versões equivalentes, pode melhorar o desempenho do programa, porque este software é cuidadosamente escrito para ser executado deforma correta e eficaz.

Dica de portabilidade 1.2

Usarfunções e classes de bibliotecas padrão, em vez de escrever suas próprias versões equivalentes, pode melhorar a portabilidade do programa, porque este software é incluído em virtualmente todas as implementações de C++.

1 .9 Java e Java Como Programar

Muitas pessoas acreditam que a próxima área importante na qual os microprocessadores terão um impacto profundo é na dos aparelhos eletrodomésticos e eletrônicos de consumo. Reconhecendo isto, a Sun Microsystems financiou um projeto corporativo interno de desenvolvimento, com nome de código Green, em 1 99 1 . O projeto resultou no desenvolvimento de uma linguagem baseada em C e C÷÷, que seu criador, James Gosling, denominou Oak (carvalho, em inglês), em homenagem a um carvalho que ele via através da janela de seu escritório na Sun. Mais tarde, descobriram que já existia uma linguagem de computador denominada Oak. Quando um grupo de pessoas da Sun estava em uma cafeteria local, o nome Java foi sugerido e pegou.

Mas o projeto Green enfrentou algumas dificuldades. O mercado para aparelhos eletrônicos de consumo inteligentes não estava se desenvolvendo tão rapidamente quanto a Sun havia previsto. Pior ainda, um importante contrato, de

cuja concorrência a Sun havia participado, foi assinado com outra empresa. Por isto, o projeto corria o risco de ser cancelado. Por pura sorte, a popularidade da World Wide Web explodiu em 1 993 e o pessoal da Sun viu o potencial imediato de usar Java para criar páginas da Web com o assim chamado conteúdo dinâmico.

A Sun anunciou Java formalmente em uma feira comercial em maio de 1 995. Normalmente, um acontecimento como este não chamaria muita atenção. Entretanto, Java despertou interesse imediato na comunidade comercial devido ao interesse fenomenal na World Wide Web. Java agora é usada para criar páginas da Web com conteúdo dinâmico e interativo, para desenvolver aplicativos empresariais de grande porte, para aumentar a funcionalidade de servidores da Web (os computadores que oferecem o conteúdo que vemos em nosso navegadores da Web), para oferecer aplicativos para apare- lhos eletrônicos de consumo (tais como telefones celulares, pagers e assistentes pessoais digitais) e muito mais.

Em 1995, estávamos cuidadosamente acompanhando o desenvolvimento de Java pela Sun Microsystems. Em novembro de 1995, participamos de uma conferência sobre Internet em Boston. Um representante da Sun Microsystems nos fez uma estimulante apresentação sobre Java. A medida em que as conversações prosseguiram, tomou-se claro para nós que Java desempenharia um papel importante no desenvolvimento de páginas da Web interativas usando multimidia. Mas vimos imediatamente um potencial muito maior para a linguagem.

Visualisamos Java como uma bela linguagem para ensinar a estudantes de programação do primeiro ano os aspectos fundamentais de gráficos, imagens, animação, audio, vídeo, bancos de dados, redes, multithreading e computação colaborativa. Pusemos mãos à obra para escrever a primeira edição de lava How to Program, que foi publicada em tempo para as aulas do outono americano de 1996. lava: Como Programar - Terceira Edição foi publicada em 1999*.

Além de sua proeminência no desenvolvimento de aplicações baseadas na Internet - e intranets, Java certamente se tomará a linguagem preferida para implementar software para dispositivos que se comuniquem através de uma rede (tais como telefones celulares, pagers e assistentes pessoais digitais). Não se surpreendam quando seus novos aparelhos de som e outros aparelhos em sua casa estiverem conectados em rede usando a tecnologia Java!

1.10 Outras linguagens de alto nível

Centenas de linguagens de alto nível foram desenvolvidas, mas só algumas obtiveram ampla aceitação. FORTRAN (FORmula TRANslator) foi desenvolvida pela IBM Corporation, entre 1954 e 1957, para ser usada no desenvolvimento de aplicativos científicos e de engenharia que exigem computações matemáticas complexas. FORTRAN é ainda amplamente usada, especialmente em aplicativos de engenharia.

COBOL (COmmon Business Oriented Language) foi desenvolvida em 1959 por fabricantes de computadores, usuários do governo e usuários industriais de computadores. COBOL é principalmente utilizada para aplicativos comerciais

que exigem manipulação precisa e eficiente de grandes quantidades de dados. Hoje em dia, mais da metade de todo o software para aplicações comerciais ainda é programado em COBOL.
Pascal foi projetada aproximadamente na mesma época que C pelo Professor Niklaus Wirth e foi planejada
para uso acadêmico. Falaremos mais sobre Pascal na próxima seção.
1.11 Programação estruturada
Durante os anos 60, muitos esforços de desenvolvimento de software de grande porte enfrentaram sérias dificuldades. Os cronogramas de software estavam sempre atrasados, os custos excediam em muito os orçamentos e os
5N. de R.: A edição brasileira foi publicada pela Bookman Companhia Editora no ano de 2000.

produtos terminados não eram confiáveis. As pessoas começaram a perceber que o desenvolvimento de software era uma atividade muito mais complexa do que haviam imaginado. Atividades de pesquisa na década de 60 resultaram na evolução da programação estruturada - uma abordagem disciplinada à construção de programas que sejam mais claros que programas não-estruturados, mais fáceis para testar e depurar e mais fáceis para modificar, O Capítulo 2 discute os princípios da programação estruturada. Os Capítulos 3 a 5 desenvolvem muitos programas estruturados.
Um dos resultados mais palpáveis desta pesquisa foi o desenvolvimento da linguagem de programação Pascal por Niklaus Wirth em 1971. Pascal, nomeado em homenagem ao matemático e filósofo do século XVII Blaise Pascal, foi projetada para ensinar programação estruturada em ambientes acadêmicos e se tomou rapidamente a linguagem de programação preferida na maioria das universidades. Infelizmente, a linguagem não dispõe de muitas das características necessárias para tomá-la útil a aplicativos comerciais, industriais e governamentais; assim, ela não foi amplamente aceita fora das universidades.
A linguagem de programação Ada foi desenvolvida sob o patrocínio do Departamento de Defesa dos Estados Unidos (DOD), durante a década de 70 e início dos anos 80. Centenas de linguagens diferentes estavam sendo usadas para produzir os enormes sistemas de software de comando e controle do DOD. O DOD quis uma única linguagem que preenchesse a maioria de suas necessidades. Pascal foi escolhida como base, mas a versão final da linguagem Ada é bastante diferente de Pascal. A linguagem foi nomeada em homenagem a Lady Ada Lovelace, filha do poeta Lord Byron. Lady Lovelace é geralmente considerada como tendo escrito o primeiro programa de computador do mundo, no início do século 19 (para o Analytical Engine Mechanical Computing Device, projetado por
= Charles Babbage). Um recurso importante de Ada é chamado multitasking; permite que os programadores especifiquem que muitas atividades devem acontecer em paralelo. As outras linguagens de alto nível amplamente utlilizadas que discutimos - inclusive C e C++ - permitem geralmente ao programador

escrever programas que executam só uma atividade de cada vez.
a
1.12 A tendência-chave em software: tecnologia de objetos
Um dos autores, HMD, lembra da grande frustração que era sentida pelas organizações de desenvolvimento de software na década de 60, especialmente por aquelas que desenvolviam projetos de grande porte. Durante seus anos na universidade, HMD teve o privilégio de trabalhar durante os verões nas equipes de desenvolvimento de sistemas operacionais para timesharing, com memória virtual, em uma empresa fabricante de computadores líder de merca- do. Esta era uma grande experiência para um estudante universitário. Mas, no verão de 1967, ele caiu na realidade quando a companhia “desistiu” de fornecer como produto comercial o sistema particular no qual centenas de pessoas estiveram trabalhando por muitos anos. Era difícil de fazer este software funcionar corretamente. Software é “coisa complexa”.
Os custos de hardware têm declinado drasticamente nos últimos anos, a ponto de os computadores pessoais terem se tornado uma comodidade. Infelizmente, os custos de desenvolvimento de software têm crescido continuamente, na medida em que os programadores desenvolvem aplicativos cada vez mais poderosos e complexos sem serem capazes de melhorar significativamente as tecnologias de desenvolvimento de software subjacentes. Neste livro, você aprenderá muitos métodos de desenvolvimento de software que podem reduzir os custos de desenvolvimento de software.
Há uma revolução em andamento na comunidade de software. Construir software rápida, correta e economicamente permanece um objetivo difícil de atingir, e isto em uma época em que a demanda por softwares novos e _____ mais poderosos está aumentando rapidamente. Objetos são, essencialmente, componentes de software reutilizáveis que modelam coisas do mundo real. Os desenvolvedores de software estão descobrindo que usar uma abordagem de implementação e projeto modulares, orientada a objetos, pode tornar os grupos de desenvolvimento de software muito mais produtivos do que é possível usando-se técnicas de programação anteriormente populares, como a programação estruturada. Os programas orientados a objetos são mais fáceis de entender, corrigir e modificar.
Melhorias na tecnologia de software começaram a aparecer com os benefícios da assim chamada programação estruturada (e as disciplinas relacionadas de análise e projeto de sistemas estruturados), sendo comprovados na década de 70. Mas foi somente quando a tecnologia de programação orientada a objetos se tomou largamente utilizada na década de 80, e muito mais amplamente utilizada na década de 90, que os desenvolvedores de software finalmente sentiram que eles tinham as ferramentas de que precisavam para obter importantes progressos no processo de desenvolvimento de software.
Na verdade, a tecnologia de objetos existe desde pelos menos meados da década de 60. A linguagem de programação C++ desenvolvida na AT&T por Bjame Stroustrup no início da década de 80 é baseada em duas

seus programas), Dicas de desempenho (técnicas que vão ajudá-lo a escrever programas que possam ser executados mais rapidamente e usando menos memória), Dicas de portabilidade (técnicas que vão ajudá-lo a escrever programas que possam ser executados, com pouca ou nenhuma modificação, em diversos computadores) e Dicas de teste e depuração (técnicas que vão ajudá-lo a remover erros de seus programas e, mais importante, técnicas que vão ajudá-lo a escrever programas sem erros desde o início). Muitas destas técnicas e práticas são apenas linhas mestras; você desenvolverá, sem dúvida, seu próprio estilo preferido de programação.j

1.13 Fundamentos de um ambiente típico de C++

Os sistemas de C++ consistem geralmente de várias partes: um ambiente de desenvolvimento de programas, a linguagem e a biblioteca padrão de C÷+. A discussão a seguir explica um ambiente de desenvolvimento de programas em C++ típico, mostrado na Fig. 1.1.

Os programas em C++ passam tipicamente por seis passos, até que possam ser executados (Fig. 1 . 1). São os seguintes: editar, pré-processar, compilar, “ligar”, link, carregar e executar. Concentramo-nos, aqui, em um sistema de C++ sob UNIX típico (Nota: os programas em C++ neste livro executarão, com pouca ou nenhuma modifica- ção, na maioria dos sistemas atuais, inclusive em sistemas baseados no Microsoft Windows). Se você não estiver usando um sistema UNIX, consulte os manuais para seu sistema ou pergunte a seu instrutor como realizar estas

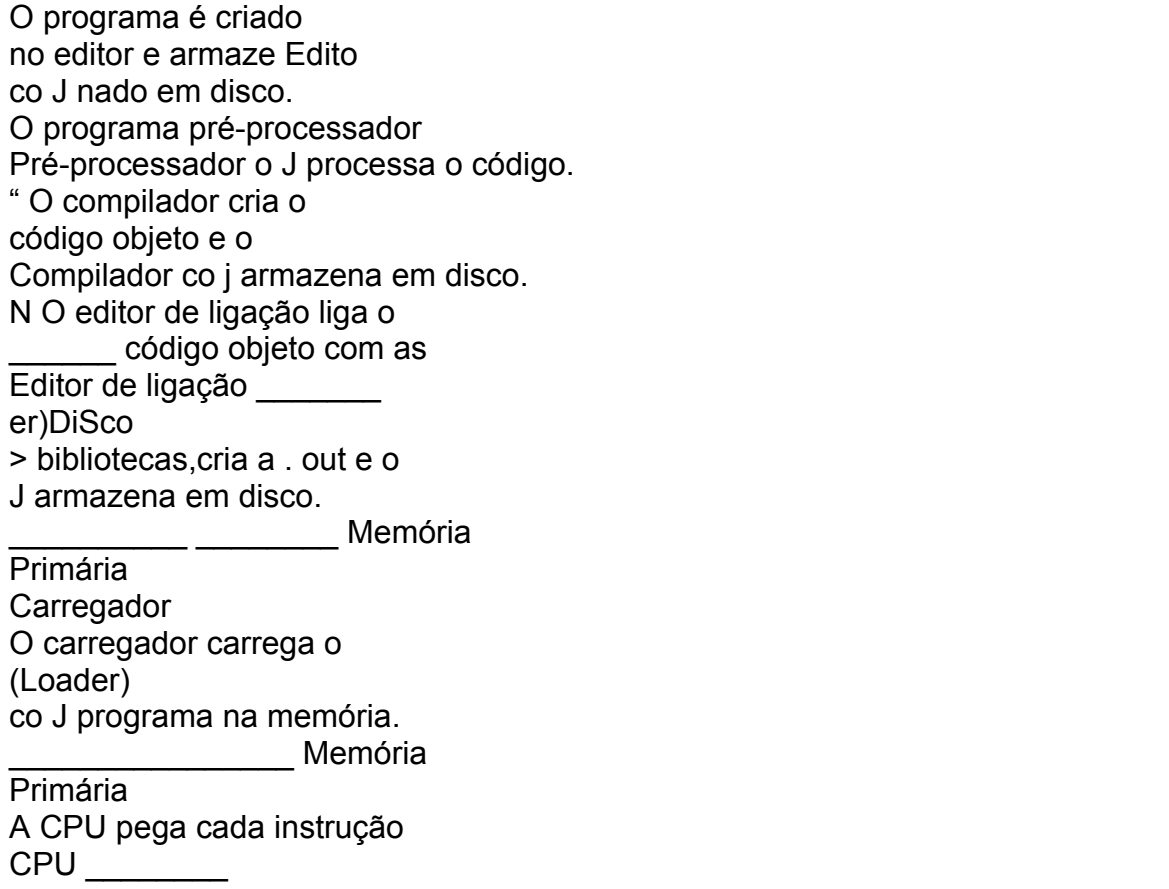
: tarefas em seu ambiente.

A primeira fase consiste em editar um arquivo. Isto é realizado com um programa editor O programador digita um programa em C++ usando o editor e, se necessário, faz correções. O programa então é armazenado em um dispositivo de armazenamento secundário, como um disco. Os nomes dos arquivos de programas em

freqüentemente terminam com as extensões . cpp. . cxx. ou C (note que C está em maiúscula). Veja a documenta- ção de seu ambiente de C+÷ para mais informação sobre extensões de nomes de arquivos. Dois editores amplamente usados em sistemas UNIX são ovi e o ernacs. Os pacotes de software de C++, como o Borland C++ e o Microsoft Visual C++ para computadores pessoais, têm editores embutidos que estão integrados naturalmente ao ambiente de programação. Aqui, assumimos que o leitor saiba como editar um programa.

Em seguida, o programador dá o comando para compilar o programa. O compilador traduz o programa em C++ para código em linguagem de máquina (também chamado de código objeto). Em um sistema C++, um programa pré-processador é executado automaticamente, antes de começar a fase de tradução pelo compilador. O pré- processador do C++ obedece a comandos especiais chamados de diretivas para o pré-processador, que indicam que certas manipulações devem ser feitas sobre o programa antes da compilação. Estas manipulações normalmente consistem em incluir outros arquivos de texto no arquivo a ser compilado e executam substituições de texto variadas. As

diretivas de pré-processador mais comuns são discutidas nos capítulos iniciais; uma discussão detalhada de todas as características do pré-processador é feita no capítulo intitulado “O pré-processador”. O pré-processador é invocado pelo compilador antes de o programa ser convertido para linguagem de máquina.
A próxima fase é chamada ligação. Os programas em C++ contêm tipicamente referências a funções definidas em outro lugar, como nas bibliotecas padrão ou nas bibliotecas privadas de grupos de programadores que trabalham juntos em um projeto particular. O código objeto produzido pelo compilador C++ contém tipicamente “buracos” devido a estas partes que estão faltando. Um editor de ligação (linker) liga o código objeto com o código das funções que estão faltando, para produzir uma imagem executável (sem pedaços faltando). Em um sistema típico com UNIX, o comando para compilar e “ligar” um programa escrito em C++ é CC. Para compilar e “ligar” um programa chamado bemvindo C. , digite
CC bemvindo.C
no prompt* do UNIX e aperte a tecla Enter (ou tecla Return). Se o programa compilar e “ligar” corretamente, é produzido um arquivo chamado a out. Isto é a imagem executável de nosso programa bemvindo. c.
5N. de R.T.: Termo utilizado para designar a mensagem ou caractere(s) que indica(m) que um programa está solicitando dados de entrada.



e a executa, possivelmente
armazenando novos valores
_____ / de dados na medida em que o
j programa vai sendo executado.
Fig. 1.1 Um ambiente típico de C++.
A próxima fase é chamada de carga (loading). Antes de um programa poder ser executado, o programa deve primeiro ser colocado na memória. Isto é feito pelo loader* (carregador), que pega a imagem executável do disco e transfere a mesma para a memória. Componentes adicionais de bibliotecas compartilhadas que suportam o programa do usuário também são carregados.
Finalmente, o computador, sob controle de sua CPU, executa o programa, uma instrução por vez. Para carregar e executar o programa em um sistema UNIX, digitamos a. out no prompt do UNIX e pressionamos Return.
*N. de R.T.: Programa normalmente fornecido junto com o sistema operacional, cuja tarefa é carregar na memória do computador um programa no formato executável.

Programas nem sempre funcionam na primeira tentativa. Cada uma das fases precedentes pode falhar por
causa de vários erros que discutiremos. Por exemplo, um programa, ao ser executado, poderia tentar dividir por zero (uma operação ilegal em computadores, da mesma maneira que é ilegal em aritmética). Isto faria o computador
; imprimir uma mensagem de erro. O programador, então, retornaria à fase editar, faria as correções necessárias e passaria pelas fases restantes novamente, para determinar se as correções estão funcionando corretamente.
Erro comum de programação 1.1
Erros como os de divisão por zero acontecem quando um programa está sendo executado; por isso, estes erros são chamados de “erros durante a execução “. Dividir por zero é geralmente um erro fatal, isto é, um erro que causa o término imediato do programa sem este ter executado seu trabalho com sucesso. Os erros não-fatais permitem que os programas sejam executados até a conclusão, freqüentemente produzindo resultados incorretos. (Nota: em alguns sistemas, dividir por zero não é um erro fatal. Consulte a
documentação do seu sistema.)
A maioria dos programas em C++ faz entrada e/ou saída de dados. Certas funções de C++ recebem sua entrada de cm (o “stream padrão de entrada “; pronunciado “cê-in”), o qual normalmente é o teclado, porém cm pode ser conectado a outro dispositivo. Os dados são freqüentemente enviados para saída em cout (o “stream padrão de
saída “ ; pronunciado “cê-aut”), que normalmente é a tela do computador, mas cout pode ser conectado a outro dispositivo. Quando dizemos que um programa imprime um resultado, normalmente queremos dizer que o resultado
é exibido na tela. Os dados podem ser enviados para saída através de outros

dispositivos, como discos e impressoras. Existe também um “stream padrão de erros”, chamado cerr. O stream cerr (normalmente conectado à tela) é
j usado para exibir mensagens de erro. E comum que os usuários direcionem os dados de saída normais, isto é, cout, para um dispositivo diferente da tela, ao mesmo tempo em que mantêm cerr direcionado para a tela, de maneira
que o usuário possa ser imediatamente informado de erros.

1 .1 4 Tendências de hardware

(A comunidade de programação progride na torrente contínua de melhorias drásticas nas tecnologias de hardware, software e comunicação. A cada ano, as pessoas normalmente esperam pagar pelo menos um pouco mais pela maioria dos produtos e serviços. No caso das áreas de computação e comunicação, tem acontecido o contrário,
especialmente no que diz respeito ao custo do hardware para suportar estas tecnologias. Durante muitas décadas, e sem perspectivas de mudança previsível no futuro, os custos de hardware têm caido rapidamente, para não dizer
‘ precipitadamente. Este é um fenômeno da tecnologia, uma outra força propulsora do estrondoso desenvolvimento econômico atual. A cada um ou dois anos, as capacidades dos computadores, especialmente a quantidade de memória que têm para executar programas, a quantidade de memória secundária (tal como armazenamento em disco) que
têm para guardar programas e dados a longo prazo e as velocidades de seus processadores - a velocidade na qual os computadores executam seus programas (isto é, fazem seu trabalho), cada uma delas tende a aproximadamente dobrar. O mesmo tem acontecido no campo das comunicações, com os custos despencando, especialmente nos
últimos anos, com a demanda por largura de banda nas comunicações atraindo uma enorme competição. Não conhe cemos nenhum outro em que a tecnologia evolua tão depressa e os custos caiam tão rapidamente.

Quando o uso de computadores explodiu, nas décadas de 60 e 70, falava-se muito nas enormes melhorias que os
computadores e as comunicações trariam para a produtividade humana. Mas estas melhorias não se concretizaram. As organizações estavam gastando grandes somas de dinheiro em computadores e certamente empregando-os eficazmente, mas sem obter os ganhos de produtividade que haviam sido esperados. Foi a invenção da tecnologia dos
o microprocessadores em circuitos integrados e sua grande proliferação no fim da década de 70 e na década de 80 que criou a base para as melhorias de produtividade da década de 90 que foram tão cruciais para a prosperidade econômica.

1.15 História da Internet

No fim da década de 60, um dos autores (HMD) era um estudante de pós-graduação no MIT. Sua pesquisa no projeto Mac do MIT (atualmente o Laboratório de Ciência da Computação - a sede do World Wide Web Consortium) foi financiada pela ARPA - a Agência para Projetos de Pesquisa Avançados do Departamento de Defesa americano. A

ARPA patrocinou uma conferência, realizada na University of Illinois em Urbana-Champaign, reunindo várias de- zenas de estudantes de pós-graduação financiados pela ARPA para compartilhar suas idéias. Durante esta conferência, a ARPA divulgou o projeto de interligar em uma rede os principais sistemas de computação de cerca de uma dezena de universidades e instituições de pesquisa financiadas pela ARPA. Eles iriam ser conectados com linhas de comunicações operando a uma velocidade - espantosa para a ocasião - de 56KB (isto é, 56.000 bits por segundo), isto em uma época em que a maioria das pessoas (das poucas que podiam) se conectavam através de linhas telefônicas a uma taxa de 1 00 bits por segundo. HMD lembra claramente da excitação provocada pela conferência. Pesquisadores de Harvard falavam em se comunicar com o "supercomputador" Univac 1 108 instalado no outro lado do país, na University of Utah, para executar cálculos relacionados com suas pesquisas em computação gráfica. Muitas outras possibilidades excitantes foram levantadas. A pesquisa acadêmica estava prestes a dar um grande salto adiante. Pouco tempo depois desta conferência, a ARPA deu início à implementação da rede que se tornou rapidamente conhecida como ARPAnet, a avó da internet de hoje.

As coisas funcionaram de forma diferente do que foi planejado originalmente. Em vez do principal benefício ser o dos pesquisadores poderem compartilhar os computadores uns dos outros, rapidamente tornou-se claro que simplesmente possibilitar que os pesquisadores se comunicassem rápida e facilmente entre eles mesmos através do que se tornou conhecido como correio eletrônico (abreviatura e-mail) viria a ser o principal benefício da ARPAnet. Isto é verdade ainda hoje na Internet, com o e-mau facilitando comunicações de todos os tipos entre milhões de pessoas em todo o mundo.

Um dos principais objetivos da ARPA para a rede era permitir que diversos usuários enviassem e recebessem informações ao mesmo tempo através das mesmas vias de comunicação (tais como linhas telefônicas). A rede operava com uma técnica denominada comutação depacotes, com a qual dados digitais eram enviados em pequenos conjuntos, denominados pacotes. Os pacotes continham dados, informações de endereçamento, informações para controle de erros e informações de seqüenciamento. A informação de endereçamento era usada para indicar a rota dos pacotes de dados até seu destino. A informação de seqüenciamento era usada para ajudar a montar novamente os pacotes (que - devido aos mecanismos complexos de roteamento - poderiam na verdade chegar fora de ordem) em sua ordem original para apresentação ao destinatário. Pacotes de muitas pessoas eram misturados nas mesmas li- nhas. Esta técnica de comutação de pacotes reduziu enormemente os custos de transmissão, se comparados aos custos de linhas de comunicação dedicadas.

A rede foi projetada para operar sem controle centralizado. Isto significava que, se uma parte da rede falhasse,

as partes que continuassem funcionando ainda seriam capazes de rotear pacotes dos remetentes até os destinatários através de caminhos alternativos. O protocolo para comunicação através da ARPAnet tornou-se conhecido como

TCP (o protocolo de controle de transmissão - Transmission Control Protocol). O TCP assegurava que as mensagens fossem roteadas adequada- mente do remetente até o destinatário e que estas mensagens chegassem intactas. Paralelamente à evolução inicial da Internet, organizações em todo o mundo estavam implementando suas próprias redes, tanto para comunicação intra-organizacional (isto é, dentro da organização) como interorganizacional (isto é, entre organizações). Surgiu uma enorme variedade de hardware e software para redes. Fazer com que estes se comunicassem entre si era um desafio. A ARPA conseguiu isto com o desenvolvimento do IP (o protocolo de conexão entre redes - Internetworking Protocol). criando uma verdadeira “rede de redes”. que é a arquitetura atual da Internet. O conjunto de protocolos combinados é agora comumente denominado de TCP/JP.

Inicialmente, o uso da Internet estava restrito às universidades e instituições de pesquisa; em seguida, as forças armadas se tornaram um grande usuário. Em um determinado momento, o governo decidiu permitir acesso à Internet para fins comerciais. Inicialmente, houve ressentimento entre as comunidades de pesquisa e das forças armadas - havia um sentimento de que os tempos de resposta ficariam degradados à medida que “a rede” se tornasse saturada com tantos usuários.

Na verdade, aconteceu exatamente o contrário. As empresas rapidamente se deram conta de que, fazendo um uso eficiente da Internet, poderiam ajustar suas operações e oferecer serviços novos e melhores para seus clientes, de modo que começaram a gastar enormes quantias em dinheiro para desenvolver e aprimorar a Internet. Isto desencadeou uma ferrenha competição entre as empresas de comunicação e os fornecedores de hardware e software para atender esta demanda. O resultado é que a largura de banda (isto é, a capacidade de transportar informação das linhas de comunicação) na Internet aumentou tremendamente e os custos despencaram. E amplamente reconhecido que a Internet desempenhou um papel significativo na prosperidade econômica que os Estados Unidos e muitas outras nações industrializadas têm experimentado ao longo da última década e provavelmente continuarão a experimentar por muitos anos.

1.16 História da World Wide Web

A World Wide Web permite aos usuários de computadores localizar e visualizar documentos baseados em multimídia (isto é, documentos com texto, gráficos, animações, áudios e/ou vídeos) sobre praticamente qualquer assunto. Embora a Internet tenha sido desenvolvida há mais de três décadas, a introdução da World Wide Web foi um evento relativamente recente. Em 1990, Tim Berners-Lee, do CERN (o Laboratório Europeu de Física de Partículas) desenvolveu a World Wide Web e diversos protocolos de comunicação que formam sua espinha dorsal.

A Internet e a World Wide Web certamente serão citadas entre as mais

importantes e profundas criações da espécie humana. No passado, a maioria dos aplicativos de computadores era executada em computadores “isolados”, isto é, computadores que não estavam conectados uns aos outros. Os aplicativos de hoje podem ser escritos para se comunicar entre as centenas de milhões de computadores do mundo. A Intemet mistura tecnologias de computação e comunicações. Ela facilita nosso trabalho. Torna as informações disponíveis instantaneamente em todo o mundo de forma conveniente. Torna possível que indivíduos e pequenas empresas obtenham visibilidade em todo o mundo. Está mudando a natureza da forma como os negócios são feitos. As pessoas podem pesquisar os melhores preços de virtualmente qualquer produto ou serviço. Comunidades com interesses especiais podem se manter em contato umas com as outras. Pesquisadores podem ser instantaneamente informados sobre os últimos avanços da ciência em todo o mundo.

1.17 Notas gerais sobre C++ e este livro

C++ é uma linguagem complexa. Os programadores de C+÷ experientes às vezes se orgulham em poder criar algum uso misterioso, contorcido, “enrolado”, da linguagem. Isto é uma prática ruim de programação. Ela torna os programas mais dificeis de serem lidos, mais propensos a apresentar comportamentos estranhos, mais difíceis de testar e depurar e mais difíceis de se adaptar a mudanças de requisitos. Este livro é dirigido a programadores novatos; assim, aqui enfatizamos a clareza do programa. A seguinte é nossa primeira “boa prática de programação”.

Boa prática de programação 1.1

Escreva seus programas em C+ + de uma maneira simples e direta. Isto é às vezes chamado de KIS (“mantenha-o simples” - Keep It Simple). Não “force” a linguagem tentando usos estranhos.

Você ouviu que C e C++ são linguagens portáveis e que os programas escritos em C e C++ pode ser executados em muitos computadores diferentes. Portabilidade é um objetivo difícil de se atingir. O documento padrão C ANSI contém uma longa lista de problemas relacionados com a portabilidade e foram escritos livros inteiros que discutem portabilidade.

Dica de portabilidade 1.3

Embora seja possível escrever programas portáveis, existem muitos problemas entre compiladores de C e C+ + diferentes e computadores diferentes, que podem tornar a portabilidade difícil de ser obtida. Simplesmente escrever programas em C e C+ + não garante portabilidade. O programador com frequência precisará lidar diretamente com variações de compilador e computador

Fizemos um cuidadoso walkthrough* do documento padrão ANSIIISO C++ e comparamos nossa apresentação com ele em relação à completude e precisão. Porém, C++ é uma linguagem rica, e existem algumas sutilezas na linguagem e alguns assuntos avançados que não cobrimos. Se você precisar de detalhes técnicos adicionais sobre C++, sugerimos que você leia o documento padrão ANSI/ISO C++. Você pode encomendar o documento padrão C+-i- no site da ANSI na Web

http://www.ansi.org/

O título do documento é “Information Technology - Programming Languages - C ++” e o número do documento é ISO/IEC 14882-1998. Se você preferir não comprar o documento, a versão antiga da minuta do padrão pode ser vista no site da World Wide Web

*N de R.T.: Termo usado para descrever uma das atividades de desenvolvimento de software, que consiste em reunir a equipe de desenvolvimento para analisar o código de um programa e discutir a implementação e as decisões tomadas pelos programadores; usada como ferramenta de
treinamento e como pane de um processo de qualidade no desenvolvimento de software.

http: //www. cygnus . com/mis c/wp/

Incluímos uma bibliografia extensa de livros e documentos sobre C++ e programação orientada a objetos. Também incluímos um apêndice de recursos para C++ que contém muitos sites da Internet relativos a C++ e à programação orientada a objetos.

Muitas características das versões atuais de C++ não são compatíveis com implementações mais antigas de
C++; assim, você pode constatar que alguns dos programas deste texto não funcionam com compiladores de C++ mais antigos.

Boa prática de programação 1.2

Leia os manuais para a versão de C+ + que você está usando. Consulte estes manuais com freqüência, para certificar-se de que esteja ciente da rica relação de recursos que C++ apresenta e deque está usando estes recursos corretamente.

Boa prática de programação 1.3

Seu computador e seu compilador são bons professores. Se, depois de ler cuidadosamente seu nanual da linguagem C++ , você não tiver certeza de como funciona um recurso de C++, experimente usar um pequeno “programa de teste” e veja o que acontece. Configure as opções do seu compilador para “nível máximo de advertências “. Estude cada mensagem que obtiver ao compilar seus programas e corrija os programas para eliminar as mensagens.

1.18 Introdução à programação de C++

A linguagem C++ facilita uma abordagem estruturada e disciplinada ao projeto de um programa de computador. Introduzimos agora a programação C++ e apresentamos vários exemplos que ilustram muitas características importantes de C++. Cada exemplo é analisado um comando por vez. No Capítulo 2, apresentamos um tratamento detalhado da programação estruturada em C++. Então, usamos a abordagem estruturada no Capítulo 5. Começando com o Capítulo 6, estudamos a programação orientada a objetos em C++. Novamente, por causa da importância central da programação orientada a objetos neste livro, cada um dos primeiros cinco capítulos conclui com uma seção intitulada “Pensando em objetos”. Estas seções especiais introduzem os conceitos da orientação a objetos e apresentam um estudo de caso que desafia o leitor a

projetar e implementar em C÷+ um programa orientado a objetos de porte significativo.

1.19 Um programa simples: imprimindo uma linha de texto

C++ usa notações que podem parecer estranhas a não-programadores. Começamos considerando um programa simples, que imprime uma linha de texto, O programa e sua saída na tela são mostrados na Fig. 1.2.

Este programa ilustra várias características importantes da linguagem C++. Discutiremos cada linha do programa em detalhe. As linhas 1 e 2

II Fig. 1.2: figOlO2.cpp

// Um primeiro programa emC++

começam cada uma com II, indicando que o restante de cada linha é um comentário. Os programadores inserem comentários para documentar programas e melhorar a legibilidade dos mesmos. Os comentários ajudam também outras pessoas a ler e entender seu programa. Os comentários não levam o computador a executar qualquer ação quando o programa for executado. Os comentários são ignorados pelo compilador C++ e não causam qualquer geração de código objeto em linguagem de máquina. O comentário Um primeiro programa em C++ simplesmente descreve o propósito do programa. Um comentário que começa com II é chamado de comentário de linha única porque o comentário termina no fim da linha em que está. [Nota: os programadores de C++ podem usar também o estilo de comentário de C, no qual um comentário - contendo possivelmente muitas linhas - começa com / e termina com */.1

Boa prática de programação 1.4

Todo programa deveria começar com um comentário descrevendo a finalidade do programa.

Fig. 1.2 Programa de impressão de texto.

A linha 3

#include <iostream>

é uma diretiva do pré-processador isto é, uma mensagem para o pré-processador C++. As linhas iníciando com são processadas pelo pré-processador antes de o programa ser compilado. Esta linha específica diz ao pré-processador para incluir no programa o conteúdo do arquivo de cabeçalho do stream de entrada/saída, <iostream>. Este arquivo deve ser incluído para qualquer programa que envia dados de saída para ateia ou recebe dados de entrada do teclado usando o estilo de entrada/saída com streams de C++. A Fig. 1.2 envia dados de saída para ateia, como logo veremos, O conteúdo de

iostream será explicado em mais detalhes mais tarde.
Erro com um de programação 1.2
Esquecer de incluir o arquivo ias tream em um programa que recebe dados de entrada do teclado ou envia dados de saída para a tela faz o compilador emitir uma mensagem de erro.

A linha 5

int main(

faz parte de todo programa em C++. Os parênteses depois de main indicam que main é um bloco de construção de programa chamado de função. Os programas em C++ contêm uma ou mais funções, exatamente uma das quais deve ser main. A Fig. 1.2 contém só uma função. Os programas em C++ começam a executar na função main, ainda que main não seja a primeira função no programa. A palavra-chave int, à esquerda de main. indica que main devolve um inteiro (número inteiro). Explicaremos o que quer dizer para uma função "devolver um valor" quando estudarmos funções a fundo, no Capítulo 3. Por ora, simplesmente inclua a palavra-chave int à esquerda de main em cada um de seus programas.
A chave à esquerda, {, linha 6, deve começar o corpo de qualquer função. Uma chave à direita, }, linha 10, correspondente deve terminar o corpo de cada função. A linha 7
std::cout « "Bem-vindo a C++!\n";
instrui o computador a imprimir na tela o string de caracteres contido entre as aspas. A linha inteira, inclusive std: : cout. o operador«, o string "Bem-vindo a c++! \n" e o ponto-e-vírgula é chamada de comando. Todo comando deve terminar com um ponto-e-vírgula (também conhecido como terminador de comando). A entrada/saída em C++ é realizada com streams de caracteres. Deste modo, quando o comando precedente for executado,

1	II Fig. 1.2: figol_02.cpp				
2	II Um primeiro programa em C++				
3	#include <iostream>				
4					
5	intmain ()				
6	{				
7	std::cout « "Bem-vindo a C++!\n";				
8					
9	return O; II indica que o programa		termin ou	co m	suces so
10	}				

B e	m-vindo a C++!				

isto enviará o stream de caracteres Bem-vindo a C++! para o objeto stream de saída padrão- s td: : cou t - que normalmente está “conectado” à tela. Discutimos as diversas características de std: : cout em detalhes no
Capítulo 11.
Note que colocamos std: : antes de cout. Isto é requerido quando usamos a diretiva do pré-processador #include <iostream>. A notação std: : cout especifica que estamos usando um nome, neste caso cout, que pertence ao “ambiente de nomes” std. Ambientes de nomes são um recurso avançado de C++. Discutimos ambientes de nomes em profundidade no Capítulo 21. Por enquanto, você deve simplesmente lembrar de incluir std: : antes de cada menção a cout, cm e cerr em um programa. Isto pode ser incômodo - na Fig. 1.14, introduzimos o comando using, que nos permitirá evitar ter que colocar std: : antes de cada uso de um nome do ambiente de nomes std.
O operador « é chamado de operador de inserção no stream. Quando este programa for executado, o valor à direita do operador, o operando à direita, é inserido no stream de saída. Os caracteres do operando à direita são normalmente impressos exatamente como eles aparecem entre as aspas duplas. Note, porém, que os caracteres \n não são exibidos na tela. A barra invertida é chamada de caractere de escape. Indica que um caractere “especial” deve ser enviado para a saída. Quando uma barra invertida for encontrada em um string de caracteres, o próximo caractere é combinado com a barra invertida para formar uma seqüência de escape. A seqüência de escape \n significa nova linha. Ela faz com que o cursor (isto é, o indicador da posição corrente na tela) mova-se para o início da próxima linha na tela. Algumas outras seqüências de escape comuns são listadas na Fig. 1.3.

Fig. 1.3 Algumas seqüências comuns de escape.
Erro comum de programação 1.3
Omitir o ponto-e-vírgula nofim de um comando é um erro de sintaxe. Um erro de sintaxe ocorre quando o compilador não pode reconhecer um comando. O compilador normalmente emite uma mensagem de erro para ajudar o programador a localizar e corrigir o comando incorreto. Os erros de sintaxe são violações da linguagem. Os erros de sintaxe são também chamados de erros de compilação, erros durante a compilação ou erros de compilação porque aparecem durante a fase de compilação do programa.
A linha 9
return O; II indica que o programa terminou com sucesso
é incluída no fim de toda função main. A palavra-chave return de C++ é um dos vários meios que usaremos para sair de uma função. Quando o comando return for usado no fim de main, como mostrado aqui, o valor O indica que o programa terminou com sucesso. No Capítulo 3, discutimos funções em detalhes, e as razões para incluir este comando se tornarão claras. Por ora, simplesmente

inclua este comando em cada programa, ou o compilador pode gerar uma mensagem de advertência em alguns sistemas.
A chave à direita, }, linha 10, indica o final de main.
Boa prática de programação 1.5
Muitos programadores fazem com que o último caractere impresso por uma função veja uma nova linha
(\n). Isto assegura que a função deixará o cursor da tela posicionado no início de uma nova linha.

Seqüência de escape	Descrição Nova linha. Posiciona o cursor da tela no início da próxima linha.
\n	
\t	Tabulação horizontal. Move o cursor da tela para a próxima posição de tabulação.
\r	Retorno do carro. Posiciona o cursor da tela no início da linha atual; não avança para a próxima linha.
\a	Alerta. Faz soar o alarme do sistema.
\\	Barra invertida. Usada para imprimir um caractere barra invertida.
\“	Aspas. Usada para imprimir um caractere aspas.

Convenções desta natureza encorajam a reusabilidade de software - uma meta-chave em ambientes de
desenvolvimento de software.
Boa prática de programação 1.6
Recue o corpo inteiro de cada função um nível de indentação nas marcas de tabulação que definem o corpo da função. Isto faz com que a estrutura funcional de um programa se destaque e ajuda a tornar os programas mais fáceis de ler.
Boa prática de programação 1.7
Estabeleça uma convenção para o tamanho dos recuos de indentação que você prefere; então, aplique uniformemente essa convenção. A tecla de tabulação pode ser usada para criar recuos, mas pontos de tabulação podem variar
Recomendamos usar espaços entre tabulações de 1/4 de polegada ou (preferível) três espaços para criar o recuo para um nível de indentação.
Bem-vindo a C++! pode ser exibido de vários modos. Por exemplo, a Fig. 1.4 usa múltiplos comandos de inserção no streum (linhas 7 e 8) e, ainda assim, produz saída idêntica à do programa da Fig. 1.2. Isto funciona porque cada comando de inserção no stream retoma a impressão no ponto em que o comando anterior havia parado a impressão. A primeira inserção no stream imprime Bem-vindo seguido por um espaço e a segunda inserção no stream começa a imprimir na mesma linha, imediatamente após o espaço deixado pela inserção anterior. Em geral, C++ permite ao programador expressar comandos de vários modos.

Um único comando pode imprimir múltiplas linhas usando caracteres nova linha, como na Fig. 1.5. Cada vez que uma seqüência de escape \n (nova linha) é encontrada no stream de saída, o cursor de tela é posicionado no início da próxima linha. Para conseguir uma linha em branco em sua saída, simplesmente coloque dois caracteres nova linha um atrás do outro, como na Fig. 1.5.

Fig. 1.5 Impressão em múltiplas linhas com um único comando usando cout (parte 1 de 2).

```
1 II Fig. 1.4: figOlO4.cpp
2 II Imprimindo uma linha usando múltiplos comandos
3 #include <iostream>
4
5 int main(
6 {
7 std: :cout « “Bem-vindo “;
8 std::cout « “a C++’\n”;
9
10 return 0; /1 indica que o programa terminou com sucesso
11
Bem-vindo a C++!
```

Fig. 1.4 Impressão em uma linha com comandos separados usando cout.

```
1 II Fig. 1.5: fig0lo5.cpp
2 1/ Imprimindo múltiplas linhas com um único comando
3 #include <iostream>
4
5 int main(
6 {
7 std: : cout « “Bem-vindo\na\n\nC++! \n”;
8
9 return 0; II indica que o programa terminou com sucesso
IO )
```



```
Bem-vindo
a
```

Fig. 1.5 Impressão em linhas múltiplas com um único comando usando cout (parte 2 de 2).

1.20 Outro programa simples: somando dois inteiros

Nosso próximo programa usa o objeto std: : cm para entrada em stream e o operador de extração stream, », para obter dois inteiros digitados no teclado

pelo usuário, computa a soma destes valores e exibe, ou imprime, o resultado usando std: : cout. O programa e uma exibição de amostra são mostrados na Fig. 1.6.

Os comentários nas linhas 1 e 2

II Fig. 1.6: figOlO6.cpp
II Programa de adição

descrevem o nome do arquivo e a finalidade do programa. A diretiva para o pré-processador C++

#include <iostream>

na linha 3 inclui o conteúdo do arquivo de cabeçalho iostream no programa.

```
1 II Fig. 1.6: figOlO6.cpp
2 // Programa de adição
3 #include <iostream>
4
II declaração
II prompt
// lê um inteiro
II prompt
// lê um inteiro
II atribuição da soma
// imprime a soma
Digite o primeiro inteiro
45
Digite o segundo inteiro
72
A soma é 117
```

Fig. 1.6 Um programa de adição.

Como já dissemos antes, todo programa começa sua execução com a função main. A chave à esquerda marca o início do corpo de main e a chave à direita correspondente marca o fim de main. A linha 7

```
5 intmain ()
6
7 int integeri, integer2, sum;
8
9 std::cout « “Digite o primeiro inteiro\n”;
10 std::cin » integerl;
11 std::cout « “Digite o segundo inteiro\n”;
12 std::cin » integer2;
13 sum = integeri + integer2;
14 std: :cout « “A soma é “ « sum « std: :endl;
15
16 return 0; // indica que o programa terminou com êxito
17
```

int integeri, integer2, sum; II declaração

é uma declaração. As palavras integeri. integer2 e sum são nomes de variáveis. Uma variável é uma posição na memória do computador onde um valor pode ser armazenado para ser usado por um programa. Esta declaração especifica que as variáveis integeri, integer2 e sum são dados do tipo int. o que significa que estas variáveis guardarão valores inteiros, isto é, números inteiros tais como 7, -11, 0, 31914. Todas as variáveis devem ser declaradas com um nome e um tipo de dados antes de poderem ser usadas em um programa. Diversas variáveis do mesmo tipo podem ser declaradas em uma declaração ou em declarações múltiplas. Podíamos ter escrito três declarações, uma para cada variável, mas a declaração precedente é mais concisa.

Boa prática de programação L8

Alguns programadores preferem declarar cada variável em uma linha separada. Este formato permite a
fácil inserção de um comentário descritivo após cada declaração.

Logo discutiremos os tipos de dados doubie (para especificar números reais, isto é, números com pontos decimais como 3.4, 0.0, - 11.19) e char (para especificar dados contendo caracteres; uma variável char pode guardar só uma única letra minúscula, uma única letra maiúscula, um único dígito ou um único caractere especial, tais como x. $, 7, , etc.).

Boa prática de programação 1.9

Coloque um espaço após cada vírgula (,) para tornar os programas mais legíveis.

Um nome de variável é qualquer identificador válido. Um identificador é uma série de caracteres que consiste em letras, dígitos e o caractere sublinhado (_), que não começa com um dígito. C++ é sensível a maiúsculas e minúsculas, isto é, letras maiúsculas e minúsculas para ela são diferentes; assim, ai e Ai são identificadores diferentes.

Dica de portabilidade 1.4

C++ permite identificadores de qualquer comprimento, mas o sistema e/ou implementação de C++ podem impor algumas restrições sobre o comprimento de identificadores. Use identificadores de 3] caracteres, ou menos, para assegurar a portabilidade dos seus programas.

Boa prática de programação 1. 10

Escolher nomes de variáveis significativos ajuda um programa a ser “autodocumentado “, isto é, torna mais fácil de entender o programa simplesmente lendo-o, em vez de ter que ler manuais ou usar comentários em excesso.

Boa prática de programação 1.11

Evite identificadores que começam com sublinhado (_) simples ou duplo, porque compiladores de C+ + podem usar nomes semelhantes para seu próprio uso interno. Isto evitará que os nomes escolhidos por você sejam confundidos com nomes que os compiladores escolhem.

As declarações de variáveis podem ser colocadas em quase qualquer lugar dentro de uma função. Porém, a declaração de uma variável deve aparecer

antes de a mesma ser usada no programa. Por exemplo, no programa da Fig. 1.6, em vez de usar uma declaração única para todas as três variáveis, poderiam ter sido usadas três declarações separadas. A declaração
int integeri;
poderia ter sido colocada imediatamente antes da linha
std: :cin » integeri;
a declaração
int integer2;

poderia ter sido colocada imediatamente antes da linha
std::cin » integer2;
e a declaração
int sum;
poderia ter sido colocada imediatamente antes da linha
suni = integeri + integer2;
Boa prática de programação 1.12
Sempre coloque uma linha em branco entre uma declaração e comandos executáveis adjacentes. Isto faz
com que as declarações se destaquem no programa, contribuindo para a clareza do mesmo.
Boa prática de programação 1.13
Se você preferir colocar declarações no início de uma função, separe essas declarações dos comandos executáveis da função com uma linha em branco, para destacar onde as declarações terminam e os comandos executáveis começam.
A linha 9
std::cout « “Digite o primeiro inteiro\n”; 1/ prompt
imprime o string Digite o primeiro inteiro (também conhecido como literal de string ou um literal) na tela e posiciona o cursor no início da próxima linha. Esta mensagem é um prompt porque ela pede ao usuário para tomar uma ação específica. Gostamos de ler o comando precedente como “std: : cout obtém o string de caracteres “Digite o primeiro inteiro\n”.
A linha 10
std::cin » integeri; II lê uni inteiro
usa o objeto cm de entrada em stream (do ambiente de nomes std) e o operador de extração de srream, ». para obter um valor do teclado. Usar o operador de extração de stream com std: : cm retira caracteres de entrada do stream padrão de entrada, que normalmente é o teclado. Gostamos de ler o comando precedente como “s td: cm fornece um valor para integeri” ou, simplesmente, “std: cm fornece integeri”.
Quando o computador executa o comando precedente, ele espera o usuário digitar um valor para a variável integeri. O usuário responde digitando um inteiro (como caracteres) e então apertando a tecla Enter (às vezes chamada de tecla Return) para enviar o número ao computador. O computador então atribui este número (ou valor), à variável integeri. Quaisquer referências subseqüentes a

integeri tio programa usarão este mesmo valor.
Os objetos strearn std: cout e std: : cm facilitam a interação entre o usuário e o computador. Como
esta interação se assemelha a um diálogo, ela é freqüentemente chamada de computação conversacional ou computação interativa.
Alinha 11
std::cout « “Digite o segundo inteiro\n”; II prompt
exibe as palavras Digi te o segundo inteiro na tela e, então, posiciona o cursor no início da próxima linha. Este comando solicita ao usuário que execute uma ação. A linha 12
std::cin » integer2; /1 lê um inteiro

obtém um valor fornecido pelo usuário para a variável integer2.
O comando de atribuição na linha 13
sum = integeri + integer2;
calcula a soma das variáveis integeri e integer2 e atribui o resultado à variável sum usando o operador de atribuição =. O comando é lido como, “suin recebe o valor de integeri + integer2.” A maioria dos cálculos são executados em comandos de atribuição. O operador = e o operador + são chamados de operadores binários porque cada um tem dois operandos. No caso do operador +, os dois operandos são integeri e integer2. No caso do operador anterior, os dois operandos são sum e o valor da expressão integeri + integer2.
Boa prática de programação 1.14
Coloque espaços dos dois lados de um operador binário. Isto faz com que o operador se destaque, tornando o programa mais legível.
A linha 14
std: :cout « “A soma é “ « sum « std: :endl; II imprime a soma
exibe o string de caracteres A soma é seguido pelo valor numérico da variável sum seguido por std: : endi (endi é uma abreviação para “fim de linha;” endi também é um nome no ambiente de nomes std) - um assim chamado manipulador de stream. O manipulador std: : endi dá saída a um caractere de nova linha e então “descarrega o buifer de saída”. Isto simplesmente significa que em alguns sistemas, nos quais os dados de saída se acumulam na máquina até que existam suficientes para que “valha a pena exibir”, std: endi força a exibição de quaisquer dados de saída acumulados até aquele instante.
Note que o comando precedente dá saída a diversos valores de tipos diferentes, O operador de inserção no stream “sabe” como dar saída a cada pedaço dos dados. Usar múltiplos operadores de inserção no stream («)em um único comando é chamado de concatenar, encadear ou cascatear as operações de inserção no stream. Deste modo, é desnecessário se ter múltiplos comandos de saída para dar saída a múltiplos pedaços de dados.
Os cálculos podem também ser executados em comandos de saída.
Poderíamos ter combinado os dois comandos das linhas 13 e 14cm um só

comando
std::cout « ‘A soma é ‘ « integeri + integer2 « std::endl;
eliminando assim a necessidade da variável sum.
A chave à direita, }, informa ao computador que se chegou ao fim da função main.
Uma característica poderosa de C++ é que os usuários podem criar seus próprios tipos de dados (exploraremos este recurso no Capítulo 6). Assim, os usuários podem então “ensinar” a C++ como receber entradas e fornecer saídas com valores destes novos tipos de dados usando os operadores » e « (isto é chamado de sobrecarga de um operador - um tópico que exploraremos no Capítulo 8).

1.21 Conceitos de memória

Nomes de variáveis, tais como integeri, integer2 e sum, correspondem, na realidade, aposições na memoria do computador. Toda variável tem um nome, um tipo, um tamanho e um valor.
No programa de adição da Fig. 1.6, quando o comando
std::cin » integeri;
é executado, os caracteres digitados pelo usuário são convertidos para um inteiro que é colocado em uma posição de memória à qual o nome integeri foi associado pelo compilador de C++. Suponha que o usuário forneça o número 45 como o valor para integeri. O computador colocará 45 na posição integeri, como mostrado na Fig. 1.7.

II atribuição a soma

integerl 45
Fig. 1.7 Posição de memória mostrando o nome e valor de uma variável.
Sempre que um valor é colocado em uma posição de memória, o valor substitui o valor anterior que havia
nessa localização. O valor anterior é perdido.
Retornando ao nosso programa de adição, quando o comando
std::cin » integer2;
é executado, suponha que o usuário digite o valor 72. Este valor é colocado na posição integer2 e a memória fica como mostrado na Fig. 1.8. Note que estas posições não são, necessariamente, posições adjacentes na memória.

Fig. 1.8 Posições de memória depois de os valores para as duas variáveis terem sido fornecidos como entrada.
Uma vez que o programa tenha obtido os valores para integeri e integer2. ele soma estes valores e
coloca a soma na variável sum. O comando
sum = integeri ÷ integer2;
que executa a adição também substitui qualquer valor que esteja armazenado em suxn. Isto acontece quando a soma calculada de integeri e integer2 é colocada na posição sum (sem levar em conta que já pode haver um valor em

sum; esse valor é perdido). Depois que suin é calculada, a memória fica como mostrado na Fig. 1.9. Note que os valores de integeri e integer2 aparecem exatamente como estavam antes de serem usados no cálculo de sum. Estes valores foram usados, mas não destruídos, quando o computador executou o cálculo. Deste modo, quando um valor é lido de uma posição de memória, o processo é não-destrutivo.

integeri 45
integer2 72
suni 117

Fig. 1.9 Posições de memória após um cálculo.

1.22 Aritmética

A maioria dos programas executa cálculos aritméticos. Os operadores aritméticos estão resumidos na Fig. 1.10. Note o uso de vários símbolos especiais não usados em álgebra. O asterisco (*) indica multiplicação e o sinal de percentagem (%) é o operador módulo que será discutido logo mais à frente. Os operadores aritméticos da Fig. 1.10

integer i	f	45	
integer 2		72	

são todos operadores binários, isto é, operadores que recebem dois operandos. Por exemplo, a expressão integerl + integer2 contém o operador binário + e o dois operandos integerl e integer2.

Fig. 1.10 Operadores aritméticos.

A divisão inteira (isto é, tanto o numerador como o denominador são inteiros) dá um resultado inteiro; por exemplo,

a expressão 7 / 4 fornece o valor 1, e a expressão 17 / 5 fornece o valor 3. Note que qualquer parte fracionária

na divisão inteira é simplesmente descartada (isto é, truncada) - não ocorre nenhum arredondamento.

C++ oferece o operador módulo, %, que dá o resto da divisão inteira. O operador módulo pode ser usado somente com operandos inteiros. A expressão x % y fornece o resto depois de x ser dividido por y. Assim, 7 % 4 dá 3. e 17 % 5 dá 2. Nos capítulos mais à frente, discutiremos muitas aplicações interessantes do operador módulo, tal como determinar se um número é múltiplo de um outro (um caso especial disto é determinar se um número é par ou ímpar).

Erro comum de programação 1.4

Tentar usar o operador módulo, %, com operandos não-inteiros é um erro de sintaxe.

As expressões aritméticas em C++ devem ser fornecidas para o computador no

formato em linha. Deste modo, expressões tais como “a dividido por b” devem ser escritas como a / b de forma que todas as constantes, variáveis e operadores apareçam em linha. A notação algébrica

a
b

geralmente não é aceitável para os compiladores, embora existam alguns pacotes de software para fins especiais que suportam a notação mais natural para expressões matemáticas complexas.

Os parênteses são usados em expressões de C++ quase do mesmo modo que nas expressões algébricas. Por
exemplo, para multiplicar a pela quantidade b + c escrevemos:

a * (b + c)

C++ aplica os operadores em expressões aritméticas em uma seqüência precisa, determinada pelas seguintes regras de precedência de operadores, que em geral são as mesmas que aquelas seguidas na álgebra:

1. Operadores em expressões contidas dentro de pares de parênteses são calculados primeiro. Assim, os parênteses podem ser usados para forçar que a ordem de cálculo aconteça em qualquer seqüência desejada pelo programador Dizemos que os parênteses estão no “nível mais alto de precedência”. Em casos de parênteses aninhados, ou embutidos, os operadores no par mais interno de parênteses são aplicados primeiro.
2. As operações de multiplicação, divisão e módulo são aplicadas em seguida. Se uma expressão contém várias operações de multiplicação, divisão e módulo, os operadores são aplicados da esquerda para a
direita. Dizemos que a multiplicação, a divisão e o módulo estão no mesmo nível de precedência.

Operação em C++	Operador aritmético	Expressão algébrica	Expressão C++	
Adição	+	f+7	f +	7
Subtração	-	p - c	p -	c
Multiplicação	*	bm	b *	m
Divisão	/	x/yorXorx÷y y	x /	y
Módulo	%	rmods	r %	s

3. As operações de adição e de subtração são aplicadas por último. Se uma expressão contém várias Operações de adição e subtração, os operadores são aplicados da esquerda para a direita. A adição e subtração têm, também, o mesmo nível de precedência.

As regras de precedência de operadores permitem a C++ aplicar os operadores na ordem correta. Quando dizemos que certos operadores são aplicados da esquerda para a direita, estamos nos referindo à associatividade dos operadores. Por exemplo, na expressão

a+b+c

os operadores de adição (+) se associam da esquerda para a direita. Veremos que alguns operadores se associam da direita para a esquerda. A Fig. 1.11 resume estas regras de precedência de operadores. Esta tabela será expandida à medida que operadores adicionais de C++ sejam introduzidos. Um quadro de precedência completo se encontra nos apêndices.

Fig. 1.11 Precedência dos operadores aritméticos.

Vamos agora considerar várias expressões levando em conta as regras de precedência de operadores. Cada exemplo lista uma expressão algébrica e seu equivalente em C++.

O exemplo a seguir é uma média aritmética de cinco termos:

a+b+c+d+e

Algebra: m =

C+-t-:m= (a+b+c+d+e) /5;

Os parênteses são necessários porque a divisão tem precedência mais alta que a adição. O valor inteiro (a + b ÷

c + d + e) é dividido por 5. Se os parênteses são erroneamente omitidos, obtemos a + b + c + d + e /

5, que é calculado incorretamente como

a+b+c+d+

5

O seguinte é um exemplo da equação de uma linha reta:

Álgebra: y = mx + b

C++:y=m * x + b;

Nenhum parêntese é necessário. A multiplicação é aplicada primeiro porque a multiplicação tem uma precedência mais alta que a adição.

Operador(es)		Operação(ões)	Ordem de avaliação (precedência)
)		Parênteses	Calculados primeiro. Se os parênteses estão aninhados, primeiro é calculado o par mais interno na expressão. Se houver vários pares de parênteses “no mesmo nível” (isto é, não aninhados), eles são calculados da esquerda para a direita.
, / ou	%	Multiplicação Divisão Módulo	Calculados em segundo lugar. Se houver vários, eles são calculados da esquerda para a direita.
+ ou -		Adição Subtração	Calculados por último. Se houver vários, eles são calculados da esquerda para a direita.

O exemplo seguinte contém as operações módulo (%), multiplicação, divisão, adição e subtração:

Álgebra: z =pr%q + w/x-y

C-t-+: z = p * r % q + w / x -
Os números circulados, abaixo do comando, indicam a ordem em que C++ aplica os operadores. Os de multiplicação, módulo e divisão são aplicados primeiro, na ordem da esquerda para a direita, (isto é, eles se associam da esquerda para a direita), uma vez que eles têm precedência mais alta que a adição e a subtração. Os de adição e subtração são aplicados em seguida. Estes também são aplicados da esquerda para a direita.
Nem todas as expressões com vários pares de parênteses contêm parênteses aninhados. Por exemplo, a expressão
a * (b + c) + c * (d + e)
não contém parênteses aninhados. Neste caso, dizemos que os parênteses estão “no mesmo nível”.
Para desenvolver uma melhor compreensão das regras de precedência de operadores, considere como um
polinômio de segundo grau é calculado:
y = a * x * x + b * x + e;
Os números circulados, abaixo do comando, indicam a ordem em que C++ aplica os operadores. Não há um operador aritmético para exponenciação em C ++; por isso, representamos x2 como x * x. Logo discutiremos a função pow (‘power”) da biblioteca padrão; essa função executa a exponenciação. Por causa de alguns problemas sutis relacionados aos tipos de dados exigidos por pow, adiamos uma explicação detalhada de pow até o Capítulo 3.
Suponha que as variáveis a, b, e e x estão inicializadas como segue: a = 2, b = 3, e = 7 ex = 5.
A Figura 1.12 ilustra a ordem em que os operadores são aplicados ao polinômio de segundo grau precedente.
Podemos acrescentar parênteses desnecessários ao comando de atribuição precedente, para clareza, como:
y = (a * x * x) + (b * x) + c;
Boa prática de programação 1.15
Como na álgebra, é aceitável se colocar parênteses desnecessários em uma expressão para torná-la mais clara. Estes parênteses são chamados de redundantes. Parênteses redundantes são usados comumente para agrupar subexpressões de uma expressão grande, para tornar a expressão mais clara. Quebrar um comando longo em uma seqüência de comandos mais curtos e mais simples também aumenta a clareza.
1.23 Tomada de decisões: os operadores relacionais e de igualdade
Esta seção introduz uma versão simples da estrutura if, a qual permite a um programa tomar decisões com base na veracidade ou falsidade de alguma condição. Se a condição é satisfeita, ou seja, se ela for true (verdadeira), o comando no corpo da estrutura if é executado. Se a condição não for satisfeita, ou seja, se for false (falsa), o comando do corpo não é executado. Logo adiante, veremos um exemplo.
As condições em estruturas if podem ser definidas usando-se os operadores de igualdade e os operadores relacionais, resumidos na Fig. 1 .13. Os operadores relacionais têm todos o mesmo nível de precedência e são associados da

esquerda para a direita. Os dois operadores de igualdade têm o mesmo nível de precedência, mais baixo que o nível de precedência dos relacionais. Os operadores de igualdade também são associados da esquerda para a direita.

78 C++ COMO PROGRAMAR

Passo]. y= 2 * 5 * 5 + 3 * 5 + 7

2 * 5
Passo2.y=10*5+3*5+ 10 * 5
Passo3. y=50÷3 * 5
3*
Passo 4. y = 50 + 15 + 7; 50 + 15 é
Passo5. y =65 +7;
65 + 7 é E1
Passo6. y = 72;

7;

(multiplicação mais à esquerda)
(multiplicação mais à esquerda)
(multiplicação antes da adição)

(adição mais à esquerda)

(última adição)

(última operação - colocar 72 em y)

Fig. 1.12 Ordem em que um polinômio de segundo grau é calculado.
Erro comum de programação 1.5
Ocorrerá um erro de sintaxe se qualquer um dos operadores ==, =, >= e <= aparece com espaços entre seus dois símbolos.
Erro comum de programação 1.6

Inverter a ordem do par de símbolos em qualquer dos operadores ! =, >= e <= (escrevendo-os como =!, e <, respectivamente) normalmente é um erro de sintaxe. Em alguns casos, escrever ! = como ! não é um erro de sintaxe, mas quase certamente será um erro de lógica.

Fig. 1.13 Operadores relacionais e de igualdade.

Operadores de igualdade	Operador C++ de	Exemplo de	Significado da	

algébricos padrão e operadores	igualdade ou relacional	condição em C++	condição em C++	
relacionais				
Operadores de igualdade				
>	>	x > y	x é maior que y	
<	<	x < y	x émenorque y	
	>=	x >= y	x é maior que ou igual a y	
	<=	x <= y	x é menor que ou igual a y	
Operadores relaciona is				
=	==	x == y	x é igual a y	
	!=	x != y	x nãoéigualay	

CAPÍTULO 1 - INTRODUÇÃO AOS COMPUTADORES E À PROGRAMAÇÃO C ++ 79

Erro comum de programação 1.7

Confundir o operador de igualdade == com o operador de atribuição . O operador de igualdade deveria ser lido como “é iguala” e o operador de atribuição como “recebe” ou “recebe o valor de” ou, ainda, “a ele é atribuído o valor de “. Algumas pessoas preferem ler o operador de igualdade como “duplo igual Como logo veremos, confundir estes dois operadores pode não necessariamente produzir um erro de sintaxe facilmente identificável, mas pode causar erros de lógica extremamente sutis.

O exemplo seguinte usa seis comandos if para comparar dois números fornecidos pelo usuário. Se a condição de um destes ifs é satisfeita, então o comando de saída (resposta) associado aquele if é executado, O programa e os diálogos de entrada/saída de três exemplos de execução são mostrados na Fig. 1.14.

```
1 II Fig. 1.14: figOll4.cpp
2 II Utilizando comandos if, operadores
3 II relacionais e operadores de igualdade
4 # include <iostream>
5
6 using std: :cout; II o programa usa cout
7 using std: :cin; II o programa usa cm
8 using std: :endl; II o programa usa endl
9
10 int main(
11
12 int nurnl, num2;
13
14 cout « “Digite dois inteiros e lhe direi\n”
15 « “quais os relacionamentos que eles satisfazem: “;
16 cm » numi » num2; 1/ lê dois inteiros
```

```
17
18 if (numl==num2)
19 cout « numi « “ é igual a “ « num2 « endl;
20
21 if ( numi != num2
22 cout « numi « “ não é igual a “ « num2 « endl;
23
24 if (numl<num2
25 cout « numi « “ é menor que “ « num2 « endl;
26
27 if ( numl >num2
28 cout « numi « “ é maior que “ « num2 « endi;
29
30 if (numl<=num2)
31 cout « numi « “ é menor que ou igual a
32 « num2 « endl;
33
34 if (numl>=num2)
35 cout « numi « “ é maior que ou igual a
36 « num2 « endi;
37
38 return 0; II indica que o programa terminou com sucesso
39
```

Fig. 1.14 Utilizando operadores relacionais e de igualdade (parte 1 de 2).

Digite dois inteiros e lhe direi		
quais os relacionamentos que eles satisfazem: 3 7		
3 não é igual a 7		
3 é menor que 7		
é menor que ou igual a 7		


```
Digite dois inteiros e lhe direi
quais os relacionamentos que eles satisfazem: 22 12
22 não é igual a 12
22 é maior que 12
22 é maior que ou igual a 12
Digite dois inteiros e lhe direi
quais os relacionamentos que eles satisfazem: 7 7
é igual a 7
é menor que ou igual a 7
é maior que ou igual a 7
```

Fig. 1.14 Utilizando operadores de relacionais e de igualdade (parte 2 de 2).

As linhas 6 a 8
using std: :cout; // o programa usa cout
using std::cin; // o programa usa cm
using std::endl; // o programa usa endi
são comandos using que nos ajudam a eliminar a necessidade de repelir o prefixo std: : . A partir do ponto em que incluímos estes comandos using, podemos escrever cout em vez de std: : cout, cm em vez de std: : cm e endl em vez de std: : endi, respectivamente, no resto do programa. [Nota: deste ponto em diante, no livro, cada exemplo contém um ou mais comandos usingi.
Alinha 12
int numi, num2;
declara as variáveis usadas no programa. Lembre-se de que variáveis podem ser declaradas em uma declaração ou em várias declarações. Se mais de um nome é declarado em uma declaração (como neste exemplo), os nomes são separados por vírgulas (,). Isto se chama uma lista separada por vírgulas.
O programa usa operações de extração do stream colocadas em cascata (linha 16) para ler dois inteiros. Lembre-se de que podemos escrever cm (em vez de std: : cm) por causa da linha 7. Primeiro, um valor é lido para a variável numl e depois um valor é lido para a variável num2.
A estrutura if nas linhas 18 e 19
if (numi num2
cout « numl « ‘ é igual a « num2 « endl;
compara os valores das variáveis numl e num2 para testar a igualdade. Se os valores são iguais, o comando na linha 19 exibe uma linha de texto indicando que os números são iguais. Se as condições são verdadeiras em uma ou mais das estruturas if que iniciam nas linhas 21, 24, 27, 30 e 34, o comando cout correspondente exibe uma linha de texto.
Note que cada estrutura if na Fig. 1.14 tem um único comando em seu corpo e que cada corpo está indentado. Indentar o corpo de uma estrutura if melhora a legibilidade do programa. No Capítulo 2, mostraremos como especificar estruturas if cujo corpo é composto de múltiplos comandos (colocando os comandos que integram o corpo entre um par de chaves, { }).
Boa prática de programação 1.16
Indente o comando que compõe o corpo de uma estrutura if para fazer com que o corpo da estrutura se
destaque, melhorando desta forma a legibilidade.

Boa prática de programação 1.17
Em um programa não deve haver mais que um comando por linha.
Erro comum de programação 1.8
Colocar um ponto-e-vírgula imediatamente após o parêntese da direita, em uma estrutura if, é freqüentemente um erro de lógica (embora não seja um erro de sintaxe). O ponto-e-vírgula pode fazer com o corpo da estrutura if seja considerado vazio, de maneira que a estrutura if não execute nenhuma ação,

independentemente do fato da condição ser verdadeira ou não. Pior ainda, o comando do corpo original da estrutura if se tornaria agora um comando em seguida à estrutura if e seria sempre executado, freqüentemente levando o programa a produzir resultados incorretos.
Note o uso dos espaços na Fig. 1.14. Nos comandos em C++, caracteres impressos como espaços em branco, tais como tabulação, nova linha e espaços, são ignorados pelo compilador. Assim, os comandos podem ser separados em várias linhas e espaçados de acordo com a preferência do programador. E incorreto separar identificadores, strings (tais como “heilo”) e constantes (tais como o número 1000) ao longo de várias linhas.
Erro comum de programação 1.9
É um erro de sintaxe colocar espaços no meio de um identificador (por exemplo, main escrito como ma
in).
Boa prática de programação 1.18
Um comando longo pode ser separado por várias linhas. Se um comando único deve ser dividido em várias linhas, escolha pontos de quebra que façam sentido, tal como depois de uma vfrgula em uma lista separada por vfrgulas, ou depois de um operador em uma expressão longa. Se um comando é dividido em duas ou mais linhas, recue todas as linhas subseqüentes.
A Fig. 1.1.5 mostra a precedência dos operadores introduzidos neste capítulo. Os operadores são mostrados de cima para baixo, obedecendo à ordem decrescente de precedência. Note que todos estes operadores, com exceção do operador de atribuição, =, se associam da esquerda para a direita. A adição é associativa à esquerda, de modo que uma expressão como x + y + z é calculada como se tivesse sido escrita como (x + y) + z. O operador de atribuição = se associa da direita para a esquerda, de modo que uma expressão como x = y = o é resolvida como se tivesse sido escrita como x = (y = 0), o que, como logo vereremos, primeiro atribui O a y e então atribui o resultado dessa atribuição - O - a x.

Fig. 1.15 Precedência e associatividade dos operadores discutidos até agora.
Boa prática de programação 1.19
Consulte o quadro de precedência dos operadores quando escrever expressões contendo muitos operadores. Confirme que os operadores na expressão são executados na ordem que você espera. Se você não tem certeza da ordem de avaliação de uma expressão complexa, separe a expressão em comandos menores ou

j Operador(es)				Associatividade	Tipo
)				esquerda para a direita	parênteses
*	/	%		esquerda para a	multiplicativos

				direita	
+	-			esquerda para a direita	aditivos
«	»			esquerda para a direita	inserção em / extração de stream
<	<	>	> =	esquerda para a direita	relacional
==	‘=			esquerda para a direita	igualdade
=				direita para a esquerda	atribuição

use parênteses para frrçar a ordem, exatamente como você faria em uma expressão algébrica. Não deixe
de observar que alguns operado res, tal como atribuição (), são associados da direita para a esquerda, jel
em vez de da esquerda para a direita. de
Introduzimos muitas características importantes de C++, incluindo exibir dados na tela, obter dados do tecla d como entrada, executar cálculos e tomar decisões. No Capítulo 2, construímos sobre estas técnicas, na medida em di que introduzimos a programação estruturada. Você se tornará mais familiar com técnicas de indentação. Estudare- cl
mos como especificar e variar a ordem em que os comandos são executados - esta ordem é chamada defluxo de h
cont role. e
ri
1.24 Pensando em objetos: introdução à tecnologia de objetos e à 1 Unified Modeling LanguageTM
e
Agora começamos nossa introdução antecipada à orientação a objetos. Veremos que orientação a objetos é um modo n
natural de pensar sobre o mundo e de escrever programas de computador. Em cada um dos cinco primeiros capítulos concentramo-nos na metodologia “convencional” da programação f
estruturada, porque os objetos que vamos construir serão compostos, em parte, por pedaços de programas estruturados, si
Então, terminamos cada capítulo com uma seção “Pensando em objetos”, na qual apresentamos uma introdução d cuidadosamente cadenciada da orientação a objetos. Nosso objetivo, nestas seções “Pensando em objetos”, é ajudá- ii lo a desenvolver uma maneira de pensar orientada a objetos, de modo que você possa imediatamente pôr em uso os
conhecimentos de programação orientada a objetos que você começa a receber no Capítulo 6. Também vamos p apresentá-lo à Unified Modeling Language (UML). A UML é uma linguagem gráfica que permite às pessoas que

constroem sistemas (isto é, projetistas de software, engenheiros de sistemas, programadores, etc.) representar seus projetos orientados a objetos usando uma notação comum.
Nesta seção obrigatória (1.24), apresentamos conceitos básicos (isto é, “pensar em objetos) e terminologia (isto é, “falar em objetos”). Nas seções “Pensando em objetos” opcionais nos finais dos Capítulos 2 a 5, consideramos aspectos mais substanciais, na medida em que atacamos um problema desafiador com as técnicas de projeto orientado a objetos (OOD, object oriented design). Analisaremos uma definição de problema típica, que requer a construção de um sistema, determinaremos os objetos necessários para implementar o sistema, determinaremos os atributos que os objetos precisarão ter, determinaremos os comportamentos que estes objetos deverão exibir e especificaremos como os objetos necessitarão interagir uns com os outros para atender aos requisitos do sistema. Faremos tudo isso antes mesmo de termos aprendido como escrever programas orientados a objetos em C++. Nas seções “Pensando em objetos” opcionais nos finais dos Capítulos 6, 7 e 9, discutimos a implementação em C++ do sistema orientado a objetos que vamos projetar nos capítulos anteriores. Esteestudo de caso vai ajudar a prepará-lo para os tipos de projetos substanciais encontrados na indústria. Se você é um estudante e seu professor não planeja incluir este estudo de caso em seu curso, considere a possibilidade de cobrir este estudo de caso por conta própria. Acreditamos que lhe valerá a pena empregar tempo para percorrer este projeto grande e desafiador. Você experimentará uma introdução sólida ao projeto orientado a objetos com a UML e irá aguçar sua habilidade de leitura de código passeando por um programa em C++ com mais de 1000 linhas, bem documentado, que resolve o problema apresentado no estudo de caso.
Começamos nossa introdução à orientação a objetos com um pouco da terminologia-chave da orientação a objetos. Olhe em sua volta no mundo real. Para onde quer que você olhe, você os vê - objetos! Pessoas, animais, plantas, carros, aviões, construções, computadores, etc. Seres humanos pensam em termos de objetos. Temos a maravilhosa habilidade da abstração que nos permite visualizar imagens em uma tela como objetos, tais como pessoas, aviões, árvores e montanhas, em vez de ver pontos coloridos isolados. Podemos, se desejarmos, pensar em termos de praias em vez de grãos de areia, florestas em vez de árvores e casas em vez de tijolos.
Poderíamos estar propensos a dividir os objetos em duas categorias - objetos animados e objetos inanimados. Objetos animados, em certo sentido, são “vivos”. Eles se movem ao nosso redor e fazem coisas. Objetos inanimados, como toalhas, parecem não fazer mesmo muita coisa. Eles somente parecem “ficar ao redor”. Todos estes
objetos, porém, têm algumas coisas em comum. Todos eles têm atributos, como tamanho, forma, cor, peso, etc. E todos eles exibem comportamentos (por exemplo, uma bola rola, salta, incha e esvazia; um bebê chora, dorme, engatinha, passeia e pisca; um carro acelera, freia e muda de direção; uma toalha absorve água; etc.).

CAPÍTULO 1 - INTRODUÇÃO AOS COMPUTADORES E À PROGRAMAÇÃO C ++ 3

Os seres humanos aprendem sobre objetos estudando seus atributos e observando seus comportamentos. Objetos diferentes podem ter atributos semelhantes e podem exibir comportamentos semelhantes. Comparações podem ser feitas, por exemplo, entre bebês e adultos e entre seres humanos e chimpanzés. Carros, caminhões, pequenas camionetes vermelhas e patins têm muito em comum.

A programação orientada a objetos (OOP object-orientedprogramming) modela objetos do mundo real com duplicatas em software. Ela se aproveita das relações de classe, nas quais objetos de uma certa classe - tal como uma classe de veículos - têm as mesmas características. Ela tira proveito de relações de herança e até de relações de herança múltipla, nas quais classes de objetos recém-criadas são derivados absorvendo características de classes existentes e adicionando características próprias suas. Um objeto da classe “conversível” certamente tem as características da classe mais genérica “automóvel”, mas a capota de um conversível sobe e desce.

A programação orientada a objetos nos dá uma maneira mais natural e intuitiva para visualizar o processo de programação, a saber, modelando objetos do mundo real, seus atributos e seus comportamentos. A OOP também modela a comunicação entre objetos. Da mesma maneira que as pessoas enviam mensagens umas às outras (por exemplo, um sargento que ordena a um soldado que se mantenha em posição de sentido), objetos também se comunicam através de mensagens.

A OOP encapsula dados (atributos) e funções (comportamento) em pacotes chamados de objetos; os dados e funções de um objeto estão intimamente amarrados. Os objetos têm a propriedade de ocultação de informações. Isto significa que, embora os objetos possam saber como se comunicar uns com os outros através de interfaces bem- definidas, normalmente não é permitido aos objetos saber como outros objetos são implementados - os detalhes de implementação ficam escondidos dentro dos próprios objetos. Certamente é possível dirigir um carro eficazmente sem ser especialista nos detalhes de como motores, transmissões e escapamentos trabalham internamente. Veremos por que a ocultação de informações é tão crucial para a boa engenharia de software.

Em C e outras linguagens de programação procedurais, a programação tende a ser orientada a ações, enquanto que, em C++, a programação tende a ser orientada a objetos. Em C, a unidade de programação é afunção. Em C++, a unidade de programação é a classe, a partir da qual os objetos são eventualmente instanciados (um termo elegante para “criados”). As classes de C ++ contêm funções (que implementam os comportamentos da classe) e dados (que implementam os atributos da classe).

Os programadores de C se concentram em escrever funções. Os grupos de ações que executam alguma tarefa comum são reunidos para formar funções e as funções são agrupadas para formar programas. Certamente, os dados são importantes em C, mas o ponto de vista é que existem dados principalmente

para suportar as ações que as funções executam. Os verbos em uma especificação de sistema ajudam o programador de C a determinar o conjunto de funções que trabalham juntas para implementar o sistema.
Os programadores de C+÷ se concentram em criar seus próprios tipos definidos pelo usuário, chamados de classes e componentes. Cada classe contém tanto dados como também o conjunto de funções que manipulam estes dados. Os componentes de dados de uma classe são conhecidos como membros de dados. Os componentes funções de uma classe são conhecidos como funções membro (tipicamente conhecidas como métodos, em outras linguagens de programação orientadas a objetos, como Java). Da mesma maneira que uma instância de um tipo primitivo da linguagem, tal como int, é chamada de variável, uma instância de um tipo definido pelo usuário (i.e., uma classe) é conhecida como objeto. O programador usa tipos primitivos como blocos de construção para construir tipos definidos pelo usuário. O foco de atenção em C++ está nas classes (com as quais criamos objetos) ao invés de nas funções. Os substantivos em uma especificação de sistema ajudam o programador de C++ a determinar o conjunto de classes a partir das quais serão criados objetos que irão trabalhar juntos para implementar o sistema.
Classes estão para objetos assim como plantas arquitetônicas estão para casas. Podemos construir muitas casas a partir de uma planta e também podemos instanciar (criar) muitos objetos a partir de uma classe. Classes também podem ter relacionamentos com outras classes. Por exemplo, em um projeto orientado a objetos de um banco, a classe CaixaDeBanco precisa se relacionar com a classe Cliente. Estes relacionamentos são chamados de associações.
Veremos que quando software é empacotado como classes, estas classes podem ser reutilizadas em sistemas de software futuros. Grupos de classes relacionadas entre si são freqüentemente empacotadas como componentes reutilizáveis. Da mesma maneira que corretores de imóveis dizem a seus clientes que os três fatores mais importantes que afetam o preço dos imóveis são “localização, localização e localização”, acreditamos que os três fatores mais importantes que afetam o futuro do desenvolvimento de software são “reutilizar, reutilizar e reutilizar.”
Realmente, com a tecnologia de objetos, construiremos a maior parte do software do futuro combinando “peças padronizadas e intercambiáveis” chamadas de classes. Este livro lhe ensinará a “elaborar classes valiosas”

para serem reutilizadas, reutilizadas e reutilizadas. Cada nova classe que você criar terá o potencial para se tornar um valioso patrimônio de software que você e outros programadores podem usar para acelerar e aumentar a qualidade de futuros trabalhos de desenvolvimento de software. Esta é uma possibilidade fascinante.
Introdução à análise e projeto orientados a objetos (OOAD, object-oriented analysis and design)
A esta altura, você provavelmente já escreveu alguns poucos programas pequenos em C++. Como você criou o código para seus programas? Se você é

como muitos programadores principiantes, pode ter ligado seu computador e simplesmente começado a digitar. Esta abordagem pode funcionar para projetos pequenos, mas o que você faria se fosse contratado para criar um sistema de software para controlar as máquinas de caixa automáticas de um banco importante ? Um projeto como este é muito grande e complexo para que se possa simplesmente sentar e sair digitando.

Para criar as melhores soluções, você deveria seguir um processo detalhado para obter uma análise dos requisitos de seu projeto e desenvolver um projeto para satisfazer tais requisitos. Você passaria por este processo e teria seus resultados revisados e aprovados por seus superiores antes de escrever qualquer código para seu projeto. Se este processo envolve analisar e projetar seu sistema de um ponto de vista orientado a objetos, nós o denominamos processo pseudocódigo de análise e projeto orientados a objetos (OOAD, object-oriented analysis and design). Programadores experientes sabem que, não importa quão simples um problema pareça ser, o tempo gasto em análise e projeto pode poupar incontáveis horas que poderiam ser perdidas ao abandonar uma abordagem de desenvolvimento de sistema mal-planejada, a meio caminho de sua implementação.

OOAD é o termo genérico para as idéias por trás do processo que empregamos para analisar um problema e desenvolver uma abordagem para resolvê-lo. Problemas pequenos como os destes primeiros poucos capítulos não requerem um processo exaustivo. Pode ser suficiente escrever pseudocódigo antes de começarmos a escrever código. (Pseudocódigo é um meio informal de representar o código de um programa. Não é uma linguagem de programação de verdade, mas podemos usá-lo como uma espécie de “esboço” para nos guiar à medida que escrevemos o código. Introduzimos pseudocódigo no Capítulo 2). Pseudocódigo pode ser suficiente para problemas pequenos, mas na medida em que os problemas e os grupos de pessoas resolvendo estes problemas aumentam em tamanho, os métodos do OOAD são mais usados. Idealmente, um grupo deveria concordar quanto a um processo estritamente definido para resolver o problema e quanto a uma
maneira uniforme de comunicar os resultados deste processo uns para os outros. Existem muitos processos diferentes de OOAD; entretanto, uma linguagem gráfica para informar os resultados de qualquer processo de OOAD se tornou largamente usada. Esta linguagem é conhecida como UnifiedModeling Language (UML). A UML foi desenvolvida em meados da década de 90, sob a direção inicial de um trio de metodologistas de software: Grady Booch, James Rumbaugh e Ivar Jacobson.

História da UML

Na década de 80, um número crescente de organizações começou a usar OOP para programar suas aplicações e surgiu a necessidade de um processo adequado para abordar a OOAD. Muitos metodologistas - incluindo Booch, Rumbaugh e Jacobson - produziram e promoveram individualmente processos separados para satisfazer esta necessidade. Cada um destes processos tinha sua própria notação, ou “linguagem” (sob a forma de diagramas gráficos), para comunicar os resultados da análise e projeto.

No início da década de 90, empresas diferentes, e até mesmo divisões diferentes de uma mesma empresa, usavam processos e notações distintos. Além disso, estas empresas queriam usar ferramentas de software que suportassem seus processos particulares. Com tantos processos, os vendedores de software achavam difícil fornecer tais ferramentas. Ficou claro que eram necessários processos e notação padronizados.

Em 1994, James Rumbaugh juntou-se a Grady Booch na Rational Software Corporation e os dois começaram a trabalhar para unificar seus já populares processos. Em seguida, juntou-se a eles Ivar Jacobson. Em 1996, o grupo liberou versões preliminares da UML para a comunidade de engenharia de software e pediu um feedback. Mais ou menos na mesma época, uma organização conhecida como Object Mana gement GroupTM (oMJM) solicitou propostas para uma linguagem comum de modelagem. O OMG é uma organização sem fins lucrativos que promove o uso da tecnologia de orientação a objetos publicando diretrizes e especificações para tecnologias orientadas a objetos. Diversas corporações - entre elas HP, IBM, Microsoft, Oracle e Rational Software -já haviam reconhecido a necessidade de uma linguagem comum para modelagem. Estas empresas constituíram a UML Partners em resposta OMG aceitou a proposta e, em 1997, assumiu a responsabilidade pela manutenção e revisão continuadas da UML.

à solicitação de propostas do OMG. Este consórcio desenvolveu e submeteu a versão 1.1 da UML para o OMG. O Em 1999, o OMG liberou a versão 1.3 da UML (a versão atual por ocasião da publicação deste livro nos EUA).

CAPÍTULO 1 - INTRODUÇÃO AOS C”

Oqueéa UML?

A Unified Modeling Language é agora o esquema .
lagem de sistemas orientados a objetos. Ela certamei.
final da década de 80. Aqueles que projetam sistemas
modelar seus sistemas.

Uma das características mais atraentes da UML é s
muitos processos do OOAD. Modeladores UML ficam livre
mas todos os desenvolvedores podem agora expressar tais sis
A UML é uma linguagem gráfica complexa e repleta de.
apresentamos um subconjunto conciso, simplificado, destes recL
leitor através de uma primeira experiência com a UML, voltada
orientação a objetos. Para uma discussão mais completa da UML, coi
e o documento com as especificações oficiais da UML 1.3 (www. o.
foram publicados. UML Distilled: Second Edition, por Martin Fowler (
detalhada à versão 1.3 da UML, com muitos exemplos. The Unified M&
Booch, Rumbaugh e Jacobson, é o tutorial definitivo para a UML.

A tecnologia de orientação a objetos está em toda a parte na indústria
ficando assim. Nosso objetivo, nestas seções “Pensando em objetos”, é incenti
tada a objetos tão cedo e tão seguido quanto possível. Iniciando na seção
“Pensa.

2, você irá aplicar a tecnologia de objetos para implementar a solução de um

proL
você ache este projeto opcional uma introdução agradável e desafiadora ao projeto
e à programação orientada a objetos.
çindo em
C++
Resumo • trições
• Um computador é um dispositivo capaz de executar computações e tomar decisões lógicas em vi
bilhões, de vezes mais rápidas do que podem as pessoas.
• Oa dados são processados em computadores sob o controle de programas.
• Os vários dispositivos (tal como o teclado, tela, discos, memória e unidades de processamento) que compú computador são chamados de hardware.
• Os programas de computador que são executados em um computador são chamados de software.
• A unidade de entrada é a “seção receptora” do computador. A maioria das informações é hoje em dia fomec. computadores através de teclados como os das máquinas de escrever.
• A unidade de saída é a “seção de expedição” do computador. Atualmente, a maioria das informações de saída são exibi telas ou impressas em papel pelos computadores.
• A unidade de memória é o “depósito” do computador, e é freqüentemente chamada de memória ou memória primária.
• A unidade de aritmética e lógica (UAL) executa cálculos e toma decisões.
• Programas ou dados que não estão sendo ativamente usados pelas outras unidades normalmente são colocados em dispositivos de armazenamento secundário (tais como discos), até que sejam novamente necessários.
• No processamento em lotes com usuário único, o computador executa um único programa de cada vez, enquanto processa os dados em grupos ou lotes.
• Sistemas operacionais são sistemas de software que tomam o uso dos computadores mais conveniente, possibilitando a obtenção de um melhor desempenho dos mesmos.
• Sistemas operacionais multiprogramados possibilitam o processamento “simultâneo” de muitos trabalhos no computador - o computador compartilha seu recursos entre vários trabalhos.
• O timesharing é um caso especial de multiprogramação em que os usuários acessam o computador através de terminais. Os programas dos usuários parecem estar sendo executados simultaneamente.
• Com a computação distribuída, a computação de uma organização é distribuída através de redes para os diversos locais onde

la de dados.

o trabalho da organização é executado.

para serem reutilizadas, reutilizadas e reutilizadas. Cada nova classe que você criar terá o potencial para se tomar um valioso patrimônio de software que você e outros programadores podem usar para acelerar e aumentar a qualidade de futuros trabalhos de desenvolvimento de software. Esta é uma possibilidade fascinante.

Introdução à análise e projeto orientados a objetos (OOAD, object-oriented analysis and design)

A esta altura, você provavelmente já escreveu alguns poucos programas pequenos em C+÷. Como você criou o código para seus programas? Se você é como muitos programadores principiantes, pode ter ligado seu computador e simplesmente começado a digitar. Esta abordagem pode funcionar para projetos pequenos, mas o que você faria se fosse contratado para criar um sistema de software para controlar as máquinas de caixa automáticas de um banco importante ? Um projeto como este é muito grande e complexo para que se possa simplesmente sentar e sair digitando.

Para criar as melhores soluções, você deveria seguir um processo detalhado para obter uma análise dos requisitos de seu projeto e desenvolver um projeto para satisfazer tais requisitos. Você passaria por este processo e teria seus resultados revisados e aprovados por seus superiores antes de escrever qualquer código para seu projeto. Se este processo envolve analisar e projetar seu sistema de um ponto de vista orientado a objetos, nós o denominamos processo pseudocódigo de análise e projeto orientados a objetos (OOAD, object-oriented analysis and design). Programadores experientes sabem que, não importa quão simples um problema pareça ser, o tempo gasto em análise e projeto pode poupar incontáveis horas que poderiam ser perdidas ao abandonar uma abordagem de desenvolvimento de sistema mal-planejada, a meio caminho de sua implementação.

OOAD é o termo genérico para as idéias por trás do processo que empregamos para analisar um problema e desenvolver uma abordagem para resolvê-lo. Problemas pequenos como os destes primeiros poucos capítulos não requerem um processo exaustivo. Pode ser suficiente escrever pseudocódigo antes de começarmos a escrever código. (Pseudocódigo é um meio informal de representar o código de um programa. Não é uma linguagem de programação de verdade, mas podemos usá-lo como uma espécie de “esboço” para nos guiar à medida que escrevemos o código. Introduzimos pseudocódigo no Capítulo 2). Pseudocódigo pode ser suficiente para problemas pequenos, mas na medida em que os problemas e os grupos de pessoas resolvendo estes problemas aumentam em tamanho, os métodos do OOAD são mais usados. Idealmente, um grupo deveria concordar quanto a um processo estritamente definido para resolver o problema e quanto a uma maneira uniforme de comunicar os resultados deste processo uns para os outros. Existem muitos processos diferentes de OOAD; entretanto, uma linguagem gráfica para informar os resultados de qualquer processo de OOAD se tornou largamente usada. Esta linguagem é conhecida como UnifiedModeling Language (UML). A UML foi desen4volvida em meados da década de 90, sob a direção inicial de um trio de metodologistas de software: Grady Booch, James Rumbaugh e Ivar Jacobson.

História da UML
Na década de 80, um número crescente de organizações começou a usar OOP para programar suas aplicações e surgiu a necessidade de um processo adequado para abordar a OOAD. Muitos metodologistas - incluindo Booch, Rumbaugh e Jacobson - produziram e promoveram individualmente processos separados para satisfazer esta necessidade. Cada um destes processos tinha sua própria notação, ou “linguagem” (sob a forma de diagramas gráficos), para comunicar os resultados da análise e projeto.
No início da década de 90, empresas diferentes, e até mesmo divisões diferentes de uma mesma empresa, usavam processos e notações distintos. Além disso, estas empresas queriam usar ferramentas de software que suportassem seus processos particulares. Com tantos processos, os vendedores de software achavam difícil fornecer tais
- ferramentas. Ficou claro que eram necessários processos e notação padronizados.
Em 1994, James Rumbaugh juntou-se a Grady Booch na Rational Software Corporation e os dois começaram a trabalhar para unificar seus já populares processos. Em seguida, juntou-se a eles Ivar Jacobson. Em 1996, o grupo liberou versões preliminares da UML para a comunidade de engenharia de software e pediu um feedback. Mais ou menos na mesma época, uma organização conhecida como Object Management GroupTM (0MGTM) solicitou propostas para uma linguagem comum de modelagem. O OMG é uma organização sem fins lucrativos que promove o uso da tecnologia de orientação a objetos publicando diretrizes e especificações para tecnologias orientadas a objetos. Diversas corporações - entre elas HP, IBM, Microsoft, Oracle e Rational Software - já haviam reconhecido a necessidade de uma linguagem comum para modelagem. Estas empresas constituíram a UML Partners em resposta à solicitação de propostas do OMG. Este consórcio desenvolveu e submeteu a versão 1.1 da UML para o OMG. O OMG aceitou a proposta e, em 1997, assumiu a responsabilidade pela manutenção e revisão continuadas da UML. Em 1999,0 OMG liberou a versão 1.3 da UML (a versão atual por ocasião da publicação deste livro nos EUA).

CAPÍTULO 1 - INTRODUÇÃO AOS COMPUTADORES E À PROGRAMAÇÃO C ++ 85
O que é a UML?
A Unified Modeling Language é agora o esquema de representação gráfica mais amplamente utilizado para modelagem de sistemas orientados a objetos. Ela certamente unificou os diversos esquemas de notação que existiam no
final da década de 80. Aqueles que projetam sistemas usam a linguagem (sob a forma de diagramas gráficos) para
modelar seus sistemas.
Uma das características mais atraentes da UML é sua flexibilidade. A UML é extensível e independente dos
muitos processos do OOAD. Modeladores UML ficam livres para desenvolver sistemas usando diversos processos, mas todos os desenvolvedores podem

agora expressar tais sistemas com um conjunto padrão de notações.
A UML é uma linguagem gráfica complexa e repleta de recursos. Em nossas seções “Pensando em objetos”, apresentamos um subconjunto conciso, simplificado, destes recursos. Usamos então este subconjunto para guiar o leitor através de uma primeira experiência com a UML, voltada para o programador/projetista principiante em orientação a objetos. Para uma discussão mais completa da UML, consulte o site do OMG na Web (www. omg. org) e o documento com as especificações oficiais da UML 1.3 (www. omg. org/uml/). Muitos livros sobre UML foram publicados. UML Distilled: Second Edition, por Martin Fowler (com Kendall Scott), oferece uma introdução detalhada à versão 1.3 da UML, com muitos exemplos. The Unified Modeling Language User Guide, escrito por Booch, Rumbaugh e Jacobson, é o tutorial definitivo para a UML.
A tecnologia de orientação a objetos está em toda a parte na indústria de software e a UML está rapidamente ficando assim. Nosso objetivo, nestas seções “Pensando em objetos”, é incentivá-lo a pensar de uma maneira orientada a objetos tão cedo e tão seguido quanto possível. Iniciando na seção “Pensando em objetos” no fim do Capítulo 2, você irá aplicar a tecnologia de objetos para implementar a solução de um problema substancial. Esperamos que você ache este projeto opcional uma introdução agradável e desafiadora ao projeto orientado a objetos com a UML e à programação orientada a objetos.
Resumo
• Um computador é um dispositivo capaz de executar computações e tomar decisões lógicas em velocidades milhões, e até bilhões, de vezes mais rápidas do que podem as pessoas.
• Oa dados são processados em computadores sob o controle de programas.
• Os vários dispositivos (tal como o teclado, tela, discos, memória e unidades de processamento) que compõem um sistema de computador são chamados de hardware.
• Os programas de computador que são executados em um computador são chamados de software.
• A unidade de entrada é a “seção receptora” do computador. A maioria das informações é hoje em dia fornecida para os computadores através de teclados como os das máquinas de escrever.
• A unidade de saída é a “seção de expedição” do computador. Atualmente, a maioria das informações de saída são exibidas em telas ou impressas em papel pelos computadores.
• A unidade de memória é o “depósito” do computador, e é freqüentemente chamada de memória ou memória primária.
• A unidade de aritmética e lógica (UAL) executa cálculos e toma decisões.
• Programas ou dados que não estão sendo ativamente usados pelas outras unidades normalmente são colocados em dispositivos de armazenamento secundário (tais como discos), até que sejam novamente necessários.
• No processamento em lotes com usuário único, o computador executa um único programa de cada vez, enquanto processa os dados em grupos ou lotes.
• Sistemas operacionais são sistemas de software que tornam o uso dos

computadores mais conveniente, possibilitando a obtenção de um melhor desempenho dos mesmos.
• Sistemas operacionais multiprogramados possibilitam o processamento “simultâneo” de muitos trabalhos no computador - o computador compartilha seu recursos entre vários trabalhos.
• O timesharing é um caso especial de multiprogramação em que os usuários acessam o computador através de terminais. Os programas dos usuários parecem estar sendo executados simultaneamente.
• Com a computação distribuída, a computação de uma organização é distribuída através de redes para os diversos locais onde o trabalho da organização é executado.

Servidores armazenam programas e dados que podem ser compartilhados por computadores clientes distribuídos ao lc uma rede, daí o termo computação cliente/servidor.
Qualquer computador pode entender apenas sua própria linguagem de máquina. Linguagens de máquina geralmente c tem em strings de números (em última instância transformados em Is e Os), que instruem os computadores para executa operações mais elementares, uma de cada vez. As linguagens de máquina são dependentes da máquina.
Abreviações semelhantes a palavras da língua inglesa formam a base das linguagens simbólicas. Os montadores (assernb
traduzem os programas em linguagem simbólica para a linguagem de máquina.
Os compiladores traduzem os programas em linguagem de alto nível para a linguagem de máquina. As linguagens de
nível contêm palavras inglesas e notações matemáticas convencionais.
Os programas interpretadores executam diretamente programas em linguagem de alto nível, sem a necessidade de compi
aqueles programas para a linguagem de máquina.
Embora programas compilados executem mais rapidamente que programas interpretados, os interpretadores são popular nos ambientes de desenvolvimento de programas nos quais os programas são recompilados freqüentemente, à medida qi novas especificações são acrescentadas e erros são corrigidos. Uma vez que um programa acabou de ser desenvolvido, um versão compilada pode então ser produzida para ser executada de forma mais eficiente.
É possível se escrever programas em C e C++ que são portáveis para a maioria dos computadores.
FORTRAN (FORmula TRANsIator) é usada para aplicatisos matemáticos.
COBOL (COmmom Business Oriented Language) é usada principalmente para aplicações comerciais que exigem a manipulação precisa e eficiente de grandes volumes de dados.
A programação estruturada é uma abordagem disciplinada à escrita de programas que são mais claros que programas não- estruturados, mais fáceis

de testar e depurar, e mais fáceis de modificar.
Pascal foi projetado para o ensino da programação estruturada em ambientes acadêmicos.
Ada foi desenvolvida sob o patrocínio do Departamento de Defesa dos Estados Unidos (DOD), usando o Pascal como base.
Multitasking permite que os programadores especifiquem atividades para serem executadas em paralelo.
Todos os sistemas C++ consistem em três partes: o ambiente, a linguagem e as bibliotecas padrão. As funções de biblioteca não são parte da linguagem C++ propriamente dita: estas funções executam operações comuns, tais como cálculos matemáticos.
Os programas em C++ tipicamente passam através de seis fases até serem executados: edição, pré-processamento, compilação, “ligação”, carga e execução.
O programador digita um programa usando um editor e fazendo correções se necessário. Os nomes de arquivo em C++ em um sistema típico baseado em UNIX terminam com a extensão .c.
Um compilador traduz um programa em C++ para código em linguagem de máquina (ou código objeto).
O pré-processador obedece a diretivas de pré-processador que tipicamente indicam os arquivos que devem ser incluídos no arquivo fonte que está sendo compilado e símbolos especiais que devem ser substituidos por textos de programa.
Um editor de ligação “liga” o código objeto com o código de funções que estão fora do programa, para produzir uma imagem executável (sem partes faltantes).
Em um sistema típico baseado em UNIX, o comando para compilar e “ligar” um programa em C÷+ é CC. Se o programa compilar e ligar corretamente, é gerado um arquivo chamado a . out. Este contém a imagem executável do programa.
Um carregador busca um programa em formato executável no disco, transferindo-o para a memória.
Um computador, sob o controle de sua CPU, executa um programa uma instrução de cada vez.
Erros como divisão por zero acontecem quando um programa é executado: por isso estes erros são chamados de erros durante a execução.
Divisão por zero geralmente é um erro fatal, isto é, um erro que faz com que o programa termine imediatamente sem ter executado com sucesso seu trabalho.
Os erros não-fatais permitem que os programas concluam sua execução, produzindo freqüentemente resultados incorretos.
Certas funções em C++ recebem seus dados de cm (o stream padrão de entrada), normalmente associado ao teclado, mas que pode ser conectado a outro dispositivo. Os dados para saída são passados a cout (o stream padrão de saída de dados), normalmente conectado à tela do computador; porém, cout pode ser conectado a outro dispositivo.

• O streatn padrão para erros é chamado de cerr. O stream cerr (normalmente conectado à tela) é usado para exibir mensagens de erro.
• Existem muitas variações entre diferentes implementações de C++ em diferentes computadores, o que toma a portabilidade uma meta difícil de ser atingida.
• C++ fornece recursos para a programação orientada a objetos.
• Os objetos são componentes de software essencialmente reutilizáveis que modelam coisas no mundo real. Os objetos são criados a partir de “modelos” chamados de classes.
• Comentários de uma só linha começam com II. Os programadores inserem comentários para documentar os programas e melhorar sua legibilidade. Os comentários não fazem o computador executar qualquer ação quando o programa é executado.
• A linha #include <iostream> diz ao pré-processador C++ para incluir o conteúdo do arquivo de cabeçalho do stream de entradaisaída no programa. Este arquivo contém informações necessárias para compilar programas que usam std: : cm, std: : cout e os operadores «e».
• A execução de programas escritos em C++ começa na função main.
• O objeto de siream std: cout. para saída - normalmente conectado à tela - é usado para fazer a saída de dados. Múltiplos itens de dados podem ser concatenados para saída usando-se os operadores de inserção no stream («).
• O objeto de stream std: cm, para entrada - normalmente conectado ao teclado - é usado para receber entrada de dados. Múltiplos itens de dados podem ser udos concatenando-se operadores de extração do stream (»).
• Todas as variáveis em um programa em C++ devem ser declaradas antes de poderem ser usadas.
• Um nome de variável em c++ é qualquer identificador válido. Um identificador é uma série de caracteres consistindo em letras, dígitos e sublinhado (_). Identificadores em C++ não podem começar com um dígito. Os identificadores em C++ podem ter qualquer comprimento; porém, alguns sistemas e/ou implementações de C++ podem impor algumas restrições sobre o comprimento dos identificadores.
• C++ é sensível a maiúsculas e minúsculas.
• A maioria dos cálculos são executados em comandos de atribuição.
• Toda variável armazenada na memória do computador tem um nome, um valor, um tipo e um tamanho.
• Sempre que um novo valor é colocado em uma posição da memória, ele substitui o valor anterior naquela posição. O valor anterior é perdido.
• Quando um valor é lido da memória, o processo é não-destrutivo, isto é, uma cópia do valor é lida, deixando o valor original inalterado na posição de memória.

• C÷+ calcula o valor de expressões aritméticas em uma seqüência precisa, determinada pelas regras de precedência e associatividade de operadores.
• O comando if permite a um programa tomar uma decisão quando uma certa condição é encontrada. O formato para um comando if é
if (condição)

comando;
Se a condição é true (verdadeira), o comando no corpo do if é executado. Se a condição não é satisfeita, isto é, a condição é false (falsa), o comando no corpo é pulado.
• As condições em comandos if são comumente formadas usando-se operadores de igualdade e operadores relacionais. O
resultado do uso destes operadores é sempre true ou false.
• Os comandos
using std::cout;
using std: :cin;
using std::endl;
são comandos using que nos ajudam a eliminar a necessidade de repetir o prefixo std: :. A partir do ponto em que incluímos estes comandos using, podemos escrever cout em vez de std : cout, cm em vez de std: : cm e endi em vez de s td: : endi. respectivamente, no resto do programa.

88 C++ COMO PROGRAMAR
• A orientação a objetos é um modo natural de pensar sobre o mundo e de escrever programas de computador.
• Os objetos têm atributos (como tamanho, forma, cor, peso, etc.) e exibem comportamentos.
• Os seres humanos aprendem sobre objetos estudando seus atributos e observando seus comportamentos.
• Objetos diferentes podem ter muitos dos mesmos atributos e exibir comportamentos semelhantes.
• A programação orientada a objetos (OOP) modela objetos do mundo real através de duplicatas em software. Ela tira partido de relações de classe, nos quais os objetos de uma certa classe têm as mesmas características. Aproveita-se de relações de herança e até herança múltipla, em que novas classes derivadas são criadas herdando características de classes existentes e ainda contendo suas próprias características únicas.
• A programação orientada a objetos fornece um modo intuitivo de ver o processo de programação, isto é, modelar objetos do mundo real, seus atributos e seus comportamentos.
• A OOP também modela a comunicação entre objetos através de mensagens.
• A OOP encapsula dados (atributos) e funções (comportamentos) em objetos.
• Os objetos têm a propriedade de ocultação de informação. Embora objetos possam saber como se comunicar uns com os outros através de interfaces bem-definidas, normalmente os objetos não têm permissão de saber detalhes de implementação de outros objetos.
• A ocultação de informações é crucial para a boa engenharia de software.
• Em C e outras linguagens de programação procedurais, a programação tende a ser orientada à ação. Os dados são certamente importantes em C, mas o ponto de vista nela adotado é que os dados existem principalmente para apoiar as ações que as funções executam.
• Os programadores de C++ se concentram em criar seus próprios tipos

definidos pelo usuário, chamados de classes. Cada classe contém tanto os dados como também o conjunto de funções que manipulam os dados. Os dados componentes de uma classe são chamados de membros de dados. As funções componentes de uma classe são chamadas de funções membro ou métodos.
Terminologia
abstração comentário (II)
ação compilador
análise componente
análise e projeto orientados a objetos (OOAD) comportamento
associação comportamentos de um objeto
associatividade da direita para a esquerda computação cliente/servidor
associatividade da esquerda para a direita computação distribuída
associatividade de operadores computador
associatividade de um operador condição
atributo corpo de uma função
atributos de um objeto criando classes valiosas
biblioteca padrão C+÷ da direita para a esquerda
Booch, Grady dados
C decisão
C padrão ANS1ISO declaração
dependente da máquina
C++ padrão ANSIJISO dispositivo de entrada
caractere de escape (\) dispositivo de saída
caractere nova linha (\n) divisão de inteiros
caracteres de espaço em branco editor
carregamento encapsulamento
clareza entrada/saída (E/S)
classe erro de compilação
comando erro de lógica

erro de sintaxe
erro durante a compilação erro durante a execução erro fatal
erro não-fatal
estrutura if
fluxo de controle
função

função membro
hardware
herança
herança múltipla
identificador
independente da máquina instanciar

irit

inteiro (int) interface interpretador

iostream

Jacobson, Ivar
ligando
linguagem de alto nível
linguagem de máquina
linguagem de programação
linguagem de programação procedural linguagem simbólica

lista separada por vírgulas main
membro de dados
memória

memória primária
mensagem
método
modelagem
multiprocessador
multiprogramação
multitasking
nome de variável
Object Managemeni Group (OMG) objeto
objeto cerr

objeto cm
objeto cout
objeto padrão de entrada (cm) objeto padrão de erro (cerr) objeto padrão de saída (cout) ocultação de informação
operador
operador binário

operador de atribuição (=)
operador de multiplicação (*)
operador módulo (%)
operadores aritméticos
operadores de igualdade == “é igual a”

= “não é igual a” operadores relacionais
> “é maior que”
<“é menor que”
>= “é maior que ou igual a” <= “é menor que ou igual a”

operando orientado a ações palavras reservadas parênteses () parênteses aninhados

patrimônio em software ponto-e-vírgula (;) posição de memória precedência

pré-processador
programa de computador programa tradutor
programação estruturada programação orientada a objetos (OOP) programação procedural projeto
projeto orientado a objetos (OOD)

prompt
pseudocódigo
Rational Software Corporation regras de precedência de operadores requisitos reusabilidade de software “reutilizar, reutilizar, reutilizar”

Rumbaugh, James sensível a maíusculas e minúsculas seqüência de escape servidor de arquivo

software
std: :cerr std: :cin

std: :cout

std: :endl

string
substantivos em uma especificação de sistema terminador de comando (;)
tipo definido pelo usuário
unidade central de processamento (CPU) unidade de aritmética e lógica (UAL)
Unified Modeling Language (UML) using
using std: :cerr
using std::cin

using std::cout using std::endl
valor de uma variável variável

verbos em uma especificação de sistema

Erros comuns de programação
1.1 Erros como os de divisão por zero acontecem quando um programa está sendo executado; por isso, estes erros são chamados de “erros durante a execução”. Dividir por zero é geralmente um erro fatal, isto é, um erro que causa

o término imediato do programa sem este ter executado com sucesso seu trabalho. Os erros não-fatais permitem que os programas sejam executados até a conclusão, freqüentemente produzindo resultados incorretos. (Nota: em alguns sistemas, dividir por zero não é um erro fatal. Consulte a documentação do seu sistema).
1.2 Esquecer de incluir o arquivo iostream em um programa que recebe dados de entrada do teclado, ou envia dados de saída para a tela, faz o compilador emitir uma mensagem de erro.
1.3 Omitir o ponto-e-vírgula no fim de um comando é um erro de sintaxe. Um erro de sintaxe ocorre quando o compilador não pode reconhecer um comando. O compilador emite normalmente uma mensagem de erro para ajudar o programador a localizar e corrigir o comando incorreto. Os erros de sintaxe são violações da linguagem. Os erros de sintaxe são também chamados de erros de compilação, erros durante a compilação, ou erros de compilação porque aparecem durante a fase de compilação do programa.
1.4 Tentar usar o operador módulo, %, com operandos não-inteiros é um erro de sintaxe.
1.5 Ocorrerá um erro de sintaxe se qualquer um dos operadores ==. =. >=. e <= aparecer com espaços entre seus dois símbolos.
1.6 Inverter a ordem do par de símbolos em qualquer dos operadores 1, >, e < (escrevendo-os como = , =>, e =<. respectivamente) normalmente é erro de sintaxe. Em alguns casos, escrever = como =! não é um erro de sintaxe, mas quase certamente será um erro de lógica.
1.7 Confundir o operador de igualdade == com o operador de atribuição =. O operador de igualdade deveria ser lido “é igual a” e o operador de atribuição como “recebe” ou “recebe o valor de” ou “ a ele é atribuído o valor de”. Algumas pessoas preferem ler o operador de igualdade como “duplo igual”. Como logo veremos, confundir estes dois operadores pode não necessariamente produzir um erro de sintaxe facilmente identificável, mas pode causar erros de lógica extremamente sutis.
1.l Colocar um ponto-e-vírgula logo imediatamente após parêntese da direita, em uma estrutura if. é freqüentemente um erro de lógica (embora não seja um erro de sintaxe). O ponto-e-vírgula faria com que o corpo da estrutura if fosse considerado vazio, assim a estrutura if não executaria nenhuma ação, independentemente do fato de sua condição ser ou não verdadeira. Pior ainda: o comando do corpo original da estrutura if agora se tornaria um comando em seqüência com a estrutura if, sendo sempre executado, freqüentemente fazendo com que o programa produza resultados incorretus.
1.9 E um erro de sintaxe colocar espaços no meio de um identificador (por exemplo, main escrito como ma in).
Boas práticas de programação
1.1 Escreva seus programas em C++ de uma maneira simples e direta. Isto é às vezes chamado de KLS (“Mantenha-o simples” - Keep It Simple). Não “force” a linguagem tentando usos estranhos.
1.2 Leia os manuais para a versão de C++ que você está usando. Consulte estes manuais com freqüência, para certificar-se de que esteja ciente da rica

relação de recursos que C++ apresenta e de que esteja usando estes recursos corretamente.
1.3 Seu computador e compilador são bons professores. Se, depois de ler cuidadosamente seu manual de linguagem C++, você não tiver certeza de como funciona um recurso de C++, experimente usar um pequeno “programa de teste” e ver o que acontece. Configure as opções do seu compilador para “nível máximo de advertências”. Estude cada mensagem que obtiver ao compilar seus programas e corrija os programas para eliminar as mensagens.
1.4 Todo programa deveria começar com um comentário descrevendo o propósito do programa.
1.5 Muitos programadores fazem com que o último caractere impresso por uma função seja um nova linha (\n). Isto assegura que a função deixará o cursor da tela posicionado no início de uma nova linha. Convenções deste natureza encorajam
a reusabilidade de software - uma meta-chave em ambientes de desenvolvimento de software.
1.6 Recue o corpo inteiro de cada função um nível de indentação nas marcas de tabulação que definem o corpo da função. Isto faz com que a estrutura funcional de um programa se destaque e ajuda a tornar os programas mais
fáceis de ler.
1.7 Estabeleça uma convenção para o tamanho dos recuos de indentação que você prefere; então, aplique uniformemente essa convenção. A tecla de tabulação pode ser usada para criar recuos, mas pontos de tabulação podem variar. Recomendamos usar espaços entre tabulações de 1/4 de polegada ou (preferível) três espaços para criar o recuo para um nível de indentação.
1.8 Alguns programadores preferem declarar cada variável em uma linha separada. Este formato permite a fácil inserção de um comentário descritivo após cada declaração.
1.9 Coloque um espaço após cada vírgula (,) para tornar os programas mais legíveis.

1.10 Escolher nomes de variáveis significativos ajuda um programa a ser “autodocumentado,” isto é, toma mais fácil de entender o programa simplesmente lendo-o, em vez de ter que ler manuais ou usar comentários em excesso.
1.11 Evite identificadores que começam com sublinhado (_) simples ou duplo, porque compiladores de C++ podem usar nomes semelhantes para seu próprio uso interno. Isto evitará que nomes escolhidos por você sejam confundidos com nomes que os compiladores escolhem.
1.12 Sempre coloque uma linha em branco entre de uma declaração e comandos executáveis adjacentes. Isto faz com que as declarações se destaquem no programa, contribuindo para a clareza do mesmo.
1.13 Se você preferir colocar declarações no início de uma função, separe essas declarações dos comandos executáveis da função com uma linha em branco,

para destacar onde as declarações terminam e os comandos executáveis começam.
1.14 Coloque espaços do dois lados de um operador binário, Isto faz com que o operador se destaque, tornando o programa mais legível.
1.15 Como na álgebra, é aceitável se colocar parênteses desnecessários em uma expressão para tomá-la mais clara. Estes parênteses são chamados de redundantes. Parênteses redundantes são usados comumente para agrupar subexpressões de uma expressão grande, para tomar a expressão mais clara. Quebrar um comando longo em uma seqüência de comandos mais curtos e mais simples também aumenta a clareza.
1.16 Indente o comando que compõe corpo de uma estrutura if para fazer com que o corpo da estrutura se destaque, melhorando dessa forma a legibilidade.
1.17 Em um programa não deve haver mais que um comando por linha.
1.18 Um comando longo pode ser separado em várias linhas. Se um comando único deve ser dividido em várias linhas, escolha pontos de quebra que façam sentido, tal como depois de uma vírgula em uma lista separada por vírgulas ou depois de um operador em uma expressão longa. Se um comando é dividido em duas ou mais linhas, recue todas as linhas subseqüentes.
1.19 Consulte o quadro de precedência dos operadores quando escrever expressões contendo muitos operadores. Confirme que os operadores na expressão são executados na ordem que você espera. Se você não tem certeza da ordem de avaliação de uma expressão complexa, separe a expressão em comandos menores ou use parênteses para forçar a ordem, exatamente como você faria em uma expressão algébrica. Não deixe de observar que alguns operadores, tal como atribuição (), são associados da direita para a esquerda, em vez de da esquerda para a direita.
Dicas de desempenho
1.1 Usar funções e classes da biblioteca padrão, em vez de escrever suas próprias versões equivalentes, pode melhorar o desempenho do programa, porque este software é cuidadosamente escrito para ser executado de forma correta e eficaz.
1.2 Reutilizar componentes de código testados em vez de escrever suas próprias versões pode melhorar o desempenho do programa, pois estes componentes são normalmente escritos para rodar de forma eficiente.
Dicas de portabilidade
1.1 Como C é uma linguagem padronizada, independente de hardware e amplamente disponível, aplicativos escritos em C podem ser freqüentemente executados com pouca, ou nenhuma modificação, em uma ampla variedade de sistemas de
computação diferentes.
1.2 Usar funções e classes da biblioteca padrão, em vez de escrever suas próprias versões equivalentes, pode melhorar a portabilidade do programa, porque este software é incluído em virtualmente todas as implementações de C++.
1.3 Embora seja possível escrever programas portáveis, existem muitos problemas entre compiladores de C e C++ diferentes e computadores diferentes,

que podem tornar a portabilidade difícil de ser obtida. Simplesmente escrever programas em C e C++ não garante portabilidade. O programador com freqüência precisará lidar diretamente com variações de compilador e computador.
1.4 C++ permite identificadores de qualquer comprimento, mas o sistema e/ou sua implementação de C++ podem impor algumas restrições sobre o comprimento de identificadores. Use identificadores de 31 caracteres, ou menos, para assegurar a portabilidade dos seus programas.
Observações de engenharia de software
1.1 Use uma “abordagem de blocos de construção” para criar programas. Evite reinventar a roda. Use pedaços existentes onde for possível - isto é chamado de”reutilização de software” e é um aspecto central da programação orientada a objetos.
1.2 Quando estiver programando em C++, você usará tipicamente os seguintes blocos de construção: classes e funções da biblioteca padrão de C++, classes e funções que você mesmo cria e classes e funções de várias bibliotecas populares não-padronizadas.

1.3 Extensas bibliotecas de classes com componentes de software reutilizáveis estão disponíveis na Internet e na World Wide Web. Moitas destas bibliotecas estão disponíveis gratuitamente.
Exercícios de auto-revisão
1.1 Preencha os espaços em branco em cada uma das seguintes frases:
a) A empresa que popularizou a computação pessoal foi a _____
b) O computador que tornou a computação pessoal viável nos negócios e na indústria foi o _____
c) Computadores processam dados sob o controle de conjuntos de instruções chamados de - do computador.
d) As seis unidades lógicas principais do computador são _____, , _. e
e) As três classes de linguagens discutidas no capítulo são , e _____
f) Os programas que traduzem programas em linguagem de alto nível para a linguagem de máquina são chamados de g) C é amplamente conhecida como a linguagem de desenvolvimento do sistema operacional
h) A linguagem foi desenvolvida por Wirth para o ensino da programação estruturada nas universida des. i) O Departamento de Defesa dos EUA desenvolveu a linguagem Ada com um recurso chamado _____ o qual permite que os programadores especifiquem que muitas atividades possam ocorrer em paralelo.
1.2 Preencha os espaços em branco em cada uma das sentenças seguintes sobre o ambiente C÷÷.
a) Programas em C÷+ são normalmente digitados em um computador usando um programa
b) Em um sistema C++, um programa _____ é executado antes de a fase de tradução do compilador começar.
c) O programa combina a saída do compilador com várias funções de biblioteca

para produzir uma
imagem executável.
d) O programa _____ transfere a imagem executável de um programa em C++ do disco para a memória.
1.3 Preencha os espaços em branco em cada uma das seguintes frases.
a) Todo programa em C++ começa sua execução na função
b) A _____ começa o corpo de toda função e a _____ termina o corpo de toda função.
c) Todo comando termina com _____
d) A seqüência de escape \n representa o caractere _____, que faz com que o cursor se posicione no início da próxima linha na tela.
e) O comando _____ é usado para tomar decisões.
1.4 Diga se cada uma das seguintes frases é verdadeiro ou falsa. Se for falsa, explique por quê. Suponha que o comando using std: :cout; é usado.
a) Comentários fazem o computador imprimir o texto depois do / / na tela quando o programa é executado.
b) A seqüência de escape \n, quando transferida para a saída com cout, faz com que o cursor se posione no início da próxima linha da tela.
c) Todas as variáveis devem ser declaradas antes de serem usadas.
d) Todas as variáveis devem receber um tipo quando são declaradas.
e) C++ considera as variáveis ni.unber e NuNbEr idênticas.
f) Declarações podem aparecer quase em qualquer lugar no corpo de uma função em C++.
g) O operador módulo (%) pode ser usado só com operandos inteiros.
h) Os operadores aritméticos *, 1, %, + e - têm todos o mesmo nível de precedência.
i) Um programa em C++ que imprime três linhas na saída deve conter três comandos de saída usando cout.
1.5 Escreva um único comando em C++ para realizar cada uma das seguintes frases (Suponha que não foram usados comandos using):
a) Declare as variáveis c, thislsAVariable, q?6354 e nuxnber como sendo do tipo int.
b) Solicite ao usuário que forneça um inteiro. Termine sua mensagem de solicitação com um sinal de dois pontos (:) seguido por um espaço e deixe o cursor posicionado depois do espaço.
c) Leia um inteiro fornecido pelo usuário através do teclado e armazene o valor fornecido em uma variável de tipo inteiro chamada age.
d) Se a variável number não for igual a 7, imprima “O número na variável não é igual 7”.

e) imprima a mensagem “Este é um programa em C++ “em uma linha.
1) imprima a mensagem “Este é um programa em C++ “em duas linhas, na qual a primeira linha termina com c++.
g) Imprima a mensagem “Este é um programa em C++ “com cada palavra da

mensagem escrita em uma linha separada.
h) Imprima a mensagem “Este é um programa em C++ “, com cada palavra separada da seguinte por uma marca de tabulação.
1.6 Escreva um comando (ou comentário) para realizar cada uma das seguintes tarefas: (Suponha que foram usados comandos using)
a) Afirme que um programa calcula o produto de três inteiros.
b) Declare as variáveis x. y. z e result como sendo do tipo irzt.
c) Escreva prompt pedindo ao usuário para digitar três inteiros.
d) Leia três inteiros do teclado e armazene-os nas variáveis x. y e z.
e) Compute o produto dos três inteiros contidos nas variáveis x. y e z e atribua o resultado à variável result.
O Imprima O produto é seguido pelo valor da variável result.
g) Devolva um valor a partir de main para indicar que o programa terminou com sucesso.
1.7 Usando os comandos que você escreveu no Exercício 1.6, escreva um programa completo que calcula e imprime o produto de três inteiros. Nota: você vai precisar escrever os comandos using necessários.
1.8 Identifique e corrija os erros em cada um dos seguintes comandos (suponha que o comando using std: : cout; seja usado):
a)if (c<7)
cout « “c é menor que 7\n”;
b)if (c=>7)
cout « “c é igual a ou maior que 7\n”
1.9 Preencha com o termo correto em linguagem de objetos” os espaços em cada uma das seguintes frases:
a) Pessoas podem olhar para uma tela de TV e ver pontos coloridos, ou elas podem dar um passo atrás e ver três pessoas sentadas em uma mesa de conferência; este é um exemplo de uma capacidade chamada
_____
b) Se virmos um carro como um objeto, o fato de que o carro seja um conversível é um atributo/comportamento (escolha um)
e) O fato de um carro poder acelerar ou desacelerar, virar para a esquerda ou virar para a direita, ou ir para frente ou para trás são todos exemplos de _____ de um objeto carro.
d) Quando uma nova classe herda características de várias classes existentes diferentes, isto é chamado de herança
e) Objetos se comunicam enviando _____ uns aos outros.
f) Objetos se comunicam uns com os outros através de bem-definidos(as).
g) Normalmente, não é permitido a um objeto saber como outros objetos são implementados; esta propriedade é chamadade _____
h) Os em uma especificação de sistema ajudam o programador de C++ a determinar as classes que serão necessáriaç para implementar o sistema.
i) Os componentes de dados de uma classe são chamados de_____ e os componentes de função de uma classe são chamados de
j) Uma instância de um tipo definido pelo usuário é um (a)
Respostas aos exercícios de auto-revisão

1.1 a) Apple. b) IBM Personal Computer. e) programas. d) unidade de entrada, unidade de saída, unidade de memória, unidade de aritmética e lógica, unidade central de processamento, unidade secundária de armazenamento. e) linguagens de máquina, linguagens simbólicas, linguagens de alto nível. f) compiladores. g) UNIX. h) Pascal. i) multitasking.
1.2 a) editor. b) pré-processador. e) editor de ligação. d) carregador (loader).
1.3 a) main. b) chave à esquerda ({), chave à direita (}). c) ponto-e-vírgula. d) nova linha, e) if.
1.4 a) Falso. Os comentários não fazem qualquer ação ser executada durante a execução do programa. Eles são usados por documentar programas e melhorar sua legibilidade.

94 C++ COMO PROGRAMAR
b) Verdadeira.
c) Verdadeira.
d) Verdadeira.
e) Falsa. C++ é sensível a maiúsculas e minúsculas; por isso, estas variáveis não são iguais.
O Verdadeira.
g) Verdadeira.
h) Falsa. Os operadores , / e % têm a mesma precedência e os operadores + e - têm uma precedência mais baixa.
i) Falsa. Um único comando de saída usando cout, contendo seqüências de escape múltiplas, pode imprimir várias linhas.
1.5 a) int c, thislsAVariable, q76354, number;
b) std::cout « “Digite um jntejro: “;
e) std: :cin » age;
d) if (number 1= 7
std::cout « “O mimero na variável não é igual a 7\n”;
e) std::cout « “Este é um programa em
f) std::cout « “Este é um program\nem C++ \n”;
g) std: :cout « ‘Este\né\num\nprograma\nem\nC++\n”;
h) std::cout « Este\té\tum\tprograina\tem\tC++\n”;
1.6 a) II Calcular o produto de três inteiros
b) int x, y, z, result;
e) cout « “Forneça três inteiros: “;
d) ciii » x » y » z;
e) result = x * y *
f) cout « “O produto é “ « result « endl;
g) return O;
1.7 II Calcular o produto de três inteiros
#unclude <iostream>
using std::cout;
using std::cin;
using std::endl;
int mairi

int x, y, z, result;
cout « “Digite três inteiros: “;
ciii » x » y » z;
result = x * y *
cout « “O produto é “ « result « endi;
retunn O;
1.8 a) Erro: ponto-e-vírgula depois do parênteses direito da condição no comando if. Correção: remova o ponto-e-vfrgula depois do parênteses direito. Nota: o resultado deste erro é que o comando de saída será executado quer a condição no comando if seja verdadeira ou não. O ponto-e-vírgula depois do parênteses direito é considerado um comando vazio
-um comando que não faz nada. Aprenderemos mais sobre o comando vazio no próximo capítulo. b) Erro: o operador relational =>. Correção: mude => para >=.
1.9 a) abstração. b) atributo. c) comportamentos. d) múltipla. e) mensagens. f) interfaces. g) ocultação de informação. h) substantivos. i) membros de dados; funções membro ou métodos. j) objeto.
Exercícios
1.10 Classifique cada um dos itens seguintes como hardware ou software:
a) CPU
b) compilador C++
e) UAL

d) pré-processador C++
e) unidade de entrada
f) um programa editor
1.11 Por que você poderia querer escrever um programa em uma linguagem independente de máquina em vez de em uma linguagem dependente de máquina? Por que uma linguagem dependente de máquina poderia ser mais apropriada para escrever certos tipos de programas?
1.12 Preencha os espaços em branco em cada uma das seguintes sentenças:
a) Que unidade lógica do computador recebe informações de fora do computador para uso pelo computador?
b) O processo de instruir o computador para resolver problemas específicos é chamado de _____ c) Que tipo de linguagem de computador usa abreviações semelhantes a palavras em inglês para instruções de linguagem de máquina?
d) Que unidade lógica do computador envia as informações que já foram processadas pelo computador a vários dispositivos, de forma que as informações possam ser usadas fora do computador?
e) Que unidade lógica do computador guarda informações?
f) Que unidade lógica do computador executa cálculos? _____
g) Que unidade lógica do computador toma decisões lógicas?
h) O nível de linguagem de computador mais conveniente para o programador escrever programas depressa e facilmente

e _____
i) A única linguagem que um computador pode entender diretamente é chamada de _____ daquele computador.
j) Que unidade lógica do computador coordena as atividades de todas as outras unidades lógicas ?
1.13 Discuta o significado de cada um dos objetos seguintes:
a)std: :cin
b)std: :cout
c)std: :cerr
1.14 Por que hoje em dia se dá tanta atenção à programação orientada a objetos em geral e a C++ em particular?
1.15 Preencha os espaços em branco em cada uma das seguintes sentenças:
a) são usados para documentar um programa e melhorar sua legibilidade.
b) O objeto usado para exibir informações na tela é _____
c) Um comando de C++ que toma uma decisão é _____
d) Cálculos são normalmente executados por comandos
e) O objeto _____ recebe como entrada valores fornecidos pelo teclado.

1.16 Escreva um único comando, ou linha, de C++ que realize cada um dos seguintes comandos:
a) Imprimaamensagem “Digite dois números’.
b) Atribua o produto das variáveis b e c à variável a.
c) Afirme que um programa executa um exemplo de cálculo de folha de pagamento (isto é, use texto que ajuda a documentar o programa).
d) Recebe como entrada do teclado três valores do tipo inteiro e os transfere para as variáveis inteiras a, b e c.
1.17 Diga quais das seguintes afirmações são verdadeiras e quais são falsas. Se falsa, explique sua resposta.
a) Os operadores de C++ são avaliados da esquerda para direita.
b) Os seguintes nomes são todos nomes válidos de variáveis: under_bar , m928134, t5, j7, suas vendas, tota1na_conta_de1e, a, b, c, z, z2.
c) O comando cout « “a = “; é um exemplo típico de um comando de atribuição.
d) Uma expressão aritmética válida em C++, sem parênteses, é avaliada da esquerda para direita.
e) Todos os seguintes são nomes de variáveis não válidos: 3g, 87, 67h2, h22. 2h.
1.18 Preencha os espaços em branco em cada uma das seguintes frases:
a) Que operações aritméticas estão no mesmo nível de precedência que a multiplicação?_____ b) Quando parênteses estão aninhados, que conjunto de parênteses é avaliado primeiro em uma expressão aritmetica?
_____
e) Uma posição na memória do computador que pode conter valores diferentes em vários momentos ao longo da execução de um programa é uma
_____

1.19 O que imprime cada um dos seguintes comandos de C++, caso imprimam qualquer coisa, quando são executados? Se não imprimem nada, então responda “nada”. Assuma que x = 2 e y = 3.
a) cout « x;
b) cout « x + x;
c) cout « ‘x&’;
d) cout « “x = “ « x;
e) cout « x + y « “ = “ « y + x;
f) z = x + y;
g) cm » x » y;
h) II cout « “x + y = “ « x + y;
i) cout « “\n”;
1.20 Qual dos comandos de C++ seguintes contêm variáveis cujos valores são substituídos?
a) cm » b » c » d » e » f;
b) p = i + j + k + 7;
c) cout « “variáveis cujos valores são sulstituidos”;
d) cout « “a = 5”;
1.21 Dada a equação algébrica y = ax3 + 7, quais dos seguintes comandos, se houver algum, são comandos corretos de C++ para expressar esta equação?
a) y = a * x * x * x + 7;
b)y=a*x*x*(x+7);
c)y= (a * x) * x * (x+ 7);
d)y=(a*x)*x*x+7;
e)y=a*(x*x*x)+7;
f) y=a*x* (x*x+7)
1.22 Indique a ordem de avaliação dos operadores em cada um dos seguintes comandos de C++ e mostre o valor de x após cada comando ser executado.
a) x 7 + 3 * 6 / 2 - 1;
b) x = 2 % 2 + 2 * 2 - 2 / 2;
c)x= (3*9* (3+ (9*3/ (3))));
1.23 Escreva um programa que pede ao usuário que forneça dois números, obtém os dois números digitados pelo usuário e imprime a soma, o produto, a diferença e o quociente dos dois números.
1.24 Escreva um programa que imprima os números de 1 a 4 na mesma linha, com cada par de números adjacentes separados por um espaço. Escreva o programa usando os seguintes métodos:
a) Usando um comando de impressão com um operador de inserção no stream.
b) Usando um comando de impressão com quatro operadores de inserção no stream.
c) Usando quatro comandos de impressão.
1.25 Escreva um programa que pede ao usuário que forneça dois inteiros, obtém os números digitados pelo usuário e então imprime o número maior seguido pelas palavras “é o maior”. Se os números são iguais, imprime a mensagem ‘Estes números são iguais”.
1.26 Escreva um programa que recebe três inteiros como entrada do teclado e

imprime a soma, a média, o produto, o menor e o maior destes números. O diálogo de tela deve aparecer como a seguir:
1.27 Escreva um programa que lê o raio de um círculo e imprime seu diâmetro, circunferência e área. Para it, use o valor constante 3,14159. Faça estes cálculos em comandos de saída. (Nota: neste capítulo, discutimos só constantes e variáveis inteiras. No Capítulo 3, discutiremos números de ponto-flutuante, isto é, valores que podem ter pontos decimais.)

Digite três inteiros diferentes: 13 27 14
A soma é 54
A média é 18
O produto é 4914
O menor é 13
O maior é 27

1.30 Escreva um programa que lê cinco inteiros e determina e imprime o maior e o menor inteiro no grupo. Use somente as técnicas de programação que você aprendeu neste capítulo.
1.31 Escreva um programa que lê um inteiro e determina e imprime se ele é par ou ímpar. (Sugestão: use o operador módulo. Um número par é um múltiplo de dois. Qualquer múltiplo de dois deixa resto zero quando dividido por 2.)
1.32 Escreva um programa que lê dois inteiros e determina e imprime se o primeiro é um múltiplo do segundo. (Sugestão: use o operador módulo).
1.33 Exiba um padrão de tabuleiro de damas com oito comandos de saída, então exiba o mesmo padrão com tão poucos comandos de saída quanto possível.
*
*
*
1.34 Diga qual a diferença entre os termos “erro fatal” e “erro não-fatal”. Por que razão você poderia preferir experimentar um erro fatal em lugar de um erro não-fatal?
1.35 Aqui espiamos um pouco mais à frente. Neste capítulo, você aprendeu sobre inteiros e o tipo int. C++ também pode representar letras maiúsculas, letras mínusculas e uma variedade considerável de símbolos especiais. C++ usa inteiros pequenos para representar internamente cada caractere diferente, O conjunto de caracteres que um computador usa e as representações em inteiros correspondentes àqueles caracteres é o que se chama de conjunto de caracteres daquele computador. Você pode imprimir um caractere simplesmente incluindo esse caractere entre apóstrofes, como com
cout « ‘A’;
Você pode imprimira inteiro equivalente a um caractere usando static_cast como segue:

cout « static cast< int > (‘A’);

1.28 Escreva um programa que imprime uma caixa, um oval, uma seta e um losango, como segue:

********* *** *
4.
* * * * *** 4.4C
* * * * ***** 4. .4’
* * * * * 4 .4’
4. 4.
* * * * *
* *
* * 4 4 *
* *
* * 4 4. * * *
* * * **
4. *
********* 444 * *
1.29 O que o código seguinte imprime?
cout « “*\n**\n***\n****\n*****\n”;

****
****
****

1
1
Ii

Isto é chamado de operação de coerção (casi) (introduzimos coerções formalmente no Capítulo 2). Quando o comando precedente for executado, imprimirá o valor 65 (em sistemas que usam o conjunto de caracteres ASCIJ).
Escreva um programa que imprime os valores inteiros equivalentes a algumas letras maiúsculas, algumas letras minúsculas, dígitos e símbolos especiais. Pelo menos, imprima os inteiros equivalentes aos seguintes caracteres: A B C a b c O 1 2 $ * + / e o caractere espaço em branco.
1.36 Escreva um programa que recebe como entrada um número de cinco dígitos, separa o número em seus dígitos individuais
e imprime os dígitos separados um do outro por três espaços cada. (Sugestão: use os operadores para inteiros divisão e módulo).
Por exemplo, se o usuário digitar 42339. o programa deve imprimir

4 2 3 3 9

1.37 Usando só as técnicas que você aprendeu neste capítulo, escreva um programa que calcula os quadrados e cubos dos números de 0 até 10 e usa marcas de tabulação para imprimir a seguinte tabela de valores:
número quadrado cubo
o o o
1 1 1
2 4 8
3 9 27
4 16 64
5 25 125
6 36 216
7 49 343
8 64 512
9 81 729
10 100 1000
l.3S Dê uma resposta breve para cada uma das seguintes perguntas sobre "pensar objetos":
a) Por que este texto optou por discutir a programação estruturada em detalhes, antes de continuar com um tratamento detalhado da programação orientada a objetos ?
b) Quais são os passos típicos (mencionados no texto) de um processo de projeto orientado a objetos?
c) Como a herança múltipla é exibida por seres humanos?
d) Que tipos de mensagens as pessoas enviam umas às outras?
e) Objetos enviam mensagens uns aos outros através de interfaces bem-definidas. Que interfaces apresenta um rádio de carro (objeto) para seu usuário (um objeto pessoa) ?
1.39 Você está provavelmente levando em seu pulso um dos tipos mais comuns de objetos do mundo - um relógio. Discuta como cada uma das condições e conceitos seguintes se aplica à noção de um relógio: objeto, atributos, comportamentos. classe. herança (considere, por exemplo, um relógio despertador), abstração, modelagem, mensagens, encapsulamento, interface, ocultação de informação, membros de dados e funções membro.

Estruturas de controle

Objetivos

• Entender as técnicas básicas de solução de problemas.
• Ser capaz de desenvolver algoritmos através do processo
de refinamento top-down, passo a passo.
• Ser capaz de usar as estruturas de seleção if, if/else e
switch para escolher ações alternativas.
• Ser capaz de usar as estruturas de repetição while. do!
while e for para executar comandos em um programa repetidamente.
• Compreender a repetição controlada por contador e a repetição controlada por sentinelas.
• Ser capaz de usar os operadores de incremento, de
decremento, operadores lógicos e de atribuição.
• Ser capaz de usar os comandos de controle de programa
break e continue.

Vamos todos dar um passo à frente.
ta Lewis Carroll
e.
A roda já deu uma volta completa.
William Shakespeare, King Lear
Quem pode controlar seu destino?
• William Shakespeare, Othello
A chave utilizada é sempre brilhante.
Benjamin Franklin

Visão Geral

2.1 Introdução
2.2 Algoritmos
2.3 Pseudocódigo

2.4 Estruturas de controle
2.5 A estrutura de seleção if
2.6 A estrutura de seleção if/else
2.7 A estrutura de repetição while
2.8 Formulando algoritmos: estudo de caso 1 (repetição controlada por contador)

2.9 Formulando algoritmos com refinamento top-down, passo a passo: estudo de caso 2 (repetição controlada por sentinela)
2.10 Formulando algoritmos com refinamento top-down, passo a passo: estudo de caso 3 (estruturas de controle aninhadas)
2.11 Operadores de atribuição
2.12 Operadores de incremento e decremento
2.13 Aspectos essenciais da repetição controlada por contador
2.14 A estrutura de repetição for
2.15 Exemplos usando a estrutura for
2.16 A estrutura de seleção múltipla switch
2.17 A estrutura de repetição do/while
2.18 Os comandos break e continue
2.19 Operadores lógicos
2.20 Confundindo os operadores de igualdade (==) e de atribuição (=)
2.21 Resumo de programação estruturada
2.22 (Estudo de caso opcional) Pensando em objetos: identificando as classes em um problema
Resumo• Terminologia Erros comuns de programação Boas práticas de programação Dicas de desempenho • Dicas de portabilidade. Observações de engenharia de software Dicas de teste e depura ção. Exercícios de auto-revisão • Respostas aos exercícios de auto-revisão • Exercícios

2.1 Introdução

Antes de escrever um programa para resolver um problema particular, é essencial se ter uma compreensão cuidadosa e profunda do problema e uma abordagem cuidadosamente planejada para resolver o problema. Ao escrever um programa, é igualmente essencial entender os tipos de blocos de construção que estão disponíveis e empregar princípios testados de construção de programas. Neste capítulo, discutimos todos estes assuntos em nossa apresentação da teoria e princípios da programação estruturada. As técnicas que você aprenderá aqui são aplicáveis à maioria das linguagens de alto nível, inclusive C++. Quando começarmos nosso tratamento da programação orientada a objetos em C++, no Capítulo 6, veremos que as estruturas de

controle que estudamos aqui no Capítulo 2 serão úteis para construir e manipular objetos.

CAPÍTULO 2 - ESTRUTURAS DE CONTROLE 101

2.2 Algoritmos

Qualquer problema de computação pode ser resolvido executando uma série de ações em uma seqüência específica. Um procedimento para resolver um problema em termos
1. das ações a serem executadas e
2. da sequência em que estas ações devem ser executadas
é chamado de algoritmo. O exemplo seguinte demonstra que é importante especificar corretamente a seqüência em que as ações serão executadas.
Considere o algoritmo de “preparar-se para ir trabalhar “ seguido por um executivo júnior para sair da cama e
ir para o trabalho: (1) saia da cama, (2) tire o pijama, (3) tome banho, (4) vista-se, (5) tome o café da manhã e (6) dirija seu carro para o trabalho.
Esta rotina leva o executivo para o seu trabalho bem preparado para tomar decisões críticas. Suponha que os
mesmos passos sejam executados em uma seqüência ligeiramente diferente: (1) saia da cama, (2) tire o pijama, (3) vista-se, (4) tome banho, (5) tome o café da manhã e (6) dirija seu carro para o trabalho.
Neste caso, nosso executivo júnior se apresentaria para trabalhar literalmente ensopado. Especificar a seqüência em que os comandos devem ser executados em um programa de computador é chamado de controle do programa. Neste capítulo, investigamos os recursos de controle de programas de C++.

2.3 Pseudocódigo

O pseudocódigo é uma linguagem artificial e informal que ajuda os programadores a desenvolver algoritmos. O pseudocódigo que apresentamos aqui é útil para desenvolver algoritmos que serão convertidos em programas estruturados em C++. O pseudocódigo é semelhante ao inglês do dia-a-dia; é conveniente e amigável, embora não seja uma linguagem de programação real para computadores.
Os programas em pseudocódigo não são realmente executados em

computadores. Em vez disso, o pseudocódigo ajuda o programador a “conceber” um programa, antes de tentar escrever o mesmo em uma linguagem de programação tal como C++. Neste capítulo, damos vários exemplos de como o pseudocódigo pode ser eficazmente usado para desenvolver programas estruturados em C++.
O estilo de pseudocódigo que apresentamos consiste puramente em caracteres, de modo que programadores podem escrever programas em pseudocódigo de maneira conveniente, usando apenas um programa editor. O computador pode exibir uma nova cópia de um programa em pseudocódigo quando necessário. Um programa em pseudocódigo cuidadosamente preparado pode ser convertido facilmente em um programa correspondente em C++. Isto é feito, em muitos casos, simplesmente substituindo os comandos de pseudocódigo pelos seus equivalentes em C++.
O pseudocódigo consiste somente em comandos executáveis - aqueles que são executados quando o programa é convertido de pseudocódigo para C++ e é executado. As declarações não são comandos executáveis. Por exemplo, a declaração

```
int i;
```

simplesmente diz ao compilador o tipo da variável i e instrui o compilador para reservar espaço na memória para a variável, mas esta declaração não provoca nenhuma ação - tal como entrada, saída ou um cálculo - que deva ocorrer quando o programa é executado. Alguns programadores escolhem listar as variáveis e mencionar brevemente o propósito de cada uma no início de um programa em pseudocódigo.

2.4 Estruturas de controle

Normalmente, os comandos em um programa são executados um depois do outro, na seqüência em que estão escritos. Isto é chamado de execução seqüencial. Vários comandos de C++ que logo discutiremos permitem ao programador especificar que o próximo comando a ser executado poder ser um outro que não o próximo na seqüência. Isto é uma transferência de controle.

Durante os anos 60, tornou-se claro que o uso indiscriminado de transferências de controle era a raíz de muitas das dificuldades experimentadas por grupos de desenvolvimento de software. O comando goto foi considerado culpado, porque

permite ao programador especificar uma transferência de controle para uma variedade muito grande de destinos possíveis em um programa. A noção da chamada programação estruturada se tornou quase sinônimo da “eliminação de goto”.
A pesquisa de Bohm e Jacopini’ demonstrou que os programas podiam ser escritos sem quaisquer comandos goto. O desafio para os programadores daquela era se tornou mudar seus estilos de programação: “escrever programas sem usar o comando goto’. Não foi senão até os anos 70 que os programadores começaram a aceitar a programação estruturada seriamente. Os resultados foram impressionantes, como perceberam grupos de desenvolvimento de software: reduções no tempo de desenvolvimento de software, término dos sistemas dentro do prazo mais freqüente e conclusão mais freqüente de projetos de software dentro do orçamento. A chave para estes sucessos é que programas estruturados são mais claros, fáceis de depurar e modificar, e tem maior probabilidade de ser isentos de erros do que os programas não estruturados.
O trabalho de Bohm e Jacopini demonstrou que todos os programas podiam ser escritos em termos de somente três estruturas de controle, isto é, a estrutura de seqüência, a estrutura de seleção e a estrutura de repetição. A estrutura de seqüência está embutida em C++. A menos que instruído contrariamente, o computador executa os comandos de C++ um depois do outro, na seqüência em que estão escritos. O segmento defluxograma da Fig. 2.1 ilustra uma estrutura de seqüência típica, uma estrutura na qual dois cálculos são executados em seqüência.
Um fluxograma é uma representação gráfica de um algoritmo ou uma representação de uma parte de um
algoritmo. Fluxogramas são desenhados usando-se certos símbolos especiais, tais como retângulos, losangos, ovais e pequenos círculos; estes símbolos são conectados por setas chamadas linhas defluxo.
Como o pseudocódigo, os fluxogramas são úteis para desenvolver e representar algoritmos, embora o pseudocódigo seja muito mais preferido pela maioria de programadores. Fluxogramas mostram claramente como operam as estruturas de controle; é por isso que nós os usamos neste texto.
Considere o segmento de fluxograma para a estrutura de seqüência na Fig. 2.1. Usamos o símbolo de retângulo, também chamado de símbolo de ação, para indicar qualquer tipo de ação, inclusive um cálculo ou uma operação de entrada/ saída. As linhas de fluxo na figura indicam a seqüência em que as ações devem ser executadas - primeiro, nota deve ser somado a total, então 1 deve ser somado a contador. C++ permite que, em uma estrutura de seqüência, tenhamos tantas ações quantas quisermos. Como logo veremos, em qualquer lugar em que uma única ação pode ser colocada, podemos colocar várias ações em seqüência.
Quando estivermos desenhando um fluxograma que representa um algoritmo completo, uma elipse contendo
a palavra “início” será o primeiro símbolo usado no fluxograma; uma elipse contendo a palavra “fim” será o último

```
total = total + nota;
contador = contador + 1;
```

Fig. 2.1 Colocando em um fluxograma a estrutura de seqüência de C++.

Bohm, C. and G. Jacopini, “Flow Diagrams, Turing Machines, and Languages with Only Two Formation Rules”, Communications ofthe ACM.
Vol. 9, N 5, May 1966, pp. 336-37 1.

símbolo usado. Quando estivermos desenhando só uma parte de um algoritmo, como na Fig. 2.1, as elipses são omitidas; em seu lugar usamos pequenos círculos, também chamados de conectores.
Talvez o mais importante símbolo na elaboração de fluxogramas seja o losango. Também chamamos o losango de símbolo de decisão, um símbolo que indica que é necessário tomar uma decisão. Discutiremos o losango na próxima seção.

C++ oferece três tipos de estruturas de seleção. A estrutura de seleção if executa (seleciona) uma ação se uma condição for true ou pula a ação se a condição for false. A estrutura de seleção if/else executa uma ação se uma condição for true e executa uma ação diferente se a condição for false. A estrutura de seleção switch executa uma de muitas ações diferentes, dependendo do valor de uma expressão inteira.
A estrutura de seleção if é uma estrutura de seleção única - seleciona ou ignora uma única ação. A estrutura de seleção if/else é uma estrutura de seleção dupla - seleciona entre duas ações diferentes. A estrutura de seleção switch é uma estrutura de seleção múltipla - seleciona a ação a ser executada dentre muitas ações diferentes.
C++ fornece três tipos de estruturas de repetição, a saber: while, do/while e for. Cada uma das palavras if. else. switch, while, do e for é uma palavra-chave de C ++. Estas palavras são reservadas pela linguagem para implementar vários recursos, tais como as estruturas de controle de C++. Palavras-chave não podem ser usadas como identificadores, tais como nomes de variáveis. Na Fig. 2.2, é mostrada uma lista completa das palavras- chave de C++.
Palavras-chave de C++
Palavras-chave comuns às linguagens de programação Cc C++
auto break case char const

continue default do double else
enum extern float for goto
if int long register return
short signed sizeof static struct
switch typedef union unsigned void
volatile while
Palavras-chave somente de C+ +
asm bool catch class constcast
delete dynamiccast explicit false friend
mime mutable namespace new operator
private protected public reinterpret_cast
static_cast template this throw true
try typeid typename using virtual
wchart
Fig. 2.2 Palavras-chave de C++.

Erro comum de programação 2.1
O uso de uma palavra-chave como um identificador é erro de sintaxe.

Bem, isso é tudo! C++ tem só sete estruturas de controle: seqüência, três tipos de seleção e três tipos de repetição. Cada programa em C++ é formado combinando-se tantas estruturas de cada tipo de estrutura de controle conforme seja necessário para o algoritmo que o programa implementa. Como com a estrutura de seqüência da Fig. 2.1, veremos que cada estrutura de controle é representada por um fluxograma com dois círculos pequenos, um no ponto de entrada da estrutura de controle e um no ponto de saída da estrutura de controle. Estas estruturas de controle de entrada e saída únicas facilitam a construção de programas - as estruturas de controle são ligadas umas às outras

conectando-se o ponto de saída de uma estrutura de controle ao ponto de entrada da próxima estrutura de controle. Isto é semelhante ao modo que uma criança empilha blocos de construção; assim, chamamos a este processo de empilhamento de estruturas de controle. Aprenderemos que existe somente um outro modo para conectar estruturas de controle - chamado de aninhamento de estruturas de controle.
Observação de engenharia de software 2.1

_____ Qualquer programa em C++ que venhamos a construir pode ser construído usando-se somente sete tipos diferentes de estruturas de controle (seqüência, if, if/else, switch, while, do/while e for), combinadas somente de dois modos (empilhamento de estruturas de controle e aninhamento de estruturas de controle).

2.5 A estrutura de seleção if

Uma estrutura de seleção é usada para se escolher cursos de ação alternativos. Por exemplo, suponha que a nota para passar em um exame seja 60. O comando em pseudocódigo

Se a nota do estudante é maior que ou igual a 60
Imprima “Aprovado”

determina se a condição “nota do estudante é maior que ou igual a 60” é true ou false. Se a condição é true. então é impresso “Aprovado” e o próximo comando em pseudocódigo na seqüência é “executado” (lembre que o pseudocódigo não é uma linguagem de programação real). Se a condição é false, o comando de impressão é ignorado e o próximo comando em pseudocódigo na seqüência é executado. Note que a segunda linha desta estrutura de seleção é indentada. Tal indentação é opcional, mas altamente recomendada porque enfatiza a estrutura inerente dos programas estruturados. Quando você converter seu pseudocódigo para o código em C++, o compilador de C++ ignora caracteres de espaçamento, como caracteres em branco, pontos de tabulação e caracteres de nova linha, usados para indentação e espaçamento vertical.

Boa prática de programação 2.1
Aplicar consistentemente convenções razoáveis de indentação em todos os seus programas melhora muito a legibilidade do programa. Sugerimos uma marca de tabulação com um tamanho fixo de cerca de 1/4 de polegada, ou três espaços, por nível de indentação.

O comando Se do pseudocódigo precedente pode ser escrito em C++ como

```
if ( nota >= 60)
cout << « “Aprovado”;
```

Note que o código em C++ corresponde de maneira próxima ao pseudocódigo. Esta é uma das propriedades do pseudocódigo que o torna uma ferramenta útil para o desenvolvimento de programas.

Boa prática de programação 2.2
O pseudocódigo éfreqüentemente usado para “bolar” um programa, durante o processo de projeto do mesmo. Após isso, o programa é convertido de pseudocódigo para C++.

O fluxograma da Fig. 2.3 ilustra a estrutura if de seleção única. Este fluxograma

contém o que talvez seja o mais
importante símbolo na elaboração de fluxogramas - o losango, também conhecido como símbolo de decisão, que
indica que uma decisão deve ser tomada. O símbolo de decisão contém uma expressão, tal como uma condição, que
pode ser ou true ou false. O símbolo de decisão tem duas linhas de fluxo que saem dele. Uma indica a direção
a ser seguida quando a expressão dentro do símbolo é true; outra indica a direção a ser tomada quando a expressão

é false. Aprendemos no Capítulo 1 que decisões podem ser baseadas em condições (contendo operadores relacionais ou de igualdade). Na verdade, uma decisão pode ser tomada com base em qualquer expressão - se o valor da expressão é zero, ela é tratada como false se o valor da expressão não é zero, ela é tratada como true. O padrão C++ oferece o tipo de dados bool para representar true e false. As palavras-chave true e false são usadas para representar valores do tipo bool.
Note que a estrutura if também é uma estrutura de entrada e saída únicas. Logo aprenderemos que os fluxogramas das demais estruturas de controle podem conter (além dos pequenos círculos e linhas de fluxo) somente retângulos, para indicar as ações que devem ser executadas, e losangos, para indicar as decisões que devem ser tomadas. Este é o modelo de programação de ação/decisão, que temos enfatizado até agora.
Podemos imaginar sete caixas, cada uma delas contendo somente estruturas de controle de um dos sete tipos. Estas estruturas de controle estão vazias. Nada está escrito nos retângulos ou nos losangos. Assim, a tarefa do programador é montar um programa juntando tantas peças de cada tipo de estrutura de controle de quantas o algoritmo necessite e, então, combinar essas estruturas de controle dos dois modos possíveis (empilhamento ou aninhamento) e preenchê-las com ações e decisões de maneira apropriada para o algoritmo. Discutiremos os vários modos segundo os quais as ações e decisões podem ser escritas.

2.6 A estrutura de seleção if/else

A estrutura de seleção if executa uma ação indicada só quando a condição é true; caso contrário, a ação é saltada. A estrutura de seleção if/else permite ao programador especificar que uma ação deve ser executada quando a condição é true e uma ação diferente quando a condição é false. Por exemplo, o comando

de pseudocódigo
Se a nota do estudante é maior que ou igual a 60
Imprime “Aprovado”
senão
imprime “Reprovado”
imprime Aprovado se a nota do estudante é maior que ou igual a 60 e imprime Reprovado se a nota do estudante é menor que 60. Em um caso ou outro, depois de ocorrer a impressão, o próximo comando de pseudocódigo da seqüência é “executado”. Note que o corpo do senão também é indentado.

Boa prática de programação 2.3
Indentar ambos os comandos do corpo de uma estrutura if/else.
Qualquer que seja a convenção de indentação que você escolha, ela deve ser aplicada cuidadosamente ao longo de todos os seus programas. É difícil ler programas que não obedecem a convenções de espaçamento uniformes.

Fig. 2.3 Representando em fluxograma a estrutura de seleção única if.

Boa prática de programação 2.4
Se existem vários níveis de indentação, cada nível deve ser indentado pelo mesmo espaço adicional.
A estrutura se/senão precedente. em pseudocódigo, pode ser escrita em C++ como

```
if ( nota >= 60)
cout <<  “Aprovado”;
else
cout << “Reprovado”;
```

O fluxograma da Fig. 2.4 ilustra bem o fluxo de controle na estrutura if /else. Uma vez mais, note que além de círculos pequenos e setas) os únicos símbolos no fluxograma são retângulos (para ações) e um losango (para uma decisão). Continuamos a enfatizar este modelo de computação com ação/decisão. Imagine novamente uma caixa grande contendo tantas estruturas vazias de seleção dupla quantas poderiam ser necessárias para construir qualquer programa em C++. O trabalho do programador é juntar estas estruturas de seleção (empilhando e aninhando) com quaisquer outras estruturas de controle

exigidas pelo algoritmo e preencher os retângulos e losangos vazios com ações e decisões apropriadas ao algoritmo que está sendo implementado.
Fig. 2.4 Representando em fluxograma a estrutura de seleção dupla if else.
C++ oferece o operador condicional (? :) bastante semelhante à estrutura i fiel se. O operador condicional é o único operador ternário de C++ - ele aceita três operandos. Os operandos, juntamente com o operador condicional, formam uma expressão condicional. O primeiro operando é uma condição; o segundo operando é o valor para a expressão condicional inteira, se a condição é true; e o terceiro operando é o valor para a expressão condicional inteira se a condição é false. Por exemplo, o comando de saída
cout << (nota >= 60 ? “Aprovado” “Reprovado”);
contém uma expressão condicional que produz como resultado (valor da expressão) o string Aprovado’ se a condição nota >= 60 for true e produz como resultado o string “Reprovado” se a condição for false. Deste modo, o comando com o operador condicional executa essencialmente o mesmo que o comando if/ else precedente. Como veremos, a precedência do operador condicional é baixa; assim, necessitamos dos parênteses na expressão precedente.
Os valores em uma expressão condicional também podem ser ações para serem executadas. Por exemplo, a
expressão condicional

1

falso

nota >= 60 ? cout << “Aprovado" cout << “Reprovado";

é lida como: “Se nota é maior que ou igual a 60 então cout « “Aprovado”; senão, cout « “Reprovado”. Isto também é comparável à estrutura if/else precedente. Veremos que operadores condicionais podem ser usados em algumas situações onde comandos if/else não podem.
Estruturas if/else aninhadas testam múltiplos casos colocando estruturas de seleção if/else dentro de outras estruturas de seleção if/else. Por exemplo, o seguinte comando em pseudocódigo imprimirá A para notas maiores que ou iguais a 90, B para notas no intervalo de 80 a 89, C para notas no intervalo 70 a 79, D para notas no intervalo 60 a 69 e R para as demais notas.
Se nota do estudante for maior que ou igual a 90

imprima “A”
senão
Se nota do estudante for maior que ou igual a 80
Imprima “B”
senão
Se nota do estudante for maior que ou igual a 70
Imprima “C”
senão
Se nota do estudante for maior que ou igual a 60
Imprima “D”
senão
Imprima “R”
Este pseucódigo pode ser escrito em como

```
if ( nota > 90)
cout << “A”;
else
if ( nota > 80)
cout << “B”;
else
if ( nota > 70
cout << “C”;
else
if ( nota >= 60)
cout << “D”;
else
cout << “R”;
```

Se nota for maior que ou igual a 90, as quatro primeiras condições serão true. mas só o comando cout depois do primeiro teste será executado. Depois de cout ser executado, a parte else do comando if/else “externo” é saltada. Muitos programadores de C++ preferem escrever a estrutura if precedente como

```
if ( nota > 90
cout << “A”;
else if (nota >=80
cout « “B”;
else if (nota >70
cout « “C”;
else if (nota >=60
cout « “D”;
else

cout « “R”;
```

As duas formas são equivalentes. A segunda forma é mais popular porque evita os níveis de indentação profundos do código para a direita. Tal estilo de indentação freqüentemente deixa pouco espaço em uma linha, forçando a quebra de linhas e diminuindo a legibilidade do programa.

1Dica de desempenho 2.1
Uma estrutura if/else aninhada pode ser muito mais rápida do que uma série de estruturas if de seleção única, por causa da possibilidade de saída logo na primeira condição satisfeita.

Dica de desempenho 2.2
_____ Em uma estrutura if/else aninhada, teste as condições que são mais prováveis de serem true no início da estrutura if/else aninhada. Isso permitirá que a estrutura if/else aninhada seja executada mais rapidamente, bem como permitirá uma saída mais rápida do que testar primeiro casos pouco freqüentes.
A estrutura de seleção if espera somente um comando no seu corpo. Para incluir vários comandos no corpo de um if, inclua os comandos entre chaves ({ e }). Um conjunto de comandos entre chaves é chamado de comando composto.

Observação de engenharia de software 2.2
_____ Um comando composto pode ser colocado em qualquer lugar de um programa em que um comando único pode ser colocado.
O exemplo a seguir inclui um comando composto na parte else de uma estrutura if/else. if (nota >= 60
cout « “Aprovado.\n”;
else {
cout << “Reprovado.\n”;
cout << “Você deve fazer este curso de novo.\n”;
Neste caso, se nota for menor que 60, o programa executa ambos os comandos no corpo do else e imprime
Reprovado.
Você deve fazer este curso de novo.
-1 Note as chaves abrangendo os dois comandos na cláusula else. Estas chaves são importantes. Sem elas, o comando cout « Você deve fazer este curso de novo.\n”;
estaria do lado de fora do corpo da parte else do if e seria executado sempre, mesmo se a nota não fosse menor que 60.

Erro comum de programação 2.2
Esquecer uma ou as duas chaves que delimitam um comando composto pode levar a erros de sintaxe ou erros de lógica em um programa.

Boa prática de programação 2.5
Colocar sempre as chaves em uma estrutura if/else (ou qualquer estrutura de controle) ajuda a prevenir sua omissão acidental, especialmente quando adicionarmos comandos à cláusula if ou else mais tarde.

Um erro de sintaxe é detectado pelo compilador. Um erro de lógica tem seu efeito durante a execução. Um erro de lógica fatal faz com que um programa se perca e termine prematuramente. Um erro de lógica não-fatal permite que um programa continue a ser executado, mas o programa pode produzir resultados incorretos.

Observação de engenharia de software 2.3
_____ Da mesma moneira que um comando composto pode ser colocado em qualquer lugar que possa ser colocado um comando único, também é possível não se ter comando algum, ou seja, um comando vazio. O comando vazio é representado colocando-se umponto-e-vírgula (;) onde seria normalmente colocado um comando.

Erro comum de programação 2.3
Colocar um ponto-e-vírgula depois da condição em uma estrutura if leva a um erro de lógica em estruturas if de seleção única e a um erro de sintaxe em estruturas if de seleção dupla (se a parte if contiver um comando em seu corpo).

Boa prática de programação 2.6
Alguns programadores preferem digitar as chaves de início efim de comandos compostos antes de digitar
os comandos individuais dentro das chaves. Isto ajuda a evitar a omissão de uma ou ambas das chaves.
Nesta seção, introduzimos a noção de um comando composto. Um comando composto pode conter declarações (como, por exemplo, acontece no corpo de main). Se esse foro caso, o comando composto é chamado de bloco. As declarações em um bloco são normalmente colocadas no início do bloco, antes

de quaisquer comandos de ação; mas as declarações podem ser misturadas com comandos de ação. Discutiremos o uso de blocos no Capítulo 3. O leitor deve evitar usar blocos até este capítulo (exceto o corpo de main, claro).

2.7 A estrutura de repetição while

Uma estrutura de repetição permite ao programador especificar que uma ação deve ser repetida enquanto alguma condição for verdadeira. O comando de pseudocódigo
Enquanto existirem mais itens em minha lista de compras
Comprar próximo item e excluí-lo da minha lista
descreve a repetição que acontece durante uma saída para compras. A condição “existirem mais itens em minha lista de compras” pode ser verdadeira ou falsa. Se ela for verdadeira, então a ação, “Comprar próximo item e excluí-lo da minha lista” é executada. Esta ação será repetidamente executada, enquanto a condição for true. O(s) comando(s) contidos na estrutura de repetição while constituem o corpo do while. O corpo da estrutura while pode ser um comando único ou um comando composto. Em algum momento, a condição se tornará false (quando o último item da lista de compras foi comprado e excluído da mesma). Neste momento, a repetição termina e o primeiro comando de pseudocódigo após a estrutura de repetição é executado.

Erro comum de programação 2.4
Não fornecer no corpo de uma estrutura while, uma ação que faça com que a condição na estrutura while se torne false em algum momento normalmente resulta em um erro chamado “laço infinito “, no qual a estrutura de repetição nunca termina de ser executada.

Erro comum de programação 2.5
Escrever a palavra-chave while com um W maiúsculo, como em While, é um erro de sintaxe (lembre-se de que C++ é uma linguagem sensível a maiúsculas e minúsculas). Todas as palavras-chave reservadas de C++, tais como while, if e else, contêm somente letras minúsculas.

Como exemplo de um while real, considere um segmento de programa projetado para achar a primeira potência de 2 maior do que 1000. Suponha que a variável produto inteira tenha sido inicializada com 2. Quando a estrutura de repetição while a seguir terminar de ser executada, produto conterá a resposta desejada:

```
int produto = 2;
while ( produto <= 1000)
produto = 2 * produto;
```

O fluxograma da Fig. 2.5 ilustra o fluxo de controle na estrutura while que corresponde à estrutura while precedente. Uma vez mais, note que (além de pequenos círculos e setas) o fluxograma contém só um retângulo e um losango. Imagine uma caixa grande cheia de estruturas while vazias que podem ser empilhadas e aninhadas com outras estruturas de controle, para formar uma implementação estruturada do fluxo de controle de um algoritmo. Os retângulos e losangos vazios são então preenchidos com ações e decisões apropriadas. O fluxograma mostra claramente a repetição. A linha de fluxo que emerge do retângulo retorna à decisão, que é testada toda vez no laço, até que se torne false. Então, a estrutura while termina e o controle passa para o próximo comando no programa.

Quando a estrutura while começa a ser executada, o valor de produto é 2. A variável produto é repetidamente multiplicada por 2, assumindo os valores 4, 8, 16, 32, 64, 128, 256, 512 e 1024, sucessivamente. Quando produto se tornar 1024, a condição da estrutura while, produto <= 1000. se torna false. Isto termina a
repetição - o valor final de produto é 1024. A execução do programa continua com o próximo comando depois do while.

2.8 Formulando algoritmos: estudo de caso 1

(repetição controlada por contador)

Para ilustrar como os algoritmos são desenvolvidos, resolveremos diversas variantes do problema de calcular a média de uma turma. Considere o seguinte problema:

Uma turma de dez estudantes resolve um teste. As notas (inteiros no intervalo de O a 100) dadas à solução deste teste estão disponíveis para você. Determine a média das notas da turma para a solução do teste.

A média da classe é igual à soma das notas dividida pelo número de estudantes. O algoritmo para resolver este problema em um computador deve receber como entrada cada uma das notas, executar o cálculo da média e imprimir o resultado.

Usemos pseudocódigo para listar as ações que devem ser executadas e para especificar a ordem em que estas ações devem ser executadas. Usamos uma repetição controlada por contador para fornecer como entrada as notas,

Fig. 2.5 Representando em fluxograma a estrutura de repetição while.

uma de cada vez. Esta técnica usa uma variável chamada de contador, para controlar o número de vezes que um conjunto de comandos será executado. Neste exemplo, a repetição termina quando o contador exceder 10. Nesta seção, apresentamos um algoritmo em pseudocódigo (Fig. 2.6) e o programa correspondente (Fig. 2.7). Na próxima seção, mostramos como algoritmos em pseudocódigo são desenvolvidos. A repetição controlada por contador é chamada, freqüentemente, de repetição definida, porque o número de repetições é conhecido antes de o laço começar a ser executado.

Note as referências no algoritmo a um total e a um contador. Um total é uma variável usada para acumular a soma de uma série de valores. Um contador é uma variável usada para contar - neste caso, contar o número de notas lidas. As variáveis que são usadas para armazenar totais devem ser normalmente inicializadas com zero antes de serem usadas em um programa; caso contrário, a soma incluirá o valor armazenado anteriormente na posição de memória do total.

As linhas 11a 14

int total, // soma das notas
gradeCounter, // número de notas digitadas
grade, // uma nota
average; // média das notas

declaram as variáveis total, gradeCounter. grade e average como sendo do tipo int. A variável grade vai armazenar o valor que o usuário fornece para o programa.

Inicialize total com zero
Inicialize contador de notas com um
Enquanto o contador de notas for menor do que ou igual a dez
Receba como entrada a próxima nota
Some a nota ao total
Some um ao contador de notas
Atribua à média da turma ao total dividido por dez
Imprima a média da turma

Fig. 2.6 Algoritmo em pseudocódigo que usa repetição controlada por contador para resolver o problema da média da turma.

```
1 // Fig. 2.7: figO2O7.cpp
2 // Programa de média da turma com repetição controlada por contador
3 #include <iostream>
4
5 using std: :cout;
6 using std: :cin;
7 using std: :endl;
8
9 intmain ()
10 {
11 int total, // soma das notas
12 gradeCounter, // número de notas digitadas
```

13 grade, 1/ uma nota
14 average; // média das notas
15
16 // fase de inicialização
17 total = 0; // limpa total
18 gradeCounter = 1; // prepara para executar o laço
Fig. 2.7 Programa em C++ e exemplo de execução para o problema de cálculo da média da turma com repetição controlada por contador (parte 1 de 2).

19
20
21
22
23
24
25
26
27
28
29
30
31
32
33 1

// fase de processamento
while (gradeCounter < 10
cout << "Forneça nota:";
cin >> grade;
total = total +grade;
gradeCounter = gradeCounter + 1;
// fase de término
average = total / 10;
cout « “A média da turma é “ «

return 0;

total = 0;
gradeCounter = 1;

Boa prática de programação 2.7 inicialize contadores e totais.
Boa prática de programação 2.8

1// repete 10 vezes
// solicita entrada de dados

// lê nota digitada
// soma nota ao total

// incrementa contador
// divisão inteira average « endi;

// indica que o programa terminou normalmente

Fig. 2.7 Programa em C++ e exemplo de execução para o problema de cálculo da média da turma com repetição controlada por contador (parte 2 de 2).
Observe que as declarações precedentes aparecem no corpo da função main. Lembre-se de que variáveis declaradas no corpo da definição de uma função são variáveis locais e podem ser usadas somente a partir da linha em que são declaradas na função até a chave à direita (}) de término da definição de função. A declaração de uma variável local em uma função deve aparecer antes que a variável seja usada nesta função.
As linhas 17 e 18

// limpa total
// prepara para executar o laço

são comandos de atribuição que inicializam total com O e gradeCounter com 1. Note que as variáveis total e gradeCounter são inicializadas antes de serem usadas em um cálculo. As variáveis contadoras são normalmente inicializadas com zero ou um, dependendo de seu uso (apresentaremos exemplos mostrando cada um destes usos). Uma variável não-inicializada contém valor lixo (também chamado de valor indefinido) - o último valor armazenado na posição de memória reservada para aquela variável.
Erro comum de programação 2.6
Se um contador ou total não é inicializado, os resultados do seu programa provavelmente serão incorretos. Este é um exemplo de erro de lógica.

Declare cada variável em uma linha separada.

Forne ça	nota:	98
Forne ça	nota:	76
Forne ça	nota:	71
Forne	nota:	87

ça		
Forneça	nota:	83
Forneça	nota:	90
Forneça	nota:	57
Forneça	nota:	79
Forneça	nota:	82
Forneça nota	:	94
A média	da turma é 81	

CAPÍTULO 2 - ESTRUTURAS DE CONTROLE 113

A linha 21
while (gradeCounter <= 10) { // repete 10 vezes
indica que a execução da estrutura while deve continuar enquanto o valor de gradeCounter for menor que ou igual a 10.
As linhas 22 e 23
cout << “Forneça nota: “; // solicita entrada de dados
cin >> grade; // lê nota digitada
correspondemte ao comando em pseudocódigo “Insira a próxima nota”. O primeiro comando exibe o prompt “Forneça nota:” na tela. O segundo comando lê o valor da nota digitada pelo usuário.
A seguir, o programa atualiza total com a nova nota digitada pelo usuário. A linha 24
total = total + grade; /1 soma nota a total
soma grade ao valor anterior de total e atribui o resultado a total.
Agora, o programa está pronto para incrementar a variável gradeCounter, para indicar que uma nova nota foi processada, e então ler a próxima nota digitada pelo usuário. A linha 25
gradeCounter = gradeCounter + 1; // incrementa contador
//soma 1 a gradeCounter, de modo que a condição na estrutura while vai, em algum momento, tornar-se falsa //e terminar o laço.
A linha 29
average = total / 10; // divisão inteira
atribui o resultado do cálculo da média à variável average. A linha 30
cout « “A média da turma é “ « average « endl;
exibe o string “A média da turma é” seguido pelo valor da variável average. Note que o cálculo da média no programa produziu um resultado inteiro. Na verdade, a soma das notas neste exemplo é 817, que, quando dividido por 10,

dá 81,7, isto é, um número com uma fração decimal. Veremos como lidar com tais números (chamados de números de ponto-flutuante) na próxima seção.
Erro comum de programação 2.7
Em um laço controlado por um contador como o contador do laço (que está sendo incrementado por um a cada repetição do laço) ao final das repetições fica com um valor que é um a mais que seu último valor legítimo (i.i., 11 no caso de “contar de 1 até 10”), usar o valor do contador em um cálculo depois do laço freqüentemente é um erro do tipo “um a mais que o esperado
Na Fig. 2.7, se na linha 29 tivéssemos usado gradeCounter em lugar de 10 para o cálculo, a saída deste programa mostraria um valor incorreto de 74.

2.9 Formulando algoritmos com refinamento top-down, passo a passo: estudo de caso 2 (repetição controlada por sentinela)

Vamos generalizar o problema da média da turma. Considere o seguinte problema:
Desenvolva um programa que calcula a média da turma e que processa um número arbitrário de notas cada vez que o programa é executado.

No primeiro exemplo de média da turma, o número de notas (10) era conhecido com antecedência. Neste exemplo, nenhuma indicação é dada de quantas notas serão digitadas. O programa deve processar um número arbitrário de notas. Como o programa pode determinar quando parar a leitura de notas? Como ele saberá quando calcular e imprimir a média da turma?
Um modo de resolver este problema é usar um valor especial, chamado de valor de sentinela (também chamado de valor sinalizador ou valor artificial), para indicar o “fim de entrada de dados”. O usuário, então, digita as notas até que todas as notas válidas foram lidas. O usuário, então, digita o valor de sentinela para indicar que a última nota foi lida. A repetição controlada por sentinela é freqüentemente chamada de repetição indefinida, porque o número de repetições não é conhecido antes de o laço começar a ser executado.
Naturalmente, o valor de sentinela deve ser escolhido de forma que não possa ser confundido com um valor aceitável fornecido como entrada. Como as notas de um teste normalmente são inteiros não-negativos, -1 é um valor de sentinela aceitável para este problema. Deste modo, uma execução do programa para

calcular a média da turma pode processar um stream de dados de entrada, tal como 95, 96, 75, 74, 89 e -1. O programa, então, computaria e imprimiria a média da turma para as notas 95, 96, 75,74 e 89 (-1 é o valor de sentinela: assim, ele não deve entrar no cálculo da média).

Erro comum de programação 2.8
Escolher um valor de sentinela que é também um valor de dados válido é um erro de lógica.
Abordamos o programa que calcula a média da turma com uma técnica chamada refinamento top-down passo a passo, uma técnica essencial para o desenvolvimento de programas bem-estruturados. Começamos com uma representação em pseudocódigo do top:
Determine a média da turma para o teste
O topo é uma única afirmação, que expõe a função global do programa. Como tal, o topo é, na realidade, uma representação completa de um programa. Infelizmente, o topo (como neste caso) raramente traz uma quantidade suficiente de detalhes para escrever o programa em C++. Assim, começamos agora o processo de refinamento. Dividimos o topo em uma série de tarefas pequenas e listamos estas tarefas na ordem em que necessitam ser executadas. Isto resulta no seguinte primeiro refinamento:
Inicializar variáveis
Receber os dados de entrada, somar e contar as notas do teste
Calcular e imprimir a média da turma
Aqui, foi usada somente a estrutura de seqüência - os passos listados devem ser executados na ordem, um depois do outro.

Observação de engenharia de software 2.4
Cada refinamento, bem como o próprio topo, é uma especificação completa do algoritmo; só varia o nível
de detalhe.

Observação de engenharia de software 2.5
Muitos programas podem ser logicamente divididos em três fases: uma fase de inicialização, que inicializa as variáveis do programa; uma fase de processamento, que recebe como entrada valores de dados e ajusta as variáveis do programa de acordo; e uma fase de finalização, que calcula e imprime os resultados finais.

A observação de engenharia de software precedente é freqüentemente de tudo que você necessita para o primeiro refinamento, seguindo o processo top-down. Para executar o próximo nível de refinamento, isto é, o segundo refinamento, comprometemo-nos com variáveis específicas. Necessitamos executar um

cálculo do total dos números, contar

quantos números foram processados, uma variável para receber o valor de cada nota como é fornecido na entrada e uma variável para guardar a média calculada. O comando de pseudocódigo
Inicializar variáveis
pode ser refinado como segue:
Inicializar total com zero
Inicializar contador com zero
Note que só as variáveis total e contador necessitam ser inicializadas antes de serem usadas; as variáveis média e nota (para a média calculada e a entrada do usuário, respectivamente) não necessitam ser inicializadas, porque seus valores serão sobrescritos à medida que são calculados ou lidos.
O comando de pseudocódigo
Receber os dados de entrada, somar e contar as notas do teste
exige uma estrutura de repetição (i.e., um laço) que sucessivamente recebe como entrada cada nota. Como não sabemos com antecedência quantas notas serão processadas, usaremos uma repetição controlada por sentinela, O usuário digitará notas válidas, uma de cada vez. Depois da última nota válida ser digitada, o usuário digitará o valor sentinela. O programa testará se foi lido o valor sentinela, após cada nota ser digitada, terminando o laço quando o valor sentinela é digitado pelo usuário, O segundo refinamento do comando de pseudocódigo precedente fica então:
Receba como entrada a primeira nota (possivelmente a sentinela)
Enquanto o usuário ainda não digitou a sentinela
Some esta nota ao total corrente
Some um ao contador de notas
Receba como entrada a próxima nota (possivelmente a sentinela)
Note que, em pseudocódigo, não usamos chaves em torno do conjunto de comandos que forma o corpo da estrutura
enquanto. Simplesmente indentamos os comandos sob o enquanto para mostrar que eles pertencem ao mesmo.
Novamente, o pseudocódigo é só uma ajuda informal para o desenvolvimento de programas.
O comando de pseudocódigo
Calcule e imprima a média da turma
pode ser refinado como segue:
Se o contador não é igual a zero
Inicialize a média com o total dividido pelo contador
imprima a média
senão

imprima “Nenhuma nota foifornecida”
Note que estamos sendo cuidadosos aqui, testando a possibilidade de divisão por zero - um erro fatal de lógica, que se não for detectado fará o programa terminar anormalmente (freqüentemente chamado “abortar”). O segundo refinamento completo do pseudocódigo para o problema da média da turma é mostrado na Fig. 2.8.

1 Erro comum de programação 2.9
Uma tentativa de dividir por zero causa um erro fatal.

Boa prática de programação 2.9
Quando executar uma divisão por uma expressão cujo valor pode ser zero, teste explicitamente esta possibilidade e trate-a apropriadamente em seu programa (tal como ao imprimir uma mensagem de erro), em vez de permitir que aconteça o erro fatal.
Na Fig. 2.6 e na Fig. 2.8, incluímos algumas linhas completamente em branco no pseudocódigo, para tornar o pseudocódigo mais legível. As linhas em branco separam estes programas em suas várias fases.
O algoritmo em pseudocódigo na Fig. 2.8 resolve a classe mais geral de problemas de cálculo da média. Este algoritmo foi desenvolvido após somente dois níveis de refinamento. As vezes, mais níveis são necessários.
Inicializar total com zero.
Inicializar contador com zero
Receba como entrada a primeira nota (possivelmente a sentinela)
Enquanto o usuário ainda não digitou a sentinela
Some esta nota ao total corrente
Some um ao contador de notas
Receba como entrada a próxima nota (possivelmente a sentinela)
Se o contador não for igual a zero
Inicialize a média com o total dividido pelo contador
Imprima a média
Senão
Imprima “Nenhuma nota foi fornecida”
Fig. 2.8 Algoritmo em pseudocódigo que usa repetição controlada por sentinela para resolver o problema da média da turma.

Observação de engenharia de software 2.6

O programador termina o processo de refinamento top-down passo a passo quando o algoritmo em pseudocódigo está especificado em detalhes suficientes para o programador ser capaz de converter o pseudocódigo para C++. Implementar o programa em C++ é então, normalmente, um processo direto.
O programa em C++ e um exemplo de execução são mostrados na Fig. 2.9. Embora somente notas de tipo inteiro sejam fornecidas, é provável que o cálculo da média produza um número com fração decimal, i.e., um número real. O tipo int não pode representar números reais. O programa introduz o tipo de dados double para tratar números com ponto decimal (também chamados de números de ponto-flutuante) e introduz um operador especial chamado de operador de coerção para forçar o cálculo da média a produzir um resultado numérico em ponto flutuante. Estes recursos são explicados em detalhes depois de o programa ser apresentado.
Neste exemplo, vemos que estruturas de controle podem ser empilhadas uma em cima da outra (em seqüência) da mesma maneira que uma criança empilha blocos de construção. A estrutura while (linhas 30 a 35) é seguida imediatamente por uma estrutura if/else (linhas 38 a 45) em seqüência. A maior parte do código neste programa é idêntica ao código na Fig. 2.7, de modo que nos concentramos, neste exemplo, nos recursos e tópicos novos.
A linha 20 declara double a variável average. Esta mudança nos permite armazenar o resultado do cálculo da média da turma como um número em ponto flutuante. A linha 24 inicializa a variável gradeCounter com 0, porque ainda não havia sido digitada nenhuma nota. Lembre-se de que este programa usa repetição controlada por sentinela. Para manter um registro preciso do número de notas digitadas, a variável gradeCounter é incrementada somente quando um valor de nota válido é digitado.
Observe que os dois comandos de entrada (linhas 28 e 34)
cin >> grade;
são precedidos por um comando de saída que solicita ao usuário os dados de entrada.


```
1 // Fig. 2.9: figo2_09.cpp
2 // Programa para cálculo da média da turma c/repetição controlada por
sentinela.
3 #include <iostream>
4
5 using std: :cout;
6 using std::cin;
7 using std: :endl;
8 using std::ios;
```

```
9
10 #include <iomanip>
11
12 using std::setprecision;
13 using std: :setiosflags;
14
15 int main ()
16
17 int total, // soma das notas
18 gradeCounter, // número de notas digitadas
19 grade; // uma nota
20 double average; // número com ponto decimal para a média
21
22 // fase de inicialização
23 total = 0;
24 gradeCounter = 0;
25
26 // fase de processamento
27 cout« << "Forneça nota ou -1 para finalizar: ";
28 cin >> grade;
29
30 while ( grade != -1
31 total = total + grade;
32 gradeCounter = gradeCounter + 1;
33 cout << “Forneça nota ou -1 para finalizar:"
34 cin >>  grade;
35
36
37 // fase de término
38 if ( gradeCounter != 0 )
39 average = static_cast< double >( total ) / gradeCounter;
40 cout << "A média da turma é “ << setprecision ( 2 )
41 « setiosflags( ios::fixed 1 ios::showpoint
42 « average « endl;
43 }
44 else
45 cout << “Nenhuma nota foi fornecida” << endl;
46
47 return 0; // indica que o programa terminou com sucesso
48
```

Fig. 2.9 Programa em C++ e exemplo de execução para o problema de cálculo da média da turma com

repetição controlada por sentinela (parte 1 de 2).

Forneça	nota	ou	-1	para	finalizar:	75	1
Forneça	nota	ou	-1	para	finalizar:	94	
Forneça	nota	ou	-1	para	finalizar:	97	
Forneça	nota	ou	-1	para	finalizar:	88	1


```
Forneça nota ou -1 para finalizar: 70
Forneça nota ou -1 para finalizar: 64
Forneça nota ou -1 para finalizar: 83
Forneça nota ou -1 para finalizar: 89
Forneça nota ou -1 para finalizar: -1
A média da turma é 82,50
```

Fig. 2.9 Programa em C++ e exemplo de execução para o problema de cálculo da média da turma com repetição controlada por sentinela (parte 2 de 2).

Boa prática de programação 2. 10
Solicite explicitamente ao usuário cada entrada pelo teclado, O lembrete deve indicar a forma da entrada e quaisquer valores de entrada especiais que devam ser fornecidos (tal como o valor de sentinela que o usuário deve digitar para terminar um laço).

Boa prática de programação 2. 11
Em um laço controlado por sentinela, os lembretes solicitando entrada de dados deveriam lembrar explicitamente ao usuário qual é o valor de sentinela.
Estude a diferença entre a lógica do programa para repetição controlada por sentinela, comparada com aquela para a repetição controlada por contador na Fig. 2.7. Na repetição controlada por contador, lemos um valor digitado pelo usuário durante cada iteração da estrutura while, pelo número especificado de iterações. Na repetição controlada por sentinela, lemos um valor (linha 28) antes que o programa atinja a estrutura while. Este valor é usado para determinar se o fluxo de controle do programa deve entrar no corpo da estrutura while. Se a condição da estrutura while é false (falsa) (i.e., o usuário já digitou a sentinela), o corpo da estrutura while não é executado (nenhuma nota foi fornecida). Se, por outro lado, a condição é true (verdadeira), o corpo começa a ser executado e o valor digitado pelo usuário é processado (somado a total, neste exemplo). Após o valor ser processado, é lido o próximo valor digitado pelo usuário antes do fim

do corpo da estrutura while. Quando a chave à direita ( )) de fechamento do corpo é atingida na linha 35, a execução continua com o próximo teste da condição da estrutura while. usando o novo valor recém-digitado pelo usuário para determinar se o corpo da estrutura while deve ser executado novamente. Observe que o próximo valor digitado pelo usuário é sempre lido imediatamente antes de ser avaliada a condição da estrutura while. Isto nos permite determinar se o valor recém-digitado pelo usuário é o valor sentinela, antes de este valor ser processado (i.e., somado a total). Se o valor digitado é o valor sentinela, a estrutura while termina e o valor não é somado a total.
Note o comando composto no laço while na Fig 2.9. Sem as chaves, os últimos três comandos no corpo do laço cairiam fora do laço, fazendo com que o computador interpretasse incorretamente este código, como segue
while (grade -1
total = total + grade;
gradeCounter = gradecounter + 1;
cout << “Forneça nota ou -1 para finalizar:
cin » grade;
Isto causaria um laço infinito se o usuário não fornecesse como entrada -1 para a primeira nota.
Cálculos de médias nem sempre produzem valores inteiros. Freqüentemente, uma média é um valor que contém uma parte fracionária, tal como 7,2 ou -93,5. Estes valores são chamados números de ponto flutuante e são representados pelos tipos de dados float e double. Uma variável do tipo double pode armazenar um valor de magnitude muito maior ou com maior precisão do que float. Por esta razão, tendemos a usar o tipo double em vez do tipo float para representar valores em ponto flutuante em nossos programas. Constantes (como 1000.0 e 05) são tratadas por C++ como sendo do tipo double.

(parei aqui)

A variável average é declarada com o tipo double para capturar o resultado fracionário de nosso cálculo. Porém, o resultado do cálculo total / counter é um inteiro, porque total e counter são ambas variáveis inteiras. Dividir dois inteiros resulta em uma divisão inteira em que qualquer parte fracionária do cálculo é perdida (i.e., truncada). Como o cálculo é executado primeiro, a parte fracionária é perdida antes de o resultado ser atribuído para average. Para produzir um cálculo de ponto flutuante com valores inteiros, devemos criar valores temporários que sejam números de ponto flutuante para o cálculo. C++ fornece o operador unário de coerção para realizar esta tarefa. O comando
average = static_cast< double > (total) / gradeCounter;
inclui o operador de coerção staticcast < double > ( ), que cria uma cópia de ponto flutuante temporária de seu operando entre parênteses - total. Usar um operador de coerção desta maneira é chamado de conversão explícita. O valor

armazenado em total ainda é um inteiro. O cálculo agora consiste em um valor de ponto flutuante (a versão double temporária de total) dividido pelo inteiro counter.
O compilador de C÷+ só sabe como calcular expressões em que os tipos de dados dos operandos são idênticos. Para assegurar que os operandos sejam do mesmo tipo, o compilador executa uma operação chamada promoção (também chamada conversão implícita) sobre operandos selecionados. Por exemplo, em uma expressão contendo os tipos de dados int e double. operandos int são promovidos a double. Em nosso exemplo, depois de counter ser promovido a double, o cálculo é executado e o resultado da divisão em ponto flutuante é atribuído a average. Mais tarde neste capítulo, discutimos todos os tipos de dados padrão e sua ordem de promoção.
Os operadores de coerção estão disponíveis para qualquer tipo de dados. O operador static cast é for-
a mado seguindo-se a palavra-chave static_cast pelos sinais de menor e maior (< e >) em volta do nome de um
a tipo de dados. O operador de coerção é um operador unário, i.e., um operador que aceita só um operando. No
a Capítulo 1, estudamos os operadores aritméticos binários. C++ também suporta versões unárias dos operadores
a mais (+) e menos (-), de modo que o programador possa escrever expressões como -7 ou +5. Operadores de
a coerção têm precedência mais alta que outros operadores unários, tais como + unários e - unários. Esta precedência
é mais alta que a dos operadores multiplicativos / e %, e mais baixa que a dos parênteses. Indicamos o operador de coerção com a notação static_cast < tipo> ( ) em nossos quadros de precedência.
O recursos de formatação da Fig. 2.9 são explicados em profundidade no Capítulo 11 e discutidos aqui brevemente. A chamada setprecision (2) no comando de saída
cout « ‘A média da turma é “ « setprecision(2
« setiosflags(ios::fixed 1 ios::showpoint
« average « eridi;
indica que a variável double average deve ser impressa com precisão de dois dígitos à direita da casa decimal (por exemplo, 92, 37). Esta chamada é denominada manipulador de streamparametrizado. Os programas que usam estas chamadas devem conter a diretiva do pré-processador
#include <iomanip>
As linhas II e 12 especificam os identificadores definidos no arquivo de cabeçalho <iomanip> que são usados neste programa. Note que endi é um manipulador de stream não-parametrizado e não exige o arquivo cabeçalho iomanip. Se a precisão não for especificada, os valores de ponto flutuante são normalmente exibidos/impressos com seis dígitos de precisão (i.e., a precisão default), embora logo vejamos uma exceção a esta regra.
O manipulador de stream setiosflags (ios: : fixed 1 ios: : showpoint) no comando precedente define duas opções de formatação para saída, i.e., ios: fixed e ios:

showpoint. O caractere barra vertical
1) separa opções múltiplas em uma chamada a setiosflags (explicaremos a notação em profundidade no Capítulo 16). [Nota: embora vírgulas (,) sejam freqüentemente usadas para separar itens de uma lista, elas não podem ser usadas com o manipulador de stream setiosflags: caso contrário, somente a última opção na lista vai ser ajustadaj. A opção ios: fixed faz com que um valor de ponto flutuante seja exibido/impresso no chamado formato de ponto fixo (em oposição à notação cientifica que discutiremos no Capítulo 11). A opção ios: : showpoint força a impressão da casa decimal e dos zeros não-significativos, ainda que o valor seja um

número inteiro, tal como 88,00. Sem a opção ios: : showpoint, tal valor seria exibido/impresso em C++ como 88, sem os zeros não-significativos e sem a casa decimal. Quando a formatação precedente é usada em um programa, o valor de saída é arredondado para o número indicado de posições decimais, embora o valor na memória permaneça inalterado. Por exemplo, os valores 87,945 e 67,543 são exibidos/impressos como 87,95 e 67,54, respectivamente.
Erro comum de programação 2. 10
Usar números de ponto flutuante de uma maneira que assume que eles são representados precisamente pode levar a resultados incorretos. Os números de ponto flutuante são representados somente aproximadamente pela maioria dos computadores.
Boa prática de programação 2.12
Não compare valores em ponto flutuante quanto à igualdade ou desigualdade. Basta testar se o valor
absoluto da diferença é menor que um valor pequeno especificado.
Apesar do fato de números de ponto flutuante não serem sempre “100% precisos”, eles tèm numerosas aplicações. Por exemplo, quando dizemos que “a temperatura normal do corpo é 36,7 °C”, não necessitamos da precisão de um grande número de dígitos. Quando olhamos a temperatura em um termômetro e lemos 36,7, ela realmente pode ser 36.69994732 10643. A questão a destacar aqui é que chamar este número simplesmente de 36,7 é suficiente para a maioria das aplicações.
Outro modo de surgirem números de ponto flutuante é através da divisão. Quando dividimos 10 por 3, o resultado é 3,3333333... com a seqüência de 3s repetindo-se infinitamente. O computador aloca uma quantidade fixa de espaço para guardar esse valor; assim, obviamente, o valor armazenado em ponto flutuante só pode ser uma aproximação.
2.10 Formulando algoritmos com refinamento top-down, passo a passo: estudo de caso 3 (estruturas de controle aninhadas)
Iremos estudar outro problema completo. Uma vez mais, formularemos o algoritmo usando pseudocódigo e o refinamento top-down passo a passo, e escreveremos um programa correspondente em C++. Vimos que estruturas de controle poderem ser empilhadas uma em cima da outra (em seqüência) da mesma maneira que uma criança constrói empilhando blocos. Neste estudo de

caso, veremos o único outro modo estruturado pelo qual estruturas de controle podem ser conectadas em C++, i.e., através do aninhamento de uma estrutura de controle dentro de outra.
Considere a seguinte definição de problema:
Uma escola oferece um curso que prepara estudantes para o exame estadual de licenciamento de corretores de imóveis. No último ano, vários dos estudantes que completaram este curso fizeram o exame de licenciamento. Naturalmente, a escola quer saber quão bem seus estudantes fbram no exame. Foi pedido a você para escrever um programa para resumir os resultados. Você recebeu uma lista destes 10 estudantes. Após cada nome está escrito um 1 se o estudante passou no exame ou um 2 se o estudante foi reprovado.
Seu programa deve analisar os resultados do exame como segue.'
1. Forneça como entrada cada resultado de teste (i.e., um 1 ou um 2). Exiba a mensagem "Digite o resultado" na tela cada vez que o programa pede outro resultado de teste.
2. Conte o número de resultados de teste de cada tipo.
3. Exiba um resumo dos resultados de teste indicando o número de estudantes que passaram e o número de estudantes que foram reprovados.
4. Se mais de 8 estudantes passaram no exame, imprima a mensagem "Aumente o preço do curso"

Depois de ler a definição do problema cuidadosamente, fazemos as seguintes observações:

1. O programa deve processar 10 resultados de teste. Um laço controlado por contador será usado.
2. Cada resultado de teste é um número - um 1 ou um 2. Cada vez que o programa lê um resultado de teste, o programa deve determinar se o número é um 1 ou um 2. Em nosso algoritmo, testamos se é um 1. Se o número não for um 1, assumimos que ele é um 2 (um exercício no fim do capítulo discute as conseqüências desta suposição).
3. Dois contadores são usados - um para contar o número de estudantes que passaram no exame e outro para contar o número de estudantes que foram reprovados no exame.
4. Depois de o programa ter processado todos os resultados, deve testar se mais de 8 estudantes passaram no exame.
Iremos proceder o refinamento top-down passo a passo. Começamos com uma representação em pseudocódigo do topo:
Analise os resultados do exame e decida se o preço deve ser aumentado
Uma vez mais, é importante enfatizar que o topo é uma representação completa do programa, mas provavelmente vários refinamentos serão necessários antes de o pseudocódigo poder ser naturalmente convertido em um programa em C++.
Nosso primeiro refinamento é
inicialize varicíveis

Obtenha as dez notas do teste e conte as aprovações e reprovações
Imprima um resumo dos resultados do teste e decida se o preço do curso deve ser aumentado
Aqui, também, embora tenhamos uma representação completa de todo o programa, é necessário um refinamento adicional. Agora definimos variáveis específicas. Serão necessários contadores para registrar as aprovações e reprovações, um contador será usado para controlar o laço de processamento e será necessária uma variável para armazenar a entrada fornecida pelo usuário.
O comando de pseudocódigo
Inicialize variáveis
pode ser refinado como segue:
Inicialize aprovações com zero
Inicialize reprovações com zero
Inicialize contador de estudantes com um
Note que só os contadores e totais são inicializados. O comando de pseudocódigo
Forneça como entrada as dez notas do teste e conte as aprovações e reprovações
exige um laço, que sucessivamente recebe como entrada o resultado de cada teste. Aqui sabemos com antecedência que existem precisamente dez resultados de teste; assim, um laço controlado por contador é adequado. Dentro do laço (i.e., aninhada dentro do laço) uma estrutura de seleção dupla determinará se cada resultado de teste é uma aprovação ou uma reprovação e, conseqüentemente, incrementará o contador apropriado, O refinamento do comando de pseudocódigo precedente fica então
Enquanto o contador de estudantes for menor que ou igual a dez
Leia o próximo resultado do teste
Se o estudante foi aprovado
Some um a aprovações

senão
Some um a reprovações
Some um ao contador de estudantes
Note o uso de linhas em branco para separar a estrutura de controle se/senão para melhorar a legibilidade do proglama. O comando de pseudocódigo
imprima um resumo dos resultados do teste e decida se o preço do curso deve ser aumentado
pode ser refinado como segue:
imprima o número de aprovações
imprima o número de reprovações
Se mais de oito estudantes foram aprovados
imprima “Aumente o preço do curso”
O segundo refinamento completo aparece na Fig. 2.10. Note que linhas em branco lambeni sao usadas para separar a estrutura while, para melhorar a legibilidade do programa.

inicialize aprovações com zero
inicialize reprovações com zero
inicialize contador de estudantes com um
Enquanto contador de estudantes for menor que ou igual a dez
Leia o próximo resultado do teste
Se o estudante foi aprovado
Some um a aprovações
senão
Some um a reprovações
Some um ao contador de estudantes
imprima o número de aprovações
imprima o número de reprovações
Se mais de oito estudantes foram aprovados
imprima “Aumente o preço do curso”

Fig. 2.10 Pseucódigo para o problema dos resultados do teste.

Este pseucódigo está agora suficientemente refinado para ser convertido em C++. O programa em C++ e dois exemplos de execuções são mostrados na Fig. 2.11.

Fig. 2.11 Programa em C++ e exemplos de execuções para o problema dos resultados do teste (parte 1
de 2).

```
1  // Fig. 2.11: fig02_11.cpp
2  // Análise dos resultados do teste
3  #include <iostream>
4
5  using std: :cout;
6  using std::cin;
7  using std::endl;
8
```



```
9
10
11
```

12
13
14
15

16
17
18
19
20

21
22
23
24
25
26
27

28
29
É 30
31
32

33

34
35

36
37
38

int main
// inicializa as variáveis nas declarações

int passes = 0,
failures = 0,
studentCounter = 1,
result;

1/’ numero de aprovações
II número de reprovações

```
II contador de estudantes
II um resultado do teste

II processa 10 estudantes; ciclo controlado por contador while ( studentCounter
<= 10
cout « “Forneça resultado (l=aprovado,2=reprovado)
cm » result;
if ( result == 1 ) II if/else aninhado no while
passes = passes + 1;
else
failures = failures + 1;
studentCounter = studentCounter + 1;
// fase de término
cout« Aprovados “ « passes « endi;
cout« “Reprovados “ « failures « endl;
if ( passes > 8
cout « “Aumente o preço do curso “ « endl;
return 0; // término normal
```

Fg. 2.11 Programa em C++ e exemplos de execuções para o problema dos resultados do teste (parte 2 de 2).

Forneça resultado (1=aprovado, 2=reprovado)		:1
Forneça resultado (1=aprovado, 2=reprovado)		: 1
Forneça resultado (1=aprovado , 2reprovado)		:1
Forneça resultado	(l=aprovado, 2=reprovado)	: 1
Forneça resultado (1=aprovado, 2=reprovado)		:2
Forneça resultado	(1=aprovado, 2=reprovado)	:1
Forneça resultado (	1=aprovado, 2=reprovado)	: 1
Forneça resultado	(1=aprovado , 2=reprovado)	:1
Forneça resultado	(1=aprovado, 2=reprovado)	:1
Forneça resultado (l=aprovado , 2reprovado) Aprovados 9 Reprovados 1 Aumente o preço do curso		:1

Forneça resultado (1=aprovado , 2reprovado)	:1
Forneça resultado (l=aprovado, 2=reprovado)	:2
Forneça resultado (1=aprovado, 2reprovado)	: 2
Forneça resultado (1=aprovado , 2=reprovado)	:1
Forneça resultado (1=aprovado, 2reprovado)	:1
Forneça resultado (1=aprovado, 2=reprovado)	:1
Forneça resultado (l=aprovado , 2=reprovado)	:2
Forneça resultado (1=aprovado, 2=reprovado)	:1
Forneça resultado (l=aprovado , 2reprovado)	:1
Forneça resultado (1=aprovado , 2=reprovado)	:2
Aprovados 6	
Reprovados 4	

As linhas 12 a 15
int passes = O, II número de aprovações
failures = O, II número de reprovações
studentCounter = 1, II contador de estudantes
result; II um resultado do teste
declaram as variáveis usadas em main para processar os resultados do exame. Note que aproveitamos um recurso de C++ que permite que a inicialização de variáveis seja incorporada às suas declarações (o é atribuído a passes. o é atribuído a failures e 1 é atribuído a studentCounter). Programas com laços podem exigir inicializações no princípio de cada repetição; tais inicializações normalmente aconteceriam em comandos de atribuição.

Boa prática de programação 2.13
inicializar as variáveis quando elas são declaradas ajuda o programador a evitar os problemas de dados
não-inicializados.

Observação de engenharia de software 2.7
A experiência mostra que a parte mais difTcil da solução de um problema em um computador está no
desenvolvimento do algoritmo para a solução. Uma vez que um algoritmo correto tenha sido especificado,
o processo de produzir um programa em C+ + que funciona, a partir do algoritmo, é normalmente direto.

Observação de engenharia de software 2.8
Muitos programadores experientes escrevem programas sem usar ferramentas de desenvolvimento de programas como pseudocódigo. Estes programadores pensam que sua meta essencial é resolver o problema em um computador e que escrever pseudocódigo somente atrasa a produção de resultado final.
Embora isto possa funcionar para problemas simples efamiliares, pode levar a sérios erros e atrasos em projetos grandes e complexos.
2.11 Operadores de atribuição
C++ oferece vários operadores de atribuição para abreviar as expressões de atribuição. Por exemplo, o comando c = c + 3;
pode ser abreviado com o operador atribuição com adição += como
c += 3;
O operador + soma o valor da expressão à direita do operador ao valor da variável à esquerda do operador e armazena o resultado na variável à esquerda do operador. Qualquer comando da forma
variável = variável expressão operador;
onde operador é um dos operadores binários +, -, *, 1, ou % (ou outros que discutiremos mais tarde no texto), pode ser escrito na forma
variável operador = expressão;
Deste modo, a atribuição c += 3 soma 3 a c. A Fig. 2.12 mostra os operadores de atribuição aritméticos, exemplos de expressões usando estes operadores e explicações.
Dica de desempenho 2.3
_____ Os programadores podem escrever programas um pouco mais rápidos e os compiladores podem compilar programas um pouco mais rapidamente quando forem usados os operadores de atribuição “abreviados “.

Alguns compiladores geram código que é executado mais rápido quando são usados operadores de atribuição “abreviados”.

Fig. 2.12 Operadores aritméticos de atribuição
1 Dica de desempenho 2.4
Muitas das dicas de desempenho que mencionamos neste texto resultam em melhorias pequenas; assim, o
leitor pode ficar tentado a ignorá-las. Freqüentemente, é obtida uma melhoria de desempenho significativa
quando uma melhoria, supostamente pequena, é introduzida em um laço que é repetido muitas vezes.
2.12 Operadores de incremento e decremento
C++ também fornece o operador unário de incremento ++ e o operador unário de decremento --, os quais
estão resumidos na Fig. 2.13. Se uma variável c é incrementada por 1, o operador de incremento (++) pode
ser usado em vez das expressões e = c+1 ou c += 1. Se um operador de

incremento ou decremento é - colocado antes de uma variável, é chamado de operador de pré-incremento ou pré-decremento, respectiva-
mente. Se um operador de incremento ou decremento é colocado depois de uma variável, é chamado de
operador de pós-incremento ou de pós-decremento, respectivamente. Pré-incrementar (pré-decrementar) uma
variável faz com que a variável seja incrementada (decrementada) por 1, sendo o novo valor da variável
usado na expressão em que ela aparece. Pós-incrementar (pós-decrementar) uma variável faz com que o _____ valor atual da variável seja primeiro usado na expressão em que ela aparece, sendo então, após, o valor da
variável incrementado (decrementado) por 1.

Fig. 2.13 Os operadores de incremento e decremento.

Operador de atribuição	Expressão exemplo	Explicação	Atribui
Assuma:intc= 3,	d=5, e= 4, f= 6, g= 12;		
+	c+7	c =c+7	10 para c
-	d-=4	d =d-4	1 parad
*=	e*=5	e =e*5	20 para e
/	f/3	f f/3	2 paraf
%=	g%=9	g =g%9	3 parag

Operador	Chamado	Exemplo de expressão	Explicação
++	pré-incremento	++a	Incrementa a por 1 e então usa o novo valor de a na expressão em que a está.
++	pós-incremento	a++	Usa o valor corrente de a na expressão em que a está e então incrementa a por 1.
- -	pré-decremento	--b	Decrementa b por 1 e então usa o novo valor de b na expressão em que b está.
- -	pós-decremento	-	Usa o valor corrente de b na expressão em que b está e então decrementa b por 1.

O programa da Fig. 2.14 demonstra a diferença entre a versão de pré-incremento e a versão de pós-incremento do operador ++. Pós-incrementar a variável e faz com que a mesma seja incrementada depois de ser usada no comando de impressão. Pré-incrementar a variável c faz com que a mesma seja incrementada antes de ser usada no comando de impressão.
O programa exibe o valor de c antes e depois que o operador ++ é usado. O operador de decremento (- -) funciona de maneira similar.
Boa prática de programação 2.14
Operadores unários deveriam ser colocados próximos a seus operandos, sem espaços intermediários.
Os três comandos de atribuição na Fig 2.11
passes = passes + 1;
failures = failures + 1;
studentCounter = studentCounter + 1;

```
1 II Fig. 2.14: figo2_14.cpp
2 II Pré-incrementando e pós-incrementando
3 #include <iostream>
4
5 using std: :cout;
6 using std::endl;
7
8 int main()
9
10 int c;
11
12 c=5;
13 cout « c « endi; II imprime 5
14 cout « c++ « endi; II imprime 5 e então pós-incrementa
15 cout « c « endi « endi; II imprime 6
16
17 c=5;
18 cout « c « endi; II imprime 5
19 cout « ++c « endi; II pré-incrementa e então imprime 6
20 cout « e « endi; II imprime 6
21
22 return 0; II término normal
23 }
```

Fig. 2.14 A diferença entre pré-incrementar e pós-incrementar.
podem ser escritos mais concisamente, usando operadores de atribuição, como
passes += 1;
failures += 1;
studentCounter += 1;

5	
5	
6	
5	
:	1

com operadores de pré-incremento como
++passes;
++failures;
++studentCounter;
ou com operadores de pós-incremento como
passes++;
failures++;
studentCounter++;
Note que, quando incrementarmos ou decrementarmos uma variável sózinha em um comando, as formas de pré- incremento e pós-incremento têm o mesmo efeito e as formas de pré-decremento e pós-decremento têm o mesmo efeito. Só quando uma variável aparece no contexto de uma expressão maior é que pré-incrementá-la e pós-incrementá-la têm efeitos diferentes (e, semelhantemente, para pré-decrementar e pós-decrementar). Além disso, o pré- incremento e o pré-decremento operam de forma um pouco mais rápida do que pós-incremento e pós-decremento.
Por ora, só um nome simples de variável pode ser usado como o operando de um operador de incremento ou
decremento (veremos que estes operadores podem ser usados com os chamados Ivalues).
Erro comum de programação 2. 11
Tentar usar o operador de incremento ou decremento em uma expressão que não seja um simples nome de
variável, por exemplo, escrever ++ (x + 1), é um erro de sintaxe.
A Fig. 2. 15 mostra a precedência e associatividade dos operadores introduzidos até aqui. Os operadores são mostrados de cima para baixo em ordem decrescente de precedência. A segunda coluna descreve a associatividade dos operadores em cada nível de precedência. Note que o operador condicional (? :), os operadores unários incremento (++) e decremento (--), mais (+), menos (-) e de coerção, e os operadores de atribuição =, +=, -=, =, 1= e %= se associam da direita para a esquerda. Todos os outros operadores no quadro de precedência de operadores da Fig. 2.15 se associam da esquerda para a direita. A terceira coluna nomeia os vários grupos de operadores.
2.13 Aspectos essenciais da repetição controlada por contador
A repetição controlada por contador exige:
o nome de uma variável de controle (ou contador de laço);
2. o valor inicial da variável de controle;
3. a condição que testa o valor final da variável de controle (i.e., se o laço deve continuar);

4. o incremento (ou decremento) pelo qual a variável de controle é modificada cada vez em uma execução do laço;

f

Fig. 2.15 Precedência dos operadores encontrados até agora no texto.

Operadores				Associatividade	Tipo
O				da esquerda para a direita	parênteses
++	--	staticcast<type> (	)	da esquerda para a direita	unários (pós-fixo)
++	--	+ -		da direita para a esquerda	unários (prefixo)
*	/	%		da esquerda para a direita	multiplicativos
+	-			da esquerda para a direita	aditivos
«	»			da esquerda para a direita	inserção/extração
<	<=	> >=		da esquerda para a direita	relacionais
==	=			da esquerda para a direita	igualdade
?:				da direita para a esquerda	condicional
	+=	-= *= /= %=		da direita para a esquerda	atribuição
,				da esquerda para a direita	vírgula


```
1 II Fig. 2.16: figO2_16.cpp
2 II Repetição controlada por contador
3 #include <iostream>

5 using std::cout;
6 using std::endl;

int main()

10 int counter = 1;

11
12
```

13
14
15
16
17
18

1

2
3
4
5
6
7
8

while (counter <= 10) { cout « counter « endl;
++counter;

}

return O;

Fig. 2.16 Repetição controlada por contador.

A declaração e inicialização de counter também podiam ter sido realizadas com os comandos
int counter;
counter = 1;
Usamos ambos os métodos para inicializar variáveis.
O comando
++counter;

Considere o programa simples mostrado na Fig. 2.16, que imprime os números de 1 até 10. A declaração int contador = 1;
nomeia a variável de controle (contador), declara a mesma como do tipo inteiro, reserva espaço para ela n memória e lhe atribui um valor inicial de 1.
Declarações que exigem inicialização são, na realidade, comandos executáveis. Em C++, é mais correto chamar uma declaração que também reserva memória - como faz a declaração precedente - uma definição.

7

8
9

// inicialização
II condição de repetição
// incremento

1

9

10

incrementa o contador de laço por 1 toda vez que o laço é executado. A condição de continuação do laço na estrutura while testa se o valor da variável de controle é menor que ou igual a 10 (o último valor para o qual a condição é

true). Note que o corpo deste while é executado até quando a variável de controle é 10. O laço termina quando a variável de controle excede 10 (i.e., counter se torna 11).

O programa na Fig. 2.16 pode ficar mais conciso inicializando counter com O e substituindo a estrutura while por

```
while ( ++counter <= 10
cout « counter « endl;
```

Este código economiza um comando, porque o incremento é feito diretamente na condição do while antes de a condição ser testada. Além disso, este código elimina as chaves em torno do corpo do while porque o while agora contém só um comando. A codificação condensada dessa forma requer alguma prática e pode levar a programas que são mais difíceis de depurar, modificar e manter.

Erro comum de programação 2.12

Como os valores em ponto flutuante podem ser aproximados, controlar laços com variáveis de ponto
flutuante pode resultar em valores imprecisos do contador e testes de término inexatos.

Boa prática de programação 2.15

Controle laços controlados por contadores com valores inteiros.

Boa prática de programação 2.16

Indente os comandos no corpo de cada estrutura de controle.

Boa prática de programação 2.17

Coloque uma linha em branco antes e depois de cada estrutura de controle, para destacá-la no programa.

Boa prática de programação 2.18

Muitos níveis de aninhamento podem tornar um programa difícil de se entender Como regra geral, tente
evitar usar mais de três níveis de indentação.
Boa prática de programação 2.19
O espaçamento vertical acima e abaixo de estruturas de controle e a indentação dos corpos das estruturas de controle dentro dos cabe çalhos dessas estruturas dão aos programas uma aparência bidimensional que melhora muito sua legibilidade.
2.14 A estrutura de repetição for
A estrutura de repetição for trata todos os detalhes da repetição controlada por contador. Para ilustrar o poder do for, iremos reescrever o programa da Fig. 2.16. O resultado é mostrado na Fig. 2.17. O programa opera como segue. Quando a estrutura for começa a ser executada, a variável de controle counter é declarada e inicializada com 1. Então, é verificada a condição de cofflinuação do laço, counter <= 10. Como o valor inicial de counter é 1, a condição é satisfeita; assim, o comando do corpo imprime o valor de counter, 1.e., 1. A variável de controle counter é então incrementada na expressão counter++ e o laço começa novamente com o teste de continuação do laço. Como a variável de controle agora é igual a 2, o valor final não é excedido e assim o programa executa novamente o comando do corpo. Este processo continua até que a variável de controle counter seja incrementada para 11. isto faz com que o teste de continuação do laço não seja satisfeito e a repetição termine. O programa continua, executando o primeiro comando depois da estrutura for (neste caso, o comando return no fim do programa).


```
1 II Fig. 2.17: figo2_17.cpp O
2 // Repetição controlada por contador com a estrutura for
3 #include <iostream>
4
5 using std: :cout;
6 using std::endl;
7
8 intmain ()
10 // inicializaçào, condição de repetição e incremento
11 1/ estão todas incluidas no cabeçalho da estrutura for.
12
13 for ( ínt counter 1; counter <= 10; counter++
14 cout « counter « endi;
15
16 return 0;
l7}
```

Fig. 2.17 Repetição controlada por contador com a estrutura for.
A Fig. 2.18 examina mais de perto a estrutura for da Fig. 2.17. Note que a estrutura for “faz tudo” - especifica cada um dos itens necessários para uma

repetição controlada por contador com uma variável de controle. Se existir mais de um comando no corpo do for, são necessárias chaves para definir o corpo do laço.
palavra- Nome da Valor final
chave variável de da variável
for controle de controle
for (int counter = 1; counter <= 7; ++counter
Valor inicial da Condição de Incremento
variável de continuação da variável
controle do laço de controle
Fig. 2.18 Componentes de um cabeçalho for típico.
Note que a Fig. 2.17 usa a condição de continuação de laço counter <= 10. Se o programador escrevesse, incorretamente, counter < 10, então o ciclo seria executado somente 9 vezes. Este é um erro de lógica comum, chamado saindo do laço uma iteração antes (off-by-one errar).
Erro comum de programação 2.13
Usar uni operador relacional incorreto ou usar um valor final incorreto de m contador de laço na condição de um while ou estrutura for pode causar erro do tipo “sair uma iteração antes
Boa prática de programação 2.20
Usar o valor final na condição de um while ou estrutura for e usar o operador relacional < ajuda a evitar erros “sair uma iteração antes “. Para um laço usado para imprimir os valores 1 a 10, por exemplo, a condição de continuação de laço deveria ser counter <= 10, em vez de counter < 10 (o que é um erro “sair uma iteração antes”) ou counter < 11 (que é, no entanto, correto). Muitos programadores, entretanto, preferem a chamada contagem baseada em zero - na qual, para contar 10 vezes o laço, coun- ter seria inicíalizado com zero e o teste de continuação do laço seria counter < 10.

O formato geral da estrutura for é
for (initialization; loopContinuationTest; increment)
statement
onde initialization inicializa a variável de controle do laço, loopContinuationTest é a condição de continuação do laço e increment incrementa a variável de controle. Na maioria dos casos, a estrutura for pode ser representada com uma estrutura whiJ.e equivalente, como segue:
initialization;
while (loopContinuationTest)
statement
increment;
Existe uma exceção a esta regra que discutiremos na Seção 2.18.
Se inicialization (a seção de inicialização) no cabeçalho da estrutura for define a variável de controle (i.e., o
tipo da variável de controle é especificado antes do nome da variável), a variável de controle só pode ser usada no ifica corpo da estrutura for, i.e., o valor da

variável de controle será desconhecido fora da estrutura for. Este uso restrito :istir do nome da variável controle é conhecido como escopo da variável, O escopo de uma variável define onde ela pode ser usada em um programa. O escopo é discutido em detalhes no Capítulo 3, “Funções”.

Erro comum de programação 2.14

Quando a variável de controle de uma estrutura for é definida na seção de inicialização do cabeçalho da
estrutura for, usar a variável de controle após o corpo da estrutura é um erro de sintaxe.

Dica de portabilidade 2.1

No padrão C++, o escopo da variável de controle declarada na seção de inicialização de uma estrutura for é diferente do escopo em compiladores de C+ + mais antigos. O código C+ + criado com compiladores C+ + antigos pode não funcionar quando compilado com compiladores compatíveis com o padrão C+ +. Duas estratégias de programação defensiva que podem ser usadas para prevenir este problema são:
defina variáveis de controle com nomes diferentes em cada estrutura for, ou, se você preferir usar o mesmo nome para a variável de controle em várias estruturas for, defina a variável de controle fora e antes do primeiro laço for.
Às vezes, as expressões de inicialization e increment são listas de expressões separadas por vírgulas. As vírgulas, tal
or- ‘ como usadas aqui, são operadores vírgula que garantem que listas de expressões sejam calculadas da esquerda para
a direita. O operador vírgula tem a precedência mais baixa de todos os operadores em C++. O valor e tipo de uma
lista de expressões separadas por vírgulas é o valor e tipo da expressão mais à direita na lista. O operador vírgula é
mais freqüentemente usado em estruturas for. Sua aplicação primária é habilitar o programador a usar expressões
de inicialização múltiplas e/ou expressões de incremento múltiplas. Por exemplo, pode haver várias variáveis de
* controle em uma única estrutura for que devem ser inicializadas e incrementadas.

Boa prática de programação 2.21

- Coloque somente expressões envolvendo as variáveis de controle nas seções de inicialização e incremento
a a de uma estrutura for. As manipulações de outras variáveis devem aparecer ou em qualquer lugar antes ilo do laço (se elas são executadas só uma vez, como em comandos de inicialização) ou no corpo do laço (se
iuna vez por iteração, como comandos de incremento ou decremento).
As três expressões na estrutura for são opcionais. Se a expressão loopContinuationTest é omitida, C++ assume que a condição de continuação do laço é sempre verdadeira, criando deste modo um laço infinito. Alguém poderia omitir inicialization se a variável de controle fosse inicializada em outro lugar do programa. A expressão increment pode

ser omitida se o incremento for calculado por comandos no corpo da estrutura for, ou se nenhum incremento for necessário. A expressão de incremento na estrutura for atua como um comando isolado no fim do corpo do for. Portanto, as expressões
counter = counter + 1
couriter += 1
++counter
counter++
são todas equivalentes à parte de incremento da estrutura for. Muitos programadores preferem a forma coun- ter++, porque o incremento ocorre depois do corpo do laço ser executado. A forma de pós-incremento parece, portanto, mais natural. Como a variável que está sendo incrementada aqui não aparece em uma expressão, tanto o pré-incremento como o pós-incremento têm o mesmo efeito. Os dois ponto-e-vírgulas na estrutura for são obrigatórios.
Erro comum de programação 2.15
Usar vírgulas em vez dos dois ponto-e-vírgulas exigidos no cabeçalho de um for é um erm de sintaxe.
Erro de programação comum 2.16
Colocar um ponto-e-vírgula imediatamente à direita do parêntese à direita do cabeçalho de um for torna
o corpo daquela estrutura for um comando vazio. Isto, normalmente, é um erro de lógica.
Observação de engenharia de software 2.9
Colocar um ponto-e-vírgula logo depois do cabeçalho de um for é às vezes usado para criar um laço chamado de laço de retardo. Tal laço for, com um corpo vazio, executa o número de vezes indicado, nada mais fazendo senão contar. Você pode usar um laço de retardo, por exemplo, para diminuir a velocidade de um programa que está produzindo exibições na tela muito rapidamente, para que você possa lê-las.
As partes de inicialização, condição de continuação do laço e de incremento de uma estrutura for podem conter expressões aritméticas. Por exemplo, assuma que x = 2 e y = 10. Sex e y não são modificados no corpo do laço, o comando
for (int j = x; j <= 4 * x * y; j + y / x)
é equivalente ao comando
for (intj 2; j<=80; j +=5)
O “incremento” de uma estrutura for pode ser negativo (sendo, de fato, um decremento e o laço, na verdade, conta para trás).
Se a condição de continuação do laço é inicialmente falsa, o corpo da estrutura for não é executado. Em vez
disso, a execução continua com o comando seguinte ao for.
A variável de controle é freqüentemente impressa ou usada em cálculos, no corpo de uma estrutura for, mas
ela não precisa ser. E comum usar a variável de controle somente para controlar a repetição sem nunca refenciá-la no corpo da estrutura for.
Boa prática de programação 2.22

Embora o valor da variável de controle possa ser mudado no corpo de um laço for, evite fazer isto, pois
esta prática pode levar a erros de lógica sutis.

CAPÍTULO 2 - ESTRUTURAS DE CONTROLE 133

O fluxograma da estrutura for é muito semelhante ao da estrutura while. A Fig. 2.19 mostra o fluxograma do comando for
counter
verdadeiro [_ <edi counter++
Corpo do laço Incrementa a
(este pode ter variável de
muitos comandos) controle

Fig. 2.19 Fluxograma de uma estrutura de repetição for típica.
O fluxograma torna claro que a inicialização acontece uma vez e que o incremento acontece todas as vezes depois de o comando do corpo ser executado. Note que (além de pequenos círculos e setas) o fluxograma contém somente retângulos e losangos. Imagine, novamente, que o programador pode acessar uma caixa de estruturas for vazias - tantas quanto o programador poderá necessitar empilhar e aninhar com outras estruturas de controle, para formar uma implementação estruturada de um algoritmo. Os retângulos e losangos são preenchidos com ações e decisões apropriadas ao algoritmo.
2.15 Exemplos usando a estrutura for
Os exemplos seguintes mostram diferentes métodos de fazer variar a variável de controle em uma estrutura for. Em cada caso, escrevemos o cabeçalho do for apropriado. Note a mudança no operador relacional para laços que decrementam a variável de controle.
a) Faça a variável de controle variar de 1 até 100 em incrementos de 1.
for (int i = 1; i <= 100; i++
b) Faça a variável de controle variar de 100 até 1 em incrementos de -1 (decrementos de 1).

for (irit i = 100; i > 1; i--

Erro comum de programação 2.17

Não usar o operador relacional apropriado na condição de continuação do laço, em um laço que conta para trás (tal como usar incorretamente i <= 1 em um laço que conta até 1) é normalmente um erro de lógica que produzirá resultados incorretos quando o programa for executado.
c) Faça a variável de controle variar de 7 até 77 em incrementos de 7.

for (int counter = 1; counter <= 10; counter++ cout « counter « endl;

Estabelece o valor inicial da variável de controle -.
Determina se o valor final da variável de controle foi atingido -a

falso

for (inti=7; i<=77; i+=7)

d) Faça a variável de controle variar de 20 até 2 em incrementos de -2.
for (int i = 20; i >= 2; i-=2
e) Faça a variável de controle variar ao longo da seguinte seqüência de valores:
2, 5, 8, 11, 14.
17, 20.
for (intj 2; j <201 j +=3
O Faça a variável de controle variar segundo seqüência seguinte de valores: 99.
88. 77, 66. 55, 44, 33, 22, 11, O.
for (int = 99; j >= 0; j -= 11)
Os próximos dois exemplos fornecem aplicações simples da estrutura for. O programa da Fig. 2.20 usa a estrutura for para somar todos os inteiros de 2 a 100.

```
1 II Fig. 2.20: figO22O.cpp
2 // Somatório com for
3 #include <iostream>
4
5 using std::cout;
6 using std: :endl;
7
8 intmain ()
10 intsum=0;
11
12 for ( int number = 2; number <= 100; number += 2
13 sum += number;
14
15 cout « “A soma é “ « sum « endi;
16
17 return 0;
18
A soma é 2550
```

Fig. 2.20 Somatório com for,
Note que o corpo da estrutura for da Fig. 2.20 na verdade podia ser fundido com a parte mais à direita do
cabeçalho de for, usando-se o operador vírgula como segue:

```
for ( int nuxnber = 2; II inicialização
number <= 100; // condição de continuação
sum += number, number += 2) // totalização e incremento
```

A atribuição sum = O também poderia ser fundida com a seção de inicialização do for.
Boa prática de programação 2.23
Embora os comandos que antecedem um for e os comandos no corpo de um for possam ser freqüentemente fundidos no cabeçalho do for, evite fazer isso porque torna o programa mais difícil de ser lido.

Boa prática de programação 2.24
Limite o tamanho de cabe çalhos de estruturas de controle a uma única linha, se possível.
O próximo exemplo calcula juros compostos usando a estrutura for. Considere a seguinte definição de problema:
Uma pessoa investe $1000,00 em uma poupança rendendo 5 por cento de juros. Assumindo que todo o rendimento dos juros é deixado depositado na conta, calcule e imprima a quantia de dinheiro na conta nofim de todo ano, por 10 anos. Use a seguinte fórmula para determinar estas quantias:
a=p(l +r)
onde
p é a quantia original investida (i.e., o principal),
r é a taxa anual de juros,
n éonúmerodeanose
a é a quantia em depósito no fim do n-ésimo ano.
Este problema envolve um laço que executa o cálculo indicado sobre o dinheiro em depósito, para cada um dos 10 anos. A solução é mostrada na Fig. 2.21.
A estrutura for executa o corpo do laço 10 vezes, fazendo variar uma variável de controle de 1 até 10 em incrementos de 1. C++ não inclui um operador de exponenciação; assim, usaremos a função pow, da biblioteca padrão, para esta finalidade. A função pow (x, y) calcula o valor de x elevado a y-ésima potência. Neste exemplo, a expressão algébrica (1 + r) é escrita como pow (1 + rate, year) , na qual a variável rate representa r e a variável year representa n. A função pow recebe dois argumentos do tipo double e retorna um valor double. Este programa não compilaria sem a inclusão de <cmath>. A função pow exige dois parâmetros double. Note que year é um inteiro. O arquivo <cmath> inclui informações que dizem ao compilador para converter o valor de year em uma representação double temporária antes de chamar a função. Estas informações estão contidas no protótipo da função pow. Os protótipos de funções são explicados no Capítulo 3. Fornecemos um resumo da função pow e outras funções da biblioteca de matemática no Capítulo 3.
Erro comum de programação 2.18
Esquecer de incluir o arquivo <cmath> em um programa que usa funções da biblioteca matemática é um
erro de sintaxe.
Note que declaramos as variáveis amount, principal e rate como do tipo double . Fizemos isto para simplificar, porque estamos lidando com partes fracionárias de moedas e necessitamos de um tipo que permita partes fracionárias em seus

valores. Infelizmente, isto pode trazer dificuldades. Uma explicação simples do que pode dar errado quando usamos float ou double para representar quantias em moedas (supondo que setprecision (2) é
usada para especificar dois dígitos de precisão na impressão): duas quantidades expressas em dólares armazenadas na máquina poderiam ser 14,234 (impressa como 14,23) e 18,673 (impressa como 18,67). Quando estas quantias forem somadas, elas produzem a soma interna 32,907, que é impressa como 32,91. Deste modo, seu relatório poderia se parecer com

```
14.23
+ 18.67
32.91
```

mas uma pessoa somando os números individuais impressos esperaria obter a soma 32,90! Você foi alertado!

Boa prática de programação 2.25
Não use variáveis do tipo float ou double para fazer cálculos monetários. A imprecisão dos números de ponto flutuante pode causar erros que resultarão em valores monetários incorretos. Nos exercícios, exploramos o uso de inteiros para executar cálculos monetários. Nota: bibliotecas de classes de C+ + de fornecedores independentes estão disponíveis para executar corretamente cálculos monetários.

```
1 II Fig. 2.21: figO22l.cpp
2 II Calculando juros compostos
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7 using std::ios;
8
9 #include <iomanip>
10
11 using std::setw;
12 using std::setiosflags;
13 using std::setprecision;
14
15 #include <cmath>
16
17 int main
18 {
19 double amount, II quantia em depósito
20 principal = 1000.0, II inicializando o capital
21 rate = .05; II taxa de juros
22
23 cout « “Ano” « setw( 21
```

```
24 « “Quantia em depósito” « endl;
25
26 Il ajusta o formato de números em ponto flutuante
27 cout« setiosflags( ios::fixed 1 ios::showpoint
28 « setprecision( 2 );
29
30 for ( int year = 1; year <= 10; year++
31 amount = principal * pow( 1.0 + rate, year );
32 cout « setw( 4 ) « year« setw( 21 ) « amount « endl;
33 }
34
•1 35 return 0;
36 }
Ano Quantia em depósito
1 1050.00
2 1102.50
3 1157.62
4 1215.51
5 1276.28
6 1340.10
7 1407.10
8 1477.46
9 1551.33
10 1628.89
```

Fig. 2.21 Calculando juros compostos com for.

O comando de saída

```
cout« setiosflags( ios::fixed 1 ios::showpoint
« setprecision( 2 );
```

antes do laço for e o comando de saída

```
cout« setw( 4 ) « year« setw( 21 ) « amount « endi;
```

no laço for são combinados para imprimir os valores das variáveis year e amount com a formatação especificada pelos manipuladores de stream parametrizados setw. setiosflags e setprecision. A chamada setw 4) especifica que o próximo valor de saída é impresso em um campo de comprimento 4, i.e., o valor é impresso
com pelo menos 4 posições de caracteres. Se o valor a ser exibido/impresso tem menos que 4 caracteres de comprimento, o valor é alinhado à direita no campo automaticamente. Se o valor a ser exibido/impresso tem mais de 4 caracteres de comprimento, o comprimento do campo é estendido para acomodar todo o valor. A chamada seti- osflags (ios: : left) pode ser usada para especificar que os valores devem ser exibidos/impressos alinhados à esquerda no campo.

O restante da formatação no comando de saída precedente indica que a variável

amount é impressa como um valor de ponto fixo com uma casa decimal (especificada com o manipulador de stream setiosflags (ios:
fixed 1 ios: : showpoint)) alinhado à direita em um campo de 21 posições de caracteres (especificado com setw(21)) e dois dígitos de precisão à direita da casa decimal (especificado com setprecision (2). Discutiremos em detalhes as poderosas capacidades de formatação de entrada/saída de C++ no Capítulo 11.
Colocamos os manipuladores de stream setiosflags e setprecision em um comando cout antes do laço for porque estes ajustes continuam valendo até que sejam mudados. Assim, eles não precisam ser aplicados durante cada iteração do laço.
Note que o cálculo 1 . O + rate, que aparece como um parâmetro para a função pow, está contido no corpo
do comando for. Na verdade, este cálculo produz o mesmo resultado toda vez que o laço é executado, de modo que repetir o cálculo é um desperdício de tempo.
Dica de desempenho 2.5
• Evite colocar expressões cujos valores não mudam dentro de laços - mas, ainda que você assim o faça, muitos dos sofisticados compiladores otimizadores atuais automaticamente colocarão tais expressões fora
dos laços no código gerado em linguagem de máquina.
Dica de desempenho 2.6
f Muitos compiladores contêm recursos de otimização que melhoram o código que você escreve; porém, ainda é melhor escrever um bom código desde o início.
Por diversão, não deixe de tentar resolver nosso problema de Peter Minuit nos exercícios do capítulo. Este problema demonstra as maravilhas dos juros compostos.
2.16 A estrutura de seleção múltipla switch
Discutimos a estrutura de seleção if de seleção única e a estrutura if/else de seleção dupla. Ocasionalmente, um algoritmo conterá uma série de decisões em que uma variável ou expressão será separadamente testada para cada um dos valores integrais constantes que ela pode assumir e ações diferentes serão executadas. C++ oferece a estrutura de seleção múltipla switch para tratar tais tomadas de decisões.
A estrutura switch consiste em uma série de rótulos case e um caso default opcional. O programa na
Fig. 2.22 usa switch para contar a quantidade de cada nota representada por letra diferente que estudantes receberam em um exame.

1
2
3
4
5
6
7

54
55
56
57

```
using std::cout; using std::cin; using std::endl;
int main
int grade,
aCount = 0,
bCount = 0,
cCount = 0,
dCount 0,
fCount = 0;

case A’ case a’:
++aCount;
break;
case ‘B’:
case ‘b’:
++bCount;
break;
case ‘C’:
case ‘c’:
++cCount;
break;
case ‘D’:
case ‘d’:
++dCount;
break;
case ‘F’:
case ‘f’:

++fCount;
break;
case case case
break;
default:

// uma nota
// número de As
// número de Bs
// número de Cs
II número de Ds
// número de Fs
```

// nota foi A maiúsculo // ou a minúsculo

II nota foi C maiúsculo // ou c minúsculo

II nota foi D maiúsculo // ou d minúsculo
// nota foi F maiúsculo II ou f minúsculo
II ignora newlines, II tabulações, II e espaços na entrada

138 C++ COMO PROGRAMAR

II Fig. 2.22: figo2_22.cpp
II Contando notas representadas por letras #include <iostream>

cout« “Forneça as notas representadas por letras.” « endl
« “Digite o caractere EOF para terminar a entrada de dados.” « endi;
21 while ((grade = cin.get ( )) = EOF
switch (grade) // switch aninhado no while

1

// necessário para sair do switch
// nota foi B maiúsculo /1 ou b minúsculo

II pega todos os outros caracteres
cout « “Fornecida nota representada por letra incorreta.’ « “ Digite urna nova nota.” « endl;

Fig. 2.22 Um exemplo usando switch (parte 1 de 2).

68
69 return 0;
70

totais para cada nota representada por letra são:”
« aCount
« bCount
« cCount
« dCount
« fCount « endl;

Forneça as notas representadas por letras.
Digite o caractere EOF para terminar a entrada de dados.

a
B
c
c
A

d
f
c
E

Fornecida nota representada por letra incorreta. Digite uma nova nota.
D

A

b

Os totais para cada nota representada por letra são:
A: 3

B: 2
C: 3
O: 2
F: 1

Fig. 2.22 Um exemplo usando switch (parte 2 de 2).

No programa, o usuário digita notas representadas por letras para uma turma. Dentro do cabeçalho do while, while ((grade = cin.get ( )) != EOF)
a atribuição entre parênteses (grade = cm . get (é executada primeiro. A função ci get ( ) lê um caractere do teclado e armazena esse caractere na variável inteira grade. A notação com ponto usada em cm . get ( ) será explicada no Capítulo 6, “Classes”. Caracteres normalmente são armazenados em variáveis do tipo char. Porém, uma característica importante de C++ é que caracteres podem ser armazenados em qualquer tipo de dado inteiro porque são representados como inteiros de 1 byte no computador. Deste modo, podemos tratar um caractere ou como um inteiro ou como um caractere, dependendo de seu uso. Por exemplo, o comando
cout « ‘0 caractere (“ « ‘a’ « “) tem o valor
« static cast< int > (‘a) « endl;
imprime o caractere a e seu valor inteiro, como segue

break; II opcional

58
59
60
61
62
63
64
65
66
67

CAPÍTULO 2 - ESTRUTURAS DE CONTROLE 139

cout «
«
«
«
«

“\n\nOs
\nB:

O caractere (a) tem o valor 97

140 C++ COMO PROGRAMAR
O inteiro 97 é a representação numérica do caractere no computador. Muitos computadores hoje usam o conjunto de caracteres ASCII (American Standard Code for Information Interchange), no qual 97 representa a letra mínuscula ‘a’. Nos apêndices, é apresentada a lista dos caracteres ASCII e seus valores decimais.
Os comandos de atribuição como um todo têm o valor que é atribuído à variável à esquerda do =. Deste modo, o valor da atribuição grade = cm . get é o mesmo que o valor retornado por cm . get ( ) e atribuído à variável grade.
O fato que comandos de atribuição têm valores pode ser útil para inicializar várias variáveis com o mesmo valor. Por exemplo,
a = b = c = O;
primeiro avalia a atribuição c = O (porque o operador = se associa da direita para a esquerda). À variável b é então atribuído o valor da atribuição c = O (que é O). Então, à variável a é atribuído o valor da atribuição b = Cc = O) (que também é O). No programa, o valor da atribuição grade = cm . get é comparado com o valor de EOF (um símbolo formado com as iniciais de “fim de arquivo” em inglês). Usamos EOF (que tem normalmente o valor
-1) como o valor sentinela. No entanto, você não digita o valor -1 nem as letras EOF como valor sentinela. O usuário digita uma combinação de teclas,

dependente do sistema, para informar “fim de arquivo”, i.e., “eu não tenho mais dados para digitar”. EOF é uma constante inteira simbólica definida no “arquivo de cabeçalho” <iostreani>. Se o valor atribuído a grade for igual a EQE, o programa termina. Optamos por representar caracteres neste programa como ints porque EOF tem um valor inteiro (repetindo, normalmente -1).

Dica de portabilidade 2.2

_____ As combinações de teclas para digitar fim de arquivo são dependentes do sistema.

Dica de portabilidade 2.3

_____ Testar a constante simbólica EOF em lugar de -1 torna os programas mais portáveis. O padrão ANSI estabelece que EOF é um valor inteiro negativo (mas não necessariamente -1). Assim, EOF pode ter valores diferentes em sistemas diferentes.

Nos sistemas UNIX e muitos outros, o fim de arquivo é indicado digitando-se a seqüência

<ctrl-d>

em uma linha. Esta notação significa apertar simultaneamente tanto a tecla ctrl como a tecla d. Em outros sistemas, tal como o VAX VMS da Digital Equipment Corporation ou o MS-DOS da Microsoft Corporation, o fim de arquivo pode ser indicado digitando-se

<ctrl-z>

Nota: em alguns casos, você pode ter que apertar Enter após a seqüência de teclas precedente.

O usuário fornece as notas pelo teclado. Quando a tecla Enter (ou Return) é apertada, os caracteres são udos pela função cm . get ( ) um caractere de cada vez. Se o caractere digitado não é fim de arquivo, a estrutura switch é executada. A palavra-chave switch é seguida pelo nome de variável grade entre parênteses. Isto é chamado de expressão de controle. O valor desta expressão é comparado com cada um dos rótulos case. Assuma
4 que o usuário digitou a letra C como uma nota. C é automaticamente comparada a cada case no switch, Se
:4 ocorre uma igualdade (case ‘C’ :), os comandos para esse case são executados. Para a letra C, cCount é incre1 mentado por 1 e sai da estrutura switch imediatamente com o comando break. Note que, diferentemente de outras estruturas de controle, não é necessário se incluir um comando case múltiplo entre chaves.

O comando break faz com que o controle do programa passe para o primeiro comando depois da estrutura switch. O comando break é usado porque, caso contrário, os cases em um comando switch seriam executados juntos. Se break não for usado em algum lugar em uma estrutura switch, então toda vez que ocorrer uma igualdade, ou correspondência, na estrutura, os comandos para todos os casos restantes serão executados. (Esta

característica às vezes é útil quando se quer executar as mesmas ações para vários cases, como no programa da Fig. 2.22.) Se nenhuma correspondência

ocorrer, ocaso default é executado e é impressa uma mensagem de erro. Cada case pode ter uma ou mais ações. A estrutura switch é diferente de todas outras estruturas no sentido
de que não são exigidas chaves antes e depois de ações múltiplas em um case de um switch. O fluxograma da estrutura de seleção múltipla switch genérica (usando um break em cada case) é mostrado na Fig. 2.23.
O fiuxograma torna claro que cada comando break no fim de um case faz com que o controle saia imediatamente da estrutura switch. Novamente, note que (além de pequenos círculos e setas) o fluxograma contém só retângulos e losangos. Imagine, novamente, que o programador tem acesso a uma caixa cheia de estruturas switch vazias - tantas quanto o programador poderia necessitar para empilhar e aninhar com outras estruturas de controle para formar uma implementação estruturada do fluxo de controle de um algoritmo. E, novamente, os retângulos e losangos são então prenchidos com ações e decisões apropriadas ao algoritmo. Estruturas de controle aninhadas são comuns, mas é raro se encontrar em um programa estruturas switch aninhadas.

falso

Fig. 2.23 A estrutura de seleção múltipla switch com breaks.

Erro comum de programação 2.19
Esquecer um comando break, quando é necessário um, em uma estrutura switch, é um erro de
lógica.
Erro comum de programação 2.20
Omitir o espaço entre a palavra case e o valor inteiro que está sendo testado, em uma estrutura swi - tch, pode causar um erro de lógica. Por exemplo, escrever case3: em vez de escrever case 3: simplesmente cria um rótulo não-usado (falaremos mais sobre isto no Capítulo 18). O problema é que a estrutura switch não executará as ações apropriadas quando a expressão de controle do switch tiver
o valor 3.
Boa prática de programação 2.26
Forneça um caso default em comandos switch. Os casos não-testados explicitamente em um comando switch sem um caso default são ignorados. Incluir um caso default chama a atenção do programador sobre a necessidade de processar condições excepcionais. Existem situações em que nenhum processamento defaul t é necessário. Embora as cláusulas case e a cláusula do caso defaul t em uma estrutura swi tch possam ocorrer em qualquer ordem, é considerada uma boa prática de programação colocar a cláusula default por último.
Boa prática de programação 2.27
Em uma estrutura switch, quando a cláusula default é listada por último, o comando break não

é necessário. Alguns programadores incluem este break por clareza e simetria com outros casos.
Na estrutura switch da Fig. 2.22, as linhas 50 a 53
case
case
case
break;
fazem o programa ignorar caracteres de nova linha, marcas de tabulação e caracteres “branco”. Ler um caractere de cada vez pode causar alguns problemas. Para que o programa leia os caracteres, eles devem ser enviados ao computador pressionando-se a tecla Enter no teclado. Isto coloca um caractere de nova linha na entrada, depois do caractere que desejamos processar. Freqüentemente, este caractere de nova linha deve ser processado especialmente para fazer o programa funcionar corretamente. Pela inclusão dos casos precedentes em nossa estrutura switch, evitamos que a mensagem de erro do caso default seja impressa sempre que um caractere de nova linha, marca de tabulação ou espaço é encontrado na entrada.
Erro com um de programação 2.21
Não processar caracteres de nova linha e outros caracteres de espaço na entrada, quando estivermos
lendo caracteres um de cada vez, pode causar erros de lógica.
Note que vários rótulos de caso listados juntos (tal como case ‘D’ : case d’ : na Fig. 2.22) simplesmente indicam que o mesmo conjunto de ações deve ocorrer para cada um dos casos.
Quando usar a estrutura switch, lembre-se de que ela só pode ser usada para testar uma expressão inteira constante, i.e., qualquer combinação de constantes de caracteres e constantes inteiras cujo valor é um inteiro constante. Uma constante de caractere é representada com o caractere específico entre apóstrofes, tal como ‘A’ . Uma constante inteira é simplesmente um valor do tipo inteiro.
Quando chegarmos à parte do livro sobre programação orientada a objetos, apresentaremos um caminho mais elegante para implementar a lógica de switch. Usaremos uma técnica chamada polimorfismo para criar programas que freqüentemente são claros, mais concisos, mais fáceis de manter e mais fáceis de estender que os programas que usam lógica de switch.

As linguagens portáveis como C-i-+ devem ter tamanhos de tipos de dados flexíveis. Aplicativos diferentes podem necessitar de inteiros de tamanhos diferentes. C++ oferece vários tipos de dados para representar inteiros. O intervalo de valores inteiros para cada tipo depende do hardware de um computador particular. Além dos tipos int e char, C++ oferece os tipos short (uma abreviação de short int) e long (uma abreviação de long int). O intervalo mínimo de valores para inteiros short é -32.768 a 32.767. Para a vasta maioria dos cálculos com inteiros, inteiros long são suficientes. O intervalo de valores

mínimo para inteiros long é -2.147.483.648 a 2.147,483.647. Na maioria dos computadores, ints são equivalentes ou a short ou a long. O intervalo de valores para um int é pelo menos o mesmo que o intervalo para inteiros short e não maior que o intervalo para inteiros long. O tipo de dados char pode ser usado para representar quaisquer dos caracteres no conjunto de caracteres do computador. O tipo de dados char também pode ser usado para representar valores inteiros peque-

nos,

Dica de portabilidade 2.4

Como ints variam de tamanho entre sistemas, use inteiros long se você espera processar inteiros fora do intervalo -32.768 a 32.767 e quer ser capaz de executar seu programa em vários sistemas de computador diferentes.

Dica de desempenho 2.7

Em situações voltadas para o desempenho, em que a memória é muito escassa ou a velocidade de execução é crucial, pode ser desejável usar tamanhos menores de inteiros.

Dica de desempenho 2.8
Usar tamanhos de inteiro menores pode resultar em um programa mais lento se as instruções da máquina para manipulá-los não forem tão eficientes quanto aquelas para os inteiros de tamanho natural (por exemplo, pode ser preciso fazer extensão do bit de sinal neles).

Erro comum de programação 2.22

Fornecer rótulos de case idênticos, em uma estrutura switch, é erro de sintaxe.

2.17 A estrutura de repetição do/while

A estrutura de repetição do/while é semelhante à estrutura while. Na estrutura while, a condição cc cominuação do laço é testada no princípio do laço antes do corpo do laço ser executado. A estrutura do/while testa a condição de continuação do laço depois do corpo do laço ser executado; assim, o corpo do laço será executado pelo menos uma vez. Quando um do/while termina, a execução continua com o comando depois da cláusula while. Note que não é necessário usar chaves na estrutura do/while se existir só um comando no corpo. Porém, as chaves são normalmente incluídas para evitar confusão entre a estrutura while e a estrutura do/while. Por exemplo,

while (condition

é normalmente considerado como o cabeçalho de uma estrutura while. Um do/while sem chaves em torno do corpo de comando único aparece como

do

state,nent

while (condition

que pode ser confuso. A última linha - while (condition) ; - pode ser mal interpretada pelo leitor como uma estrutura while contendo um comando vazio. Deste modo, o do/while com um comando é freqüentemente escrito como segue, para evitar confusão:

do

.ltatenlent
1 while (condition

Boa prática de programação 2.28
Alguns programadores sempre incluem chaves em uma estrutura do/while mesmo que as chaves não sejam necessárias. Isto ajuda a eliminar a ambigüidade entre a estrutura while e a estrutura do/while contendo um único comando.
Erro comum de programação 2.23
Laços infinitos são causados quando a condição de continuação do laço, em uma estrutura while. for ou do/whi le, nunca se torna falsa. Para prevenir este erro, tenha certeza de que o valor da condição muda em algum lugar no cabeçalho ou no corpo do laço, de maneira que a condição possa, em algum momento, se tornar falsa.
O programa na Fig. 2.24 usa uma estrutura de repetição do/while para imprimir os números de 1 até 10. Note que
a variável de controle counter é pré-incrementada no teste de continuação do laço. Note também o uso das chaves
para incluir o único comando do corpo da estrutura do/while.

```
1 Il Fig. 2.24: figO2_24.cpp
2 Il usando a estrutura de repetição do/while
3 #include <iostream>
4
5 using std: :cout;
6 using std::endl;
7
```

```
8 intmain ()
9 {
10 int counter = 1;
11
12 do{
13 cout « counter «
14 } while ( ++counter < 10 );
15
16 cout « endi;
17
18 return 0;
19
2345678910
```

Fig. 2.24 Usando a estrutura do/while.

O fluxograma da estrutura do/while é mostrado na Fig. 2.25. Este fluxograma torna claro que a condição de continuação do laço não é executada até depois da ação ser executada pelo menos uma vez. Novamente, note que (além de pequenos círculos e setas) o fluxograma contém só um retângulo e um losango. Imagine, novamente, que o programador tem acesso a uma caixa cheia de estruturas do/while vazias - tantas quantas o programador possa necessitar empilhar e aninhar em outras estruturas de controle para formar uma implementação estruturada do fluxo de controle de um algoritmo. E, novamente, os retângulos e losangos são então prenchidos com ações e decisões apropriadas ao algoritmo.

2.18 Os comandos break e continue

Os comandos break e continue alteram o fluxo de controle, O comando break. quando executado em uma estrutura while, for, do/while ou switch, provoca a saída imediata da estrutura. A execução do programa continua com o primeiro comando depois da estrutura. Os usos mais comuns do comando break são sair antecipadamente de um laço ou saltar o restante de uma estrutura switch (como na Fig. 2.22). A Fig. 2.26 demonstra o

iao
le

comando break em uma estrutura de repetição for. Quando a estrutura if descobre que x se tornou 5, o break é executado. Isto termina o comando for e o programa continua com o cout depois do for. O laço é executado completamente só quatro vezes.

Note que a variável de controle x neste programa é definida fora do cabeçalho da estrutura for. Isto porque
pretendemos usar a variável de controle tanto no corpo do laço como depois do laço completar sua execução.

```
1 II Fig. 2.26: figo2_26.cpp
2 // Usando o comando break em uma estrutura for
3 #include <iostream>

4

using std::cout;
using std: :endl;

int main

x é declarado aqui para que possa ser usado após o laço int x;

for ( x 1; x <= 10; x++

if ( x == 5
break; /1 sai do laço somente se x é 5

ou
em
, se

que
aves
```

Fig. 2.25 O fluxograma da estrutura de repetição do/while.

lo de que que)ossa [luxo isões

```
5
6
7
8
9
10
1].
12
13
14
15
16
17
18
19
```

20
21
22
23

1

cout « x «
cout « “\nSaiu do laço com x igual a « x « endi; return 0;

uma
rama cipatra o

123	4			
Saiu do		laço	com x igual a 5	
Fig. 2.26 U		sanc io	o comando brea	k em uma estrutura for.

O comando continue, quando executado em uma estrutura while. for ou do/while. salta os comandos restantes no corpo dessa estrutura e prossegue com a próxima repetição do laço. Em estruturas while e dof while. o teste de continuação do laço é feito logo depois do comando continue ser executado. Na estrutura for, a expressão de incremento é executada e então é feito o teste de continuação do laço. Anteriormente, declaramos que a estrutura while pode ser usada na maioria dos casos para representar a estrutura for. A única exceção ocorre quando a expressão de incremento na estrutura while vem após o comando continue. Neste caso, o incremento não é executado antes da condição de repetição/continuação ser testada e o while não é executado da mesma maneira que o for. A Fig. 2.27 usa o comando continue em um estrutura for para saltar o comando de saída na estrutura e começar a próxima repetição do laço.

```
1 II Fig. 2.27: fig0227.cpp
2 // Usando o comando continue em uma estrutura for
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 int main O
9 {
10 for ( int x = 1; x <= 10; x++
11
12 if(x5)
13 continue; // salta o restante do código no laço
```

```
14 // somente se x é 5
15
16 cout « x «
17
18
19 cout « “\nusou continue para pular a impressão do valor 5’
20 « endi;
21 returnO;
22
1 2 3 4 6 7 8 9 10
Usou contínue para pular a impressão do valor 5
```

Fig. 2.27 Usando o comando continue em uma estrutura for.

Boa prática de programação 2.29

Alguns programadores consideram que break e continue violam a programação estruturada. Como os efeitos destes comandos podem ser obtidos por técnicas de progrwnação estruturada que logo aprenderemos, estes programadores não usam break e continue.

fDica de desempenho 2.9

_____ Os comandos break e continue, quando usados corretamente, são executados mais rapidamente que
as técnicas estruturadas correspondentes que logo aprenderemos.

Observação de engenharia de software 2.10

Existe um certo conflito, ou incompatibilidade, entre realizar uma engenharia de software de qualidade e obter o software de melhor desempenho. Freqüentemente, uma destas metas é alcançada às custas da outra.

ilta os comandos ,s while e do!
ido. Na estrutura 3rmente, declara- A única exceção
ie. Neste caso, o lo é executado da Itar o comando de

2.19 Operadores lógicos

Até agora estudamos somente condições simples, tais como counter <= 10, total > 1000 e nuniber ! = sentinelValue. Expressamos estas condições em termos dos operadores relacionais>. <, > e < e dos operadores de igualdade == e ! . Cada decisão testa precisamente uma condição. Para testar condições múltiplas ao tomar uma decisão, executamos estes testes em comandos separados, ou em if ou estruturas if/else aninhadas.

C++ oferece operadores lógicos que são usados para especificar condições mais complexas combinando condições simples. Os operadores lógicos são: & & (E lógico), 1 1 (OU lógico) e ! (NAO lógico, também chamado de negação lógica). Examinaremos exemplos de cada um destes.

Suponhamos que desejemos nos assegurar de que duas condições são ambas true, antes de escolhermos um certo caminho de execução. Neste caso, podemos usar o operador lógico &&, como segue:

Este comando if contém duas condições simples. A condição gender == 1 poderia ser avaliada, por exemplo, para determinar se uma pessoa é uma mulher. A condição age >= 65 é avaliada para determinar se uma pessoa é idosa. A condição simples à esquerda do operador && é avaliada primeiro, porque a precedência de == é mais alta que a precedência de &&. Se necessário, a condição simples à direita do operador && é avaliada em seguida, porque a precedência de >= é mais alta que a precedência de & & (como discutiremos em breve, o lado direito de uma expressão lógica E só é avaliado se o lado esquerdo é (true) . O comando if, então, analisa a condição combinada

Esta condição é true se, e somente se, ambas as condições simples são true. Finalmente, se esta condição combinada é realmente true, então a contagem de seniorFemales é incrementada por 1. Se uma, ou ou ambas, das condições simples são false. então o programa salta o incremento e continua no comando em seguida ao if. A condição combinada precedente pode ser tomada mais legível acrescentando-se parênteses redundantes

Erro comum de programação 2.24
Embora 3 < x < 7 seja uma condição matematicamente correta, ela não é avaliada corretamente em
C++. Use (3 < x && x < 7) para obter a avaliação apropriada em C++.
A tabela da Fig. 2.28 resume o operador &&. A tabela mostra todas as quatro combinações possíveis de valores
false e true para a expressão 1 e a expressão2. Tais tabelas são freqüentemente chamadas de tabelas verdade.
C++ avalia como false ou true todas as expressões que incluem operadores relacionais, operadores de igualdade e/ou operadores lógicos.

expressãol expressão2 expressãol && expressão2 1

false false true true

false

Fig. 2.28 Tabela verdade para o operador && (E lógico).

are de qualidade e içada às custas da

Dica de portabidade 2.5
Por compatibilidade com versões antigas do padrão C+ +, o valor true do tipo bool pode também ser representado por qualquer valor diferente de zero e o valor bool fal se também pode ser representado como o valor 0.

if (gender == 1 && age >= 65 ++seniorFemales;

gender == 1 && age >= 65

gender == 1) && (age > 65

estruturada. Como que logo aprendeis rapidamente que

true false true

false false false true

Agora vamos considerar o operador 1 1 (OU logico). Suponhamos que desejemos nos assegurar, em um certo
ponto de um programa, que ou uma ou ambas de duas condições são true antes de escolhermos um certo caminho de execução. Neste caso, usamos o operador 1 1 como no segmento de programa seguinte:
if (semesterAverage >= 90 1 1 finalExaju > 90
cout « “A nota do estudante é A” « endi;
A condição precedente também contém duas condições simples. A condição simples semesterAverage > 90
é avaliada para determinar se o estudante merece um “A” no curso por causa de um desempenho altamente consistente ao longo do semestre. A condição simples finalExam >= 90 é avaliada para determinar se o estudante
merece um “A” no curso por causa de um desempenho excelente no exame final. O comando if, então, considera a
condição combinada
sezuesterAverage >= 90 1 1 finalExam > 90
e premia o estudante com um “A” se qualquer uma ou ambas as condições simples são true (verdadeiras). Note que a mensagem “A nota do estudante é A” só não é impressa quando ambas as condições simples são falsas. A Fig. 2.29 é uma tabela verdade para o operador lógico 0k (1 1).

expressãol expressão2 expressãol 1 1 expressão2

false false false
false true true
true false true
true true true
Fig. 2.29 Tabela verdade para o operador 1 1 (OU lógico).
O operador & & tem uma precedência mais alta que o operador I. Ambos os operadores se associam da esquerda para a direita. Uma expressão contendo operadores & & ou 1 1 é avaliada somente até sua verdade ou falsidade ser conhecida. Deste modo, a avaliação da expressão
gender == 1 && age >= 65
parará imediatamente se gender não for igual a 1 (i.e., a expressão inteira é

false) e continuará se gender for
igual a 1 (i.e., a expressão inteira pode ainda ser true se a condição age >= 65 for true).
Erro comum de programação 2.25
Em expressões que usam o operador &&, é possível que uma condição - chamaremos esta condição de condição dependente - possar exigir que uma outra condição seja true para que faça sentido avaliar a condição dependente. Neste caso, a condição dependente deve ser colocada depois da outra condição ou então pode ocorrer um erro.
Dica de desempenho 2.10
f . Em expressões que usam o operador &&, se as condições separadas são independentes uma da outra, coloque à esquerda a condição com maior probabilidade de ser false. Em expressões que usam o operador I I , coloque mais à esquerda a condição que tem maior probabilidade de ser true. Isto pode reduzir o tempo de execução de um programa.
C++ oferece o operador ! (negação lógica) para possibilitar ao programador "inverter" o significado de uma condição. Diferentemente dos operadores && e 1 I que combinam duas condições (operadores binários), o operador de negação lógica tem somente uma única condição como operando (operador unário). O operador de negação

if (! (grade sentinelValue
cout « A próxima nota é " « grade « endi;

Os parênteses em torno da condição grade == sentinelValue são necessários porque o operador de negação lógica tem uma precedência mais alta que o operador de igualdade. A Fig. 2.30 é uma tabela verdade para o operador de negação lógica.

Na maioria dos casos, o programador pode evitar o uso da negação lógica expressando a condição diferentemente com um operador relacional ou de igualdade apropriado. Por exemplo, o comando precedente pode também ser

escrito como segue:

if (grade sentinelVa].ue
cout « "A próxima nota é " «grade « endi;

Esta flexibilidade freqüentemente pode ajudar um programador a expressar uma condição de uma maneira mais

"natural" ou conveniente.

A Fig. 2.31 mostra a precedência e associatividade dos operadores de C++ introduzidos até aqui. Os operadores são mostrados de cima para baixo, em

ordem decrescente de precedência.

Fig. 2.31 Precedência e associatividade de operadores.

erto
nho

lógica é colocado antes de uma condição quando estivermos interessados em escolher um caminho de execução se a condição original (sem o operador de negação lógica) for fa].se, tal como no segmento de programa a seguir:

=90
nsislante
era a

Note

5 SO

Fig. 2.30 Tabela verdade para o operador ‘(negação lógica).

uerda
de ser

ão de
‘liara
ão ou

am o
pode

uma peraação

120 Confundindo os operadores de igualdade (==) e de atribuição (=)

Existe um tipo de erro que os programadores de C++, não importa quão experientes sejam, tendem a cometer tão freqüentemente que percebemos que este problema merecia uma seção separada. O erro é trocar, acidentalmente, os

expres	‘expressão

são	1
false	true
true	false

Operaclores		Associatividade	Tipo
O		esquerda para a direita	parênteses
++	--	static_cast<type> () esquerda para a direita	unário (pós-fixo)
++	--	+ - direita para a esquerda	unário (prefixo)
*	/	% esquerda para a direita	multiplicativo
+	-	esquerda para a direita	aditivo
«	»	esquerda para a direita	inserção/extração
<	<=	> >= esquerda para a direita	relacional
=	=	esquerda para a direita	igualdade
&&		esquerda para a direita	E lógico
1		esquerda para a direita	OU lógico
?:		direita para a esquerda	condicional
	+	* /= %= direita para a esquerda	atribuição
,		esquerda para a direita	vírgula

operadores == (igualdade) e = (atribuição). O que torna estas trocas tao prejudiciais e o tato de que elas normalmente não causam erros de sintaxe. Ao contrário, comandos com estes erros compilam em geral corretamente e os programas executam até o fim, provavelmente gerando resultados incorretos através de erros de lógica durante a execução.
Existem dois aspectos de C++ que causam estes problemas. Um deles é que qualquer expressão que produz um valor pode ser usada na parte de decisão de qualquer estrutura de controle. Se o valor for O, ele será tratado como false e se o valor não for zero, será tratado como true. A segunda é que atribuições em C++ produzem um valor, i.e., o valor que é atribuído à variável do lado esquerdo do operador de atribuição. Por exemplo, suponha que pretendamos escrever
if (payCode 4
cout « Você obteve urna gratificação’” « endi;
mas que, porém, escrevamos acidentalmente
if (payCode 4
cout « “Você obteve urna gratificação!” « endi;
O primeiro comando if concede corretamente uma gratificação à pessoa cujo paycode é igual a 4. O segundo comando if - o que tem o erro - avalia a

expressão de atribuição na condição if obtendo a constante 4. Como qualquer valor diferente de zero é interpretado como true, a condição neste comando if é sempre true e a pessoa sempre recebe uma gratificação, não importando qual seja o paycode verdadeiro! Pior ainda, o paycode foi modificado, quando se supunha que ele seria somente testado!

Erro comum de programação 2.26

Usar o operador == para atribuição, ou usar o operador = para igualdade, constitui erros de lógica.

® Dica de teste e depura ção 2.1

Os programadores escrevem normalmente condições tais como x 7 com o nome da variável à esquerda e o da constante à direita. Invertendo estes nomes, deforma que a constante fique à esquerda e o nome da variável à direita, como em 7 == x, o programador que substituir acidentalmente o operador == pelo = estará protegido pelo compilador. O compilador tratará essa troca como um erro de sintaxe porque só um nome de variável pode ser colocado à esquerda de um comando de atribuição. Pelo menos este evitará a devastação potencial do erro de lógica durante a execução.

Os nomes de variáveis são chamados de lvalues (ou seja. “valores à esquerda” - left values, em inglês) porque podem ser usados à esquerda de um operador de atribuição. As constantes são chamadas de rtalues (ou seja, “valores à direita” - right values) porque podem ser usadas somente à direita de um operador de atribuição. Note que lvalues também podem ser usados como rvalues, mas não vice-versa.

O outro lado da moeda pode ser igualmente desagradável. Suponha que o programador quer atribuir um valor a uma variável com um comando simples como

```
x = 1;
```

mas, em vez disso, ele escreve

```
x == 1;
```

Aqui, também, não temos um erro de sintaxe. Em vez disso, o compilador simplesmente avalia a expressão condicional. Se x for igual a 1, a condição é true e a expressão retorna o valor true. Se x não for igual 1, a condição é false e a expressão retorna o valor false. Independentemente de qual valor é retornado, não existe nenhum operador de atribuição, assim o valor é simplesmente perdido, permanecendo inalterado o valor de x e provavelmente causando um erro de lógica durante a execução. Infelizmente, não temos à mão um truque disponível para ajudá-lo a evitar este problema!

® Dica de teste e depura ção 2.2

Use seu editor de texto para procurar todas as ocorrências de = em seu programa e conferir se você tem o
operador correto em cada lugar

2.21 Resumo de programação estruturada

Da mesma maneira que os arquitetos projetam construções empregando a sabedoria coletiva de sua profissão, assim deveriam os programadores projetar seus programas. Nosso campo é mais jovem que a arquitetura e nossa sabedoria coletiva é consideravelmente mais escassa. Aprendemos que a programação estruturada produz programas que são mais fáceis de entender do que programas não-estruturados e que, conseqüentemente, são mais fáceis para testar, depurar, modificar e até provar sua correção no sentido matemático. A Fig. 2.32 resume as estruturas de controle de C++. Os círculos pequenos são usados na figura para indicar
o ponto único de entrada e o ponto único de saída de cada estrutura. Conectar símbolos individuais de fluxogra ma

arbitrariamente pode levar a programas não-estruturados. Portanto, os profissionais de programação decidiram combinar símbolos de fluxograma, de forma a compor um conjunto limitado de estruturas de controle e construir programas estruturados corretamente combinando estruturas de controle de duas maneiras simples.

estrutura if (seleção única)

estrutura if/else (seleção dupla)
FI T

Fig. 2.32 As estruturas de repetição de seqüência, seleção e repetição com uma única entrada/única saída em C++.

Para simplificar, são usadas somente estruturas de controle de entrada e saída única e existe uma única maneira para entrar, bem como uma única maneira para sair de cada estrutura de controle. Conectar estruturas de controle em seqüência para formar programas estruturados é simples - o ponto de saída de uma estrutura de controle é conectado ao ponto de entrada da próxima estrutura de controle, i.e., as estruturas de controle são simplesmente colocadas uma depois da outra em um programa; temos chamado isto de “empilhamento de estruturas de controle”. As regras para formar programas estruturados também permitem que as estruturas de controle sejam aninhadas.

nte raao.
luz

[no or, jue

do

de

Seq.iência

Seleção

T
+

Repetição

(à
(or xe
[OS

_ 1
_____ 1
estrutura switch
(seleção múltipla)

ue
loue

estrutura while
O estrutura do/while
estrutura for
O

lor

T

ao
,a
;te

A Fig. 2.33 mostra as regras para formar programas corretamente estruturados. As regras assumem retângulo de um fluxograma possa ser usado para indicar qualquer ação, inclusive entrada/saída. As regras taff
supõem que começamos com o fluxograma mais simples (Fig. 2.34).

Regras para formar programas estruturados

Fig. 2.34 O fluxograma mais simples.

Aplicar as regras da Fig. 2.33 sempre resulta em um fluxograma estruturado, com uma aparência limpa como a blocos de construção. Por exemplo, aplicando repetídamente a regra 2, o fluxograma mais simples resulta em u fluxograma estruturado, contendo muitos retângulos em seqüência (Fig. 2.35). Note que a regra 2 gera uma pilha estruturas de controle; assim, iremos chamar a regra 2 de regra de empilhamento.

CD

Regra 2

CD

Fig. 235 Aplicando repetidamente a regra 2 da Fig. 2.33 ao fluxograma mais simples.

1)	Comece com o" fluxograma mais simples" (Fig. 2.34).
2)	Qualquer retângulo (ação) pode ser substituído por dois retângulos (ações) em seqüência.
3)	Qualquer retângulo (ação) pode ser substituído por qualquer estrutura de controle
	(seqüência, if, if/else. switch, while, doJwhi].e ou for).
4)	As regras 2 e 3 podem ser aplicadas tão freqüentemente quanto você queira e em qualquer ordem.
Fig	. 2.33 Regras para formar programas estruturados.

A regra é chamada de regra de aninhamento. Aplicando repetidamente a regra 3, o fluxograma mais
simples resulta em um fluxograma com estruturas de controle nitidamente aninhadas. Por exemplo, na Fig. 2.36, o retângulo no fluxograma mais simples primeiro é substituído por uma estrutura de seleção dupla (if/else).
Então, a regra 3 é novamente aplicada a ambos os retângulos na estrutura de seleção dupla, substituindo cada um destes retângulos por estruturas de seleção dupla. As caixas tracejadas ao redor de cada estrutura de seleção dupla representam o retângulo que foi substituído no fluxograma original mais simples.

A regra 4 gera estruturas maiores, mais complicadas e mais profundamente aninhadas, O fluxograma que
surge da aplicação das regras na Fig. 2.33 constitui o conjunto de todos os fluxogramas estruturados possíveis e, conseqüentemente, o conjunto de todos

os programas estruturados possíveis.
Ji..
_____ Regra 3
Regra 3
t
1

Regra 3

Fig. 2.36 Aplicando a regra 3 da Fig. 2.33 ao fluxograma mais simples.

A beleza da abordagem estruturada está no fato de usarmos somente sete peças simples de entrada / saída única e as montarmos somente de duas maneiras. A Fig. 2.37 mostra o tipo de blocos de construção empilhados que surgem da aplicação da regra 2 e os tipos de blocos de construção aninhados que surgem da aplicação da regra 3. A Fig. também mostra o tipo de blocos de construção sobrepostos que não pode aparecer em fluxogramas estruturados (por causa da eliminação do comando goto).
Se as regras na Fig.2.33 são seguidas, não é possível criar um fluxograma não-estruturado (como o da Fig. 2.38). Se você não tiver certeza se um dado fluxograma é estruturado ou não, aplique as regras da Fig. 2.33 na ordem inversa, para tentar reduzi-lo ao fluxograma mais simples. Se o fluxograma é redutível ao tluxograma mais simples, então o fluxograma original é estruturado; caso contrário, ele não o é.
Blocos de construção empilhados
1 ___
II
Blocos de construção sobrepostos
(ilegal em programas estruturados)
Fig. 2.37 Blocos de construção empilhados, aninhados e sobrepostos.
A programação estruturada promove a simplicidade. Bohm e Jacopini nos demonstraram que somente três formas de controle são necessárias:
• Seqüência
• Seleção
• Repetição
A seqüência é trivial. A seleção é implementada em uma de três formas:
• Estrutura if (seleção única)
• Estrutura if/else (seleção dupla)
• Estrutura switch (seleção múltipla)

Blocos de construção aninhados
‘1

1

Fig. 2.38 Um fluxograma não-estruturado.

CAPÍTULO 2 - ESTRUTURAS DE CONTROLE 155
e as De fato, é simples provar que a estrutura if simples é suficiente para satisfazer qualquer forma de seleção - tudo inda que pode ser feito com a estrutura if/else e a estrutura switch pode ser implementado pela combinação de bém estruturas if (embora talvez não tão clara e eficientemente).
sa da A repetição é implementada em um de três maneiras:
Fi • estrutura while
rdem • estrutura do/while
pies, estrutura for
É simples provar que a estrutura while é suficiente para implementar qualquer forma de repetição. Tudo que pode ser feito com a estrutura do/while e a estrutura for pode ser feito com a estrutura while (embora talvez não tão suavemente).
Combinar estes resultados mostra que qualquer forma de controle que possa ser necessária em um programa
em C++ pode ser expressa em termos de:
• seqüência
• estrutura if (seleção)
• estrutura while (repetição)
e estas estruturas de controle podem ser combinadas só de dois modos - empilhamento e aninhamento. A programação estruturada realmente promove a simplicidade.
Neste capítulo discutimos como compor programas a partir de estruturas de controle contendo ações e decisões. No Capítulo 3, introduziremos outra unidade de estruturação de programa denominadafunção. Aprenderemos a compor programas grandes combinando funções que, por sua vez, são compostas de estruturas de controle. Também discutiremos como funções promovem a reusabilidade de software. No Capítulo 6, introduzimos outra unidade de estruturação de programa de C++ denominada classe. Então, criaremos objetos a partir de classes e continuaremos com nosso tratamento da programação orientada a objetos. Agora continuamos nossa introdução a objetos, introduzindo um problema que o leitor atacará com as técnicas do projeto orientado a objetos.
2.22 (Estudo de caso opcional) Pensando em objetos: identificando as classes em um problema
Agora iniciamos nosso estudo de caso opcional de projeto/implementação orientado a objetos. Estas seções “Pensando em objetos”, nos finais deste e dos próximos capítulos, vão deixá-lo à vontade com a orientação a objetos, examinando um estudo de caso de simulação de elevador. Este estudo de caso vai proporcionar-lhe uma experiência de projeto e implementação completa, substancial e cuidadosamente cadenciada. Nos Capítulos 2 a 5, executaremos as várias etapas de um projeto orientado a objetos (OOD, object-oriented design) usando a UML. Nos Capítulos 6, 7 e 9, implementaremos o simulador de elevador usando as técnicas de programação orientada a objetos (OOP. object-

oriented programming) em C÷+. Apresentamos este estudo de caso em um formato totalmente resolvido. Isto não é um exercício, mas sim uma experiência de aprendizado ponta a ponta que termina com um walkrhrough* detalhado do código em C++. Oferecemos este estudo de caso para que você se acostume com os tipos de problemas de porte que são enfrentados em empresas. Esperamos que você aprecie esta experiência.
rmas Definição do problema
Uma empresa pretende construir um edifício comercial de dois andares e equipá-lo com um elevador. A empresa quer que você desenvolva um simulador em software orientado a objetos em C++ que modele a operação do elevador para determinar se este elevador satisfará ou não às suas necessidades.
N. de R.T.: Termo usado para descrever uma das atividades de desenvolvimento de software, que consiste em reunir a equipe de desenvolvimento para analisar o código de um programa e discutir a implementação e as decisões tomadas pelos programadores; usada como ferramenta de treinamento e como parte de um processo de qualidade no desenvolvimento de software.

Seu simulador deve incluir um relógio que inicia com sua hora, em segundos, ajustado para zero. O relógic “bate” (incrementa o tempo em uma unidade) a cada segundo; ele não mantém controle de horas e minutos. Seu simulador também deve incluir um scheduler* que começa o dia escalonando aleatoriamente dois momentos: a hora em que uma pessoa vai entrar no pavimento 1 e apertar o botão do andar para chamar o elevador e a hora em que uma pessoa vai entrar no pavimento 2 e apertar o botão do andar para chamar o elevador. Cada um destes momentos é um valor inteiro aleatório, no intervalo fechado de 5 a 20 (i.e., 5, 6, 7 20). [Nota: você aprenderá como escalonar horários aleatórios no Capítulo 3]. Quando a hora do relógio fica igual ao mais cedo destes dois momentos, o schedu/er cria uma pessoa, a qual, então, entra no andar apropriado e aperta o botão do andar. [Nota: é possível que estes dois momentos escalonados aleatoriamente sejam idênticos, caso em que as pessoas entrarão nos dois pavimentos e apertarão os botões dos andares ao mesmo tempo]. O botão do andar se ilumina, indicando que foi pressionado. [Nota: a iluminação do botão do andar ocorre automaticamente quando o botão é pressionado e não requer nenhuma programação; a luz embutida no botão se apaga automaticamente quando o botão é desligado]. O elevador começa o dia esperando, com sua porta fechada, no pavimento 1. Para economizar energia, ele se move somente quando necessário. O elevador alterna a direção de movimento entre subir e descer.
Para simplificar, o elevador e cada um dos andares tem capacidade para uma pessoa. O scheduler inicialmente verifica se um andar está desocupado, antes de criar uma pessoa para entrar naquele andar. Se o andar está ocupado, o scheduler retarda a criação da pessoa em um segundo (dando, assim, oportunidade para que o elevador busque a pessoa e libere o andar). Depois

que uma pessoa entra em um andar, o scheduler gera o próximo momento aleatório (entre 5 e 20 segundos mais tarde) em que uma pessoa entrará naquele andar e apertará o botão do andar.
Quando o elevador chega em um andar, ele desliga o botão do elevador e faz soar a campainha (que está no seu interior). O elevador então sinaliza sua chegada ao andar, O andar, em resposta, desliga o botão do andar e liga a luz de chegada do elevador do andar. A porta se abre. [Nota: a porta no andar abre automaticamente, junto com a porta do elevador, e não necessita de nenhuma programação]. O passageiro, se houver, sai do elevador e uma pessoa, se houver uma esperando naquele andar, entra no elevador. Embora cada andar tenha capacidade para uma pessoa, suponha que haja espaço suficiente em cada andar para que uma pessoa espere naquele andar enquanto o passageiro do elevador, se houver, sai.
Uma pessoa que entra no elevador aperta o botão, que se ilumina (automaticamente, sem programação) quando pressionado e se apaga quando o elevador chega no andar e libera o botão. [Nota: como só existem dois andares, é necessário somente um botão dentro do elevador; este botão simplesmente diz ao elevador para se mover para o outro andar.] Ao chegar em um andar, se uma pessoa não entra no elevador e o botão no outro andar não foi pressionado, o elevador fecha sua porta e permanece naquele andar até que um botão em um andar seja pressionado.
Para simplificar, suponha que todas as atividades que acontecem quando o elevador chega em um andar e até que feche a porta durem zero segundos. [Nota: embora estas atividades tenham duração zero, elas ainda acontecem seqüencialmente, por exemplo, a porta do elevador precisa abrir antes de o passageiro sair do mesmo]. O elevador leva cinco segundos para se mover de qualquer andar para o outro. Uma vez por segundo, o simulador informa a hora para o scheduler e para o elevador, O scheduler e o elevador usam a hora para determinar quais ações cada um deles deve tomar naquele momento em particular, por exemplo, o scheduler pode decidir que é hora de criar uma pessoa; e o elevador, se estiver se movendo, pode descobrir que é hora de chegar no pavimento de destino.
O simulador deve exibir mensagens na tela descrevendo as atividades que ocorrem no sistema. Estas incluem uma pessoa apertando o botão em um andar, o elevador chegando em um andar, o relógio batendo, uma pessoa entrando no elevador, etc. A saída deve ser parecida com a seguinte:

1

Digite tempo de execução: 30
(scheduler escalona próxima pessoa para o andar 1 no tempo 5)
(scheduler escalona próxima pessoa para o andar 2 no tempo 17)
“ INÍCIO DA SIMULAÇÃO DO ELEVADOR ***
TEMPO:
elevador parado no andar 1

* N. de R.T.: Termo usado para designar, em sistemas de software (principalmente sistemas operacionais), a rotina ou conjunto de rotinas que selecionam as tarefas que serão executadas e em que ordem, com base em prioridades definidas pelo sistema.

---

para zero, O relógio oras e minutos. Seu iomentos: a hora em a hora em que uma stes momentos é um será como escalonar mentos, o scheduler ssível que estes dois pavimentos e apertaressionado. [Nota: a er nenhuma prograva dor começa o dia nte quando necessáeduler inicialmente
andar está ocupado, o elevador busque a momento aleatório D do andar.
inha (que está no seu do andar e liga a luz e, junto com a porta lor e uma pessoa, se le para uma pessoa, quanto o passageiro
)rogramação) quanistem dois andares, ira se mover para o andar não foi presseja pressionado.
em um andar e até is ainda acontecem tesmol. O elevador idor informa a hora
ões cada um deles criar uma pessoa; ‘o.
ma. Estas incluem endo, uma pessoa

TEMPO: 2
elevador parado no andar 1
TEMPO: 3
elevador parado no andar 1
TEMPO: 4
elevador parado no andar 1
TEMPO: 5
scheduler cria a pessoa 1
pessoa 1 entra no andar 1
pessoa 1 aperta o botão do andar no andar 1
botão do andar 1 chama o elevador
(scheduler escalona próxima pessoa para o andar 1 no tempo 20) elevador desliga o botão
elevador toca a campainha
andar 1 desliga o botão
andar 1 acende a luz
elevador abre a porta no andar 1
pessoa 1 entra no elevador no andar 1
pessoa 1 aperta o botão do elevador
botão do elevador diz ao elevador que se prepare para sair andar 1 apaga a luz
elevador fecha a porta no andar 1
elevador começa a subir para o andar 2 (chega no tempo 10)
TEMPO: 6
elevador subindo

TEMPO: 7
elevador subindo
TEMPO: 8
elevador subindo
TEMPO: 9
elevador subindo
TEMPO: 10
elevador chega no andar 2
elevador desliga o botão
elevador toca a campainha
andar 2 desliga o botão
andar 2 acende a luz
elevador abre a porta no andar 2
pessoa 1 sai do elevador no andar 2
andar 2 apaga a sua luz
elevador fecha a porta no andar 2
elevador parado no andar 2
TEMPO: 11
elevador parado no andar 2
TEMPO: 12

elevador parado no andar 2
TEMPO: 13
elevador parado no andar 2

A

ijunto de rotinas que

TEMPO: 14
elevador parado no andar 2

TEMPO: 15
elevador parado no andar 2
TEMPO: 16
elevador parado no andar 2
TEMPO: 17
scheduler cria a pessoa 2
pessoa 2 entra no andar 2
pessoa 2 aperta o botão do andar no andar 2
botão do andar 2 chama o elevador
(scheduler escalona próxima pessoa para o andar 2 no tempo 34) elevador desliga o botão

elevador toca a campainha
andar 2 desliga o botão
andar 2 acende a luz
elevador abre a porta no andar 2
pessoa 2 entra no elevador no andar 2
pessoa 2 aperta o botão do elevador
botão do elevador diz ao elevador que se prepare para sair andar 2 apaga a luz
elevador fecha a porta no andar 2
elevador começa a descer para o andar 1 (chega no tempo 22)
TEMPO: 18
elevador descendo
TEMPO: 19
elevador descendo
TEMPO: 20
scheduler cria a pessoa 3
pessoa 3 entra no andar 1
pessoa 3 aperta o botão do andar no andar 1
botão do andar 1 chama o elevador
(scheduler escalona próxima pessoa para o andar 1 no tempo 26)
elevador descendo
TEMPO: 21
elevador descendo
TEMPO: 22
elevador chega no andar 1
elevador desliga o botão
elevador toca a campainha
andar 1 desliga o botão
andar 1 acende a luz
elevador abre a porta no andar 1
pessoa 2 sai do elevador no andar 1
pessoa 3 entra no elevador no andar 1
pessoa 3 aperta o botão do elevador
botão do elevador diz ao elevador que se prepare para sair andar 1 apaga a luz
elevador fecha a porta no andar 1
elevador começa a subir para o andar 2 (chega no tempo 27)
TEMPO: 23
elevador subindo

TEMPO: 24
elevador subindo

TEMPO: 25
elevador subindo
TEMPO: 26
scheduler cria a pessoa 4

pessoa 4 entra no andar 1
pessoa 4 aperta o botão do andar no andar 1
botão do andar 1 chama o elevador
(scheduler escalona próxima pessoa para o andar 1 no tempo 35)
elevador subindo
TEMPO: 27
elevador chega no andar 2
elevador desliga o botão
elevador toca a campainha
andar 2 desliga o botão
andar 2 acende a luz
elevador abre a porta no andar 2
pessoa 3 sai do elevador no andar 2
andar 2 apaga a luz
elevador fecha a porta no andar 2
elevador começa a descer para o andar 1 (chega no tempo 32)
TEMPO: 28
elevador descendo
TEMPO: 29
elevador descendo
TEMPO: 30
elevador descendo
*** FIM DA SIMULAÇÃO DO ELEVADOR ***
LNosso objetivo (ao longo destas seções “Pensando em objetos” nos Capítulos 2 a 9) é implementar um simulador em software que funcione e modele a operação do elevador durante o número de segundos especificado pelo usuário do simulador.
Analisando e projetando o sistema
Nesta e nas próximas seções “Pensando em objetos”, executamos as etapas de um projeto orientado a objetos para o sistema do elevador. A UML é projetada para uso com qualquer processo OOAD - existem muitos destes processos. Um método popular é o Rational Unifled ProcessTM desenvolvido pela Rational Software Corporation. Para este estudo de caso, apresentamos nosso próprio processo de projeto simplificado para sua primeira experiência com OOD/UML.
Antes de começarmos, precisamos examinar a natureza das simulações. Uma simulação consiste em duas partes. Uma contém todos os elementos que pertencem ao mundo que queremos simular. Estes elementos incluem o elevador, os pavimentos, os botões, as luzes, etc. Chamemos esta parte de parte do mundo. A outra parte contém todos os elementos necessários para simular este mundo. Estes elementos incluem o relógio e o scheduler. Chamamos esta parte de parte controladora. Manteremos estas duas partes em mente à medida que projetamos nosso sistema.
Diagramas de caso de uso
Quando os desenvolvedores iniciam um projeto, eles raramente começam com uma definição detalhada do problema. como a que fornecemos no começo desta

seção (Seção 2.22). Este documento e outros normalmente são o resultado da fase de análise orientada a objetos (OOA, object-oriented analvsis). Nesta fase, você entrevista as pessoas que querem que você construa o sistema e as pessoas que vão em algum momento usar o sistema. Você usa a informação obtida nestas entrevistas para compilar uma lista de requisitos do sistema. Estes requisitos guiam você e seus companheiros desenvolvedores enquanto projetam o sistema. Em nosso estudo de caso, a definição do proble

ma contém os requisitos de sistema para o sistema de elevador. O resultado da fase de análise destina-se a especificar claramente o que se pretende que o sistema faça. O resultado da fase de projeto destina-se a especificar claramente como o sistema deve ser construído para fazer o que é necessário.

A UML oferece o diagrama de casos de uso para facilitar o processo de reunir requisitos. O diagrama de casos de uso modela as interações entre os clientes externos do sistema e os casos de uso do sistema. Cada caso de uso representa um recurso diferente que o sistema oferece ao cliente. Por exemplo, uma máquina de caixa automática tem diversos casos de uso, incluindo "depósito", "retirada" e "transferência de fundos".

A Fig. 2.39 mostra o diagrama de caso de uso para o sistema do elevador. A figura estilizada representa um ator. Atores são quaisquer entidades externas, tais como pessoas, robôs, outros sistemas, etc. que usam o sistema. Os únicos atores em nosso sistema são as pessoas que querem andar no elevador. Portanto, modelamos um ator chamado "Pessoa". O "nome" do ator aparece sob a figura estilizada.

A caixa do sistema (i.e., o retângulo de contorno na figura) contém os casos de uso para o sistema. Observe que a caixa está rotulada com "Sistema do elevador". Este título mostra que este modelo de caso de uso focaliza os comportamentos do sistema que queremos simular (i.e., elevador transportando pessoas), e não os comportamentos da simulação (i.e., criar pessoas e escalonar chegadas).

A UML modela cada caso de uso como uma elipse. Em nosso sistema simples, os atores usam o elevador somente para uma finalidade: ir para outro andar. O sistema oferece somente um recurso a seus usuários; portanto, "Ir para outro andar" é o único caso de uso em nosso sistema do elevador.

A medida que constrói seu sistema, você se apóia no diagrama de caso de uso para assegurar que todas as necessidades dos clientes são atendidas. Nosso estudo de caso contém somente um caso de uso. Em sistemas maiores, diagramas de caso de uso são ferramentas indispensáveis que ajudam os projetistas do sistema a manter o foco na satisfação das necessidades dos usuários. O objetivo do diagrama de caso de uso é mostrar os tipos de interações que os usuários têm com um sistema, sem fornecer os detalhes destas interações.

Identificando as classes em um sistema
A próxima etapa em nosso processo de OOD é identificar as classes em nosso problema. Em algum momento, vamos descrever estas classes de uma maneira formal e implementá-las em C++ (começamos a implementar o simulador do elevador em C++ no Capítulo 6). Primeiro, revisamos a definição do problema e localizamos todos os substantivos; muito provavelmente, eles representam a maioria das classes (ou instâncias de classes) necessárias para implementar o simulador do elevador. A Fig. 2.40 é uma lista destes substantivos.
Lista de substantivos na definição do problema
empresa andar 2
edifício porta do elevador
elevador energia
simulador capacidade
relógio botão do elevador

Fig. 2.39 Diagrama de caso de uso para o sistema do elevador.

Fig. 2.40 Lista de substantivos na definição do problema (parte 1 de 2).

Fig. 2.40 Lista de substantivos na definição do problema (parte 2 de 2).
A seguir, escolhemos somente os substantivos que desempenham tarefas importantes em nosso sistema. Por esta razão, vamos omitir os seguintes:
• empresa
• simulador
• hora
• energia
• capacidade
Não precisamos modelar “empresa” como uma classe, porque a empresa não faz parte da simulação; ela simplesmente quer que modelemos o elevador. O “simulador” é todo o nosso programa em C++, não uma classe isolada. A “hora” é uma propriedade do relógio, não uma entidade autônoma. Não modelamos “energia” em nossa simulação (embora companhias de eletricidade, gás ou óleo pudessem certamente estar interessadas em fazer isto em seus programas de simulação) e, finalmente, “capacidade” é uma propriedade do elevador e dos pavimentos - não uma entidade autônoma separada.
Determinamos as classes para nosso sistema classificando os substantivos restantes em categorias. Cada substantivo remanescente da Fig. 2.40 se refere a uma ou mais das seguintes categorias:
• edifício
• elevador
• relógio
• scheduler
• pessoa (pessoa esperando em um andar, passageiro do elevador)

• andar (andar 1, andar 2)
• botão do andar
• botão do elevador
• campainha
• luz
• porta

Estas categorias são provavelmente as classes que precisaremos implementar para nosso sistema. Observe que criamos uma categoria para os botões dos andares e uma categoria para o botão do elevador. Os dois tipos de botões desempenham tarefas diferentes em nossa simulação - os botões nos andares chamam o elevador e o botão no elevador diz ao elevador para começar a se mover para o outro andar.

Podemos agora modelar as classes em nosso sistema com base nas categorias que derivamos. Por convenção, vamos usar letras maiúsculas para iniciar nomes de classes. Se o nome de uma classe contém mais de uma palavra, juntamos todas as palavras e começamos cada uma delas com letra maiúscula (por exemplo, NomeComVarias- Palavras). Usando esta convenção, criamos as classes Elevator. Clock, Scheduler, Person, Floor, Doar, Building, FloorButton. ElevatorButton. Beil e Light*. Construímos nosso sistema usando todas estas classes como blocos de construção. Antes de começarmos a construir o sistema, no entanto, precisamos entender melhor como as classes se relacionam entre si.

* N. de T.: “Elevador”, “Relógio”, “Scheduler”, “Pessoa”, “Andar”, “Porta”, “Edifício”, “Botão do andar”, “Botão do elevador’, “Campainha” e “Luz”, respectivamente.

Lista de substantivos na definição do problema	
hora	campainha do elevador
scheduler	luz de chegada do elevador em um andar
pessoa	pessoa esperando no andar
andar 1	passageiro do elevador
botão do andar	

Diagramas de classes

A UML nos possibilita modelar as classes no sistema de elevador e seus relacionamentos através de diagramas de classes. A Fig. 2.41 mostra como representar uma classe usando a UML. Aqui, modelamos a classe Elevator. Em um diagrama de classes, cada classe é modelada como um retângulo. Este retângulo pode, então, ser dividido em três partes. A parte superior contém o nome da classe.

A parte do meio contém os atributos da classe. Discutimos atributos na seção “Pensando em objetos” no fim do Capítulo 3. A parte inferior contém as operações da classe. Discutimos operações na seção “Pensando em objetos” no fim do Capítulo 4.
Elevator
Fig. 2.41 Representando uma classe na UML.
As classes se relacionam com outras classes através de associações. A Fig. 2.42 mostra como nossas classes Buil- ding, Elevator e Floor se relacionam umas com as outras. Observe que os retângulos neste diagrama não estão subdivididos em três seções. A UML permite o truncamento dos símbolos de classes desta maneira, de modo a criar diagramas mais legíveis.

2
2 Serviços li
Floor Elevator

Fig. 2.42 Associações entre classes em um diagrama de classes.
Neste diagrama de classes, uma linha cheia que conecta classes representa uma associação. Uma associação é um relacionamento entre classes. Os números junto às linhas expressam valores de multiplicidade. Os valores de multiplicidade indicam quantos objetos de uma classe participam da associação. Pelo diagrama, vemos que dois objetos da classe Floor participam da associação com um objeto da classe Building. Portanto, a classe Building tem um relacionamento um-para-dois com a classe Floor: também podemos dizer que a classe Floor tem um relacionamento dois-para-um com a classe Building. A partir do diagrama, você pode ver que a classe Building tem um relacionamento um-para-um com a classe Elevator e vice-versa. Usando a UML, podemos modelar muitos tipos de multiplicidade. A Fig. 2.43 mostra os tipos de multiplicidade e como representá-los.
Uma associação pode ter um nome. Por exemplo, a palavra “Serviços” acima da linha que conecta as classes Floor e Elevator indica o nome daquela associação - a seta mostra a direção da associação. Esta parte do diagrama é lida assim: “um objeto da classe Elevator presta serviços a dois objetos da classe Floor”.
O losango cheio preso às linhas de associação da classe Building indica que a classe Building tem um relacionamento de composição com as classes Floor e Elevator. A composição implica um relacionamento todo! parte. A classe que tem o símbolo de composição (o losango cheio) em sua extremidade da linha de associação é o todo (neste caso, Building) e a classe na outra extremidade da linha de associação é a parte (i.e., Floor e Elevator).2
2 De acordo com as especificações da UML 1.3, as classes em um relacionamento de composição possuem as três propnedades seguintes: 1) somente uma classe no relacionamento pode representar o todo (i.e., o losango pode ser colocado somente em uma das extremidades da linha de associação); 2) a composição implica em tempos de vida coincidentes para as partes com o todo e o todo é responsável pela cnação e destmição de suas partes; 3) uma parte pode pertencer somente a um todo de cada vez, embora a parte possa ser

removida e anexada a Outro todo, que então assume a responsabilidade pela parte.

CAPÍTULO 2 - ESTRUTURAS DE CONTROLE 163

Fig. 2.43 Tabela de multiplicidade

A Fig. 2.44 mostra o diagrama de classes completo para o sistema do elevador. Todas as classes que criamos estão modeladas, bem como as associações entre estas classes. [Nota: no Capítulo 9, expandimos nosso diagrama de classes usando o conceito orientado a objetos de herançal.

1

passageiro

Fig. 2.44 Diagrama completo de classes para a simulação do elevador.

Símbolo	Significado
O	Nenhum.
1	Um.
m	Um valor inteiro.
0.1	Zero ou um.
m..n	No mínimo m, mas não mais do que n.
*	Qualquer inteiro não-negativo.
0..’	Zero ou mais.
1..*	Umou mais.

164 C++ COMO PROGRAMAR

A classe Building está representada próximo ao topo do diagrama e é composta de quatro classes, incluindo Clock e Scheduler. Estas duas classes compõem a parte controladora da simulação.3 A classe Building também é composta pelas classes Elevator e Floor (observe o relacionamento um-para-dois entre a classe Building e a classe Floor).
As classes Floor e Elevator estão modeladas próximas à base do diagrama. A classe Floor é composta de um objeto de cada uma das classes Light e FloorButton. A classe Elevator é composta de um objeto de cada uma das classes ElevatorButton. classe Door e classe Beli.
As classes envolvidas em uma associação também podem desempenhar papéis. Papéis ajudam a tornar claro

o relacionamento entre duas classes. Por exemplo, a classe Person faz o papel de “passageiro aguardando” em
associação com a classe Floor (porque a pessoa está esperando o elevador). A classe Person faz o papel de
passageiro em sua associação com a classe Elevator. Em um diagrama de classes, o nome do papel de uma classe
é colocado em qualquer um dos lados da linha de associação, próximo ao retângulo da classe. Cada classe em uma
associação pode desempenhar um papel diferente.
A associação entre a classe Floor e a classe Person indica que um objeto da classe Floor pode relacionar-se com zero ou um objeto da classe Person. A classe Elevator também se relaciona com zero ou um objeto da classe Person. A linha tracejada que une estas duas linhas de associação indica uma restrição no relacionamento entre as classes Person. Floor e Elevator. A restriçào diz que um objeto da classe Person pode participar de um relacionamento com um objeto da classe Floor ou com um objeto da classe Elevator, mas não com os dois objetos ao mesmo tempo. A notação para este relacionamento é a palavra “xor” (que significa “ou exclusivo”) colocada entre chaves. A associação entre a classe Scheduler e a classe Person afirma que um objeto da classe Scheduler cria zero ou mais objetos da classe Person.
Diagramas de objetos
A UML também define diagramas de objetos, que são similares aos diagramas de classes, exceto pelo fato de modelarem objetos e links - links são relacionamentos entre objetos. Assim como os diagramas de classes, os diagramas de objetos modelam a estrutura do sistema. Diagramas de objetos apresentam uma fotografia da estrutura enquanto o sistema está em execução - isto oferece informação sobre quais objetos estão participando do sistema em um determinado instante no tempo.
A Fig. 2.45 modela uma fotografia do sistema quando ninguém está no edifício (i.e., não existe nenhum objeto da classe Person no sistema neste instante). Nomes de objetos são normalmente escritos na forma objectName:ClassName.
A primeira palavra em um nome de objeto não inicia com letra maiúscula, mas as palavras seguintes sim. Todos os nomes de objetos em um diagrama de objetos são sublinhados. Omitimos o nome do objeto para alguns objetos no diagrama (por exemplo, objetos da classe FloorButton). Em sistemas grandes, muitos nomes de objetos serão usados no modelo. Isto pode dar origem a diagramas congestionados, difíceis de ler. Se o nome de um objeto em particular é desconhecido ou se não é necessário incluir o nome (i.e., podemos nos preocupar somente com o tipo do objeto), podemos deixar de fora o nome do objeto. Neste caso, simplesmente mostramos os dois pontos e o nome da classe.
Agora já identificamos as classes para este sistema (embora possamos descobrir outras nas fases posteriores do processo de projeto). Também examinamos o caso de uso do sistema. Na seção “Pensando em objetos” no fim do Capítulo 3, usamos este conhecimento para examinar como o sistema muda ao longo do tempo. A medida que expandirmos nossos conhecimentos, também

descobriremos novas informações que nos possibilitarão descrever nossas classes em maior profundidade.
1. Você aprenderá a implementar a “aleatoriedade” no próximo capítulo (Capítulo 3), no qual estudamos a geração de números aleatórios. A geração de números aleatórios o ajuda a simular processos aleatórios,
O relacionamento composto entre as classes Building e as classes Clock e Scheduler representa uma decisão de projeto tomada por você. Consideramos a classe Building como sendo parte tanto da porção “mundo” quanto da porção “controladora” de nossa simulação. Em nosso projeto, atribuímos ao edifício a responsabilidade de executar a simulação.
Restrições em diagramas UML. podem ser escritas com o que é conhecido como a Object Constraint Language (OCL) - Linguagem de Restrição de Objetos. A OCI. foi criada para permitir que modeladores expressassem restrições em um sistema de uma forma claramente detinida. Para aprender mais, visite www-4.ibm.com/software/ad/5tandardilocl.html.

Notas

Fig. 2.45 Diagrama de objetos do edifício vazio.

tais como sortear “cara ou coroa” com uma moeda e o lançamento de dados. Também o ajudará a simular a chegada aleatória de pessoas para usar o elevador.
2. Como o mundo real é tão orientado a objetos, será bastante natural para você desenvolver este projeto, muito embora você ainda não tenha estudado formalmente a orientação a objetos.

Perguntas

1. Como você poderia decidir se o elevador pode dar conta do volume de tráfego previsto?
2. Por que poderia ser mais complicado implementar um edifïcio de três pavimentos (ou mais alto)?
3. E comum em grandes edifícios ter muitos elevadores. Veremos no Capítulo 6 que, uma vez que tenhamos criado um objeto elevador, é fácil criar tantos quanto queiramos. Que problemas ou oportunidades você prevê em ter vários elevadores, cada um deles podendo apanhar e deixar passageiros em todos os andares de um grande edifício?
4. Para simplificar, demos ao nosso elevador e a cada andar a capacidade de um passageiro. Que problemas ou oportunidades você prevê para poder aumentar estas capacidades?

Resumo

• Um procedimento para resolver um problema em termos das ações a serem executadas e a ordem em que estas ações devem ser executadas é chamado de algoritmo.

• Especificar a ordem em que os comandos devem ser executados em um programa de computador é chamado de controle do programa.

• Pseudocódigo ajuda o programador a pensar sobre" um programa antes de tentar escrevê-lo em uma linguagem de programação, tal como C++.

• Declarações são mensagens para o compilador informando-lhe os nomes e atributos das variáveis e solicitando-lhe que reserve espaço às mesmas.

Uma estrutura de seleção é usada para escolher cursos alternativos de ação.

• A estrutura de seleção if executa uma ação especificada só quando a condição é verdadeira.

• A estrutura de seleção if/else especifica ações separadas a serem executadas quando a condição é verdadeira e quando a condição é falsa.

166 c++ COMO PROGRAMAR

Sempre que mais de um comando deve ser executado, onde normalmente um único comando é esperado, estes comandos devem ser colocados entre chaves, formando um comando composto. Um comando composto pode ser colocado em qualquer lugar no qual possa ser colocado um único comando.

Um comando vazio, indicando que nenhuma ação deve ser executada, é indicado colocando-se um ponto-e-vírgula (;) onde normalmente seria colocado um comando.

Uma estrutura de repetição especifica que uma ação deve ser repetida enquanto alguma condição permaneça verdadeira.

O formato da estrutura de repetição while é

while (condição)
comando

Um valor que contém uma parte fracionária é chamado de número de ponto flutuante e é representado pelo tipo de dados float ou double.

O operador de coerção unário static_cast < double > ( ) cria uma cópia temporária em ponto flutuante de seu operando.

C++ oferece os operadores aritméticos de atribuição +=, -=, =, 1= e %= que ajudam a abreviar certos tipos comuns de expressões.

C++ oferece os operadores de incremento (++) e decremento (--) para incrementar ou decrementar uma variável por 1. Se o
operador é pré-fixado à variável, a variável é incrementada ou decrementeda por 1 primeiro e então usada em sua expressão.
Se o operador é pós-fixado à variável, a variável é usada em sua expressão e então incrementada ou decrementada por 1.

Um laço é um grupo de instruções que o computador executa repetidamente até que alguma condição de término seja satisfeita. Duas formas de repetição são: repetição controlada por contador e repetição controlada por sentinela.

Uni contador de laço é usado para contar as repetições de um grupo de instruções. É incrementado (ou decrementado) normalmente por 1 toda vez que o grupo de instruções é executado.

Os valores de sentinela são geralmente usados para controlar a repetição quando o número preciso de repetições não é conhecido com antecedência e o laço inclui comandos que obtêm dados toda vez que o laço é executado. Um valor de sentinela é digitado no final da entrada de itens de dados válidos para o programa. As sentinelas devem ser diferentes de itens de dados válidos.

A estrutura de repetição for trata de todos os detalhes da repetição controlada por contador. O formato geral da estrutura for é

for (initialization; loopContinuationTest; increment)
statement

onde initialization inicializa a variável de controle do laço. loopContinuationTest é a condição de continuação do laço e increment incrementa a variável de controle.

A estrutura de repetição do/while testa a condição de continuação do laço no fim do mesmo: assim, o corpo do laço será executado pelo menos uma vez. O formato para a estrutura do/while é

do

comando

while (condição)

O comando break, quando executado em uma das estruturas de repetição (for, while e do/while) . provoca a saída imediata da estrutura.

O comando continue, quando executado em uma das estruturas de repetição (for. while e do/while) . salta quaisquer comandos restantes no corpo da estrutura e prossegue com a próxima repetição do laço.

O comando swi tch processa uma série de decisões em que uma variável ou expressão particular é testada para os valores que ela pode assumir e ações diferentes são executadas. Na maioria dos programas, é necessário se incluir um comando

break depois dos comandos para cada um dos case.Vários cases podem executar os mesmos comandos, listando-se os rótulos dos vários cases em conjunto, antes dos comandos executáveis comuns aos cases. A estrutura switch só pode

testar expressões constantes inteiras. Um case multicomando não precisa ser incluído entre chaves. Em sistemas UNIX e muitos outros, o fim de arquivo é indicado digitando-se a seqüência
<ctrl-d>
sozinha em uma linha. No VMS e no DOS, o fim de arquivo é indicado digitando-se a seqüência

<ctrl-z>

possivelmente seguida pelo pressionamento da tecla Enter.

• Os operadores lógicos podem ser usados para formar condições complexas combinando condições. Os operadores lógicos são &&, 1 1 e !. significando o E lógico, o OU lógico e o NÃO lógico (negação), respectivamente.
• Qualquer valor não-zero é implicitamente convertido para true; O (zero) é implicitamente convertido para false.
Terminologia

1

algoritmo

bloco

bool break

caracteres em branco
caso default no switch
char
comando composto
comando vazio (;)
comprimento de campo
condição de continuação do laço
conjunto de caracteres ASCII
contador de laço
continue
corpo de um laço
definição
divisão inteira
double
E lógico (&&)
EOF
erro “saída uma repetição antes do fim”
erro de lógica
erro de sintaxe
erro fatal
erro não-fatal
estrutura de controle
estrutura de repetição do/while
estrutura de repetição for
estrutura de repetição while
estrutura de seleção dupla
estrutura de seleção if
estrutura de seleção if/else
estrutura de seleção múltipla
estrutura de seleção switch
estrutura de seleção unica
estruturas de controle aninhadas
estruturas de controle de entrada única / saída única

estruturas de repetição
execução seqüencial
false
float
formato de ponto fixo
funçãocin.get(
função pow
inicialização
ios: : fixed
ios: :left

ios: :showpoint
laço de retardo
laço infinito
long
lvalue (“left value”)
manipulador de stream parametrizado manipulador de stream setiosflags
manipulador de stream setprecision manipulador de stream setw modelo
ação/decisão
negação lógica (!)
operador -
operador
operador &&
operador?:
operador 1
operador ++
operador condicional (?:) operador de coerção
operador de decremento (-) operador de incremento (++) operador de pós-decremento operador de pós-incremento operador de pré-decremento operador de pré-incremento operador ternário
operador unário

estruturas de controle empilhadas

operadores aritméticos de atribuição: +=, -=, ‘=, 1= e %=

168 C++ COMO PROGRAMAR

operadores lógicos
OUlógico(I 1)
palavra-chave
programação estruturada
pseudocódigo
refinamento top-down, passo a passo
repetição
repetição controlada por contador
repetição definida
repetição indefinida
repetindo
Terminologia de “Pensando em objetos”

rótulo de case rvalue (“right value”) seleção
short staticcast<type> transferência de controle true
valor “lixo” valor indefinido valor sentinela

análise e projeto orientados a objetos (OOAD) análise orientada a objetos (DOA)

associação
ator
caixa do sistema
caso de uso
composição
diagrama de caso de uso
diagrama de classes
diagrama de objetos
estrutura estática de um sistema
identificar as classes em um sistema
linha cheia nos diagramas de classes e de objetos da UML link
losango cheio nos diagramas de classes e de objetos da UML multiplicidade
nome de associação

Object Constraint Language (DCL) “o que” versus “como” papel
porção controladora de uma simulação porção mundo de uma simulação projeto orientado a objetos (OOD) Rational Unified Process relacionamento dois para um relacionamento um para dois relacionamento um para um requisitos do sistema restrição
retângulo em diagrama de classes da UML simulador em software
xor

Erros comuns de programação
2.1 Ouso de uma palavra-chave como um identificador é erro de sintaxe.
2.2 Esquecer uma ou as duas chaves que delimitam um comando composto poder levar a erros de sintaxe ou erros de lógica em um programa.
2.3 Colocar um ponto-e-vírgula depois da condição em uma estrutura if leva a um erro de lógica em estruturas if de seleção única e a um erro de sintaxe em estruturas if de seleção dupla (se a parte if contiver um comando em seu corpo).
2.4 Não fornecer, no corpo de uma estrutura while. uma ação que faça com que a condição na estrutura while se torne false em algum momento normalmente resulta em um erro chamado “laço infinito”, no qual a estrutura de repetição nunca termina de ser executada.
2.5 Escrever a palavra-chave while com um w maiúsculo, como em While. é um erro de sintaxe (lembre-se de que C++ é uma linguagem sensível a maiúsculas e minúsculas). Todas as palavras-chave reservadas de C++, tais como while. if e else, contêm somente letras minúsculas.
2.6 Se um contador ou total não é inicializado, os resultados do seu programa provavelmente serão incorretos. Este é um exemplo de erro de lógica.
2.7 Em um laço controlado por um contador, como o contador do laço (que está sendo incrementado por um a cada repetição do laço) ao final das repetições fica com um valor que é um a mais que seu último valor legítimo (i.e., li no caso de “contar de 1 até 10”). usar o valor do contador em um cálculo depois do laço freqüentemente é um erro do tipo “um a mais que o esperado”.
2.8 Escolher um valor de sentinela que é também um valor de dados válido é um

erro de lógica.
2.9 Uma tentativa de dividir por zero causa um erro fatal.

CAPÍTULO 2 - ESTRUTURAS DE CONTROLE 169
2.10 Usar números de ponto flutuante de uma maneira que assume que eles são representados precisamente pode levar a resultados incorretos. Os números de ponto flutuante são representados somente aproximadamente pela maioria dos computadores.
2.11 Tentar usar o operador de incremento ou decremento em uma expressão que não seja um simples nome de variável, por exemplo, escrever ++ (x + i>, é um erro de sintaxe.
2.12 Como os valores em ponto flutuante podem ser aproximados, controlar laços com variáveis de ponto flutuante pode resultarem valores imprecisos do contador e testes de término inexatos.
2.13 Usar um operador relacional incorreto ou usar um valor final incorreto de um contador de laço na condição de um while ou estrutura for pode causar erro do tipo “sair uma iteração antes”.
2.14 Quando a variável de controle de uma estrutura for é definida na seção de inicialização do cabeçalho da estrutura for. usar a variável de controle após o corpo da estrutura é um erro de sintaxe.
2.15 Usar vírgulas em vez dos dois ponto-e-vírgulas exigidos no cabeçalho de um for é um erro de sintaxe.
2.16 Colocar um ponto-e-vírgula imediatamente à direita do parêntese à direita do cabeçalho de um for toma o corpo daquela estrutura for um comando vazio. Isto, normalmente, é um erro de lógica.
2.17 Não usar o operador relaciona! apropriado na condição de continuação do laço, em um laço que conta para trás (tal como usar incorretamente i <= 1 em um laço que conta até 1) é normalmente um erro de lógica que produzirá resultados
incorretos quando o programa for executado.
2.1X Esquecer de incluir o arquivo <cmath> em um programa que usa funções da biblioteca de matemática é um erro de sintaxe.
2.19 Esquecer um comando break, quando é necessário um em uma estrutura switch, é um erro de lógica.
2.20 Omitir o espaço entre a palavra case e o valor inteiro que está sendo testado, em uma estrutura switch. pode causar um erro de lógica. Por exemplo, escrever case3: em vez de escrever case 3: simplesmente cria um rótulo não-usado (falaremos mais sobre isto no Capítulo 18). O problema é que a estrutura switch não executará as ações apropriadas quando a expressão de controle do switch tiver o valor 3.
2.21 Não processar caracteres de nova linha e outros caracteres de espaço na entrada, quando estivermos lendo caracteres um de cada vez, pode causar erros de lógica.
2.22 Fomecer rótulos de case idênticos, em uma estrutura switch, é erro de sintaxe.
2.23 Laços infinitos são causados quando a condição de continuação do laço,

em estruturas while, for ou do/while, nunca se toma falsa. Para prevenir este erro, tenha certeza de que o valor da condição muda em algum lugar no cabeçalho
ou no corpo do laço, de maneira que a condição possa, em algum momento, se tomar falsa.
2.24 Embora 3 < x < 7 seja uma condição matematicamente correta, ela não é avaliada corretamente em C++. Use (3 < x && x < 7) para obter a avaliação apropriada em C++.
2.25 Em expressões que usam o operador &&. é possível que uma condição - chamaremos esta condição de condição dependente - possa exigir que uma outra condição seja true para que faça sentido avaliar a condição dependente. Neste caso,
a condição dependente deve ser colocada depois da outra condição ou então pode ocorrer um erro.
2.26 Usar o operador == para atribuição, ou usar o operador = para igualdade, constitui erros de lógica.
Boas práticas de programação
2.1 Aplicar consistentemente convenções razoáveis de indentação em todos os seus programas melhora muito a legibilidade do programa. Sugerimos uma marca de tabulação com um tamanho fixo de cerca de 1/4 polegada, ou três espaços por
nível de indentação.
2.2 O pseudocódigo é freqüentemente usado para “bolar” um programa durante o processo de projeto do mesmo. Após isso, o programa é convertido de pseudocódigo para C++.
2.3 Indentar ambos os comandos do corpo de uma estrutura if/else.
2.4 Se existem vários níveis de indentação, cada nível deve ser indentado pelo mesmo espaço adicional.
2.5 Colocar sempre as chaves em uma estrutura if/else (ou qualquer estrutura de controle) ajuda a prevenir sua omissão acidental, especialmente quando adicionarmos comandos à uma cláusula if ou else mais tarde.
2.6 Alguns programadores preferem digitar as chaves de início e fim de comandos compostos antes de digitar os comandos individuais dentro das chaves. Isto ajuda evitar a omissão de uma ou ambas as chaves.
2.7 Inicialize contadores e totais.
2.8 Declare cada variável em uma linha separada.
2.9 Quando executar uma divisão por uma expressão cujo valor pode ser zero, teste explicitamente esta possibilidade e tratea apropriadamente em seu programa (tal como ao imprimir uma mensagem de erro), em vez de permitir que aconteça o
erro fatal.
2.10 Solicite explicitamente ao usuário cada entrada pelo teclado. O lembrete deve indicar a forma da entrada e quaisquer valores de entrada especiais que devam ser fomecidos (tal como o valor de sentinela que o usuário deve digitar para
terminar um laço).

2.11 Em um laço controlado por sentinela, os lembretes solicitando entrada de dados deveriam lembrar explicitamente ao usuário qual é o valor de sentinela.
2.12 Não compare valores de ponto flutuante quanto à igualdade ou desigualdade. Basta testar se o valor absoluto da diferença é menor que um valor pequeno especificado.
2.13 Inicializar as variáveis quando elas são declaradas ajuda o programador a evitar os problemas de dados não-inicializados.
2.14 Operadores unários deveriam ser colocados próximos a seus operandos, sem espaços intermediários.
2.15 Controle laços controlados por contadores com valores inteiros.
2.16 Indente os comandos no corpo de cada estrutura de controle.
2.17 Coloque uma linha em branco antes e depois de cada estrutura de controle, para destacá-la no programa.
2.18 Muitos níveis de aninhamento podem tomar um programa difícil de se entender. Como regra geral, tente evitar usar mais de três níveis de indentação.
2.19 O espaçamento vertical acima e abaixo das estruturas de controle e a indentação dos corpos das estruturas de controle dentro dos cabeçalhos dessas estruturas dão aos programas uma aparência bidimensional que melhora muito sua legibilidade.
2.20 Usar o valor final na condição de um while ou estrutura for e usar o operador relacional < ajuda a evitar erros “sair uma iteração antes”. Para um laço usado para imprimir os valores 1 a 10, por exemplo, a condição de continuação de laço deveria ser counter <= 10. em vez de counter < 10 (o que é um erro “sair uma iteração antes”) ou counter < 11 (que é, no entanto, correto). Muitos programadores, entretanto, preferem a chamada contagem baseada em zero - na qual, para contar 10 vezes o laço, counter seria inicializado com zero e o teste de continuação do laço seria counter <lo.
2.21 Coloque somente expressões envolvendo as variáveis de controle nas seções de inicialização e incremento de uma estrutura for. As manipulações de outras variáveis devem aparecer ou em qualquer lugar antes do laço (se elas são executadas só uma vez, como em comandos de inicialização) ou no corpo do laço (se elas são executadas uma vez por iteração, como comandos de incremento ou decremento).
2.22 Embora o valor da variável de controle possa ser mudado no corpo de um laço for, evite fazer isso, pois esta prática pode levar a erros de lógica sutis.
2.23 Embora os comandos que antecedem um for e os comandos no corpo do for possam ser freqüentemente fundidos no cabeçalho do for, evite fazer isso, porque pode tomar o programa mais difícil de ser lido.
2.24 Limite o tamanho de cabeçalhos de estruturas de controle a uma única linha, se possível.
2.25 Não use variáveis do tipo float ou double para fazer cálculos monetários. A imprecisão dos números de ponto flutuante pode causar erros que resultarão em valores monetários incorretos. Nos exercícios, exploramos o USO de inteiros para executar cálculos monetários. Nota: bibliotecas de classes de C++ de fornecedores independentes estão disponíveis para executar corretamente

cálculos monetários.
2.26 Forneça um caso default em comandos switch. Os casos não-testados explicitamente em um comando switch sem um caso default são ignorados. Incluir um caso default chama a atenção do programador sobre a necessidade de processar condições excepcionais. Existem situações em que nenhum processamento default é necessários Embora as cláusulas case e a cláusula do caso default em uma estrutura switch possam ocorrerem qualquer ordem, é considerado uma boa prática de programação colocar a cláusula default por último.
2.27 Em uma estrutura switch. quando a cláusula default é listada por último, o comando break não é necessário. Alguns programadores incluem este break por clareza e simetria com outros casos.
2.28 Alguns programadores sempre incluem chaves em uma estrutura do/while. mesmo que as chaves não sejam necessárias. Isto ajuda a eliminar a ambigüidade entre a estrutura while e a estrutura do/while contendo um único comando.
2.29 Alguns programadores consideram que break e contine violam a programação estruturada. Como os efeitos destes comandos podem ser obtidos por técnicas de programação estruturada que logo aprenderemos, estes programadores não
usam break e continue.
Dicas de desempenho
2.1 Uma estrutura if/else aninhada pode ser muito mais rápida que uma série de estruturas de seleção única if. por causa da possibilidade de saída logo na primeira condição satisfeita.
2.2 Em uma estrutura if/else aninhada, teste as condições que são mais prováveis de serem true no início da estrutura if/else aninhada. Isso permitirá que a estrutura if/else aninhada seja executada mais rapidamente, bem como permitirá uma saída mais rápida do que testar primeiro casos pouco freqüentes.
2.3 Os programadores podem escrever programas um pouco mais rápidos e os compiladores podem compilar programas um pouco mais rapidamente quando forem usados os operadores de atribuição “abreviados”. Alguns compiladores geram
código que é executado mais rápido quando são usados operadores de atribuição “abreviados”.
2.4 Muitas das dicas de desempenho que mencionamos neste texto resultam em melhorias pequenas; assim o leitor pode ficar tentado a ignorá-las. Freqüentemente, é obtida uma melhoria de desempenho significativa quando uma melhoria, supostamente pequena, é introduzida em um laço que é repetido muitas vezes.

2.5 Evite colocar expressões cujos valores não mudam dentro de laços- mas, ainda que você assim o faço muitos dos sofisticados compiladores otimizadores atuais automaticamente colocarão tais expressões fora dos laços no código gerado em

linguagem de máquina.
2.6 Muitos compiladores contêm recursos de otimização que melhoram o código que você escreve; porém, ainda é melhor escrever um bom código desde o início.
2.7 Em situações voltadas para o desempenho, em que a memória é muito escassa ou a velocidade de execução é crucial, pode ser desejável usar tamanhos menores de inteiro.
2.8 Usar tamanhos de inteiro menores pode resultar em um programa mais lento se as instruções da máquina para manipulálos não forem tão eficientes quanto aquelas para os inteiros de tamanho natural (por exemplo, pode ser preciso fazer
extensão do bit de sinal neles).
2.9 Os comandos break e continue, quando usados corretamente, são executados mais rapidamente que as técnicas estruturadas correspondentes que logo aprenderemos.
2.10 Em expressões que usam o operador &&, se as condições separadas são independentes uma da outra, coloque à esquerda a condição com maior probabilidade de ser false. Em expressões que usam o operador 1 1, coloque mais à esquerda a
condição que tem maior probabilidade de ser true. Isto pode reduzir o tempo de execução de um programa.
Dicas de portabilidade
2.1 No padrão C++, o escopo da variável de controle declarada na seção de inicialização de uma estrutura for é diferente do escopo em compiladores de C++ mais antigos. O código C++ criado com compiladores C++ antigos pode não funcionar quando compilado com compiladores compatíveis com o padrão C++. Duas estratégias de programação defensiva que podem ser usadas para prevenir este problema são: defina variáveis de controle com nomes diferentes em cada estrutura for, ou, se você preferir usar o mesmo nome para a variável de controle em várias estruturas for, defina a variável de controle fora e antes do primeiro laço for.
2.2 As combinações de teclas para digitar fim de arquivo são dependentes do sistema.
2.3 Teste a constante simbólica EOF em lugar de -1 torna os programas mais portáveis. O padrão ANSI estabelece que EOF é um valor inteiro negativo (mas não necessariamente -1). Desta forma, EOF pode ter valores diferentes em sistemas
diferentes.
2.4 Como ints variam de tamanho entre sistemas, use inteiros long se você espera processar inteiros fora do intervalo
-32.768 a 32,767 e quer ser capaz de executar seu programa em vários sistemas de computador diferentes.
2.5 Por compatibilidade com versões antigas do padrão C++, o valor true do tipo bool pode também ser representado por qualquer valor diferente de zero e o valor false do tipo bool também pode ser representado como o valor O.
Observações de engenharia de software

2.1 Qualquer programa em C++ que venhamos a construir pode ser construído usando-se somente sete tipos diferentes de estruturas de controle (seqüência, if, if/else, switch. while, do/while e for), combinadas somente de dois modos (empilhamento de estruturas de controle e aninhamento de estruturas de controle).
2.2 Um comando composto pode ser colocado em qualquer lugar de um programa em que um comando único pode ser colocado.
2.3 Da mesma maneira que um comando composto pode ser colocado em qualquer lugar que possa ser colocado um comando único, também é possível não se ter comando algum, ou seja, um comando vazio. O comando vazio é representado colocando-se um ponto-e-vírgula (;) onde seria normalmente colocado um comando.
2.4 Cada refinamento, bem como o próprio topo, é uma especificação completa do algoritmo; só varia o nível de detalhe.
2.5 Muitos programas podem ser logicamente divididos em três fases: uma fase de inicialização, que inicializa as variáveis do programa; uma fase de processamento, que recebe como entrada valores de dados e ajusta as variáveis do programa de acordo; e uma fase de finalização, que calcula e imprime os resultados finais.
2.6 O programador termina o processo de refinamento top-down passo a passo quando o algoritmo em pseudocódigo está especificado em detalhes suficientes para o programador ser capaz de converter o pseudocódigo para C++. Implementar o programa em C++ é então, normalmente, um processo direto.
2.7 A experiência mostra que a parte mais difícil da solução de um problema em um computador está no desenvolvimento do algoritmo para a solução. Uma vez que um algoritmo correto tenha sido especificado, o processo de produzir um programa em C++ que funciona, a partir do algoritmo, é normalmente direto.
2.8 Muitos programadores experientes escrevem programas sem usar ferramentas de desenvolvimento de programas como pseudocódigo. Estes programadores pensam que sua meta essencial é resolver o problema em um computador e que escrever pseudocódigo somente atrasa a produção de resultado final. Embora isto possa funcionar para problemas simples e familiares, pode levar a sérios erros e atrasos em projetos grandes e complexos.

2.9 Colocar um ponto-e-vírgula logo depois do cabeçalho de um for é às vezes usado para criar um laço chamado de laço de retardo. Tal laço for, com uni corpo vazio, executa o número de vezes indicado, nada mais fazendo senão contar. Você pode usar um laço de retardo, por exemplo, para diminuir a velocidade de um programa que está produzindo exibições na tela muito rapidamente, para que você possa lê-las.
2.1 O Existe um certo conflito, ou incompatibilidade entre realizar uma engenharia de software de qualidade e obter o software de melhor desempenho. Freqüentemente, uma destas metas é alcançada às custas da outra.

Dicas de teste e depura ção
2.1 Os programadores escrevem normalmente condições tais como x == 7 com o nome da variável à esquerda e o da constante à direita. Invertendo estes nomes, de forma que a constante fique à esquerda e o nome da variável à direita, como era 7 == x. o programador que substituir acidentalmente o operador == pelo = estará protegido pelo compilador.
O compilador tratará esta troca como um erro de sintaxe porque só um nome de variável pode ser colocado à esquerda de um comando de atribuição. Pelo menos, este evitará a devastação potencial de um erro de lógica durante a execução.
2.2 Use seu editor de texto para procurar por todas as ocorrências de = em seu programa e conferir se você tem o operador correto em cada lugar.
Exercícios de auto-revisão
Os Exercícios 2.1 a 2. 10 correspondem às Seções 2.1 a 2.12.
Os Exercícios 2.11 a 2.13 correspondem às Seções 2.13 a 2.21.
2.1 Responda cada uma das seguintes perguntas.
a) Todos os programas podem ser escritos em termos de três tipos de estruturas de controle:
_____ e _____
b) A estrutura de seleção _____ é usada para executar uma ação quando uma condição é true e outra ação quando a condição é false.
c) A repetição de um conjunto de instruções um número de vezes específico é chamada de repetição
d) Quando não é conhecido com antecedência quantas vezes um conjunto de comandos será repetido, um valor pode ser usado para terminar a repetição.
2.2 Escreva quatro comandos de C++ diferentes, cada um somando i à variável inteira x.
2.3 Escreva comandos de C++ para implementar cada uma das seguintes frases:
a) Atribua a soma de x e y a z e incremente o valor de x por 1 depois do cálculo.
b) Teste se o valor da variável counter é maior que 10. Se for, imprima “Counter é maior que 10.”
c) Decremente a variável x por 1 e então subtrai-a da variável total.
d) Calcule o resto após q ser dividido por divisor e atribua o resultado a q. Escreva estes comandos de dois modos diferentes.
2.4 Escreva um comando de C++ para executar cada uma das seguintes tarefas.
a) Declare as variáveis sum e x como do tipo int.
b) Inicialize a variável x com 1
c) Inicialize a variável sum com 0.
d) Some a variável x à variável sum e atribua o resultado à variável sum.
e) Imprima “A soma é : “, seguido pelo valor da variável sum.
2.5 Combine os comandos que você escreveu para o Exercício 2.4 em um programa que calcula e imprime a soma dos inteiros de 1 até 10. Use a estrutura while para iterar através do cálculo e comandos de incremento. O laço deve terminar quando o valor de x se tornar 11.

2.6 Determine os valores de cada variável depois de o cálculo ser executado. Assuma que, quando cada comando começa a ser executado, todas as variáveis têm o valor inteiro 5.
a) product * x++;
b) quotient 1= ++x;

CAPÍTULO 2 - ESTRUTURAS DE CONTROLE 173
2.7 Escreva comandos simples de C++ que façam o seguinte:
a) Receba como entrada a variável inteira x com cm e».
b) Receba como entrada a variável inteira y com cm e».
c) Inicialize a variável inteira i com 1
d) Inicialize a variável inteira power com 1
e) Multiplique a variável power por x e atribua o resultado a power.
1) Incremente a variável y por 1.
g) Teste y para ver se ela é menor que ou igual a x.
h) Exiba o valor da variável inteira power com cout e «.
2.8 Escreva um programa em C++ que usa os comandos do Exercício 2.7 para calcular x elevado à potência y. O programa deve ter uma repetição controlada por uma estrutura while.
2.9 Identifique e corrija os erros em cada um dos seguintes comandos:
a) while (c <= 5) {
product = c;
++c;
b) cm « value;
c) if (gender == 1
cout« “Mulher’ « endl;
else;
cout« “Homem” « endi;
2.14) O que está errado com a seguinte estrutura de repetição while:
while (z > O)
sum + z;
2.11 Determine se as seguintes frases são verdadeiras ou falsas. Se a resposta for falsa, explique por quê.
a) O caso default é obrigatório na estrutura de seleção switch.
h) O comando break é exigido no caso default de uma estrutura de seleção switch para sair da estrutura corretamente.
c) Aexpressão (x>y && a<b) é true seaexpressãox>yé true ouaexpressãoa<bé true.
d) Uma expressão contendo o operador 1 1 é true se pelo menos um, ou ambos, os seus operandos são true.
2.12 Escreva um comando de C++, ou um conjunto de comandos de C++, para executar cada um dos seguintes comandos:
a) Some os inteiros ímpares entre 1 e 99 usando uma estrutura for. Suponha que as variáveis sum e count tenham sido declaradas como inteiras.
b) Imprima o valor 333.546372 em um campo de comprimento de 15 caracteres com precisões de 1, 2, e 3. Imprima cada número na mesma linha. Alinhe cada

número à esquerda em seu campo. Quais os três valores impressos?
c) Calcule o valor de 2 . 5 elevado à potência 3 usando a função pow. Imprima o resultado com uma precisão de 2 em um campo de 10 posições de comprimento. O que é impresso?
d) Imprima os inteiros de 1 até 20 usando um laço while e a variável contadora x. Assuma que a variável x foi declarada, mas não inicializada. Imprima somente 5 inteiros por linha. Sugestão: use o cálculo x % 5. Quando o
valor deste é 0, imprima um caractere nova linha; caso contrário imprima um caractere de tabulação horizontal.
c) Repita o Exercício 2.12 (d) usando uma estrutura for.
2.13 Ache o(s) erro(s) em cada um dos segmentos de código seguintes e explique como corrigi-lo(s).
a) x 1;
while (x <= 10);
x++;
b) for (y =.1; y 1.0; y+= .1
cout « y « endl;
c) switch (n) {
case 1:
cout « “O número é 1” « endi;
case 2:
cout « “O número é 2” « endl;

break;
default:
cout « “O número não é 1 ou 2” « endi;
break:
d) O código seguinte deve imprimir os valores de 1 a 10.
n = 1;
while (n < 10
cout « n++ « endi;
Respostas aos exercícios de auto-revisão
2.1 a) Seqtiência, seleção e repetição. b) if/else. c) Controlada por contador ou definida. d) Sentinela, sinal, flag ou dummy.
2.2 xx+1;
x += 1;
2.3 a) z = x++ + y;
b) if (count > 10
cout « “Conta é maior que 10” « endi;
c) total -= ---x;
d) q %= divisor;
q = q % divisor;
2.4 a) int sum, x;
b) x = 1;
c) sum = 0;

d) sum += x; ou sum sum + x;
c) cout « “A soma é: “ « sum « endi;
2.5 Ver abaixo.

```
1 Il Calcular a soma dos inteiros de 1 até 10
2 #include <iostream>
3
4 using std::cout;
5 using std::endl;
6
7 intmaín ()
8 {
9 int sum, x;
10 x=l;
11 sum=O;
12 while ( x<= 10)
13 sun+=x;
14 ++x;
15 1
16 cout « “A soma é: “ « sum « endl;
17 return 0;
18 }
```

2.6 a) product = 25, x = 6;
b) quotient = 0, x = 6;

CAPÍTULO 2 - ESTRUTURAS DE CONTROLE 175
2.7 a) cm » x;
b) cm » y;
c) i = 1;
d) power = 1;
e) power *z x;oupower = power * x;
f) i++;
g) if (i < y
h) cout « power « endi;
2.8 Ver abaixo.

```
1 Il eleva x à potência y
2 #include <iostream>
3
4 using std: :cout;
5 using std::cin;
6 using std: :endl;
7
8 intmain ()
9 {
10 int x, y, i, power;
li
12 i1;
```

```
13 power = 1;
14
15 cout « “Digite o valor (inteiro) da base: “;
16 cm » x;
17
18 cout « “Digite o valor (inteiro) do expoente:
19 cm » y;
20
21 while(i<y) {
22 power = x;
23 ++i;
24
25
26 cout « power « endl;
27 return 0;
28
```

2.9 a) Erro: falta a chave à direita do corpo do while.
Correção: adicione a chave à direita, de fechamento do comando de ++c;
b) Erro: usada inserção em stream em vez de extração de stream. Correção: mude « para ».
c) Erro: o ponto-e-vírgula depois de else resulta em um erro de lógica. O segundo comando de saídasempre será executado.
Correção: remova o ponto-e-vírgula depois do else.
2.10 O valor da variável z nunca é mudado na estrutura while. Então, se a condição de continuação do laço (z > 0) for true. é criado um laço infinito. Para prevenir o laço infinito, z deve ser decrementada de forma que seu valor, em algum momento, se torne menor que O.
2.11 a) Falsa. Ocaso default é opcional. Se nenhuma ação default é necessária, então não há necessidade de um caso
default.
b) Falsa. O comando break é usado para sair da estrutura switch. O comando break não é necessário quando o caso default é o último caso.
c) Falsa. Ambas as expressões relacionais devem ser true para que a expressão inteira possa ser true quando se usa o operador &&.
d) Verdadeira.

2.12 a) sum = 0;
for (count = 1; count <= 99; count += 2
sum += count;
b) cout « setiosflags (ios::fixed 1 ios::showpoint 1 ios::left)
« setprecision(1) « setw(15) « 333.546372
« setprecision(2) « setw(15) « 333.546372
« setprecision(3) « setw(15) « 333.546372
« endi;
A saída é:

333.5 333.55 333.546
c) cout « setiosflags(ios::fixed 1 ios::showpoint
« setprecision(2) « setw(10) « pow(2.5, 3
« endi;
A saída é:
15.63
d) x = 1;
while (x < 20) {
cout « x;
if (x % 5 == O
cout « endi;
ei se
cout « \t';
e)for (x= 1; x<=20; x++) {
cout « x;
if (x % 5 == O
cout « endi;
eise
cout « '\t';
ou
for (x = 1; x <= 20; x++)
if (x % 5 == O
cout « x « endi;
eise
cout « x « '\t';
2.13 a) Erro: o ponto-e-vírgula depois do cabeçalho while causa um laço infinito.
Correção: substitua o ponto-e-vírgula por
uma { ou remova tanto os ; como a }.
b) Erro: usar um número de ponto flutuante para controlar uma estrutura de repetição for. Correção: use um inteiro e
execute o cálculo apropriado, a fim de obter os valores que você deseja.
for (y = i; y 10; y++
cout « (staticcast< doubie > (y) / 10) « endi;
e) Erro: comando break faltando nos comandos para o primeiro case.
Correção: acrescente um comando break no fim dos comandos pertencentes ao primeiro case. Note que isto não é necessariamente um erro se o programador quiser que os comandos do case 2: sejam executados toda vez que os comandos do case 1: sejam executados.
d) Erro: operador relacional impróprio usado na condição de continuação de repetição do whiie. Correção: use < em lugar de <ou mude 10 para 11.
Exercícios
Os Exercícios 2.14 a 2.38 correspondem às Seções 2.1 a 2.12.
Os Exercícios 2.39 a 2.63 correspondem às Seções 2.13 a 2.21.

2.14 Identifique e corrija o(s) erro(s) em cada um dos seguintes comandos:

a) if (age >= 65);
cout « “Idade é maior que ou igual a 65” « endl;
else
cout « “Idade é menor que 65 « endi”;
b) if (age >= 65
cout « “Idade é maior que ou igual a 65” « endi;
else;
cout « “Idade é menor que 65 « endl”;
c) int x = 1, total;
while (x < 10) {
total + x;
d) While (x <= 100)
total += x;
e) while (y > O) {
cout « y « endi;
++y;
2.15 O que é impresso pelo seguinte programa?

```
1 #include <iostreain>
2
3 using std::cout;
4 using std::endl;
5
6 intmain ()
7 {
8 int y, x = 1, total = 0;
9
10 while (x<10) {
li yx*x;
12 cout « y « endl;
13 total += y;
14
15 }
16
17 cout « “O total é “ « total « endi;
18 return 0;
19 }
```

Para os Exercícios 2.16 a 2.19 execute cada um destes passos:
a) Leia a definição do problema.
b) Formule o algoritmo usando pseudocódigo e o refinamento top-down, passo a passo.
c) Escreva um programa em C++.
d) Teste, depure e execute o programa em C++.
2.16 Os motoristas estão preocupados com a quilometragem obtida por seus automóveis. Um motorista fez um controle de vários tanques de gasolina gastos, anotando os quilômetros de rodagem obtidos por tanque. Desenvolva um programa em

que recebe como entrada os quilômetros rodados e os litros gastos para cada tanque. O programa deve calcular e exibir os quilômetros por litro obtidos para cada tanque. Depois de processar todas as informações fornecidas, o programa deve calcular e imprimir os quilômetros por litro obtidos por todos os tanques combinados (média).

Digite os litros usados (-1 para fim): 12.8
Digite os quilômetros dirigidos: 287
Os quilômetros/litro para este tanque foram 22.421875
Digite os litros usados (-1 para fim) : 10.3
Digite os quilômetros dirigidos: 200
Os quilômetros/litro para este tanque foram 19.417475
Digite os litros usados (-1 para fim): 5
Digite os quilômetros dirigidos: 120
Os quilômetros/litro para este tanque foram 24.000000
Digite os litros usados (-1 para fim) : -1
A média geral de quilômetros/litro foi 21.601423

2.17 Desenvolva um programa em C++ que determinará se um cliente de uma loja de departamentos excedeu o limite de crédito em conta corrente. Para cada cliente, os seguintes fatos estão disponíveis:

a) Número da conta (um inteiro)
b) Saldo no princípio do mês
c) Total de todos os itens comprados a crédito por este cliente neste mês
d) Total de todos os créditos do cliente nesta conta neste mês
e) Limite de crédito permitido

O programa deve receber como entrada cada um destes fatos, calcular o novo saldo (= saldo inicial + débitos - créditos) e
determinar se o novo saldo excede o limite de crédito do cliente. Para aqueles clientes cujo limite de crédito for excedido, o
programa deve exibir o número da conta do cliente, o limite de crédito, o novo saldo e a mensagem “Limite de crédito excedido.”

Digite número da conta (-1 para encerrar): 100
Digite saldo inicial: 5394.78
Digite débitos totais: 1000.00
Digite créditos totais: 500.00
Digite limite de crédito: 5500.00
Conta: 100
Limite de crédito: 5500.00
Saldo: 5894.78
Limite de crédito excedido.
Digite número da conta (-1 para encerrar): 200
Digite saldo inicial: 1000.00
Digite débitos totais: 123.45
Digite créditos totais: 321.00

Digite limite de crédito: 1500.00
Digite número da conta (-1 para encerrar): 300
Digite saldo inicial: 500.00
Digite débitos totais: 274.73
Digite créditos totais: 100.00
Digite limite de crédito: 800.00
Digite número da conta (-1 para encerrar): -1

2.18 Uma grande empresa química paga a seu pessoal de vendas com base em comissão. O pessoal de vendas recebe $200 por semana mais 9% de suas vendas brutas daquela semana. Por exemplo, um vendedor que vende $5000 em substâncias químicas em uma semana recebe $200 mais 9% de $5000, ou um total de $650. Desenvolva um programa em C++ que recebe como entrada as vendas brutas de cada vendedor na semana anterior e que calcula e exibe o salário daquele vendedor. Processe os números de cada vendedor um de cada vez.

Digite vendas em dólares (-1 para encerrar): 5000.00
Salário é: $650.00
Digite vendas em dólares (-1 para encerrar): 6000.00
Salário é: $740.00
Digite vendas em dólares (-1 para encerrar): 7000.00
Salário é: $830.00
Digite vendas em dólares (-1 para encerrar): -l

2.19 Desenvova um programa em C++ que determinara o pagamento bruto para cada um de vazios empregados. A empresa paga o salário-hora “normal” para as primeiros 40 horas trabalhadas por cada empregado e paga “uma vez e meia” o salário-hora normal para todas as horas excedentes a 40 horas. Você recebe uma lista dos empregados da empresa, o número de horas que cada empregado trabalhou na semana anterior e o salário-hora de cada empregado. Seu programa deve receber estas informações como entrada para cada empregado e deve determinar e exibir o valor bruto a ser pago para o empregado.

Digite horas trabalhadas (-1 para encerrar): 39
Digite salário-hora do trabalhador ($00.00): 10.00
O salário é $390.00
Digite horas trabalhadas (-1 para encerrar): 40
Digite salário-hora do trabalhador ($00.00): 10.00
O salário é $400.00
Digite horas trabalhadas (-1 para encerrar): 41
Digite salário-hora do trabalhador ($00.00): 10.00
O salário é $415.00
Digite horas trabalhadas (-1 para encerrar): -1

2.20 O processo de achar o maior número (i.e., o máximo de um grupo de números) é usado freqüentemente em aplicações de computador. Por exemplo, um programa que determina o vencedor de uma competição de vendas receberia como entrada o número de unidades vendidas por cada vendedor. O

vendedor que vende mais unidades ganha a competição. Escreva um programa em pseudocódigo e, então, um programa em C++, que recebe como entrada uma série de tO números e que determina e imprime o maior dos números. Sugestão: seu programa deve usar três variáveis como segue:
counter: Um contador para contar até 10 (i.e., para manter o controle de quantos números foram fornecidos como entrada e para determinar quando todos os tO números foram processados).
number: O número corrente efetivo fornecido como entrada para o programa.
largest: O maior número encontrado até agora
2.21 Escreva um programa em C++ que utiliza laços e a seqüência de escape para marcas de tabulação, \t, para imprimir a seguinte tabela de valores:

```
N 10*N 100*N 1000*N
1 10 100 1000
2 20 200 2000
3 30 300 3000
4 40 400 4000
5 50 500 5000
```

2.22 Usando uma abordagem semelhante à do Exercício 2.20, ache os dois maiores valores dos 10 números. Nota: você deve fornecer cada número como entrada somente uma vez.
2.23 Modifique o programa na Fig. 2.11 para validar as entradas do mesmo. Em qualquer entrada, se o valor entrado for diferente de 1 ou 2, execute um ciclo de espera até que o usuário entre um valor correto.

2.24 O que imprime o seguinte programa?

```
1 #include <iostream>
2
3 using std::cout;
4 using std;:encil;
5
6 int main(
7 {
8 int count = 1;
9
10 while ( count < 10
11 cout« ( count % 2 ? ****“ :
12 «endi;
13 ++count;
14 )
15
16 return O;
17
```

2.25 O que imprime o seguinte programa?

```
1 #include <iostream>
2
```

```
3 using std::cout;
4 using std::endl;
5
6 intmain ()
7 {
8 int row = 10, column;
9
10 while ( row > 1
11 column = 1;
12
13 while ( co1urin < 10
14 cout « ( row % 2 ? “<“
15 ++co1imn;
16 }
17
18 --row;
19 cout « endi;
20 }
21
22 return 0;
23
```

2.26 (Outro problema de “else sem correspondente”) Determine a saída para cada um dos seguintes itens quando x é 9 e y é 11 e quando x é 11 e y é 9. Note que o compilador ignora a indentação em um programa em C++. Além disso, o compilador de C++ sempre associa um else com o if anterior a menos que seja instruído para fazer o contrário pela colocação de chaves { }. Como, à primeira vista, o programador pode não estar certo sobre a qual if um else corresponde, isto é conhecido como o problema de “else sem correspondente”. Eliminamos a indentação do código a seguir para tornar o problema mais interessante. (Sugestão: aplique as convenções de indentação que você aprendeu.)

a)
```
if (x < 10 )
if (y > 10 )
cout «“*****“ «endi;
```



```
else
cout «“#####“ «endi;
cout «“$$$$$“ «endi;
```

b)
```
if ( x < 10
if ( y > 10
cout «“*****“ «endi;
else {
cout «“#####“«endl;
```

cout «“$$$$$“ «endi;
2.27 (Outro problema de “else sem correspondente”) Modifique o código seguinte para produzir a saiaa mostraaa. Use técnicas apropriadas de indentação. Você não pode fazer qualquer mudança, exceto inserir chaves. O compilador ignora a indentação em um programa em C++. Eliminamos a indentação do código seguinte para tomar o problema mais interessante. Nota: é possível que nenhuma modificação seja necessária.

```
if (y == 8 )
if (x == 5
cout « “@@@@@“ «endl;
else
cout « “#####“ « endi;
cout « “$$$$$“ « endi;
cout « “&&&&&“ « endi;
```

a) Assumindo-se que x = 5 e y = 8, é produzida a seguinte saída.

2.28 Escreva um programa que lê o tamanho do lado de um quadrado e então imprime um quadrado daquele tamanho com asteriscos e espaços em branco. Seu programa deve funcionar para quadrados com lados de todos os tamanhos entre 1 e 20. Por exemplo, se seu programa lê um tamanho de 5, deve imprimir

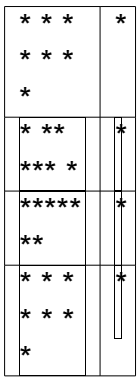
b)	Assumindo-se que x = 5 e y = 8, é produzida a seguinte saída.
@@@ @@	
e)	Assumindo-se que x = 5 e y = 8, é produzida a seguinte saída.
@@@@ @	
&&&&&	
d)	Assumindo-se que x = 5 e y = , é produzida a seguinte saída. Nota: os últimos três comandos de saída após o else são todos partes de um comando composto.
##### $$$ $$	
&&&&&	

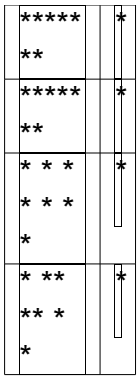
-.-
* *

2.29 Um palíndromo é um número ou uma frase de um texto que, lido ao contrário, é o mesmo que lido normalmente. Por exemplo, cada um dos seguintes inteiros de cinco dígitos são palíndromos: 12321, 55555, 45554 e 11611. Escreva um programa que lê um inteiro de cinco dígitos e determina se ele é um palíndromo. (Sugestão: use os operadores divisão e módulo para separar os dígitos de um número).
2.30 Forneça um inteiro contendo somente Os e is (i.e., um inteiro “binário”) e

imprima seu equivalente decimal. (Sugestão:
use os operadores módulo e divisão para “pegar” os dígitos do número binário, um de cada vez, da direita para a esquerda. Da mesma maneira que no sistema de numeração decimal, no qual o dígito mais à direita tem um valor posicional de 1 e o próximo dígito à esquerda tem um valor posicional de 10, depois 100, depois 1000, etc., no sistema numeração binário o dígito mais à direita tem um valor posicional de l,o próximo dígito à esquerda tem um valor posicional de 2, depois 4, depois 8, etc. Assim, o número decimal 234 pode ser interpretado como 4 * 1 + 3 * 10 + 2 * 100. O equivalente decimal do número binário 1101 é: 1 * 1+0*2+1 *4+i*8ou 1+O+4+8ou13).
2.31 Escreva um programa que exibe o seguinte padrão de tabuleiro de damas

Seu programa deve usar somente três comandos de saída, um de cada uma das formas seguintes:
cout « “*
cout«
cout « endi;
2.32 Escreva um programa que fica imprimindo os múltiplos do inteiro 2, ou seja, 2, 4, 8, 16, 32, 64, etc. Seu laço não deve terminar (i.e., você deve criar um laço infinito). O que acontece quando você executa este programa?
2.33 Escreva um programa que lê o raio de um círculo (como valor double) e calcula e imprime o diâmetro, a circunferência e a área. Use o valor 3,14159 para it.
2.34 O que está errado com o comando seguinte? Forneça o comando correto para fazer o que o programador provavelmente estava tentando fazer.
cout « ++(x + y);
2.35 Escreva um programa que lê três valores double diferentes de zero e que determina e imprime se eles podem representar os lados de um triângulo.
2.36 Escreva um programa que leia três inteiros diferentes de zero e que determina e imprime se eles poderiam ser os lados de um triângulo retângulo.
2.37 Uma empresa quer transmitir dados pelo telefone, mas está preocupada que seus telefones possam ser grampeados. Todos os seus dados são transmitidos como inteiros de quatro dígitos. Eles lhe pediram para escrever um programa que codifica





CAPÍTULO 2 - ESTRUTURAS DE CONTROLE 183
seus dados de forma que possam ser transmitidos com mais segurança. Seu programa deve ler um inteiro de quatro dígitos e codificá-lo como segue: substitua cada dígito por (dígito mais 7) módulo 10. Então, troque de posição o primeiro dígito com o terceiro e o segundo dígito com o quarto e imprima o inteiro codificado. Escreva um programa separado, que recebe como entrada um inteiro de quatro dígitos codificado e o decodifica para obter o número original.
2.38 O fatorial de um inteiro n, não-negativo, é representado por n! (pronunciado “fatorial de n”) e é definido como segue:
n! = n (n- 1). (n- 2) . . . . (n-1) (para valores de n maiores que ou iguais a 1)
e
n! = 1 (para n 0).
Por exemplo, 5! = 5.4.3. 2 1, que é 120.
a) Escreva um programa que lê um inteiro não-negativo e que calcula e imprime seu fatorial.
b) Escreva um programa que estima o valor da constante matemática e usando a fórmula:
e= 1
e) Escreva um programa que calcula o valor de e usando a fórmula
e= 1
2.39 Ache o(s) erro(s) em cada um dos seguintes itens:
a) for (x = 100, x >= 1, x++
cout « x « endl;
b) O código seguinte deve imprimir se o value inteiro é par ou ímpar:
switch(value % 2) {
case 0:
cout « “Inteiro par” « endi;
case 1:
cout « “Inteiro ímpar” « endi;
e) O código seguinte deve exibir os inteiros ímpares de 19 até 1:
for (x19;x>1; x+2)
cout « x « endi;
d) O código seguinte deve exibir os inteiros pares de 2 até 100:
counter = 2;
do {

```
cout « counter « endi;
counter += 2;
} While ( counter < 100 );
```

2.40 Escreva um programa que soma uma seqüência de inteiros. Assuma que o primeiro inteiro lido especifica o número de valores restantes a ser fornecidos. Seu programa deve ler só um valor por comando de entrada. Um seqüência típica de entrada poderia ser

5 100 200 300 400 500

onde o número 5 indica que os 5 valores subseqüentes devem ser somados.

2.41 Escreva um programa que calcula e imprime a média de vários inteiros. Assuma que o último valor digitado é a sentinela

9999. Uma seqüência típica de entrada poderia ser

10 8 11 7 9 9999

indicando que a média de todos os valores precedentes a 9999 deve ser calculada.

2.42 O que faz o seguinte programa?

```
1 #include <iostream>
2
3 using std::cout;
4 using std::cin;
5 using std::endl;
6
7 intmain ()
8 {
9 int x, y;
10
11 cout « “Digite dois inteiros no intervalo 1-20: “;
12 cin»x»y;
13
14 for ( int i = 1; i < y; i++
15
16 for ( int j = 1; j <= x; j++
17 cout « ‘@‘;
18
19 cout « endi;
20
21
22 return 0;
23 }
```

2.43 Escreva um programa que acha o menor de vários inteiros. Suponha que o primeiro valor lido especifica o número de valores restantes e que o primeiro número não é um dos inteiros a comparar.

2.44 Escreva um programa que calcula e imprime o produto dos inteiros ímpares de 1 até 15.

2.45 A função fatorial é freqüentemente usada em problemas de probabilidade. O fatorial de um inteiro positivo n (escrito n!
e pronunciado “fatorial de n”) é igual ao produto dos inteiros positivos de 1 até n. Escreva um programa que calcula os fatoriais
dos inteiros de 1 até 5. Imprima o resultado em formato de tabela. Que dificuldade pode lhe impedir de calcular o fatorial de 20?
2.46 Modifique o programa de juros compostos da Seção 2.15 para repetir seus passos para taxas de juros de 5 por cento, 6 por cento, 7 por cento, 8 por cento, 9 por cento e 10 por cento. Use um laço for para fazer variar a taxa de juros.
2.47 Escreva um programa que imprime os padrões seguintes, separadamente, um abaixo do outro. Use laços for para gerar os padrões. Todos os asteriscos (*) devem ser impressos por um único comando, da forma cout « * ; (isto faz com que os asteriscos sejam impressos lado alado). Sugestão: os últimos dois padrões exigem que cada linha comece com um número apropriado de espaços em branco. Crédito extra: combine seu código dos quatro problemas separados em um único programa que imprime todos os quatro padrões lado a lado, fazendo um uso inteligente de laços for aninhados.
(A) (B) (C) (D)

```
* ********** ********** *
** ********* ********* **
*** ***
**** ******* ******* ****
***** ****** ****** *****
****** ***** ***** ******
******* ****
******** *** *** ********
********* ** ** *********
* **********
```


2.4$ Uma aplicação interessante de computadores é desenhar gráficos e gráficos de barras (às vezes chamados de “histogramas”). Escreva um programa que lê cinco números (cada um entre 1 e 30). Para cada número lido, seu programa deve imprimir uma linha contendo aquela quantidade de asteriscos adjacentes. Por exemplo, se seu programa lê o número sete, deve imprimir *******.
2.49 Uma loja de vendas pelo correio vende cinco produtos diferentes, cujos preços de varejo são: produto 1 - $2.Q8, produto
2- $4.50, produto 3 -$9.98, produto 4- $4.49 e produto 5- $6.87. Escreva um programa que lê uma série de pares de números,
como segue:
a) Número do produto
b) Quantidade vendida em um dia
Seu programa deve usar um comando switch para ajudar a determinar o preço de varejo para cada produto. Seu programa deve calcular e exibir o valor de varejo total de todos os produtos vendidos na última semana.

2.50 Modifique o programa da Fig. 2.22, de forma que calcule a nota média da turma. Uma nota ‘A’ vale 4 pontos, ‘B ‘vale 3 pontos, etc.
2.51 Modifique o programa na Fig. 2.21, de maneira que só use inteiros para calcular os juros compostos. (Sugestão: trate
todas as quantidades monetárias como números inteiros de centavos. Então, “quebre” o resultado em sua parte em dólares e
sua parte em centavos, usando as operações divisão e módulo. Insira um ponto entre ambas.)
2.52 Assuma que i = 1, j = 2, k = 3 e m = 2. O que imprime cada um dos seguintes comandos? Os parênteses são necessários em cada caso?
a) cout« (i == 1) « eridi;
b) cout« (j == 3) « endi;
e) cout« (i >= 1 && j < 4) « endi;
d) cout« (m <= 99 && k < m) « endl;
e) cout« (j >= i II k==m) « endl;
cout« (k + m < j 3 - j >= k) « endi;
g) cout« (!) « endi;
h) cout« ((j - m)) «endi;
i) cout« (! (k > m)) « eridi;
2.53 Escreva um programa que imprime uma tabela dos equivalentes binários, octais e hexadecimais dos números decimais, no intervalo 1 a 256. Se não estiver familiarizado com estes sistemas de numeração, leia primeiro o Apêndice C.
2.54 Calcule o valor de t com a série infinita
44444
= 4--+---÷---+.
3 5 7 9 11
Imprima uma tabela que mostra o valor de it aproximado por 1 termo desta série, por dois termos, por três termos, etc. Quantos termos desta série você tem que usar antes de obter pela primeira vez 3,14 ? 3,141 ? 3,1415 ? 3,14159 ?
2.55 (Triplas pita góricas) Um triângulo retângulo pode ter lados que são todos inteiros. O conjunto de três inteiros que representam os lados de triângulo retângulo é chamado de tripla pitagórica. Estes três lados devem satisfazer a relação de que a soma dos quadrados de dois dos lados é igual ao quadrado da hipotenusa. Ache todas as triplas pitagóricas para ladol, lado2 e hipotenusa, todos menores que 500. Use um laço for triplamente aninhado que testa todas as possibilidades. Este é um exemplo de calcular pela “força bruta”. Você aprenderá, em cursos de Ciência da Computação mais avançados, que existem muitos problemas interessantes para os quais não existe nenhuma abordagem algorítmica conhecida além de usar a pura força bruta.
2.56 Uma empresa paga a seus empregados como gerentes (que recebem um salário semanal fixo), trabalhadores horistas (que recebem um valor fixo por hora para as primeiras 40 horas que trabalham e “metade a mais”, ou veja, 1,5 vezes sua remuneração horária normal, por hora extra trabalhada), trabalhadores por comissão (que recebem $250 mais 5,7% de sua venda semanal bruta) ou “tarefeiros” (que recebem uma quantia fixa de dinheiro por

iteni, para cada uma dos itens que produzem - cada tarefeiro nesta empresa trabalha só com um tipo de item). Escreva um programa para calcular o pagamento semanal para cada empregado. Você não sabe o número de empregados com antecedência. Cada tipo de empregado tem seu próprio código de pagamento: os gerentes têm código de pagamento 1, os horistas têm código 2, os comissionados tem código 3 e os tarefeiros têm código 4. Use um switch para calcular o pagamento de cada empregado baseado no código de paga-

mento do empregado. Dentro do switch, solicite ao usuário (i.e., o funcionário que cuida da folha de pagamento) que digite os dados adequados de que seu programa necessita para calcular o pagamento de cada empregado com base no seu código de pagamento.

2.57 (Leis de De Morgan) Neste capítulo, discutimos os operadores lógicos && e . As Leis de De Morgan podem às vezes tornar mais conveniente para nós a forma de formular uma expressão lógica. Estas leis afirmam que a expressão (condição] && condição2) é logicamente equivalente à expressão (!condiçãol 1 1 ‘condição2). Da mesma forma, a expressão ! (condição] 1 condição2) é logicamente equivalente à expressão (‘condiçãol && ! condição2). Use as Leis de De Morgan para escrever expressões equivalentes a cada uma das expressões seguintes e, então, escreva um programa para mostrar que a expressão original e a nova expressão, em cada caso, são equivalentes:

a) !(x<5) && !(y>=7)
b) ‘(a==b) II ‘(g !=5)
c) !((x<=8) && (y>4))
d)!((i>4)II(j<=6))

2.58 Escreva um programa que imprime a forma losango abaixo. Você pode usar comandos de saída que imprimem um único asterisco (*) ou um único espaço em branco. Maximize seu uso de repetições (com estruturas for aninhadas) e minimize o número de comandos de saída.

*****

*

2.59 Modifique o programa que você escreveu para o Exercício 2.58 para ler um número ímpar, no intervalo de 1 a 19, que especifica o número de linhas do losango. Seu programa deve, então, exibir um losango do tamanho apropriado.

2.60 Uma crítica ao comando break e ao comando continue é que não são estruturados. Realmente, os comandos break e continue podem sempre ser substituidos por comandos estruturados, embora fazer isto seja às vezes algo complicado. Descreva, em linhas gerais, como você eliminaria todos os comandos break de um laço em um programa e substituiria esse comando por algum equivalente estruturado. (Sugestão: o comando break sai de um laço dentro do corpo do laço. O outro caminho para sair do laço é não satisfazer o

teste de continuação do laço. Considere usar no teste de continuação do laço um segundo teste que indica “saída prematura por causa de uma condição de ‘quebra”). Use a técnica que você desenvolveu aqui para remover o comando break do programa da Fig. 2.26.

2.61 O que faz o segmento do seguinte programa?

```
1 for ( i = 1; i <= 5; i++ ) {
2 for(j=1;j<=3;j++){
3 for (k = 1; k <= 4; k++)
4 cout «‘*‘;
5 cout « endi;
6 }
7 cout « endi;
8 }
```

2.62 Descreva, em linhas gerais, como você removeria qualquer comando continue de um laço em um programa e substituria esse comando por algum equivalente estruturado. Use a técnica que você desenvolveu aqui para remover o comando continue do programa da Fig. 2.27.

2.63 (Canção “The Twelve Days of Christmas”) Escreva um programa que usa estruturas de repetição e estruturas switch para imprimir a canção “The Twelve Days of Christmas”. Uma estrutura switch deve ser usada para imprimir o dia (i.e., “Primeiro,” “Segundo,” etc.). Uma estrutura switch separada deve ser usada para imprimir o restante de cada verso.

*

****

** *** * * *

On the first day of Christmas, my true love gave to me a partridge in a pear tree.
On Lhe second day of Christmas, my true love gave Lo me two turtie doves,
and a partridge in a pear tree.
On the third day of Christmas, my true love gave to me three french hens.
two turtie doves,
and a partridge in a pear tree.
On the fourth day of Christmas, my troe love gave Lo me tour calling birds.
three french bens,
two turtie doves,
and a partridge in a pear tree.
On Lhe tifth day of Christmas, my true love gaveto me five gold rings,
four calling birds,
three french hens,
two turtie doves,
and a partridge in a pear tree.
Ou Lhe sixth day of Christmas, my true love gave Lo me six geese alaying,

tive gold rings,
four ca]ling birds,
three french hens,
two turtie doves,
and a partridge in a pear tree.
On the seventh day of Christmas, my true love gave to me seven swans a-swimming,
six geese a-laying,
tive gold rings,
four calling birds,
three french bens,
two turtie doves,
and a partridge in a pear tree.
On the eighth day of Christmas, my true love gave Lo me eight maids a-milking.
seven swans a-swimming,
six geese a-laying.
five gold rings,
four calling birds,
three french bens,
two turtie doves,
and a partridge in a pear tree.
On the ninth day of Christmas, my true love gave to me níne ladies waiting,
eight maids a-milking,
seven swans a-swimming,
six geese a-laying,
five gold rings,
four calling rings,
three french bens,
two turtie doves,
and a partridge in a pear tree.
On the tenth day of Christmas, my true love gave to me ten Jords a-leaping,
nine ladies waiting,
eight maids a-milking,
seven swans a-swimming,
six geese a-laying,
five gold rings,
four calling birds,
three french bens,
two turtie doves,
and a partridge in a pear tree,
On the eeventh day of Christmas, my true love gave Lo me eleven pipers piping.
teu lords a-leaping,
nine ladies waiting,
eight maids a-milking,

seven swans a-swimming,
six geese a4aying,
five gold rings,
four caHing birds,
three french bens,
two turtie doves,
and a partridge in a pear tree.
On the twelfth day of Christmas, my true love gave to me twelve drummers drumming,
eleven pipers piping,
ten lords a-leaping,
nine ladies waiting,
eight maids a-milking,
seven swans a-swimming,
six geese a-laying.
five gold rings,
four calling birds,
three french bens,
two turtie doves,
and a partridge in a pear tree.
O Exercício 2.64 corresponde à Seção 2.22, “Pensando em objetos”.
2.64 Descreva em 200 palavras, ou menos, o que é um automóvel e o que faz, Liste os substantivos e verbos separadamente. No texto, declaramos que cada substantivo pode corresponder a um objeto que necessita ser construído para implementar um sistema, neste caso um carro. Escolha cinco dos objetos que você listou e, para cada um, liste vários atributos e vários comportanientos. Descreva brevemente como estes objetos interagem uns com os outros e outros objetos em sua descrição. Você acabou de executar vários dos passos-chave em um projeto orientado a objetos típico.
2.65 (O Problema de Peter Minuit) Uma lenda conta que, em 1ó26, Peter Minuit comprou Manhattan por $24,00, dando
outros bens em troca. Ele fez um bom investimento? Para responder esta pergunta, modifique o programa de juros compostos da
Fig. 2.21 para começar com um principal de $24,00 e para calcular a quantia paga em juros sobre o depósito se esse dinheiro
continuasse aplicado, em depósito, até este ano (374 anos, até 2000). Rode o programa com taxas de juro de de 5%, 6%, 7%, 8%,
9% e 10%, para observar as maravilhas dos juros compostos.

3

Funções

Objetivos

• Entender como construir programas modularmente a partir de pequenas partes chamadas funções.
• Ser capaz de criar novas funções.
• Entender os mecanismos usados para passar informações entre funções.
• Apresentar técnicas de simulação usando geração aleatória de números.
• Entender como a visibilidade dos identificadores limitada a regiões específicas dos programas.
• Entender como escrever e usar funções que chamam a si próprias.

A forma nunca segue afunção.
Louis Henri Suilivan
E pluribus unum.
(Um composto por muitos.)
Virgil
Chama o dia de ontem, faze com que o tempo atrás
retorne.
William Shakespeare
Richard II
Chamem-me Ismael.
Herman Melvilie
Moby Dick

Quando você me chamar assim, sorria.
Owen Wister

Visão Geral

3.1 Introdução
3.2 Componentes de programas em C++÷
3.3 Funções da biblioteca matemática
3.4 Funções
3.5 Definições de funções
3.6 Protótipos de funções
3.7 Arquivos de cabeçalho
3.8 Geração de números aleatórios
3.9 Exemplo: um jogo de azar e apresentando enum
3.10 Classes de armazenamento
3.11 Regras de escopo
3.12 Recursão
3.13 Exemplo usando recursão: a série de Fibonacci
3.14 Recursão *versus* iteração
3.15 Funções com listas de parâmetros vazias
3.16 Funções *mime*

3.17 Referências e parâmetros por referência
3.18 Argumentos default
3.19 Operador unário de resolução de escopo
3.20 Sobrecarga de funções
3.21 Gabaritos de funções
3.22 (Estudo de caso opcional) Pensando em objetos: identificando os atributos de uma classe

Resumos Terminologia Erros comuns de programação Boas práticas de programação Dicas de desempenho Dicas de portabilidade Dicas de teste e depura ção. Observações de engenharia de software. Exercícios de auto-revisão . Respostas dos exercícios de auto-revisão • Exercícios

3.1 Introdução

A maioria dos programas de computador que resolvem problemas do mundo real são muito maiores do que os programas apresentados nestes primeiros capítulos. A experiência tem mostrado que a melhor maneira de desenvolver e manter um programa grande é construí-lo a partir de pequenas partes ou componentes, sendo cada uma delas mais fácil de manipular que o programa original. Essa técnica é chamada de *dividir para conquistar.* Este capítulo descreve os recursos da linguagem C++ que facilitam o projeto, a implementação, a operação e a manutenção de programas grandes.

3.2 Componentes de programas em C++

Os módulos em C÷+ são chamados de *funções* e *classes.* Os programas em C++ são escritos tipicamente combinando-se funções novas que o programador escreve com "funções pré-empacotadas" disponíveis na *biblioteca padrão de C+ +* e combinando-se classes novas que o programador escreve com "classes pré-empacotadas", disponíveis em várias bibliotecas de classes. Neste capítulo, vamos nos concentrar em funções; discutiremos classes em detalhes a partir do Capítulo 6.

1

1

As funções são normalmente chamadas em um programa escrevendo-se o nome da função seguido pelo parêntese esquerdo, seguido pelo *argumento* (ou uma lista de argumentos separados por vírgulas) da função, seguido pelo parêntese direito. Por exemplo, um programador que desejasse calcular e imprimir a raiz quadrada de 900 . O poderia escrever

```
cout « sqrt ( 900.0 );
```

Quando este comando é executado, a função sqrt da biblioteca matemática é chamada para calcular a raiz quadra- dado número contido entre os parênteses (900 . 0). O número 900 .0 é o *argumento* da função sqrt. O comando anterior imprimiria 30 . 00. A função sqrt recebe um argumento do tipo double e retorna um resultado do tipo double. Todas as funções da biblioteca matemática retornam o tipo de dado double. Para usar as funções da biblioteca matemática, inclua o arquivo de cabeçalho <cmath>.

Erro de comum programação 3.1

Esquecer de incluir o arquivo de cabeçalho <cmath> *ao usar funções da biblioteca matemática é um erro de sintaxe. Para cada função da biblioteca padrão usada em um programa deve ser incluído um arquivo de cabeçalho padrão.*

Os argumentos de funções podem ser constantes, variáveis ou expressões. Se cl = 13 . 0. d = 3 . O e f = 4 . 0. o comando

```
cout « sqrt( cl + d * f );
```

calcula e imprime a raiz quadrada de 13 . o + 3 . O * 4 O = 25 . 0, ou seja, 5 (porque C++ normalmente não introduz zeros à direita nem a casa decimal, em um número em ponto flutuante que não tem parte fracionária).

Algumas funções da biblioteca matemática de C++ estão resumidas na Fig. 3.2. Na figura, as variáveis x e y são do tipo double.

1

Fig. 3.2 Funções comumente usadas da biblioteca matemática.

Método	Descrição	Exemplo
ceil (**x**)	arredondax para o menor inteiro não menorquex	ceil (**9.2) é** 10.0 ceil(-9.8) *é* -9.0
cos (**x**)	co-seno trigonométrico de x (x em radianos)	cos (**O .O**) é 1. O
exp (**x**)	função exponencial e	exp (1.0) é 2 .71828 exp(2.0) é **7.38906**
fabs (**x)**	valor absoluto de x	fabs (**5. 1) é 5** .**1** fabs(0.0) é **0.0** fabs(-8.76) **é 8.76**
floor (**x**)	arredonda x para o maior inteiro não maiorquex	floor (9 .2) **é 9** .0 floor(-9.8) **é** -10.0
fmod(x, y	) restodex/ycomonúmerode ponto flutuante	fmod(13.657, 2.333) é **1.992**
log (**x**)	logaritmo natural de x (base *e)*	**log** (**2** .**718282**) **é 1** .**O log(7.389056**) é **2.0**
loglO (**x**)	logaritmo de x (base 10)	10gb (**10** .O) é **1** .**O** loglO(100.0) é **2.0**
pow(x, y)	x elevado à potência dey (xY)	**pow(2,7**) é **128 pow(9, .5**) **é 3**
sin (**x**)	seno trigonométrico dcx (x em radianos)	sin (**0.0**) é **O**
sqrt(x)	raizquadradadex	sqrt(900.0) **é** 30.0 sqrt(9.0) é **3.0**
tan (**x**)	tangente trigonométrica de x (x em radianos)	tan (**O .O**) é **O**


```
12 int main()
13
14 cout « "Digite o comprimento do lado do seu cubo:
15
16 double side;
17
18 cm » side;
19 cout « 'Volume do cubo com lado
20 « side « -é-« cube( side )« endl;
21
22 return 0;
23
Digite o comprimento do lado do seu cubo: 3.5
Volume do cubo com lado 3.5 é 42.875
```

Figura 3.19 Usando uma função **inline** para calcular o volume de um cubo (parte 2 de 2).

3.17 Referências e parâmetros por referência

Duas maneiras de invocar funções, em muitas linguagens de programação, são *chamadas por valor* e *chamadas por referência.* Quando os argumentos são passados através de uma chamada por valor, é feita uma *cópia* do valor dos argumentos e a mesma é passada para a função chamada. Modificações na cópia não afetam o valor original da variável, na função que realizou a chamada. Quando um argumento é passado através de uma chamada por referência, a função chamadora permite que a função chamada realmente modifique o valor original da variável. Isso evita os *efeitos colaterais* acidentais que tão freqüentemente atrapalham e retardam o desenvolvimento de sistemas corretos e confiáveis de software. Até aqui, neste capítulo, os argumentos nos programas mostrados foram passados através de chamadas por valor.

Dica de desempenho 3.10

Uma desvantagem da chamada por valor é que, se um grande item de dados está sendo passado, copiar esses dados pode consumir um tempo de processamento considerável.
Nesta seção, introduzimos *parâmetros referência* - o primeiro de dois modos que C++ oferece para executar uma chamada por referência. Com uma chamada por referência, a função que chama dá à função chamada a possibilidade de acessar e modificar os dados da função que chamou, caso a função chamada assim o queira.
Dica de desempenho ***3.11***
Chamada por referência **é** *bom sob o ponto de vista do desempenho porque elimina o* overhead *de copiar grandes volumes de dados.*
Observação de engenharia de software 3.18
___ Chamada por referência pode diminuir a segurança porque afunção chamada pode corromper os dados da função que chamou.
Agora, mostraremos como obter as vantagens de desempenho de uma chamada por valor ao mesmo tempo que obtemos os benefícios importantes, sob o ponto de vista da engenharia de software, da proteção dos dados da função que chamou de uma possível corrupção.
Um parâmetro passado por referência é um *alias* do seu argumento correspondente. Para indicar que um parâmetro é passado por referência, simplesmente coloque um "e comercial" (&) depois do tipo do parâmetro, no protótipo da função; use a mesma convenção quando listar o tipo do parâmetro no cabeçalho da função. Por exemplo, a declaração int &count

Agora que estamos discutindo a omissão de algumas coisas, deve ser observado que a função definida em um arquivo antes de qualquer chamada à função não requer um protótipo de função separado. Neste caso, o cabeçalho da função funciona como o protótipo da função.
Erro comum de programação 3.25
A *menos que sejam fornecidos protótipos de função para todas as funções ou, então, que cada função seja definida antes de ser usada, os programas em C+ + não são compilados.*

3.16 *Funções mime*

Implementar um programa como um conjunto de funções é bom sob o ponto de vista de engenharia de software, mas chamadas a funções envolvem um *overhead* durante a execução. C++ provê *funções mIme* para ajudar a reduzir o *overheadda* chamada a uma função, especialmente para pequenas funções. O qualificador mime antes do tipo de retorno da função na definição da função "aconselha" o compilador a gerar uma cópia do código da função no próprio local (quando apropriado), para evitar uma chamada de função. A desvantagem é que múltiplas cópias da função são inseridas no programa (tornando, assim, o programa maior) em vez de ter uma única cópia da função para a qual o controle é passado cada vez que é chamada a função. O compilador pode ignorar o qualificador mime e, normalmente, faz isto para todas as funções, exceto as menores.
Observação de engenharia de software 3.16
Qualquer mudança em uma em uma função mime *pode exigir que todos os clientes da função sejam recompilados. Isso pode ser relevante em algumas situações de desenvolvimento e manutenção de programas.*
Boa prática de programação 3.11
O qualificador mime *deveria ser usado somente com funções pequenas, freqüentemente usadas.*
Dica de desempenho 3.9
f *Usar funções* mime *pode reduzir o tempo de execução, mas pode aumentar o tamanho do programa.*
A Fig. 3.19 usa a função *mime* cube para calcular o volume de um cubo de lado s. A palavra-chave cons t na lista de parâmetros da função cube diz ao compilador que a função não modifica a variável s. Isto garante que o valor de s não seja alterado pela função quando o cálculo é executado. A palavra-chave const é discutida em detalhes nos Capítulos 4, 5 e 7.
Observação de engenharia de software 3.17
___ Muitos programadores não se preocupam em declarar parâmetros passados por valor como const, *ainda que a função chamada não deva modificar o argumento passado. A palavra-chave* const *está somente protegendo uma cópia do argumento original, não o próprio argumento original.*

Fig. 3.19 Usando uma função mime para calcular o volume de um cubo (parte 1 de 2).

```
1  // Fig. 3.19: fig03_19.cpp
2  // Usando uma função mime para calcular
3  // o volume de um cubo.
```

```
4  #include <iostreain>
5
6  using std::cout;
7  using std::cin;
8  using std::endi;
9
10 mime doubie cube( const double s ) { return s * s * s;
11
```


3.15 Funções com listas de parâmetros vazias

Em C++, uma listas de parâmetros vazia *é* especificada escrevendo-se **void.** ou simplesmente nada, entre os parênteses. A declaração

void print

especifica que a função print não recebe quaisquer argumentos e não retorna um valor. A Fig. 3.18 mostra as duas maneiras de declarar e usar, em C++, funções que não recebem argumentos.

```
1 II Fig. 3.18: figO3l8.cpp
2 II Funções que não recebem argumentos
3 #include <iostreain>
4
5 using std: :cout;
6 us±ng std: :endl;
7
8 void functionlQ;
9 void function2 ( void );
10
11 int main O
12
13 functionlO;
14 function2O;
15
16 return 0;
17
18
19 void functionl()
20
21 cout « "functionl não recebe argumentos" « endl;
22
23
24 void function2( void
25
26 cout « "function2 também não recebe argumentos" « endl;
27
```

Fig. 3.18 Duas maneiras de declarar e usar funções que não recebem argumentos.

Boa prática de programação 3.10

Sempre inclua protótipos de função, ainda que seja possível omiti-los quando as funções são definidas antes que sejam usadas. Incluir os protótipos evita amarrar o código à ordem na qual as funções são definidas (que pode facilmente mudar à medida que um programa evolui).

Dica de portabilidade 3.3

__O significado de uma lista de parâmetros de função vazia em C++ é completamente diferente do que em C. Em C, significa que toda a verificação de argumentos é desativada (i.e., a chamada dafirnção pode passar quaisquer argumentos que queira). Em C++, significa que afunção não recebe argumentos. Assim, a compilação em C++ de programas escritos em C que usam este recurso pode acusar erros de sintaxe.

functionl	não recebe	argumentos	
furzction2	também não	recebe	argumentos

Fig. 3.17 Resumo dos exemplos e exercícios sobre recursão no livro.

Vamos reiterar algumas observações que fazemos repetidamente ao longo do livro. A boa engenharia de software é importante. O desempenho elevado é importante. Infelizmente, estes objetivos freqüentemente são conflitantes. A boa engenharia de software é fundamental para tornar administrável a tarefa de desenvolvimento dos sistemas de software maiores e mais complexos de que necessitamos. Um alto desempenho destes sistemas é fundamental para concretizar os sistemas do futuro, que exigirão capacidades de computação do hardware cada vez maiores. Onde se encaixam as funções neste cenário?

Observação de engenharia de software ***3.15***

___Funcionalizar programas de maneira clara e hierárquica promove a boa engenharia de software. Mas isso tem um preço.

Dica de desempenho 3.8

___Um programa muito dividido em funções - em comparação com um programa monolítico, não dividido em funções (i.e., composto de um único módulo) - executa um número potencialmente maior de chamadas de funções e isto consome tempo de execução e espaço do(s) processador(es) de um computador Porém, programas monolíticos são difíceis de programai testa,ç depuran manter e de acrescentar novas funcionalidades.

Assim, divida seus programas em funções de forma criteriosa, sempre tendo em mente o delicado equilíbrio entre desempenho e a boa engenharia de software.

Capítulo	Exemplos e exercícios de recursão
Capítulo 3	Função fatorial Função de Fibonacci Máximo divisor comum Soma de dois inteiros Multiplicação de dois inteiros Elevar um inteiro a uma potência inteira Torres de Hanói Imprimir dados udos do teclado na ordem inversa Visualizar a recursão
Capítulo 4	Soma dos elementos de um array Imprimir um array Imprimir um array na ordem inversa Imprimir um *string* na ordem inversa Verificar se um *string* ***é*** um palíndromo Valor mínimo em um array Classificação seletiva (“sort de seleção”) *Eight Queens* (“Oito Damas”) Pesquisa linear Pesquisa binária
Capítulo 5	***Quicksort*** Percorrer um labirinto Imprimir na ordem inversa um *string* lido do teclado
Capítulo ***15***	Inserção em uma lista encadeada Exclusão de uma lista encadeada Pesquisa em uma lista encadeada Imprimir uma lista encadeada na ordem inversa Inserção em uma árvore binária Percorrer uma árvore binária no modo *preorder* Percorrer uma árvore binária no modo *morder* Percorrer uma árvore binária no modo *postorder*

chamadas, ou seja, o número de chamadas recursivas que serão executadas para calcular o n-ésimo número de Fibonacci é da ordem de 2’. Isso foge rapidamente do controle. Calcular apenas o 20° número de Fibonacci exigiria na ordem de 220 ou cerca de um milhão de chamadas, calcular o **300** número de Fibonacci exigiria na ordem de 2° ou cerca de um bilhão de chamadas, e assim por diante. Os cientistas da computação chamam isso de *complexidade exponencial.* Problemas dessa natureza humi Iham até mesmo os computadores mais poderosos! Questões de complexidade em geral e complexidade exponencial em particular são analisados detalhadamente em uma disciplina avançada de Ciência da Computação chamada “Complexidade de algoritmos”.

Dica de desempenho 3.6

f *.z Evite programas recursivos semelhantes ao da função* fibonacci *que resultem em uma “explosão” exponencial de chamadas.*

3.14 Recursão *versus* iteração

Nas seções anteriores, estudamos duas funções que podem ser facilmente implementadas, tanto de uma maneira recursiva como iterativa. Nesta seção, comparamos os dois métodos e analisamos os motivos que levam o programador a escolher um método em detrimento do outro para uma determinada situação.

Tanto a iteração como a recursão se baseiam em uma estrutura de controle; a iteração usa uma estrutura de repetição; a recursão usa uma estrutura de seleção. Tanto a iteração como a recursão envolvem repetições: a iteração usa explicitamente uma estrutura de repetição; a recursão obtém repetição por intermédio de chamadas repetidas de funções. A iteração e a recursão envolvem um teste de encerramento: a iteração termina quando uma condição de continuação do laço se torna falsa; a recursão termina quando um caso básico é reconhecido. Tanto a iteração com repetição controlada por contador quanto a recursão chegam gradualmente ao fim: a iteração fica modificando um contador até que ele assuma um valor que faça a condição de continuação do laço se tomar falsa; a recursão fica produzindo versões mais simples do problema original até chegar ao caso básico. Tanto a iteração como a recursão podem ocorrer infinitamente: ocorre um laço infinito com a iteração se o teste de continuação do laço nunca se tomar falso; ocorre um laço infinito com a recursão se a etapa de recursão não reduzir gradualmente o problema de forma que ele convirja para o caso básico.

A recursão apresenta muitos aspectos negativos. Ela usa repeticlamente o mecanismo e, em conseqüência, o *overhead,* de chamada de funções. Isso pode custar caro, tanto em tempo do processador como em espaço de memória. Cada chamada recursiva faz com que outra cópia da função (na realidade, apenas as variáveis da função) seja criada; isso pode consumir memória de forma considerável. Normalmente, a iteração ocorre dentro de uma função, e assim, o *overhead* de repetidas chamadas de funções e a alocação de mais memória não existe. Então, por que escolher a recursão?

Observação de engenharia de software ***3.14***

___Qualquer problema que pode ser resolvido recursivamente também pode ser resolvido interativamente (não recursivamente). Uma abordagem recursiva é escolhida em detrimento de uma abordagem iterativa, quando ela reflete o problema de modo mais natural, resultando em programa mais fácil de compreender e depurar Outra razão para se optar por uma solução recursiva é quando uma solução iterativa não é fácil de ser encontrada.

Dica de desempenho 3.7

___Evite usar a recursão em situações que exigem um alto desempenho. Chamadas recursivas consomem tempo e exigem mais memória.

Erro comum de ***programação*** *3.24*

Fazer uma função não-recursiva chamar acidentalmente a si mesma, tanto direta como indiretamente (através de outra função), é um erro de lógica.

A maioria dos livros sobre programação introduzem a recursão muito mais à frente do que fizemos neste. Achamos que a recursão é um tópico tão rico e complexo que é melhor ser apresentado mais cedo e espalhar os exemplos pelo restante do texto. A Fig. 3.17 apresenta um resumo dos exemplos e exercícios de recursão neste livro.

a chamada original a fibonacci. A Fig. 3.16 mostra como a função fibonacci calcularia fibonacci (3) . abreviamos fibonacci simplesmente como f para tornar a figura mais legível.

Essa figura traz à tona algumas questões interessantes sobre a ordem na qual os compiladores da linguagem **C++** calcularão os operandos dos operadores. Essa é uma questão diferente da ordem na qual os operadores são aplicados a seus operandos, ou seja, a ordem estabelecida pelas regras de precedência entre operadores. Na Fig. 3.16. parece que. ao calcular f (3 **)** . serão feitas duas chamadas recursivas. a saber. f **(** 2 **)** e f (1 **).** Mas, em que ordem serão feitas essas chamadas?

Fig 3.16 Conjunto de chamadas recursivas à função **fibonacci.**

A maioria dos programadores simplesmente considera que os operandos serão calculados da esquerda para a direita. Estranhamente, a liguagem C++ não especifica a ordem na qual os operandos da maioria dos operadores (incluindo +) devem ser calculados. Portanto, o programador não pode presumir a ordem na qual serão feitas essas chamadas. Na realidade, poderia ser executada inicialmente a chamada para f **(** 2 **)** e depois a chamada para f **(** 1 **)** . ou as chamadas poderiam ser executadas na ordem inversa. Neste programa e na maioria dos outros programas, o resultado final seria o mesmo. Mas, em alguns programas, o cálculo de um operando pode causar *efeitos colaterais* que poderiam afetar o resultado final da expressão.

A linguagem C++ especifica a ordem de avaliação de somente quatro operadores, a saber: **&&,** . o operador vírgula **(,)** e ?: Os três primeiros são operadores binários, cujos dois operandos, com certeza, são avaliados da esquerda para a direita. O último operador é o único operador ternário de C++. Seu operando da extremidade esquerda é sempre avaliado em primeiro lugar: se o operando da extremidade esquerda levar a um valor diferente de zero, o operando do meio é avaliado a seguir e o último operando é ignorado; se o operando da extremidade esquerda levar ao valor zero, o operando da extremidade direita é avaliado a seguir e o operando do meio é ignorado.

Erro comum de programação 3.23

Escrever programas que dependam da ordem de cálculo dos operandos de operadores diferentes de **&&,** 1 *?: e o operador vírgula (,)pode levar a erros, porque os compiladores poderiam não calcular necessariamente os operandos na ordem em que o programador espera.*

Dica de portabilidade 3.2

___Programas que dependem da ordem de cálculo de operandos de operadores diferentes de **&&, 1 1,?:** *e o operador virgula (,)podem funcionar de modo distinto em sistemas com compiladores diferentes.*

Uma palavra de advertência deve ser dita a respeito de programas recursivos, como o que usamos aqui para gerar os números de Fibonacci. Cada nível de recursão na função fibonacci tem um efeito de duplicação do número de

returnf_(**1 + f** (O

‘Ir ____

return **1** return O


```
13 unsigned long result, number;
14
15 cout « “Digite um inteiro:
16 cm » nuxnber;
17 result = fibonacci ( nuxnber );
18 cout « “Fibonacci (“ « number « “) = « result « endi;
19 return 0;
20
21
22 // Definição recursiva da função fibonacci
23 unsigned long fibonacci( unsigned long n
24
25 if ( n == 0 1 n == 1 ) // caso base
26 return n;
27 else // caso recursivo
28 return fibonacci( n - 1 ) + fibonacci( n - 2 );
29
```

```
Digite um inteiro : 5
Fibonacci (5) = 5
Digite um inteiro: 6
Fibonacci (6) = 8
Digite um inteiro: 10
Fibonacci (10) 55
Digite um inteiro: 20
Fibonacci (20) = 6765
Digite um inteiro: 30
Fibonacci (30) 832040
Digite um inteiro: 35
Fibonacci (35) = 9227465
```

Fig. 3.15 Gerando os números de Fibonacci recursivamente (parte 2 de 2).

A chamada a fibonacci a partir de main não é uma chamada recursiva, mas todas as chamadas subseqüentes a fibonacci são recursivas. Cada vez que fibonacci é chamada, o caso básico é testado imediatamente - se n é igual a O ou 1. Se isso for verdadeiro, o valor de n é retornado. E interessante observar que, se n for maior do que 1, a etapa de recursão gera *duas* chamadas recursivas, cada uma delas para um problema ligeiramente menor do que

Digite um Fibonacci	inteiro: O (0) = O	
Digite um Fibonacci	inteiro: (1) =1	1
Digite um Fibonacci	inteiro : (2) =1	2
Digite um Fibonacci	inteiro : (3) =2	3
Digite um Fibonacci	inteiro : (4) =3	4

seja, a de que a linguagem não é facilmente estendida para atender às exigências especiais de várias aplicações. Como veremos na seção do livro dedicada à programação orientada a objetos, C++ é uma linguagem extensível, que permite a criação de inteiros arbitrariamente grandes, se assim desejarmos.

Erro comum de programação 3.21

Esquecer-se de retornar um valor de uma função recursiva, quando isto é necessário, fará com que a maioria dos compiladores produza uma mensagem de advertência.

Erro comum de programação 3.22

Omitir o caso básico, ou escrever incorretamente a etapa de recursão, deforma que ela não convirja para o caso básico, ocasionará uma recursão infinita, o que pode eventualmente esgotar a memória. Isso é análogo ao problema de um laço infinito em uma solução iterativa (não-recursiva). A recursão infinita também pode ser causada pelo fornecimento de um dado incorreto de entrada.

3.13 Exemplo usando recursão: a série de Fibonacci

A série de Fibonacci

O, 1, 1,2, 3,5, 8, 13, 21,

começa comO e 1 e tem a propriedade de que cada número subseqüente de Fibonacci é a soma dos dois números de Fibonacci anteriores.

A série ocorre na natureza e, em particular, descreve uma forma de espiral. A razão (relação produzida pela divisão do maior, o sucessor, pelo menor, o predecessor imediato) dos números sucessivos de Fibonacci converge para um valor constante de 1,618.... Esse número também ocorre repetidamente na natureza e foi chamado de *relação áurea* ou *proporção áurea.* As pessoas tendem a achar a relação áurea esteticamente agradável. Os arquitetos freqüentemente **projetam** janelas, quartos e edifícios cujos comprimentos e larguras estão nas proporções da relação áurea. Muitas vezes, os cartões postais são criados com proporções de altura/largura respeitando a relação áurea.

A série de Fibonacci pode ser definida recursivamente como segue:

flbonacci(O) = O

fibonacci(1) =1

fibonacci(n) -flbonacci(n - 1) +flbonacci(n -2)

O programa da Fig. 3.15 calcula recursivamente o *i-ésimo* número de Fibonacci usando a função fibonacci. Note que os números de Fibonacci tendem a crescer rapidamente. Portanto, escolhemos o tipo de dados unsigned long para o tipo do parâmetro e do valor de retomo na função fibonacci. Na Fig. 3.15, cada par de linhas de saída mostra uma execução distinta do programa.

Fig. 3.15 Gerando os números de Fibonacci recursivamente (parte 1 de 2).

1	*II* **Fig. 3.15: figO3_15.cpp**			
2	**// Função recursiva para**	**fibonacci**		
3	**#include <iostreaxn>**			
4				
5	**using std: :cout;**			
6	**using std::cin;**			
7	**using std::endl;**			
8				
9	**unsigned long fibonacci(**	**unsigned**	**long**	**);**
10				

11	int main O			
12	{			


```
1 // Fig. 3.14: fig03_14.cpp
2 /1 Função recursiva para cálculo de fatoriais
3 #include <iostream>
4
5 using std::cout;
6 using std: :endl;
7
8 #include <iomanip>
9
10 using std::setw;
11
12 unsigned long factorial( unsigned long );
13
14 int main ()
15
16 for ( int i = 0; i <= 10; i++
17 cout « setw ( 2 ) « i « "! = · « factorial( i ) « endl;
18
19 return 0;
20
21
22 // Definição recursiva da função factorial
23 unsigned long factorial (unsigned long number)
24 {
25 if (number <= 1) // caso base
26 return 1;
27 else // caso recursivo
28 return number * factorial( nuniber - 1);
29
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9’ = 362880
10! = 3628800
```

Fig. 3.14 Cálculo de fatoriais com uma função recursiva.

A função factorial foi declarada de modo a receber um parâmetro do tipo uns igned long e retornar um resultado do tipo unsigned long. Isso é uma abreviação de unsigned long int. A especificação de C++ determina que uma variável do tipo unsigned long int seja armazenada em pelo menos 4 bytes (32 bits), podendo assim conter um valor no intervalo O a 4294967295. (O tipo de dados long int também é armazenado em pelo menos 4 bytes e pode armazenar um valor de pelo menos -2147483648 a 2147483647). Como pode ser observado na Fig. 3.14, os valores dos fatoriais se tornam grandes rapidamente. Escolhemos o tipo de dados unsigned long para que o programa possa calcular fatoriais maiores que 7! em computadores com inteiros pequenos (como 2 hytes, por exemplo). Infelizmente, a função factorial produz valores grandes tão rapidamente que até mesmo unsigned long não nos ajuda a calcular muitos valores de fatoriais antes de o valor máximo possível de uma variável uns igned long ser superado.

Como exploramos nos exercícios, double pode ser necessário em último caso para o usuário que deseje calcular fatoriais de números grandes fsso indica uma deficiência da maioria das linguagens de programação. ou

com 1! igual a 1 e 0! igual a 1 por definição. Por exemplo, *5!* é o produto *5 .4.* 3 .2. 1, que é igual a 120. O fatorial de um inteiro, number. maior ou igual a 0, pode ser calculado *iterativamente* (não-recursivamente)

usando for como se segue:
factorial = **1;**
for (int counter = nuniber; counter **>= 1;** counter--)
factorial *= counter;
Chega-se a uma definição recursiva para a função fatorial observando-se o seguinte relacionamento:
n! =n.(n-])!
Por exemplo, *5! é* claramente igual aS * *4!,* como é mostrado a seguir:
5! = 5.4.3.2.
5! 5. (4.3.2.1)
5! 5(4!)
O cálculo de *5!* prosseguiria da forma mostrada na Fig. 3.13. A Fig. 3.13a mostra como a sucessão de chamadas recursivas se processaria até 1! ser calculado como 1, o que encerraria a recursão. A Fig. 3.13b mostra os valores enviados de volta à função chamadora em cada chamada recursiva, até que o valor final seja calculado e retornado.
Valor final = 120

r

5!=5*24_120 é o valor retornado
5 * **4!**
41=4*6=24 é o valor retornado
4 * **3!**
3!=3*2=6 é o valor retornado
3 * **2'**
2 * = é o valor retornado
1 éo valor retornado
1
a) Processamento de chamadas b) Valores retornados de cada chamada recursiva.
recursivas
Fig. 3.13 Cálculo recursivo de ***5!.***
O programa da Fig. 3.14 usa a recursão para calcular e imprimir fatoriais de inteiros de O a 10 (a escolha do tipo de dado unsigned long será explicada em breve). A função recursiva factorial inicialmente verifica se uma condição de término é true (verdadeira), i.e., se nuxnber é menor ou igual a 1. Se number for menor ou igual a 1. factorial retorna 1, nenhuma recursão se faz mais necessária e o programa é encerrado. Se nuniber for maior do que 1, o comando
return number * factorial (**nuinber** - **1)**
expressa o problema como o produto de number por uma chamada recursiva a factorial calculando o fatorial de number - 1. Observe que factorial (nuiriber - i é um problema ligeiramente mais simples do que o cálculo original factorial (number).

Uma variável global x é declarada e inicializada com o valor **1.** Essa variável global fica oculta em qualquer bloco (ou função) que declare uma variável x. Em main. a variável local x é declarada e inicializada com o valor 5. A seguir, essa variável é impressa para mostrar que a variável global x é ignorada em main. Em seguida, é definido um novo bloco em main, com outra variável local x inicializada com o valor 7. Essa variável é impressa para mostrar que x do bloco externo de main é ignorada. A variável x com o valor 7 é eliminada automaticamente, depois da execução do bloco, e a variável local x do bloco externo de main é impressa novamente para mostrar que não está mais sendo ignorada. O programa define três funções que não utilizam nenhum argumento e não retornam nenhum valor. A função a define uma variável automática x e a inicializa como 25. Quando a é chamada, a variável é impressa, incrementada e impressa novamente, antes do término da função. Cada vez que essa função é chamada, a variável automática x é reinicializada como 25. A função b declara uma variável x static e a inicializa como
50. As variáveis locais declaradas como static conservam seus valores, mesmo quando estão fora do escopo. Quando b é chamada, a variável x é impressa, incrementada e impressa novamente antes do término da função. Na próxima chamada a essa função, a variável static local x terá o valor 51. A função c não declara variável alguma. Portanto, quando ela faz referência à variável x, o x global é utilizado. Quando c é chamada, a variável global é impressa, multiplicada por 10 e impressa novamente antes do término da função. Na próxima vez que a função e for chamada, a variável global ainda terá seu valor modificado, 10. Finalmente, o programa imprime mais uma vez a variável local x, em main, para mostrar que nenhuma das chamadas de funções modificou o valor de x, porque todas as funções faziam referência a variáveis em outros escopos.

3.12 Recursão
Geralmente, os programas que analisamos são estruturados como funções que fazem chamadas entre si de uma maneira disciplinada, hierárquica. Para alguns tipos de problemas, é útil ter funções que chamem a si mesmas. Uma *função recursiva* é uma função que chama a si mesma, direta ou indiretamente (por meio de outra função). A recursão é um tópico complexo, analisado exaustivamente em cursos avançados de Ciência da Computação. Nesta seção e na seção seguinte, são apresentados exemplos simples de recursão. Este livro contém um tratamento abrangente da recursão. A

Fig. 3.17 (no final da Seção 3.14) resume os exemplos e exercícios de recursão do livro.
Em primeiro lugar, vamos analisar a recursão conceitualmente e depois examinaremos vários programas contendo funções recursivas. Os métodos recursivos para solução de problemas possuem vários elementos em comum. Uma função recursiva é chamada para resolver um problema. Na realidade, a função só sabe resolver o(s) caso(s) mais simples, ou o(s) chamado(s) *caso(s) básico(s).* Se a função for chamada com um caso básico, ela simplesmente retorna um resultado. Se a função for chamada com um problema mais complexo, ela divide o problema em duas partes conceituais: uma parte que a função sabe como resolver e outra que ela não sabe. Para viabilizar a recursão, a segunda parte deve ser parecida com o problema original, mas ser uma versão um pouco mais simples ou menor do que ele. Por esse novo problema ser parecido com o problema original, a função dispara (chama) uma nova cópia de si mesma para lidar com o problema menor - o que é conhecido por *chamada recursiva* ou *etapa de recursão.* A etapa de recursão também inclui a palavra-chave return, porque seu resultado será combinado com a parte do problema que a função sabe como resolver para formar um resultado que será enviado de volta para a função original de chamada, possivelmente main.
A etapa de recursão é executada enquanto a chamada original para a função estiver ativa, i.e., ainda não tiver sido concluída. A etapa de recursão pode levar a outras chamadas recursivas, à medida que a função continuar a dividir cada problema em duas partes conceituais. Para a recursão chegar ao fim, cada vez que a função chamar a si mesma com uma versão ligeiramente mais simples do problema original, essa seqüência de problemas cada vez menores deve finalmente convergir para o caso básico. Nesse instante, a função reconhece o caso básico, envia um resultado de volta para a cópia anterior da função e ocorre uma seqüência de passagem de resultados na ordem inversa, até que a chamada original da função envie, finalmente, o resultado final para main. Tudo isso parece um tanto esquisito, se comparado com a maneira convencional de resolver problemas que usamos até aqui. Na verdade, é necessário muita prática com programas recursivos antes de o processo parecer natural. Como exemplo da utilização desses conceitos, vamos escrever um programa recursivo para realizar um conhecido cálculo da matemática.
O fatorial de um inteiro não-negativo *n,* escrito *n!* (e pronunciado "fatorial de *n* "), é o produto
n


```
32 b O; // x local estático conserva seu valor anterior
33 c o; // x global também conserva seu valor
34
35 cout « "x local em main é -« x « endl;
36
37 return 0;
38
39
40 void a( void
41
42 int x = 25; // inicializada sempre que a é chamada
43
44 cout « endl « 'x local em a é -« x
45 -depois de entrar em a" « endl;
46 ++x;
47 cout « x local em a é « x
48 « -antes de sair de a" « endl;
49
50
51 void b( void
52 {
53 static int x = 50; // somente inicialização estática
54 // na primeira vez que b é chamada
55 cout « endl« "x local estático é « x
56 « -ao entrar em b" « endl;
57
58 cout « 'x local estático é -« x
59 « -ao sair de b" « x « endl;
60
61
62 void c( void
63
64 cout « endl « "x global é « x
65 « -ao entrar em c" « endl;
66 x*10;
67 cout « "x global é « x « ao sair de c" « endi;
```

```
68
x local no escopo externo de main é 5
x local no escopo interno de main é 7
x local no escopo externo de main é 5
x local em a é 25 depois de entrar em a
x local em a é 26 antes de sair de a
x local estático é 50 ao entrar em b
x local estático é 51 ao sair de b
x global é 1 ao entrar em c
x global é 10 ao sair de c
x local em a é 25 depois de entrar em a
x local em a é 26 antes de sair de a
x local estático é 51 ao entrar em b
x local estático é 52 ao sair de b
x global é 10 ao entrar em c
x global é 100 ao sair de c
x local em main é 5
```

Fig. 3.12 Um exemplo de escopos (parte 2 de 2).

de variáveis. Quando os blocos estão aninhados e um identificador em um bloco externo tem o mesmo nome de um identificador em um bloco interno, o identificador no bloco externo é "escondido" até que o bloco interno seja encerrado. Durante a execução de um bloco interno, este vê o valor de seu próprio identificador local e não o valor do identificador de mesmo nome no bloco externo. As variáveis locais declaradas static também possuem escopo de bloco, muito embora existam desde o instante em que o programa começou a ser executado. Dessa maneira, a classe de armazenamento não afeta o escopo de um identificador.

Os únicos identificadores com *escopo de protótipo de função* são os utilizados na lista de parâmetros de um protótipo de função. Como já mencionado, os protótipos de função não necessitam de nomes em suas listas de parâmetros - apenas os tipos são exigidos. Se um nome foi usado na lista de parâmetros de um protótipo de função, ele será ignorado pelo compilador. Os identificadores usados em um protótipo de função podem ser reutilizados em qualquer lugar do programa sem ambigüidade.

Erro comum de programação 3.20

Usar acidentalmente, em um bloco interno, o mesmo nome que é usado para um identificador em um bloco mais externo, quando, na realidade, o programador quer que o identificador do bloco externo esteja ativo durante o processamento do bloco interno, geralmente causa um erro de lógica.

Boa prática de programação 3.9

Evite nomes de variáveis que escondam nomes em escopos mais externos, isso pode ser conseguido simplesmente evitando-se, em um programa, o uso de identificadores duplicados.

O programa da Fig. 3.12 demonstra aspectos de escopo com variáveis globais, variáveis locais automáticas e variáveis locais static.

```
1 // Fig. 3.12: fig03_12.cpp
2 // Um exemplo de escopos
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 void a( void ); // protótipo de função
9 void b( void ); // protótipo de função
10 void c( void ); // protótipo de função
11
12 int x = 1; // variável global
13
14 int main()
15
16 int x = 5; // variável local para main
17
18 cout << "x local no escopo externo de main é " << x << endl;
19
20 { // inicia novo escopo
21 int x = 7;
22
```

```
23 cout « "x local no escopo interno de main é " « x « endl;
24 } // encerra novo escopo
25
26 cout « "x local no escopo externo de main é " « x « endl;
27
28 a O; // a tem x local automático
29 b O; // b tem x local estático
30 c O; // c usa x global
31 a O; // a reinicializa x local automático
```

Fig. 3.12 Um exemplo de escopos (parte 1 de 2).

tado, isso não significa que esses identificadores possam ser usados em todo o programa. A classe de armazenamento e o escopo (onde um nome pode ser usado) são aspectos diferentes, como veremos na Seção 3.11.

Há dois tipos de identificadores com classe de armazenamento estática: identificadores externos (como variáveis globais e nomes de função) e variáveis locais declaradas com o especificador de classe de armazenamento s tatic.As variáveis globais e nomes de função pertencem, por default, à classe de armazenamento extern. As variáveis globais são criadas colocando-se declarações de variáveis fora de qualquer definição de função e conservam seus valores ao longo de toda a execução do programa. As referências a variáveis e funções globais podem ser feitas em qualquer função que venha após suas declarações ou definições no arquivo.

Observação de engenharia de software 3.12

Declarar uma variável como global em vez de local permite que ocorram efeitos colaterais indesejáveis quando uma função que não necessita de acesso à variável modifica, acidental ou intencionalmente, essa variável, Em geral, o uso de variáveis globais deveria ser evitado, exceto em certas situações com exigências especiais de desempenho.

Observação de engenharia de software 3.13

___As variáveis usadas apenas em uma determinada função devem ser declaradas como variáveis locais naquela função, em vez de serem declaradas como variáveis globais.

Variáveis locais declaradas com a palavra-chave static são conhecidas apenas na função na qual são definidas, mas, diferentemente das variáveis automáticas, as variáveis locais static conservam seus valores quando a função é encerrada. Na próxima vez em que a função for chamada, a variável local static conterá o valor que tinha quando a função foi executada pela última vez. O comando seguinte declara a variável local count como static e a inicializa com o valor 1.

static int count = **1;**

Todas as variáveis numéricas da classe de armazenamento estática são inicializadas com o valor zero, se não forem inicializadas explicitamente pelo programador. (As variáveis estáticas ponteiros, discutidas no Capitulo *5,* também são inicializadas com o valor zero).

Os especificadores de classes de armazenamento extern e static possuem significado especial quando aplicados explicitamente a identificadores externos. No Capítulo 18, "Tópicos sobre código legado em C", discutiremos o uso de **extern e static** com identificadores externos e programas com múltiplos arquivos-fonte.

3.11 Regras de escopo

A parte do programa na qual um identificador tem significado é conhecida como seu *escopo.* Por exemplo, quando declaramos uma variável local em um bloco, podem ser feitas referências a ela apenas naquele bloco ou em blocos aninhados naquele bloco. Os quatro escopos de um identificador são *escopo de função, escopo de arquivo, escopo de bloco* e *escopo de protótipo de função.* Mais tarde, veremos dois outros escopos - *escopo de classe* (Capítulo 6) e *escopo de ambiente de nomes* (Capítulo 21).

Um identificador declarado fora de qualquer função tem *escopo de arquivo.* Tal identificador é "conhecido" por todas as funções desde o local onde é declarado até o final do arquivo. Variáveis globais, definições de funções e protótipos de funções colocados fora de uma função possuem escopo de arquivo.

Rótulos (identificadores seguidos de dois pontos, como inicio:) são os únicos identificadores com *escopo defunção.* Os rótulos podem ser usados em qualquer lugar de uma função na qual aparecem, mas não pode ser feita qualquer referência a eles fora do corpo da função. Os rótulos são usados em estruturas switch (como rótulos de case) e em comandos goto (ver Capítulo 18). Os rótulos são detalhes de implementação que as funções ocultam umas das outras. Essa ocultação - chamada mais formalmente de *ocultação de informações* - é um dos princípios mais importantes da boa engenharia de software.

Os identificadores declarados dentro de um bloco possuem *escopo de bloco.* O escopo de bloco termina na chave à direita final **(})** do bloco. As variáveis locais declaradas no início de uma função possuem escopo de bloco, assim como os parâmetros da função, que também são variáveis locais dela. Qualquer bloco pode conter declarações

Apenas variáveis podem ter classe de armazenamento automática. As variáveis locais de uma função (as declaradas na lista de parâmetros ou no corpo da função) são normalmente da classe de armazenamento automática. A palavra-chave auto declara explicitamente variáveis da classe de armazenamento automática. Por exemplo, a declaração a seguir indica que as variáveis x e y, do tipo double. são variáveis locais automáticas e existem apenas no corpo da função na qual a declaração aparece:

auto double x, y;

As variáveis locais possuem classe de armazenamento automático por default; portanto, a palavra-chave auto raramente é usada. No restante do texto, vamos nos referir a variáveis com classe de armazenamento automática simplesmente como variáveis automáticas.

Dica de desempenho 3.3

___O armazenamento automático é um meio de economizar memória porque as variáveis automáticas são criadas quando o bloco na qual são declaradas é ativado e são destruídas quando o bloco é deixado.

Observação de engenharia de software 3.11

___O armazenamento automático é mais um exemplo do princípio do menor pri vilégio. Por que armazenar variáveis na memória e deixá-las disponíveis, quando na realidade não são mais necessárias?

Os dados, na versão de um programa em linguagem de máquina, são normalmente carregados em registradores para cálculos e outros processamentos.

Dica de desempenho 3.4

___O especificador de classe de armazenamento **regis** ter *pode ser colocado antes de uma declaração de variável automática para sugerir que o compilador mantenha a variável em um dos registradores de hardware de alta velocidade do computador em vez de na memória. Se as variáveis forem usadas freqüentemente como contadores ou para cálculo de totais, elas podem ser conservadas em registradores de hardware, e o overhead de carregar repetidamente as variáveis da memória para os registradores e armazenar os resultados novamente na memória pode ser eliminado.*

Erro comum de programação ***3.19***

Usar diversos especcadores de classe de armazenamento para o mesmo idenaficador é um erro de sintaxe. Somente um especificador de classe de arinazenamento pode ser aplicado a um identificador Por exemplo, se você incluir **regis ter,** *não inclua também* **auto.**

O compilador pode ignorar as declarações regis ter. Por exemplo, pode não haver número suficiente de registradores disponíveis para uso pelo compilador. A declaração a seguir indica que a variável inteira contador pode ser colocada em um dos registradores do computador e inicializada com 1:

register int contador = 1;

A palavra-chave regis ter só pode ser usada com variáveis da classe de armazenamento automática.

Dica de desempenho 3.5

Freqüentemente, as declarações **regis ter** *são desnecessárias. Os atuais compiladores otimizadores são capazes de reconhecer as variáveis usadas repetidamente e podem decidir colocá-las em registradores sem que haja a necessidade de o programador incluir uma declaração* **regis ter.**

As palavras-chave extern e static são usadas para declarar identificadores para variáveis e funções da classe de armazenamento estática. Os identificadores da classe de armazenamento estática existem desde o instante em que o programa começa a ser executado. Para variáveis, o armazenamento é alocado e inicializado uma vez, quando o programa começa a ser executado. Para funções, o nome da função existe quando o programa começa a ser executado. Entretanto, muito embora as variáveis e os nomes das funções existam quando o programa começa a ser execu

Erro comum de programação 3.18

Após uma constante de enumeração ter sido definida, tentar atribuir outro valor para a constante de enumeração é um erro de sintaxe.

Boa prática** de **programação 3.7

Use somente letras maiúsculas nos nomes de constantes de enumeração. Isto faz *com que estas constantes se destaquem no programa, lembrando ao programador que constantes de enumeração não são variáveis.*

***Boa** prática de **programação** 3.8*

Usar constantes de enumeração em vez de constantes inteiras torna um programa mais compreensível.

Depois do primeiro lançamento, se o jogo terminar, a estrutura while ***é*** ignorada porque gameStatus não será igual a CONTINUE. O programa prossegue para a estrutura if/else que imprime "Jogador ganha" se ganeS tatus for igual a WON e "Jogador perde" se ganeS tatus for igual a LOST.

Depois do primeiro lançamento, se o jogo não terminar, a soma (sun) é armazenada na variável myPoint ("meu ponto"). A execução prossegue com a estrutura while porque gameStatus é O. Cada vez que a estrutura while é percorrida, roliDice ***é*** chamada para produzir uma nova variável sum. Se sum for igual a inyPoint, gaineStatus ***é*** definido com o valor WON para indicar que o jogador ganhou, a condição no while torna-se falsa, a estrutura if/else

imprime “Jogador ganha” e a execução termina. Se sum for igual a **7,** ganeStatus ***é*** definido com o valor LOST para indicar que o jogador perdeu, a condição no whiJ.e toma-se falsa, o comando if/ else imprime “Jogador perde” e a execução ***é*** encerrada.

Observe a interessante estrutura de controle do programa. Usamos duas funções - main e roliDice - e as estruturas switch. while. if/else e estruturas if aninhadas. Nos exercícios, examinaremos várias características interessantes do jogo de *craps.*

3.10 Classes de armazenamento

Nos Capítulos 1 a 3, usamos identificadores para nomes de variáveis. Os atributos de variáveis incluem nome, tipo e valor. Neste capítulo, também usamos identificadores como nomes de funções definidas pelo usuário. Na realidade, cada identificador em um programa tem outros atributos, incluindo *classe de armazenamento, escopo* e *ligação.*

C++ fornece cinco classes de armazenamento, indicadas pelos *especificadores de classes de armazenamento:* auto, register, extern, mutable e static. O especificador da classe de armazenamento de um identificador ajuda a determinar sua classe de armazenamento, escopo e ligação. Esta seção discute os especificadores de classe de armazenamento auto, register. **extern e static.** O especificador de classe de armazenamento mutable (discutido em detalhes no Capítulo 21) é usado exclusivamente com os tipos definidos pelo usuário de C++ chamados de *classes* (apresentados nos Capítulos 6 e *7).*

A *classe de armazenamento* de um identificador determina o período durante o qual aquele identificador existe na memória. Alguns identificadores existem durante pouco tempo, alguns são repetidamente criados e eliminados e outros existem durante toda a execução do programa. Nesta seção, discutimos duas classes de armazenamento: estática e automática.

O *escopo* de um identificador é onde se pode fazer referência àquele identificador dentro de um programa. Pode-se fazer referência a alguns identificadores ao longo de todo o programa; outros podem ser referenciados apenas a partir de partes limitadas de um programa. A Seção 3.11 discute o escopo dos identificadores.

A *ligação* de um identificador determina, para um programa com vários arquivos-fonte, se um identificador é conhecido apenas no arquivo-fonte atual ou em qualquer arquivo-fonte com as declarações adequadas.

Os especificadores de classes de armazenamento dos identificadores podem ser divididos em duas classes de armazenamento: *classe de armazenamento automática* e *classe de armazenamento estática.* As palavras-chave auto e register são utilizadas para declarar variáveis da classe de armazenamento automática. As variáveis da classe de armazenamento automática são criadas quando o bloco no qual são declaradas é ativado e existem enquanto o bloco estiver ativo, sendo destruídas quando a execução do programa sai do bloco.

valor 2. Os identificadores em uma enum devem ser únicos; no entanto, constantes de enumeração separadas podem ter o mesmo valor inteiro.

```
Jogador jogou 6 + 5
Jogador vence
Jogador jogou 6 + 5
Jogador vence
Jogador jogou 4 + 6 = 10
Ponto é 10
Jogador jogou 2 + 4 = 6
Jogador jogou 6 + 5 = 11
Jogador jogou 3 4 3 6
Jogador jogou 6 + 4 10
Jogador vence
Jogador jogou 1 + 3 = 4
Ponto é 4
Jogador jogou 1 + 4 = 5
Jogador jogou 5 + 4 = 9
Jogador jogou 4 + 6 = 10
Jogador jogou 6 + 3 = 9
Jogador jogou 1 + 2 = 3
Jogador jogou 5 + 2 = 7
Jogador perde
```

Fig. 3.11 Exemplos de resultados do jogo de *craps.*

IBoa prática de programação ***3.6***

Use maiúscula na primeira letra de um identificador usado como o nome de um tipo definido pelo usuário.

Variáveis do tipo Status definidas pelo *usuário* somente podem receber a atribuição de um dos três valores definidos na enumeração. Quando o jogo é **vencido, gameStatus recebe** o valor WON . Quando o jogo é perdido, gameStatus recebe o valor LOST. Caso contrário. ganieStatus recebe o valor CONTINUE, de maneira que os dados

possam ser lançados novamente.

Erro comum de programação 3.17

Atribuir o inteiro equivalente a uma constante de enumeração a uma variável do tipo da enumeração é um erro de sintaxe.

Uma outra enumeração popular é

```
enum Meses { JA = 1, FEV, MAR, ABR, MAl, JUN, JUL, AGO,
SET, OUT, NOV, DEZ };
```

que cria um tipo definido pelo usuário chamado **Meses** com constantes de enumeração representando os meses do ano. Como o primeiro valor na enumeração precendente é inicializado explicitamente com 1. os demais são incrementados por 1, resultando nos valores 1 a **12.** Qualquer constante de enumeração pode ter atribuído a ela um valor inteiro na definição da enumeração e cada uma das constantes de enumeração subseqüentes terão um valor que é 1 a mais que a constante precedente.


```
ero adicio 2
case 11: II vence na primeira jogada
28 gameStatus = WON;
29 break;
30 case 2:
31 case 3:
nsecutivOs
32 case 12: II perde na primeira jogada
desejado). 33 gameStatus = LOST;
es que não 34 break;
35 default: II memoriza o "Ponto
36 gameStatus = CONTINUE;
37 myPoint = sum;
38 cout « "Ponto é - « myPoint « endi;
eafunção 39 break; II otiona1
40 }
41
42 while ( gameStatus ==CONTINUE ) { II continua jogando
43 sum = roliDiceO;
44
45 if ( sum =myPoint ) II vence fazendo o ponto
nOS e salas 46 gaineStatus = WON;
47 else
48 if ( suin ==7 ) II perde obtendo o valor 7
49 gameStatus = LOST;
tos. Depois 50
7ou 11 no 51
ido craps), 52 if ( gameStatus =WON )
esta soma 53 cout « "Jogador ganha" « endi;
54 else
eu ponto . 55 cout « "Jogador perde" « endi;
56
57 return 0;
58 }
59
60 int rollDice( void
61
62 int diel, die2, workSum;
63
64 diel = 1 + rand() % 6;
65 die2 = 1 + rand() % 6;
66 workSum = diel + die2;
67 cout « "Jogador fez - « diel « + - « die2
68 « - = - « workSum «endl;
69
70 return workSum;
71
```

Fig. 3.10 Programa para simular o jogo de *craps* (parte 2 de 2).

Observe que o jogador deve lançar dois dados no primeiro lançamento e deve fazer isso em todos os lançamentos subseqüentes.

Definimos uma função rollDice para lançar os dados e calcular e imprimir a soma dos pontos de suas faces. A função rollDice é definida apenas uma vez, mas é chamada em dois locais no programa. E interessante observar **que rollDice não** recebe argumentos, portanto indicamos void na lista de parâmetros. A função roilDice retorna a soma dos dois dados; assim, um tipo de retorno int é indicado no cabeçalho da função.
O jogo é razoaveiniente complicado. O jogador pode vencer ou perder no primeiro lançamento ou pode vencer ou perder em qualquer lançamento subseqüente. A variável gaineStatus é usada para controlar isso. A linha
enum Status { CONTINUE, WON, LOST };
cria um *tipo definido pelo usuário* chamado de *enumeração.* Uma enumeração, introduzida pela palavra-chave enum e seguida por um *nome de tipo* (neste caso Status), é um conjunto de constantes inteiras representadas por identificadores. Os valores destas *constantes de enumeração* começam em **0.** a menos que especificado de outra forma, e são incrementados por **1.** Na enumeração precente, CONTINUE recebe o valor O, **WON.** o valor 1 e LOST, o

ajustar a escala de rand com o operador módulo (i.e., 6) e o número inicial do intervalo é igual ao número adicionado a rand % 6 (i.e., 1). Podemos generalizar esse resultado, como segue
n = a +rand O **% b;**
onde a é **o** *valor de deslocamento da escala* (que é igual ao primeiro número do intervalo de inteiros consecutivos desejado) e b é o fator de ajuste de escala (que é igual à amplitude do intervalo de inteiros consecutivos desejado). Nos exercícios, veremos que é possível escolher inteiros aleatoriamente a partir de conjuntos de valores que não sejam intervalos de inteiros consecutivos.

IErro comum de programação 3.16

Usar **srand** *no lugar de* **rand** *para gerar números aleatórios é um erro de sintaxe, porque a função* srand *não retorna um valor*

3.9 Exemplo: um jogo de azar e apresentando enum

Um dos jogos de azar mais populares é um jogo de dados conhecido como *craps,* que é jogado em cassinos e salas de jogos de todo o mundo. As regras do jogo são simples:
Um jogador joga dois dados. Cada dado tem seis faces. Essas faces contém 1, 2, *3, 4, 5 ou 6 pontos. Depois de os dados pararem, a soma dos pontos nas faces superiores é calculada. Se a soma for 7 ou 11 no primeiro lançamento, o jogador vence. Se a soma for 2, 3 ou 12 no primeiro lançamento (chamado craps), ojogadorperde (i. e., a "banca" vence). Sea somafor4, 5,6,8, 9ou lOno primeiro lançamento, esta soma se torna o "ponto" do jogador Para vencer você deve continuar lançando os dados até "fazer seu ponto". O jogador perde se tirar um 7 antes de fazer o ponto.*
O programa da Fig. 3.10 simula o jogo de *craps.* A Fig. 3.11 mostra vários exemplos de execução.

```
1 // Fig. 3.10: figO3lO.cpp
2 // Craps
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <cstdlib>
9
10 #include <ctime>
11
12 using std::time
13
14 int rollDice( void ); // Protótipo da função
15
16 int main()
17
18 enum Status { CONTINUE, WON, LOST };
19 int sum, myPoint;
20 Status gameStatus;
21
22 srand( time( O ) );
23 sum = roilDiceO; // primeira jogada dos dados
24
25 switch ( sum ) {
26 case 7:
```

Fig. 3.10 Programa para **simular o jogo** de *craps* (parte 1 de **2).**

```
12
13 #include <cstdlib>

14

int main

unsigned seed;

cout « "Forneça a semente: "; cm » seed;
srand( seed

for ( int i = 1; i < 10; i++
cout « setw ( 10 ) « 1 + rand () % 6;

if ( i % 5 == O
cout « endi;

}

30 return 0;
31
```

Isso faz com que o computador leia seu relógio para obter automaticamente o valor da semente. A função time (com o argumento 0, como escrito no comando precedente) retoma a "hora de calendário" atual do dia, em segundos. Esse valor é convertido para um inteiro unsigned e é usado como a semente para o gerador de números aleatórios, O protótipo de função para time está em <ctime>.

Dica de desempenho 3.2

f *Afimnção* srand *necessita ser chamada somente uma vez em um programa para obter o efeito de randomização desejado. Chamá-la mais de uma vez* é *redundante e, portanto, reduz o desempenho do programa.*

Os valores produzidos diretamente por rand sempre estão no intervalo:

```
O rand O RAND MAX
```

Mostramos anteriormente como escrever um único comando em C++ para simular o lamçamento de um dado de seis lados, com o seguinte comando:

```
face = 1 + rand O % 6;
```

que sempre atribui (aleatoriamente) à variável face um valor inteiro, no intervalo 1 **face** 6. Observe que a amplitude desse intervalo (i.e., o número de inteiros consecutivos no intervalo) é 6 e o número inicial do intervalo é Examinando o comando anterior, vemos que o comprimento do intervalo é determinado pelo número usado para

```
15
16
```

```
17
18
19
20
21
22
23
24
25
26
27
28
29
```

Fig. 3.9 Randomizando o programa de lançamento de um dado (parte 2 de 2).

Forneça a semente: 67				
1	6	5	1	4
5	6	3	1	2

Forneça a semente: 432				
4	2	6	4	3
2	5	1	4	4

Forneça a semente: 67				
1	6	5	1	4
5	6	3	1	2

Como mostra a saída do programa, ao fazer o ajuste e o deslocamento da escala, utilizamos a função rand para simular realisticamente o lançamento de um dado de seis faces. Note que o programa nunca deveria passar pelo caso default na estrutura switch; porém, incluímos este caso por razões de boa prática de programação. Depois de estudarmos arrays no Capítulo 4, mostraremos como substituir elegantemente toda a estrutura switch por um comando em uma única linha.



Dica de teste e depura ção 3.1

Inclua um caso **`default`** *em uma estrutura* **`switch`** *para detectar erros mesmo que você esteja absolutamente seguro de que não há erros.*

Executar novamente o programa da Fig. 3.7 produz

5 5 3 5 5
2 4 2 5 5
5 3 2 2 1
5 1 4 6 4

Observe que foi impressa exatamente a mesma seqüência de valores. Como esses números podem ser aleatórios? Ironicamente, essa repetição de valores é uma característica importante da função rand. Ao depurar um programa, essa repetição é essencial para assegurar que correções feitas no programa funcionam adequadamente.

Na realidade, a função rand gera *números pseudo-aleatórios.* Chamar rand repetidamente produz uma seqüência de números que parece ser aleatória. Entretanto, a seqüência se repete cada vez que o programa é executado. Depois de o programa ser completamente depurado, ele pode ser condicionado de modo a produzir uma seqüência diferente de números em cada execução. Isso é chamado de *randomização* e é realizado com a função srand da biblioteca padrão. A função srand utiliza um argumento inteiro unsigned para ser a *semente* da função rand. de forma que seja produzida uma seqüência diferente de números aleatórios a cada execução do programa.

O uso de srand é demonstrado na Fig. 3.9. No programa, usamos o tipo de dado unsigned, que é uma abreviação de uns igned int. Um int é armazenado em pelo menos dois bytes de memória e pode ter valores positivos e negativos. Uma

variável do tipo unsigned int também é armazenada em pelo menos dois bytes de memória. Um uns igned int de dois bytes só pode ter valores não-negativos, no intervalo de O a 65535. Um uns igned int de quatro bytes só pode ter valores não-negativos, no intervalo de O a 4294967295. A função srand recebe um valor unsigned int como argumento. O protótipo de função para a função srand é encontrado no arquivo de cabeçalho <cstdlib>.
Vamos executar o programa várias vezes e observar os resultados. Observe que uma seqüência *diferente* de números aleatórios é obtida cada vez que o programa é executado, desde que, em cada execução, seja fornecida uma semente diferente.
Se desejássemos randomizar sem necessidade de fornecer uma semente a cada vez, poderíamos usar um comando como
srand(time(O));

Fig. 3.9 Randomízando o programa de lançamento de um dado (parte 1 de 2).

```
1  // Fig. 3.9: figO3O9.cpp
2  // Randomizando o programa de lançamento de um dado
3  #include <iostream.h>
4
5  using std::cout;
6  using std::cin;
7  using std: :endl;
8
9  #include <iomanip>
10
11 using std::setw;
```



```
7
8
9
10
11
12
13
14
15
16 int frequencyl = 0,
17 frequency3 = 0,
18 frequency5 = 0,
19 face;
20
21
22
23
24 switch ( face
25 case 1:
26 ++frequencyl;
27 break;
28 case 2:
29 ++frequency2;
30 break;
31 case 3:
32 ++frequency3;
33 break;
34 case 4:
35 ++frequency4;
36 break;
37 case 5:
38 ++frequency5;
39 break;
40 case 6:
41 ++frequency6;
42 break;
43 default:
44 cout « 'Não deve chegar aqui nunca ";
45
```

```
46
47
48 cout « ‘Face” « setw( 13
49 « “\n 1” « setw(
50 « “\n 2” « setw(
51 « “\n 3” « setw(
52 « “\n 4” « setw(
53 « “\n 5” « setw(
54 « “\n 6” « setw(
55
56 return O
57


#include <iomanip> using std::setw; #include <cstdlib> int main()


frequency2 = 0,
frequency4 = 0,
frequency6 = 0,


for ( int roli = 1; roil <= 6000; face = 1 + rand() % 6;


roll++ ){


}

« “Freqüência”
13 ) « frequencyl
13 ) « frequency2
13 ) « frequency3
13 ) « frequency4
13 ) « frequency5
13 ) « frequency6 « endl;


Face Freqüência
1 987
2 984
3 1029
4
5
6


974
1004
1022
```

Fig. 3.8 Lançando um dado de seis faces 6000 vezes (parte 2 de 2).

<cstdlib>. Para produzir inteiros no intervalo de 0 a *5,* usamos o operador módulo *(%)* juntamente com rand, como segue

```
rand() % 6
```

Isso é chamado de *ajuste de escala.* O número 6 é chamado *de fator de escala.* A seguir, *deslocamos* a escala dos números produzidos adicionando 1 ao nosso resultado anterior. A Fig. 3.7 confirma que os

resultados estão contidos no intervalo de 1 a 6.

```
1 // Fig. 3.7: fígO3_07.cpp
2 // Inteiros em uma escala ajustada e deslocada gerados
3 #include <iostreani>

4

using std::cout;
using std::endl;

8 #incJ-ude <iomanip>

9

10 using std::setw;
11

12 #include <cstdlib>

13

int main Q
```

Para mostrar que esses números ocorrem com probabilidades aproximadamente iguais, vamos simular 6000 lançamentos de um dado com o programa da Fig. 3.8. Cada inteiro de 1 a 6 deve ser obtido aproximadamente 1000

vezes.

de nuxi o cálcul

```
5
6
7

por 1+randO%6

14
15
16
17
18
19
20
21
22

for ( nt i = 1; i <= 20; i++
cout « setw( 10 ) « ( 1 + (rand () % 6
if ( i % 5 == O
cout « endl;
```

```
23 return 0;
24 )
```

4

4

4

4
4t
4
47

48
49

50

Fig. 3.7 Inteiros em uma escala ajustada e deslocada produzidos por 1 + rand ( ) % 6.

51
52

53

54
55
56
Lac ***Fig.*** 3.1

Fig. 3.8 Lançando um dado de seis faces 6000 vezes (parte 1 de 2).

1	*II* **Fig. 3.8: figo3_08.cpp**	
2	*1/* **Lançar uni dado de**	**seis faces 6000 vezes**
3	**#include <iostreani>**	
4		
5	**using std: :cout;**	
6	**using std::endl;**	

Fig. 3.6 Arquivos de cabeçalho da biblioteca padrão (parte 2 de 2).

3.8 Geração de números aleatórios

Vamos agora fazer um breve e, esperamos, divertido desvio para uma aplicação popular de programação que é a dos jogos e simulações. Nesta seção e na próxima, desenvolveremos um programa de jogo bem-estruturado que inclui múltiplas funções. O programa utiliza a maioria das estruturas de controle que já estudamos.

Há algo no ambiente de um cassino que estimula todos os tipos de pessoas, desde os jogadores profissionais nas suntuosas mesas de mogno e feltro para o jogo de dados *craps,* até os pequenos apostadores em máquinas caça-níqueis. E *ofator sorte,* a possibilidade de que a sorte converta um punhadinho de dinheiro em uma montanha de riqueza. O

fator sorte pode ser introduzido em aplicações de computador usando a função da biblioteca padrão rand.
Considere o seguinte comando:
i = rand
A função rand gera um inteiro entre O e RAND MAX (uma constante simbólica definida no arquivo de cabeçalho <cstdlib>) . O valor de RAND_MAX deve ser pelo menos 32767, que é o valor máximo de um inteiro de dois bytes (i.e., 16 bits). Se rand produz inteiros verdadeiramente aleatórios, todos os números entre O e RAND_MAX terão a mesma *chance* ou *probabilidade* de serem escolhidos cada vez que rand é chamada.
Freqüentemente, o conjunto de valores produzido por rand é diferente do necessário para uma aplicação específica. Por exemplo, um programa que simula o lançamento de uma moeda pode exigir apenas O para "cara" e 1 para "coroa". Um programa de jogo de dados que simula um dado de seis faces exigiria inteiros aleatórios no intervalo de 1 a 6. Um programa que prevê o próximo tipo de espaçonave (entre quatro possibilidades) que surgirá no horizonte em um videogame pode exigir números aleatórios no intervalo de 1 a 4.
Para demonstrar rand, vamos desenvolver um programa para simular 20 lançamentos de um dado de seis faces e imprimir o valor de cada lançamento. O protótipo de função para a função rand pode ser encontrado em

Arquivo de cabeçalho da biblioteca padrão	**Explicação**
<functional>	Contém classes e funções usadas por algoritmos da biblioteca padrão.
<memory>	Contém classes e funções usadas pela biblioteca padrão para alocar memória para os contêineres da biblioteca padrão.
<iterator>	Contém classes para acessar dados nos contêineres da biblioteca padrão.
<algorithm>	Contém funções para manipular dados nos contêineres da biblioteca padrão.
<exception> <stdexcept>	Estes arquivos de cabeçalho contêm classes que são usadas para tratamento de exceções (discutidas no Capítulo 13).
<string>	Contém a definição da classe string da biblioteca padrão (discutida no Capítulo 19, *"Strings").*
<sstreaxn>	Contém protótipos de funções para funções que realizam operações de entrada e saída para strings na memória (discutidas no Capítulo 14).
<locale>	Contém protótipos de classes e funções normalmente usadas para o processamento de *streams* para processar dados na forma natural para diferentes idiomas (por exemplo, formatos de moedas, classificação de *strings,* apresentação de caracteres, etc).
<limits>	Contém classes para definir os limites de tipos de dados numéricos em diferentes plataformas computacionais.
<typeinfo>	Contém classes para a identificação de tipos em tempo de execução (determinar os tipos de dados em tempo de execução).

ser incluído usando-se a diretiva de pré-processador #include. Por exemplo, o arquivo de cabeçalho square h pode ser incluído em nosso programa através da diretiva
#include 'square .h'
no início do programa. A Seção 17.2 apresenta informações adicionais sobre como incluir arquivos de cabeçalho.

<*cassert*> Contém macros e informações para adicionar diagnósticos que ajudam o programador a realizar depuração. A versão antiga deste arquivo de cabeçalho é <assert h>.
<cctype> Contém protótipos para funções que examinam caracteres em busca de determinadas propriedades dos caracteres e para funções que podem ser usadas para converter letras minúsculas em maidsculas e vice-versa. Este arquivo de cabeçalho substitui o arquivo de cabeçalho <ctype h>.
<cfloat> Contém limites do sistema para o tamanho dos números de ponto flutuante. Este arquivo de cabeçalho substitui o arquivo de cabeçalho <float h>.
<climits> Contém os limites do sistema para os tamanhos dos números inteiros. Este arquivo de cabeçalho substitui o arquivo de cabeçalho <limits h>.
<crnath> Contém protótipos de funções da biblioteca matemática. Este arquivo de cabeçalho substitui o arquivo de cabeçalho <math h>.
<cstdio> Contém protótipos de funções da biblioteca padrão de entrada/saída e as informações utilizadas por elas. Este arquivo de cabeçalho substitui o arquivo de cabeçalho
<stdio .h>.
<cstdiib> Contém protótipos de funções para conversão de números em texto e texto em números, alocação de memória, números aleatórios e outras funções com várias finalidades. Este arquivo de cabeçalho substitui o arquivo de cabeçalho <stdlib .h>.
<cstring> Contém protótipos de funções para processamento de strings. Este arquivo de cabeçalho substitui o arquivo de cabeçalho <string h>.

<ctime> Contém protótipos e tipos de funções para manipular horários e datas. Este arquivo de cabeçalho substitui o arquivo de cabeçalho <time h>.
<iostream> Contém protótipos de função para as funções padrões de entrada e saída. Este arquivo de cabeçalho substitui o arquivo de cabeçalho <iostream. h>.
<iomanip> Contém protótipos de função para os manipuladores de *streams* que permitem formatação de *streams* de dados. Este arquivo de cabeçalho substitui o arquivo de
cabeçalho <iomanip h>.
<*fstream*> Contém protótipos de função para funções que executam operações de entrada a partir de arquivos em disco e saída para arquivos em disco (discutidos no Capítulo
14). Este arquivo de cabeçalho substitui o arquivo de cabeçalho <fstream. h>.
<utility> Contém classes e funções que são usadas por muitos arquivos de cabeçalho da biblioteca padrão.
<vector>,<list>, Os arquivos de cabeçalho contêm classes que implementam contêjneres da biblioteca <deqiie>,<qiieue>, padrão. Contêineres são usados para armazenar dados durante a execução de um <stack>,<zmap>, programa. Discutimos estes arquivos de cabeçalho no capítulo intitulado "A <set>,<bitset> biblioteca padrão de gabaritos".
Fíg. 3.6 Arquivos de cabeçalho da biblioteca padrão (parte 1 de 2).

Arquivo de cabeçalho da biblioteca padrão	**Explicação**

quando o tipo de um argumento não corresponde ao tipo do parâmetro especificado na definição da função. A Fig. 3.5 lista os tipos de dados, classificados do "maior tipo" para o "menor tipo".
Tipos de dados

Fig. 3.5 Hierarquia de promoção para os tipos de dados primitivos.
Converter valores para tipos inferiores pode resultar em valores incorretos. Portanto, um valor só pode ser convertido para um tipo inferior atribuindo-se explicitamente o valor a uma variável do tipo inferior ou usando-se um operador de coerção. Os valores dos argumentos da função são convertidos para os tipos dos parâmetros de seu protótipo se forem atribuídos diretamente a variáveis daqueles tipos. Se nossa função square que usa um parâmetro inteiro (Fig. 3.3) fosse chamada com um argumento de ponto flutuante, este seria convertido para int (um tipo inferior), e normalmente square retornaria um valor incorreto. Por exemplo, square (4 . **5**) retornaria 16 e não 20.25.

Erro comum de programação 3.14

Converter *de um tipo superior de dados em uma hierarquia de promoção para um tipo inferior pode modificar o valor dos dados.*

Erro comum de programação 3.15

Esquecer um protótipo de função causa um erro de sintaxe se a definição daquela função aparecer depois de sua chamada no programa.

Observação de engenharia de software 3.10

___ *Um protótipo de função colocado fora de qualquer definição de função se aplica a todas as chamadas daquela função que aparecem depois de seu protótipo no arquivo. Um protótipo de função colocado dentro de uma função se aplica somente às chamadas realizadas naquela função.*

3.7 Arquivos de cabeçalho

Cada biblioteca padrão tem um *arquivo de cabeçalho* correspondente, contendo os protótipos de todas as funções daquela biblioteca e definições dos vários tipos de dados e constantes necessários por elas. A Fig. 3.6 lista alguns arquivos de cabeçalho comuns da biblioteca padrão C++ que poderiam ser incluídos em programas C++. O termo "macros", que é usado diversas vezes na Fig. 3.6, é discutido em detalhes no Capítulo 17, "O pré-processador". Os arquivos de cabeçalho que terminam em .h são arquivos de cabeçalho no "estilo antigo", que foram substituídos pelos arquivos de cabeçalho da biblioteca padrão C++.
O programador pode criar arquivos de cabeçalho personalizados. Os nomes dos arquivos de cabeçalho definidos pelo programador também devem terminar com h. Um arquivo de cabeçalho definido pelo programador pode

long double			
double			
`float`			

unsigned long	**int**	**(sinônimo de unsigned**	**long)**
long int		**(sinônimo de long)**	
unsigned int		**(sinônimo de unsigned)**	
int			
uns igned short	**int**	**(sinônimo de unsigned**	**short)**
short int		**(sinônimo de short)**	
unsigned char			
char			
bool		**(false** toma-se O, true	torna-se 1)

O protótipo da função maximum na Fig. 3.4 é
int maximum(int, int, **int);**
Esse protótipo de função declara que maximum utiliza três argumentos do tipo int e retorna um resultado do tipo int. Observe que o protótipo da função é idêntico à primeira linha da definição da função maximum, exceto pelo fato de que os nomes dos parâmetros (x, y e z) não são incluídos.

Boa prática de programação 3.5

Muitos programadores usam nomes de parâmetros nos protótipos de função para fins de documentação. O compilador ignora esses nomes.

***Erro** comum de programação 3.11*

Esquecer o ponto-e-vírgula no final de um protótipo definção causa um erro de sintaxe.

A parte do protótipo de uma função que inclui o nome da função e os tipos dos seus argumentos é chamada de *assinatura da função* ou, simplesmente, *assinatura.* A assinatura da função não inclui o tipo do valor devolvido pela função.

IErro comum de programação 3.12

Uma chamada de função que não corresponde ao protótipo da função é um erro de sintaxe.

Erro comum de programação 3.13

É *um erro de sintaxe se o protótipo da função e a definição da mesma estiverem em desacordo.*

Por exemplo, na Fig. 3.4, se o protótipo da função tivesse sido escrito como
void rnaximum(int, int, int);
o compilador acusaria um erro, porque o tipo de retorno void no protótipo da função difere do tipo de retorno int no cabeçalho da função.

Outro recurso importante dos protótipos de funções é a *coerção de argumentos,* ou seja, a imposição do tipo apropriado aos argumentos. Por exemplo, a função sqrt da biblioteca matemática pode ser chamada com um argumento inteiro e a função funcionará corretamente, embora o protótipo da função em <cmath> especifique um argumento double. O comando
cout « **sqrt(4);**
calcula corretamente sqrt ***(4)*** e imprime o valor 2. O protótipo da função faz com que o compilador converta o valor do argumento inteiro ***4*** para o valor double ***4*** . O antes do mesmo ser passado para sqrt . Em geral, os valores de argumentos que não correspondem precisamente aos tipos de parâmetros no protótipo da função são convertidos para o tipo apropriado antes de a função ser chamada. Essas conversões podem levar a resultados incorretos se as *regras de promoção* de C++ não forem obedecidas. As regras de promoção especificam como tipos de dados podem ser convertidos para outros tipos sem perda dos dados. Em nosso exemplo sqrt anterior, um int é convertido automaticamente em double sem modificação de seu valor. Entretanto, um double convertido a um int trunca a parte fracionária do valor double. Converter tipos inteiros grandes para tipos inteiros pequenos (p.ex., long para short) também pode resultar em valores modificados.

As regras de promoção são aplicadas automaticamente a expressões que contêm valores de dois ou mais tipos de dados (também chamadas de *expressões de tipo misto).* O tipo de cada valor em uma expressão de tipo misto é promovido automaticamente para o tipo máximo na expressão (na realidade, uma versão temporária de cada valor é criada e usada na expressão - os valores originais permanecem inalterados). Um outro uso comum da promoção é

int max =

```
if ( y > max max =

if ( z > maz) max z;

38 return max;

39 }
```

[

Os três inteiros são digitados. A seguir. os inteiros são passados para **maximum.** que determina o maior inteiro. Este valor é devolvido a **main** pelo comando return em maximurn. O valor devolvido é, então, impresso.

3.6 Protótipos de funções

Um dos recursos mais importantes de C+÷ **é** o *protótipo de função.* Um protótipo de função diz ao compilador o nome da função, o tipo dos dados retornados pela função, o número de parâmetros que a função espera receber, os tipos dos parâmetros e a ordem na qual esses parâmetros são esperados. O compilador usa protótipos de funções para validar as chamadas de funções. As versões anteriores de C++ não realizavam esse tipo de verificação; portanto, era possível chamar funções de forma inadequada sem que o compilador detectasse os erros. Tais chamadas poderiam levar a erros fatais durante a execução ou a erros não-fatais que causavam erros lógicos sutis e difíceis de detectar. Os protótipos de funções corrigem essa deficiência.

seiva ção le adesoftware38

Protótipos de funções são obrigatórios em C++. Use diretivas **#include** *dopré-processadorpara obter protótipos de todas as funções da biblioteca padrão a partir dos arquivos de cabeçalho das bibliotecas apropriadas. Use também* **#include** *para obter os arquivos de cabeçalho que contêm os protótipos de funções usados por você ou pelos membros de sua equipe.*

Observação de engenharia de software ***3.9***
_____ *Um protótipo de função não é obrigatório se a definição da função aparece antes do primeiro uso da função no programa. Em tal caso, a definição da função também serve como o protótipo da função.*

```
// Definição da função maximuiu
1/ x, y e z, abaixo, são parâmetros
li para a definição da função maximum
int maximum( int x, int y, int z


25
26
27
28
29
30
31
32
33
34
35
36
37
```

Fg. 3.4 **Função maximum** definida pelo programador (parte 2 de 2).

```
Forneça três inteiros: 22 85 17
O maior é: 85
```

```
Forneça três inteiros: 92 35 14
O maior é: 92
```

```
Forneça três inteiros: 45 19 98
O maior é: 98
```


Observação de engenharia de software 3.6

Os programas devem ser escritos como conjuntos de pequenas funções. Isso torna os programas mais fáceis de escrever; depurar manter e modificar

Observação de engenharia de software 3.7

Uma função que exige um grande número de parâmetros pode estar realizando tarefas demais. Pense na possibilidade de dividir a função em funções menores, que realizem tarefas separadas. O cabeçalho da função deveria estar contido em uma linha, se possível.

Erro comum de programação 3.10

É um erro de sintaxe se o protótipo da função, o arquivo de cabeçalho e as chamadas da função não concordarem quanto ao número, tipo e ordem dos argumentos e parâmetros e quanto ao tipo do valor de retorno.

Há três maneiras de retomar controle para o ponto no qual uma função foi chamada. Se a função não fornecer um valor como resultado, o controle é retomado simplesmente quando a chave que indica o término da função é alcançada, ou ao se executar o comando

return;

Se a função fomecer um valor como resultado, o comando

return *expressão;*

retorna o valor de *expressão* para a função que realizou a chamada.

Nosso segundo exemplo utiliza uma função definida pelo programador, denominada **maximum,** para determinar e devolver o maior entre três inteiros (Fig. 3.4).

```
1 // Fig. 3.4: fig03_04.cpp
2 // Encontrar o maior de três inteiros
3 #include <iostream>
4
5 using std::cout;
6 using std::cin;
7 using std::endl;
8
9 int maximum ( int, int, int ); // protótipo da função
10
11 int main()
12 {
13 int a, b, c;
14
15 cout « Forneça três inteiros:
16 cin»a»b»c;
17
18 // a, b e c, abaixo, são argumentos
19 // para a chamada da função maximum
20 cout « "O maior é: " « maximum( a, b, c ) « endl;
21
22 return 0;
23 }
24
```

Fig. 3.4 Função maximum definida pelo programador (parte 1 de 2).

Erro comum de programação 3.3

Esquecer de retornar um valor de uma função que deve fazê-lo é erro de sintaxe.

*Erro comum de programação **3.4***

Retornar um valor de uma função cujo tipo de retorno foi declarado void *causa um erro de sintaxe.*
A *lista de parâmetros* é uma lista separada por vírgulas, contendo as declarações dos parâmetros recebidos pela função quando ela é chamada. Se uma função não recebe nenhum valor, a *lista de parâmetros* é void ou simplesmente é deixada vazia. Deve ser listado, explicitamente, um tipo para cada parâmetro na lista de parâmetros de uma função.
Erro comum de programação 3.5
Declarar parâmetros da função do mesmo tipo como float x, y *em vez de* float x, float y. A *declaração de parâmetros* float *x, y geraria na realidade um erro de compilação, porque os tipos são obrigatórios para cada parâmetro na lista de parâmetros.*
Erro comum de ***programação*** *3.6*
Colocar um ponto e vírgula após o parêntese direito, ao encerrar a lista de parâmetros de uma definição de função, é um erro de sintaxe.
Erro comum de programação 3.7
Definir um parâmetro de função novamente como variável local dentro da função é um erro de sintaxe.
Boa prática de programação 3.3
Embora não seja incorreto fazê-lo, não use os mesmos nomes para os argumentos passados para uma função e os parâmetros correspondentes na definição da função. Isso ajuda a evitar ambigüidades.
Erro ***comum de programação*** *3.8*
Os O em uma chamada de função são, na realidade, um operador de C++. Eles fazem com que afunção seja chamada. Esquecer os O em uma chamada de função que não aceita argumentos não é um erro de sintaxe. A função não é invocada quando você provavelmente pretendia que ela fosse.
As declarações e os *comandos* entre chaves formam o *corpo da função,* também chamado de *bloco.* Um bloco é simplesmente um comando composto que inclui declarações. As variáveis podem ser declaradas em qualquer bloco e os blocos podem estar aninhados. *Uma fimção não pode ser definida no interior de outra função sob quaisquer circunstâncias.*
Erro comum de programação ***3.9***
Definir uma função dentro de outra função é um erro de sintaxe.
Boa prática de programação ***3.4***
Escolher nomes significativos para funções e parâmetros torna os programas mais legíveis e ajuda a evitar o uso excessivo de comentários.
Observação de engenharia de software ***3.5***
Uma função deveria caber inteira na janela de um editor. Independentemente de quão longa seja uma função, ela deveria executar bem uma tarefa. Funções pequenas favorecem a reutilização do software.


```
16 return 0;
17
18
19 /l Definição da função
20 int square( int y
21 {
22 return y *y;
23 }
```

```
1 4 9 16 25 36 49 64 81 100
```

Fig. 3.3 Criando e usando uma função definida pelo programador (parte 2 de 2).

Boa prática de programação 3.2
Coloque uma linha em branco entre as definições das funções para separá-las e melhorar a legibilidade do programa.
A função square *é chamada* ou *invocada* em main com o comando
square(**x**
A função square recebe uma cópia do valor de x no *parâmetro* y. Então, square calcula y * y. O resultado é passado de volta para o ponto em main onde square foi chamada. Esse processo é repetido dez vezes, usandos-se a estrutura de repetição for.
A definição de square mostra que ela espera receber um parâmetro inteiro y. A palavra- chave int antes do nome da função indica que square retorna um resultado inteiro, O comando return em square passa o resultado do cálculo de volta para a função chamadora.

A linha 8
int square(int); *II* protótipo da função
é um *protótipo de função.* O tipo de dados int entre parênteses informa ao compilador que **square** espera receber um valor inteiro da função que faz a chamada. O tipo de dados int à esquerda do nome da função square informa ao compilador que square retorna um resultado inteiro à função que faz a chamada. O compilador consulta o protótipo da função para verificar se as chamadas de square contêm o tipo correto do valor de retorno, o número correto de argumentos e os tipos corretos dos argumentos e se os argumentos estão na ordem correta, O protótipo da função não é requerido se a definição da função aparece antes do primeiro uso da função no programa. Em um caso assim, a definição da função também funciona
como protótipo da função. Se as linhas 20 a 23, na Fig. 3.3, estivessem antes de main, o protótipo da função *de* na linha 8 seria desnecessário. Os protótipos de funções são analisados com mais detalhes na Seção 3.6. slmpl(
O formato de uma definição de função é eos bI
circun
tipo do valor de retorno nome da função (lista de parâmetros)
declarações e comandos
O *nome da função* é qualquer identificador válido. O *tipo do valor de retorno* é o tipo de dado do resultado devolvido pela função que fez a chamada. Um *tipo do valor de retorno* void indica que a função não retorna um valor.
Erro comum de programação 3.2
Omitir o tipo do valor de retorno em uma definição de função é um erro de sintaxe.
u

3.4 Funções

As funções permitem ao programador modularizar um programa. Todas as variáveis declaradas em definições de função são *variáveis locais* - elas só são conhecidas na função na qual são definidas. A maioria das funções tem uma lista de *parâmetros* que provêem os meios para transferir informações entre funções. Os parâmetros de uma função também são variáveis locais.
Observação de engenharia de soflware 3.2
___ Em programas que contêm muitas funções, main deveria ser implementada como um grupo de chamadas afiwções que executam o "grosso" do trabalho do programa.
Existem várias motivações para se "funcionalizar" um programa. A abordagem dividir para conquistar toma o desenvolvimento de programas mais administrável. Uma outra motivação é a *reutilização de software* - usar funções existentes como blocos de construção para criar programas novos. A capacidade de reutilização do software é um dos fatores mais importantes para a programação orientada a objetos. Com boa definição e nomenclatura das funções, os programas podem ser criados a partir de funções padronizadas que realizem tarefas específicas, em vez de serem construídos usando código personalizado. Um terceiro motivo é evitar a repetição de código em um programa. Incluir código em uma função permite que ele seja executado em vários locais de um programa simplesmente chamando a função.
Observação de engenharia de software 3.3
___ Cada função deve se limitar a realizar uma tarefa simples e bem-definida e o nome da função deve expressar efetivamente aquela tarefa. Isto promove a reutilização do software.
IObservação de engenharia de software 3.4
_____ Se você não puder escolher um nome conciso que expresse o que a função faz, é possível que sua função esteja tentando realizar muitas tarefas diferentes. Normalmente, é melhor dividir tal função em várias funções menores.

3.5 Definições de funções

Cada programa que apresentamos consiste em uma função denominada main que chamou as funções da biblioteca padrão para realizar suas tarefas. Agora, vamos analisar como os programadores escrevem suas próprias funções personalizadas.
Considere um programa que use a função **square** para calcular os quadrados dos números inteiros de 1 a 10 (Fig. 3.3).

```
1 II Fig. 3.3: figO3O3.cpp
2 // Criando e usando uma função definida pelo programador
3 #include <iostream>
5 using std: :cout;
6 using std: :endl;
8 int square( int ); // protótipo da função
9
10 int main()
11
12 for ( int x =1; x <10; x++
```

```
13 cout « square ( x ) «
15 cout « endl;
```

Fig. 3.3 Criando e usando uma função definida pelo programador (parte 1 de 2).

em um cabeçalho de função pode ser lida como" count é uma referência para um int". Na chamada da função, simplesmente mencione o nome da variável e ela será passada por referência. Então, mencionar a variável pelo nome do seu parâmetro, no corpo da função chamada, na realidade faz uma referência à variável original na função que chamou e a variável original pode ser modificada diretamente pela função chamada. Como sempre, o protótipo e o cabeçalho da função devem estar de acordo.

A Fig. 3.20 compara a chamada por valor e a chamada por referência com parâmetros passados por referência. Os "estilos" dos argumentos nas chamadas a squareByValue e squareByReference são idênticos, ou seja, ambas as variáveis são simplesmente referidas por nome. Sem verificar protótipos ou definições de funções, não é possível dizer, somente com base nas chamadas, se as funções podem modificar seus argumentos. Porém, como os protótipos de função são obrigatórios, o compilador não tem problemas para resolver a ambigüidade.

Erro comum de programação 3.26

Como parâmetros por referência são referidos somente por nome no corpo da função chamada, o programador pode, inadvertidamente, tratar parâmetros passados por referência como parâmetros passados por valor. isto pode ocasionar efeitos colaterais inesperados, se as cópias originais das variáveis são alteradas pela função que está chamando.

```
1 //Fig. 3.20: figO32O.cpp
2 //Comparando uma chamada por valor com uma chamada por referência
3 // com referências.
4 #include <iostream>
5
using std::cout;
using std::endl;
int squareByValue( int );
void squareByReference ( int & );
int main()
int x =2, z =4;


6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
```

```
cout « "x = " « x «
« "Valor retornado
« squareByValue( x
« "x = " « x «
cout « "z = " « z «
squareByReference ( z );
cout « "z = " « Z «
return 0;


antes de squareByValue\n"
por squareByValue:
« endl
depois de squareByValue\n" « endl;
antes de squareByReference" « endl;
depois de squareByReference" « endi;


int squareByValue( int a
return a = a; // argumento da chamadora não modificado
void squareByReference( int &cRef
cRef = cRef; // argumento da chamadora modificado
```

Fig. 3.20 Um exemplo de chamada por referência (parte 1 de 2).


```
x = 2 antes de sqi.iareByValue
Valor retornado por squareByValue: 4
x 2 depois de squareByValue
z = 4 antes de sqi.iareByReference
z = 16 depois de squareByReference
```

Fig. 3.20 Um exemplo de chamada por referência (parte 2 de 2).

No Capítulo 5, discutimos ponteiros; veremos que ponteiros oferecem uma forma alternativa de chamada por referência, na qual o estilo da chamada indica claramente uma chamada por referência (e o potencial para modificar os argumentos da função que chamou).

Dica de desempenho 3.12

f *Para passar objetos grandes, use um parâmetro por referência constante para simular a aparência e segurança de uma chamada por valor e evitar o* overhead *de passar uma cópia do objeto grande.*

Para especificar uma referência para uma constante, coloque o qualificador const antes do especificador do tipo, na declaração do parâmetro.

Note a colocação do **&** na lista de parâmetros da função squareByReference. Alguns programadores de C++ preferem escrever int& cRef em vez de int &cRef.

Observação de engenharia de software 3.19

___ *Tanto para fins de clareza como de desempenho, muitos programadores de C+ + preferem que argumentos modificáveis sejam passados para funções por meio de ponteiros, argumentos pequenos não-modificáveis sejam passados por chamadas por valor e argumentos grandes não-modificáveis sejam passados para funções usando referências para constantes.*

‘

Referências também podem ser usadas como *aliases* para outras variáveis dentro de uma função. Por exemplo, o código

```
int count = 1; // declara a variável inteira count
int &cRef = count; // cria cRef como um alias para count
++cRef; // incrementa count (usando seu alias)
```

incrementa a variável count usando seu *alias* **cref.** Variáveis de referência devem ser inicializadas em suas declarações (ver Fig. 3.21 e Fig. 3.22) e não podem ser novamente atribuídas como *alias* para outras variáveis. Uma vez que uma referência é declarada como um *alias* para outra variável, todas as operações supostamente executadas no *alias* (i.e., a referência) são na realidade executadas na própria variável original. O *alias* é simplesmente um outro nome para a variável original. Pegar o endereço de uma referência e comparar referências não causa erros de sintaxe; em vez disso, cada operação na realidade ocorre na variável para a qual a referência é um *alias.* Um argumento por referência deve ser um *Ivalue,* não uma constante ou expressão que retorna um *rvalue.*

```
1 // Fig. 3.21: figO3_21.cpp
```

```
2 // Referências devem ser inicializadas
3 #include <iostream>
4
5 using std: :cout;
6 using std: :endl;
7
```

Fig. 3.21 Usando uma referência não-inicializada (parte 1 de 2).

Erro comum de programação 3.27

Declarar múltiplas referências em um só comando, supondo que o & se distribui por uma lista de nomes de variáveis separados por vírgulas. Para declarar as variáveis x, y e **z,** *como referências para inteiros, use a notação* int &x = *a, &y* = b, &z = b **;** *em vez da notação incorreta* int& x = *a, y* = b, z = c; *ou de outra notação incorreta freqüentemente encontrada:* **int &x, y, z;.**

Funções podem retornar referências, mas isto pode ser perigoso. Quando retomar uma referência para uma variável declarada na função chamada, a variável deveria ser declarada static naquela função. Caso contrário, a referência referir-se-á a uma variável automática que é descartada quando a função termina; tal variável é dita "indefinida" e o comportamento do programa será imprevisível (alguns compiladores emitem mensagens de advertência quando isto é feito). Referências para variáveis indefinidas são chamadas de *refrrências sem correspondente.*

Erro comum de programação 3.28
Não inicializar uma variável de referência quando ela é declarada é um erro de sintaxe.

Erro comum de programação 3.29
Tentar atribuir a uma referência previamente declarada um alias para outra variável é um erro de lógica. O valor da outra variável é simplesmente atribuído para a localização para a qual a referência já é um alias.

```
1 II Fig. 3.22: fig0322.cpp
2 // Referências devem ser inicializadas
3 #include <iostream>


4


5 using std: :cout;
6 using std: :endl;


7


int main Q


int x = 3, &y; II Erro: y deve ser inicializado


cout « "x = " y = 7;
cout « 'x = " «


return O;


8 int main O
9
10 int x = 3, &y = x; II y é agora um alias para x
11
```

```
12 cout « X = " « x « endl « "y = " « y « endi;
13 y=7;
14 cout « 'X = ' « x « endi « "y = " « y « endi;
15
16 return O;
17 )
```

Fig. 3.21 Usando uma referência não-inicializada (parte 2 de 2).

```
8
9
10
11
12
13
14
15
16
17
```

```
x « endl « "y = " « y « endl;
x « endi « "y = " « y « endi;
```

Fig. **3.22** Tentando usar uma referência não-inicializada (parte 1 de 2).

Mensagem de erro do compilador Borland C+ + com linha de comando

Error E2304 figo3_22.cpp 10: Reference variable y' must be initialized

in function main()

Mensagem de erro do compilador Microsoft Visual C+ +

```
__figo3_22.cpp(10) : error C2530 : 'y' : references must be initialized
```

Fig. Fig. 3.22 Tentando usar uma referência não-inicializada (parte 2 de 2).

No Ca , ***Erro comum de programação 3.30***

renda,

argum *Retomar um ponteiro ou referência para uma variável automática em uma função chamada é um err lógica. Alguns compiladores emitirão uma mensagem de advertência quando isto ocorrer em um progrt*

3.18 Argumentos default

Chamadas a funções devem geralmente passar um valor particular de um argumento. O **programador pode espe** Para e car que tal argumento seja um *argumento default* e o programador pode fornecer um valor default para esse a na dec mento. Quando um argumento default é omitido na chamada de uma função, o valor default daquele argumen automaticamente inserido pelo compilador e passado na chamada.

C++ Argumentos default devem ser os argumentos mais à direita (finais) na lista de parâmetros da função. Qua se chama uma função com dois ou mais argumentos default, se um argumento omitido não é o argumento m direita na lista de argumentos, todos os argumentos à direita desse argumento também devem ser omitidos. A mentos default devem ser especificados com a primeira ocorrência do nome da função - tipicamente no protól Valores default podem ser constantes, variáveis globais ou chamadas de função. Argumentos default também po ser usados com funções mime.

A Fig. 3.23 demonstra a utilização de argumentos default no cálculo do volume de uma caixa. O prótotip função para boxVolume, na linha 5, especifica que todos os três argumentos recebem valores default de 1. - que os valores default devem ser definidos somente no protótipo da função. Também note que fornecemos nome Refere variáveis no protótipo da função, por razões de legibilidade. Como sempre, nomes de variáveis não são requer codigc em protótipos de funções.

A primeira chamada para boxVolume (linha 12) não especifica argumentos e, assim, usa todos os valores default. A segunda chamada (linha 14) passa um argumento length e, por isso, usa valores default par argumentos width e height. A terceira chamada (linha 16) passa argumentos length e width e, assim. um valor default para o argumento height. A última chamada (linha 18) passa argumentos para length. wié e height e, por isso, não usa valores default.

incren

dec lar ***Boa prática de programação 3.12***

vez qu

no *alic Usar argumentos default pode simplificar a codificação de chamadas de funções. Contudo, alguns*
nome *gramadores acham que especificar explicitamente todos os argumentos é mais claro.*

Erro comum de programação 3.31

Especificar e tentar usar um argumento default que não é o argumento final (mais à direita) da list argumentos (sem, simultaneamente, tomar default os outros argumentos mais à direita) é um erro de Sint

1

```
2 , 1 II Fig. 3.23: figo3_23.cpp
3 2 II Usando argumentos default
4 3 #inciude <iostreain>
51 4
6 5 using std::cout;
6 using std::endl;
```

Fig. 3 Fig. 3.23 Usando argumentos default (parte 1 de 2).


```
7
8 int boxVolume ( int length =1, int width =1, int height =1 );
•ized g
10 int main()
11
12 cout« "O volume default da caixa é: -« boxVolume()
d 13 « "\n\nO volume de uma caixa com comprimento 10,\n"
e 14 « "largura 1 e altura 1 é: -« boxVolume( 10
____15 « "\n\nO volume de uma caixa com comprimento l0,\n"
16 « 'largura 5 e altura 1 é: '« boxVolume( 10, 5
17 « "\n\nO volume de uma caixa com comprimento 10,\n"
18 « "largura 5 e altura 2 é: -« boxVolume( 10, 5, 2
___19 « endl;
éumerrode 20
npmgrama. 21 return O;
24 /l Calcular o volume da caixa
25 int boxVolume(int length, int width, int height)
. 26 {
Jueespeciii- 27 return length * width * height;
aesseargu- 28
argumento é
O volume default da caixa é: 1
ao. Quando O volume de uma caixa com comprimento 10,
lento mais a largura 1 e altura 1 é: 10
tidos. Arguonrotóti
O volume de uma caixa com comprimento 10,
• largura 5 e altura 1 é: 50
bem podem
O volume de uma caixa com comprimento 10,
prótotipoda largura 5 e altura 2 é: 100
tde 1. Note
osnomesde Fig. 3.23 Usando argumentos default (parte 2 de 2).
requeridos
```

dos os três

3.19 Operador unário de resolução de escopo

Cault para os
assim, usa E possível declarar variáveis locais e globais com o mesmo nome. C++ oferece o *operador unário de resolução de* th. width. *escopo (:* :) para acessar uma variável global quando uma variável local com o mesmo nome está no escopo. O
operador unário de resolução de escopo não pode ser usado para acessar uma variável local do mesmo nome a partir de um bloco externo. Uma variável global pode ser acessada diretamente, sem o operador unário de resolução de escopo, se o nome da variável global não for o mesmo que o nome de uma variável local no escopo. No Capítulo 6, discutimos o uso do *operador binário de resolução de escopo* com classes.

alguns pro- ...

A Fig. 3.24 mostra o uso do operador unano de resoluçao de escopo com variaveis locais e globais com o mesmo nome. Para enfatizar que as versões locais e globais da variável constante P1 são distintas, o programa declara

uma das variáveis como double e a outra como float.
da ksia de Erro comum de programação 3.32
de sintaxe. Tentar acessar uma variável não-global em um bloco externo usando o operador unário de resolução de escopo é um erro de sintaxe se não existir uma variável global com o mesmo nome no bloco externo e um erro de lógica se existir uma.

```
1 // Fig. 3.24: figo3_24.cpp
2 // Usando o operador unário de resolução de escopo
3 #include <iostream>
```

Fig. 3.24 Usando o operador unário de resolução de escopo (parte 1 de 2).


```
4
5 using std::cout;
6 using std: :endl;
7
8 #include <iomanip>
9
10 using std::setprecision;
11
12 const double P1 = 3.14159265358979;
13
14 int main O
15
16 const float P1 = static cast< float >( ::PI );
17
18 cout « setprecision( 20
19 « "Valor local float de P1 = " « P1
20 « '\nValor global double de P1 = « ::PI « endl;
21
22 return 0;
23 )
```

Saída do compilador Borland C+ + com linha de comando

Fig. 3.24 Usando o operador unário de resolução de escopo (parte 2 de 2).

*Boa prática de programação **3.13***
Evite usar variáveis com o mesmo nome para propósitos diferentes em um programa. Embora isso seja permitido em várias circunstâncias, pode gerar confusão.
ch

3.20 Sobrecarga de funções 01

C++ possiblita que sejam definidas várias funções com o mesmo nome, desde que estas funções tenham conjuntos de parâmetros diferentes (pelo menos quanto a seus tipos). Este recurso é chamado de *sobrecarga de funções.* Quando é chamada uma função sobrecarregada, o compilador C++ seleciona a função apropriada pelo exame da quantidade, dos tipos e da ordem dos argumentos na chamada. A sobrecarga de funções é comumente usada para criar várias funções do mesmo nome que executam tarefas semelhantes, mas sobre tipos de dados diferentes.
Boa prática de programação 3.14
Sobrecarregarfunções que executam tarefas proximamente relacionadas torna um programa mais legível e compreensível.
A Fig. 3.25 usa a função sobrecarregada square para calcular o quadrado de um int e o quadrado de um double. No Capítulo 8, discutimos como sobrecarregar operadores para definir como eles devem operar sobre objetos de tipos de dados definidos pelo usuário. (Na verdade, temos usado muitos operadores sobrecarregados até aqui, inclusive o operador de inserção em *streams* « e o operador de extração de *st reams* ». Falaremos mais sobre sobrecarregar « e » no Capítulo 8). A Seção 3.21 introduz gabaritos de funções para gerar automaticamente funções sobrecarregadas que executam tarefas idênticas sobre tipos de dados diferentes. No Capítulo 12, discutimos gabari to de função e gabaritos de classe em detalhes. F

Valor local float de P1 =	**3.141592741012573242**

Valor global double de P1	= **3.141592653589790007**
Saída do compilador Microsofi C+ +	
Valor local float de P1 =	**3.1415927410125732**
Valor global double de P1	**3.14159265358979**


```
1 // Fig. 3.25: fig0325.cpp
2 // Usando funções sobrecarregadas
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 int square( int x ) { return x * x;
9
10 double square ( double y ) { return y *
11
12 int main O
13
14 cout« 'O quadrado do inteiro 7 é -« square( 7
15 « "\nO quadrado do double 7.5 é -« square( 7.5
16 « endi;
17
18 return O;
19
O quadrado do inteiro 7 é 49
O quadrado do double 7.5 é 56.25
```

Fig. 3.25 Usando funções sobrecarregadas.

Funções sobrecarregadas são distinguidas por suas *assinaturas* - *uma assinatura* é uma combinação do nome de uma função e seus tipos de parâmetros. O compilador associa cada identificador de função com o número e os tipos de seus parâmetros (às vezes chamado de *adulteração* ou *decoração do nome)* para garantir uma *ligação segura quanto ao tipo* dos parâmetros. A ligação segura quanto ao tipo assegura que a função sobrecarregada apropriada seja chamada e que os argumentos estejam em conformidade com os parâmetros. Os erros de ligação são detectados e acusados pelo compilador. A Fig. 3.26 foi compilada com o compilador C++ da Borland. Em vez de mostrar a saída produzida pela execução do programa (como normalmente fazemos), mostramos os nomes de função adulterados produzidos em linguagem simbólica pelo C++ da Borland. Cada nome adulterado começa com @ seguido pelo nome da função. A lista de parâmetros adulterada começa com $q. Na lista de parâmetros para a função **nothing2,** c representa um char, i representa um int, pf representa um float* e pd representa um double*. Na lista de parâmetros para função nothingl. i representa um int, f representa um float, c representa um char e pi representa um int*. As duas funções square são diferenciadas por suas listas de parâmetros; uma especifica d significando double e a outro especifica i significando int. Os tipos de retomo das ftinções não são especificados nos nomes adulterados. A adulteração de nomes de função é específica de cada compilador. Funções sobrecarregadas podem ter tipos de retomo diferentes ou não, mas devem ter listas de parâmetros diferentes.

```
1 // Fig. 3.26: fig0326.cpp
2 // Alterando nomes
3
4 int square( int x ) { return x * x;
5
6 double square ( double y ) { return y * y; }
7
8 void nothingl( int a, float b, char c, int *d
9 { } // corpo de função vazio
10
11 char *nothing2( char a, int b, float *c, double *d
12 { return O;
13
14 int main()
15
```

Fig. 3.26 Alterando o nome para possiblitar a ligação segura quanto aos tipos (parte 1 de 2).


```
16 return O;
```

```
17
main
@ nothing2 $qcipfpd
@nothingl$qifcpi
@square$qd
@square$qi
```

Fig. 3.26 Alterando o nome para possiblitar a ligação segura quanto aos tipos (parte 2 de 2).

Erro comum de programação 3.33

Criarfunções sobrecarregadas com listas de parâmetros idênticas e tipos diferentes de retorno é um erro de sintaxe. O compilador usa somente as listas de parâmetros para distinguir entre funções do mesmo nome. Funções sobrecarregadas não necessitam ter o mesmo número de parâmetros. Os programadores devem ser cautelosos quando sobrecarregarem funções com parâmetros default, pois isto pode causar ambigüidades.

Erro comum de programação 3.34

Uma função com parâmetros default omitidos pode ser chamada de forma idêntica a outra função sobrecarregada; isto é um erro de sintaxe. Por exemplo, terem um programa tanto uma função que explicitamente não aceita nenhum argumento e uma função com mesmo nome cujos parâmetros são todos default, provoca um erro de sintaxe quando éfeita uma tentativa de usar aquele nome de função em uma chamada sem passar nenhum argumento.

3.21 Gabaritos de funções

Funções sobrecarregadas são usadas normalmente para executar operações semelhantes que envolvem lógicas de programação distintas para tipos de dados diferentes. Se a lógica dos programas e suas operações são idênticas para os vários tipos de dados, isto pode ser codificado mais compacta e convenientemente usando-se *gaba ritos de função.* O _____ programador escreve uma única definição do gabarito de função. Dados os tipos dos parâmetros fornecidos nas chama da desta função, C++ gera automaticamente *Jlnções gabarito* separadas para tratar cada tipo de chamada de forma
apropriada. Deste modo, detinindo-se um único gabarito da função, define-se também uma famflia inteira de soluções. Todas as definições de gabaritos de função começam com a palavra-chave template seguida por uma lista de parâmetros de tipo formais para o gabarito de função colocada entre os símbolos **<** e >. Todo parâmetro de tipo formal é precedido pela palavra-chave typename ou pela palavra-chave class. Os *parâmetros de tipo formais* são tipos primitivos ou tipos definidos pelo programador, usados para especificar os tipos dos parâmetros da função, especificar o tipo de retorno da função e para declarar variáveis dentro do corpo da definição da função. A definição de função vem a seguir e é definida como qualquer outra função.
A definição de gabarito de função a seguir é usada também na Fig. 3.27.

```
teniplate <class T> II ou template< typename T >
T maximuxn( T valuel, T value2, T value3
T max valuel;

if ( value2 > max
max = value2;

if ( value3 > max
max = value3;
return max;
```


Este gabarito de função declara um parâmetro de tipo formal único, T, como o tipo dos dados que serão testados pela função maximo. Quando o compilador descobre uma chamada a maximo no código fonte do programa, o tipo dos dados passados para maximo substitui T ao longo de toda a definição do gabarito e C++ cria uma função completa para determinar o máximo de três valores do tipo de dados especificado. Então, a função recém-criada é compilada. Assim, gabaritos na realidade são uma maneira de gerar código. Na Fig. 3.27, três funcões são instanciadas - uma espera receber três valores `int, uma espera três valores double e uma espera três valores` char. O gabarito de função criado para o tipo int é:

```
int maximum( int valuel, int value2, int value3
int max = valuel;
if ( value2 > max
max value2;
if ( value3 > max
max = value3;
return max;
```

`O nome de` um parâmetro de tipo deve ser único na lista de parâmetros formais de `uma definição de gabarito particular. A Fig. 3.27 ilustra o uso do gabarito de função maximo`

para determinar o maior de três valores int, três valores double e três valores char.

```
1 // Fig. 3.27: figo3_27.cpp
2 // Usando um gabarito de função
3 #include <iostream>
4
5 using std::cout;
6 using std::cin;
7 using std::endl;
8
9 template < class T >
10 T maximum ( T valuel, T value2, T value3
11 {
12 T max =valuel;
13
14 if ( value2 > max
15 max =value2;
16
17 jf ( value3 > max)
18 max =value3;
19
20 return max;
21 }
22
23 int main O
24 {
25 int inti, int2, int3;
26
27 cout « Forneça três valores inteiros:
28 cm » intl » int2 » int3;
29 cout « "O maior valor inteiro é:
30 « maximuni( inti, int2, int3 ); // versão para int
31
```

Fig. 3.27 Usando um gabarito de função (parte 1 de 2).


```
32 double doublel, double2, double3;
33
34 cout « '\nForneça três valores double: ";
35 cm » doublel » double2 » double3;
36 cout « 'O maior valor double é:
37 « maximum( doublel, double2, double3); // versão para double
38
39 char chari, char2, char3;
40
41 cout « "\nForneça três caracteres: ";
42 cm » chari » char2 » char3;
43 cout « "O maior valor de caractere é:
44 « maximwn( chari, char2, char3 ) // versão para char
45 « endi;
46
47 return 0;
48 }
Forneça três valores inteiros: 1 2 3
O maior valor inteiro é: 3
Forneça três valores double: 3.3 2.2 1.1
O maior valor double é: 3.3
Forneça três caracteres: A C 2
O maior valor de caractere é: C
```

Fig. 3.27 Usando um gabarito de função (parte 2 de 2).

IErro comum de programação 3.35

Não colocar a palavra-chave das s *ou a palavra-chave* typename *antes de cada parâmetro de tipo de um gabarito de*

função é um erro de sintaxe.

3.22 (Estudo de caso opciona') Pensando em objetos: identificando os atributos de uma c'asse

Na seção "Pensando em objetos" no fim do Capítulo 2, começamos a primeira fase de um projeto orientado a objetos (OOD) para o nosso simulador de elevador - identificar as classes necessárias para implementar o simulador. Começamos listando os substantivos na definição do problema e criamos uma classe separada para cada categoria de substantivos que executa um trabalho importante na simulação do elevador. Representamos, então, as classes e seus relacionamentos em um diagrama de classes UML. Classes possuem *atributos* e *operações.* Atributos de classes são implementados em programas C++ como dados; operações de classes são implementadas como funções. Nesta seção, determinaremos muitos dos atributos de classes necessários para implementar o simulador do elevador. No Capítulo 4, determinaremos as operações. No Capítulo 5, vamos nos concentrar nas interações, freqüentemente chamadas de *colabo rações,* entre os objetos no simulador de elevador.

Considere os atributos de alguns objetos do mundo real. Os atributos de uma pessoa incluem altura e peso. Os atributos de um rádio incluem sua sintonia de estações, o ajuste de volume e se ele está ajustado para AM ou FM. Os atributos de um carro incluem as leituras de seu velocímetro e odômetro, a quantidade de combustível no tanque, que marcha está engrenada, etc. Os atributos de um computador pessoal incluem fabricante (por exemplo. Apple, LBM ou Compaq), tipo de tela (por exemplo, monocromática ou a cores), tamanho da memória principal (em megabytes), tamanho do disco rígido (em gigabytes), etc.

Atributos descrevem classes. Podemos identificar os atributos de nosso sistema procurando palavras e frases descritivas na definição do problema. Para cada palavra ou frase descritiva que encontramos, criamos um atributo e atribuímos aquele atributo a uma classe. Também criamos atributos para representar qualquer dado de que uma classe possa necessitar. Por exemplo, a classe **Scheduler** precisa saber a hora para criar a próxima pessoa a entrar em cada um dos andares. A Fig. 3.28 é uma tabela que lista as palavras ou frases da definição do problema que descrevem cada classe.

Fig 3.28 Palavras e frases descritivas na definição do problema.

Observe que, para as classes Bell e Building, não foi listado nenhum atributo. À medida que progredirmos ao longo deste estudo de caso, continuaremos a adicionar, modificar e eliminar informações sobre cada uma das classes em nosso sistema.

A Fig. 3.29 é um diagrama de classes que lista alguns dos atributos para cada classe em nosso sistema - estes atributos são criados a partir das palavras e frases descritivas da Fig. 3.28. No diagrama de casos da UML, os atributos de uma classe são colocados no compartimento do meio do retângulo que representa a classe. Considere o seguinte atributo da classe Elevator:

capacity : int = 1

currentFloor: int = 1 direction : enum = up capacity : int = 1 arrivalTime : int moving : bool = false

1

Elevator

Clock

time: int = O

Door

open : bool = false

Floor

Scheduler

capacity: int = 1 occupied : bool false

floorlArrivalTime : int tloor2ArrivalTime : int

Beli

<nenhum ainda>

FloorButton

pressed : bool = false

Person
ID: int

Light

on : bool = false

ElevatorButton

pressed : bool = false

Buildincj

<nenhum ainda>

Fig. 3.29 Diagrama de classes mostrando os atributos.

Classe	Palavras e frases descritivas
Elevator	começa o dia esperando no pavimento 1 do edifício alterna direções: subindo e descendo capacidade de 1 leva 5 segundos para se mover de um andar para o outro elevador se movendo
Clock	começa o dia marcando a hora 0
Scheduler	[escalona os tempos de chegada de pessoas paraj um inteiro aleatório entre 5 e 20 segundos em direção ao futuro a partir do momento atual (para cada andar)
Person	número da pessoa (da saída)
Floor	capacidade de 1 está ocupado / desocupado
FloorButton	foi pressionado
ElevatorButton	foi pressionado
Door	porta aberta / porta fechada

Beil	nenhuma na definição do problema
Light	luz acesa / apagada
Building	nenhuma na definição do problema

Esta listagem contém três componentes de informação sobre o atributo, O atributo tem um *nome* - **capacity.** O atributo também tem um *tipo,* **int.** O tipo depende da linguagem usada para escrever o sistema de software. Em C++, por exemplo, o valor pode ser um tipo primitivo, tal como int. char ou float, bem como um tipo definido pelo usuário, como uma classe (começaremos nosso estudo de classes no Capítulo 6, onde veremos que cada nova classe é, em essência, um novo tipo de dado).

Também podemos indicar um ***valor*** *inicial* para cada atributo, O atributo capacitiem um valor inicial de 1. Se um atributo em particular não tem um valor inicial especificado, somente seu nome e tipo (separados por dois pontos) são mostrados. Por exemplo, o atributo floorArrrivalTime da classe Scheduler é do tipo int. Aqui não mostramos nenhum valor inicial porque o valor deste atributo é um número aleatório que ainda não conhecemos; o número aleatório vai ser determinado durante a execução. Por enquanto, não nos preocuparemos demais com os tipos ou valores iniciais dos atributos. Incluímos somente a informação que podemos vislumbrar na definição do problema.

Diagramas de mapa de estados

Objetos em um sistema podem ter ***estados.*** Estados descrevem a condição de um objeto em um determinado instante no tempo. *Diagramas de mapa de estados* (também chamados de diagramas de *estados)* nos dão uma maneira de expressar como, e sob quais condições, os objetos em um sistema mudam de estado.

A Fig. 3.30 é um diagrama de mapa de estados simples que modela os estados de um objeto da classe FloorButton ou da classe E].evatorButton. Cada estado, em um diagrama de mapa de estados, é representado por um retângulo com cantos arredondados, com o nome do estado colocado dentro dele. Um círculo cheio com uma seta presa a ele aponta para o estado inicial (i.e., o estado "Não pressionado"). As linhas cheias com setas indicam *transições* entre estados. Um objeto pode passar de um estado para outro em resposta a um *evento.* **Por** exemplo, as classes FloorButton e ElevatorButton mudam do estado "Não pressionado" para o estado "Pressionado" em resposta ao evento "apertar botão". O nome do evento que causa a transição é escrito junto à linha que corresponde àquela transição (podemos incluir mais informações sobre eventos, como veremos).

pressionaJ

apertar botão soltar botão

iodJ

Fig. 3.30 Diagrama de estados para as classes FloorButton e ElevatorButton.

A Fig. 3.31 mostra o diagrama de mapa de estados para a classe Elevator. O elevador tem três estados possíveis: "Esperando", "Atendendo andar" (i.e., o elevador está parado em um andar, mas está ocupado desligando o botão do elevador ou se comunicando com o andar, etc.) e "Movendo-se". O elevador inicia no estado "Esperando". Os eventos que provocam as transições estão indicados junto as linhas de transição apropriadas.

[nenhuma chamada adicional]

apertar botão [no outro andar]! arrivalTime = currentTime + 5

apertar botão [precisa se mover]

apertar botão

Atendendo andar

saída! fechar a porta

Fig 3.31 Diagrama de estados para a classe Elevator.

Examinemos os eventos neste diagrama de mapa de estados. O texto

apertar botão [precisa se mover]

nos diz que o evento "apertar botão" faz com que o elevador passe do estado "Atendendo andar" para o estado "Movendo-se". A *condição guarda,* entre colchetes, indica que a transição acontece somente se o elevador precisa se mover, O texto completo do evento indica que o elevador passa do estado "Atendendo andar" para o estado "Movendo-se" em resposta ao evento "apertar botão" somente se o elevador precisar se mover. Similarmente, as transições do elevador do estado "Esperando" para o estado "Atendendo andar" indicam quando é apertado um botão no andar em que o elevador está atualmente.

O texto junto à linha de transição do estado "Esperando" para o estado "Movendo-se" indicam que esta transição ocorre no caso de um botão ser apertado (se o botão é apertado no outro andar). A barra ***("1")*** indica que uma *ação* acompanha esta mudança de estado. O elevador executa a ação de calcular e ajustar a hora em que vai chegar ao outro andar.'

Uma transição de estado também pode ocorrer no caso de uma certa condição ser verdadeira, O texto

quando [currentTime == arrivalTime]

indica que o elevador passa do estando "Movendo-se" para o estado "Atendendo andar" *quando* a hora atual na simulação é igual à hora em que o elevador deve chegar em um andar.

O texto que está junto à linha de transição do estado "Atendendo andar" para o estado "Esperando" indica que o elevador passa para o estado "Esperando" a partir do estado "Atendendo andar" com a condição de que não exista nenhuma chamada adicional para atendimento pelo elevador.2

Diagramas de atividades

O *diagrama de atividades* é uma variação do diagrama de mapa de estados. O diagrama de atividades se concentra nas atividades que um objeto executa; em outras palavras, o diagrama de atividades modela o que um objeto faz durante sua existência.

O diagrama de mapa de estados na figura anterior (Fig. 3.3 1) não traz nenhuma informação sobre para qual estado o elevador passa se duas pessoas diferentes no edifício apertam um botão ao mesmo tempo em andares diferentes. Ele também não contém informação sobre como o elevador decide se precisa se mover. O diagrama de atividades na Fig. 3.32 acrescenta novas informações às apresentadas no diagrama de mapa de estados, modelando as atividades que o elevador executa em resposta a uma chamada.

Atividades são representadas por elipses. O nome da atividade é colocado dentro da elipse. Uma linha cheia com uma seta conecta duas atividades, indicando a ordem na qual as atividades são executadas. Assim como no diagrama de mapa de estados, o círculo cheio indica o ponto de partida da seqüência de atividades. A seqüência de atividades modelada neste diagrama é executada sempre que é apertado um botão (i.e., se qualquer um dos botões de andar está agora no estado "Pressionado"). Quando esta condição é verdadeira, o elevador precisa tomar uma decisão (representada pelo losango).3 O elevador opta por uma entre várias atividades neste ponto, com base em certas condições. Cada linha (ou caminho) saindo do losango representa um destes diferentes conjuntos de atividades. Uma condição de guarda colocada junto a cada caminho indica sob que circunstâncias aquele caminho é executado.

Em nosso diagrama, o elevador executa um de três diferentes conjuntos de atividades quando é pressionado um botão. Se o elevador está em movimento (i.e., no estado "Movendo-se"), o elevador não pode executar imediatamente qualquer outra atividade, de modo que a seqüência de atividades no caminho atual simplesmente termina.

Em um sistema de elevador do mundo real, um sensor no elevador poderia fazer com que ele parasse em um andar. Em nosso simulador de elevador, sabemos que o elevador leva cinco segundos para se mover de um andar para o outro. Assim, em nossa simulação, o elevador pode simplesmente agendar sua própria chegada em um andar e o elevador irá parar naquela hora agendada.

Em um sistema de elevador do mundo real, o elevador provavelmente passa de um destes estados para o outro depois de decorrido um certo intervalo de tempo. Queremos programar um simulador, mas não queremos nos preocupar com os detalhes de como o elevador vai "saber" quando não existe nenhuma chamada adicional. Portanto, simplesmente dizemos que o elevador muda de estado no caso de não haver mais nenhuma chamada.

[botão do andar atual pressionado]

Fig. 3.32 Diagrama de atividades modelando a lógica do elevador para responder a pressionamentos de botões.

Um círculo cheio circundado por outro círculo (às vezes chamado de "olho de boi") indica o ponto final de um diagrama de atividades.
Se o botão do andar é pressionado no andar em que está o elevador, o elevador desliga o botão, toca a campainha
e abre a porta. Se o botão no andar em que o elevador está não é pressionado, o elevador precisa primeiro fechar a
porta, mover para o outro andar e então parar no outro andar, antes que ele possa atender ao outro andar. Observe que
a UML modela a junção de caminhos de decisão com outro losango pequeno. Depois que o elevador abre a porta, a
seqüência de atividades termina.

Conclusão

Para recapitular, expandimos nossos conhecimentos sobre as classes de nosso sistema (como vamos continuar a fazer nos próximos vários capítulos) e representamos este conhecimento em nosso diagrama de classes. Também usamos diagramas de mapa de estados e diagramas de atividades para obter mais informações sobre como funciona nosso sistema. Embora ainda não tenhamos discutido os detalhes da programação orientada a objetos em C++, já temos uma quantidade significativa de informações sobre nosso sistema. Nas seções "Pensando em objetos" nos finais dos Capítulos 4 e 5, determinamos as operações associadas com nossas classes e como nossas classes interagem (i.e., colaboram) umas com as outras.

Nota
1. Neste capítulo, você apreendeu como implementar a "aleatoriedade". O comando

```
arrivalTime = currentTime + ( 5 + rand() % 16 );
```

pode ser usado para programar aleatoriamente a próxima chegada de uma pessoa em um andar.

Resumo

• O melhor caminho para desenvolver e manter um programa grande é dividi-lo em vários módulos de programa menores, sendo cada um deles mais administrável que o programa original. Módulos são escritos em C++ como classes e funções.
• Um função é invocada por uma chamada de função. A chamada da função menciona a função por nome e fornece informação (como parâmetros) de que a função chamada necessita para executar sua tarefa.
• A finalidade da ocultação de informações é as funções terem acesso apenas às informações de que necessitam para completar suas tarefas. Este é um meio de implementar o princípio do mínimo privilégio, um dos princípios mais importantes da boa engenharia de software.
• O tipo de dados double é um tipo de ponto flutuante como float. Uma variável do tipo double pode armazenar um valor de magnitude e precisão muito maiores do que float pode armazenar.
• Cada parâmetro de uma função pode ser uma constante, uma variável ou uma expressão.
• Uma variável local é conhecida só em uma definição da função. Funções não podem saber os detalhes de implementação de qualquer outra função (inclusive variáveis locais).
• O formato geral da definição de uma função é
tipo do valor de retorno nome da função (lista de parâmetros
declarações e instruções
O tipo do valor de retorno indica o tipo do valor enviado de volta para a função que fez a chamada. Se uma função não retorna um valor, o *tipo do valor de retorno* é declarado como void. *O nome da função é* qualquer identificador válido. A *lista de parâmetros é* uma lista separada por vírgulas que contém as declarações das variáveis que serão passadas à função. Se uma função não recebe valor algum, a *lista de parâmetros é* declarada void. *O corpo da função* é o conjunto de declarações e comandos que formam a função.
• Os argumentos passados a uma função devem ser equivalentes em número, tipo e ordem aos parâmetros na definição da função.
• Quando um programa encontra uma chamada de função, o controle é transferido do ponto de chamada para a referida função, a função chamada é executada e o controle retorna a quem chamou.
• Uma função chamada pode devolver o controle a quem chamou de três maneiras. Se a função não retorna um valor, o controle é devolvido quando a chave à direita final da função é alcançada ou pela execução do comando

return;
Se a função retornar um valor, o comando
return *expressão;*
retorna o valor de *expressão.*
• Um protótipo de função declara o tipo de retorno da função e declara o número, os tipos e a ordem dos parâmetros que a função espera receber.

Este símbolo não deve ser confundido como losango grande usado em fluxogramas como aqueles apresentados na Seção 2.21.

• Os protótipos de funções permitem que o compilador verifique se as funções são chamadas corretamente. • Á
• O compilador ignora nomes variáveis mencionados no protótipo da função. . H
• Cada biblioteca padrão tem um arquivo de cabeçalho correspondente, contendo os protótipos de função para todas as funções daquela biblioteca, bem como definições de constantes simbólicas necessárias para aquelas funções. Á
• Os programadores podem e devem criar e incluir seus próprios arquivos de cabeçalho.
• Quando um parâmetro é passado em uma chamada por valor, uma cópia do valor da variável é feita e a cópia é passada para Á a função chamada. Alterações da cópia na função chamada não afetam o valor da variável original. T
• A função rand gera um inteiro entre O e RAND-MAX, que é definido como sendo pelo menos 32767. e
• Os protótipos de função para rand e srand estão contidos em <cstdlib>. C
• Os valores produzidos por rand podem ser ajustados em escala e deslocados para produzir valores em um intervalo específico.
• Para randomizar a saída de rand, use a função srand da biblioteca padrão.
• O comando srand normalmente é inserido em um programa só depois de o programa estar completamente depura- •
do. Enquanto se faz a depuração, é melhor omitir srand. Isto garante a "repetibilidade", que é essencial para provar d que as correções em um programa de geração de números aleatórios funcionam corretamente.
• Para randomizar sem necessidade de fornecer uma semente toda vez, podemos usar srand (time (O)). A função time normalmente retorna o "tempo de calendário" em segundos. O protótipo da função time está localizado no arquivo de cabeçalho <etime>. • (
• A equação geral para ajustar a escala e deslocar um número aleatório é
n = a-I- rand() % b;
onde a é o valor do deslocamento (que é igual ao primeiro número no intervalo desejado de inteiros sucessivos), e b é o fator 6 de ajuste de escala (que é igual ao comprimento do intervalo de inteiros sucessivos desejado).
• Uma enumeração, introduzida pela palavra-chave enum e seguida por um nome de tipo, é um conjunto de constantes do tipo .l
inteiro representadas por identificadores. • S
• Os valores destas constantes de enumeração começam em O, a menos que especificado de maneira diversa, e são incrementados u pori. u
• Os identificadores em uma erium devem ser únicos, mas constantes de enumeração separadas podem ter o mesmo valor inteiro.
• A qualquer constante de enumeração pode ser explicitamente atribuído um valor inteiro na enumeração.
• Cada identificador de variável tem os atributos classe de armazenamento, escopo e ligação.
• C++ fornece cinco especificadores de classe de armazenamento: auto. register. extern, mutable e static. •
• A classe de armazenamento de um identificador determina quando aquele identificador existe na memória.
• O escopo de um identificador é onde o identificador pode ser referenciado em um programa.
• A ligação de um identificador determina, em um programa com vários arquivos-fonte, que um identificador é conhecido apenas no arquivo-fonte atual ou em qualquer arquivo-fonte com declarações adequadas. e
• As variáveis da classe de armazenamento automática são criadas quando se ingressa no bloco onde elas são declaradas, . l existem enquanto o bloco estiver ativo e são destruídas quando o bloco chega ao fim. Normalmente, as variáveis locais de uma função são da classe de armazenamento automática por default. •
• O especificador de classe de armazenamento register pode ser colocado antes de uma declaração de variável automática c para sugerir que o compilador mantenha a variável em um dos registradores de hardware de alta velocidade do computador. • O compilador pode ignorar declarações register. A palavra-chave register pode ser usada apenas com variáveis da 1 classe de armazenamento automática.
• As palavras-chave extern e static são usadas para declarar identificadores para variáveis e funções da classe de armazenamento estática. e

• As variáveis com classe de armazenamento estática são alocadas e inicializadas quando a execução do programa é iniciada.
• Há dois tipos de identificadores da classe de armazenamento estática: identificadores externos (como variáveis globais e nomes de funções) e variáveis locais declaradas com o especificador de classe de armazenamento static.
• As variáveis globais são criadas colocando-se declarações de variáveis fora de qualquer definição de função e conservam seus valores ao longo de toda a execução do programa.
• As variáveis locais declaradas como static conservam seus valores quando termina a função na qual são declaradas.
• Todas as variáveis numéricas da classe de armazenamento estática são inicializadas com o valor zero se não forem inicializadas explicitamente pelo programador.
• Os escopos de um identificador são: escopo de função, escopo de arquivo, escopo de bloco e escopo de protótipo de função.
• Os rótulos são os únicos identificadores com escopo de função. Os rótulos podem ser usados em qualquer lugar da função onde aparecem, mas não se pode fazer referência a eles fora do corpo da função.
• Um identificador declarado fora de qualquer função tem escopo de arquivo. Tal identificador é "conhecido" por todas as funções,

desde o ponto em que o identificador é declarado até o final do arquivo.
• Os identificadores declarados dentro de um bloco possuem escopo de bloco. O escopo de bloco termina na chave à direita (}) de encerramento do bloco.
• As variáveis locais declaradas no início de uma função possuem escopo de bloco, assim como os parâmetros da função, que são considerados variáveis locais por ela.
• Qualquer bloco pode conter declarações de variáveis. Quando houver blocos aninhados e um identificador externo ao bloco tiver o mesmo nome que um identificador interno a ele, o identificador do bloco externo fica "escondido" até que termine a execução do bloco interno.
• Os únicos identificadores com escopo de protótipo de função são os utilizados na lista de parâmetros de um protótipo de função. Os identificadores empregados em um protótipo de função podem ser reutilizados em qualquer lugar do programa, sem ambigüidade.
• Uma função recursiva é uma função que chama a si mesma, direta ou indiretamente.
• Se uma função recursiva é chamada com um caso básico, ela simplesmente retorna um resultado. Se a função é chamada com um problema mais complexo, ela divide o problema em duas partes conceituais: uma parte que a função sabe como tratar e uma versão ligeiramente menor do problema original. Como esse novo problema se parece com o problema original, a função ativa uma chamada recursiva para trabalhar com o problema menor.
• Para uma recursão terminar, cada vez que uma função recursiva chamar a si mesma com uma versão ligeiramente mais simples do problema original, a seqüência de problemas cada vez menores deve convergir para o caso básico. Quando a função reconhecer o caso básico, o resultado é devolvido para a chamada anterior da função e segue-se uma seqüência de retornos, até que a chamada original da função devolve, em algum momento, o resultado final.
• O padrão C++ não especifica a ordem na qual os operandos da maioria dos operadores devem ser calculados. C++ especifica a ordem de cálculo dos operandos dos operadores **&&. 1 1** .o operador vírgula **(,)** e ?:. Os três primeiros são opera- dores binários, cujos dois operandos são calculados da esquerda para a direita. O último operador é o único operador ternário de C++. Seu operando da extremidade esquerda é avaliado em primeiro lugar; se o operando mais à esquerda tiver valor diferente de zero, o operando do meio é avaliado em seguida e o último operando é ignorado; se o operando mais à esquerda tiver o valor zero, o terceiro operando é avaliado a seguir e o operando do meio é ignorado.
• Tanto a iteração como a recursão se baseiam em uma estrutura de controle: a iteração usa uma estrutura de repetição; a recursão, uma estrutura de seleção.
• Tanto a iteração como a recursão envolvem repetições: a iteração usa explicitamente uma estrutura de repetição; a recursão consegue a repetição através de chamadas repetidas da função.
• Tanto a iteração como a recursão envolvem testes de encerramento: a iteração se encerra quando a condição de continuação do laço se torna falsa; a recursão termina quando um caso básico é reconhecido.
• A iteração e a recursão podem ocorrer infinitamente: ocorre um laço infinito com a iteração se o teste de continuação do laço nunca se tornar falso; acontece recursão infinita se a etapa de recursão não reduzir o problema de forma que convirja para o caso básico.

• A recursão ativa repetidamente o mecanismo, e conseqüentemente o *overhead,* de chamadas de funções. Isso pode custar caro, tanto em termos de tempo do processador como em espaço de memória.
• Os programas em C++ não compilam a menos que um protótipo de função seja fornecido para cada função ou que cada função seja definida antes de ser usada.
• Uma função que não retorna um valor é declarada com um tipo de rctorno void. Uma tentativa de retornar um valor da função ou de usar o resultado da invocação da função na função que chamou é um erro de sintaxe. Uma lista de parâmetros vazia é especificada com parênteses vazios ou void em parênteses.
• Funções *mime* eliminam o *overhead* de uma chamada de função. O programador usa a palavra-chave mime para aconselhar o compilador a gerar o código da função em linha (quando possível), para minimizar chamadas de função. O compilador pode escolher optar por ignorar a especificação mune.
• C++ oferece uma forma direta de chamada por referência usando parâmetros de referência. Para indicar que um parâmetro de função é passado por referência, coloque um & após o tipo do parâmetro, no protótipo da função. Na chamada da função, mencione a variável por seu nome e ela será passada por referência. Na função chamada, mencionar a variável por seu nome local na realidade faz referência à variável original na função que chamou. Deste modo, a variável original pode ser diretamente modificada pela função chamada.
• Parâmetros por referência também podem ser criados para uso local como *aliases* para outras variáveis dentro de uma função. As variáveis de referência devem ser inicializadas em suas declarações e não lhe podem ser atribuídas a *aliases* de outras variáveis. Uma vez que uma variável de referência tenha sido declarada como um nome alternativo para outra variável, todas as operações supostamente executadas sobre o nome alternativo são realmente executadas sobre a variável por ele referida.
• C++ permite ao programador especificar parâmetros default e seus valores default. Se um parâmetro default é omitido em uma chamada de uma função, o valor default daquele parâmetro é usado. Os parâmetros default devem ser os parâmetros mais à direita (finais) na lista de parâmetros de uma função. Os parâmetros default devem ser especificados com a primeira ocorrência do nome da função. Os valores default podem ser constantes, variáveis globais ou chamadas de função.
• O operador unário de resolução de escopo (: :) possibilita a um programa acessar uma variável global quando uma variável localdo mesmo nome está no escopo.
• É possível definir várias funções com o mesmo nome, mas com os tipos dos parâmetros diferentes. Isto é chamado de sobrecarregar uma função. Quando é chamada uma função sobrecarregada, o compilador seleciona a função apropriada examinando o número e os tipos de parâmetros na chamada.
• Funções sobrecarregadas podem ter valores de retorno diferentes ou não, mas devem ter listas de parâmetros diferentes. Se duas funções diferem somente pelo tipo de retorno, isto resultará em um erro de compilação.
• Gabaritos de função possibilitam a criação de funções que executam as mesmas operações sobre tipos diferentes de dados, mas o

gabarito da função é definido somente uma vez.

Terminologia

ajustar escala classe de armazenamento
argumento em uma chamada de função classe de armazenamento automática
argumentos default de uma função classe de armazenamento estática
armazenamento automático coerção de argumentos
arquivo de cabeçalho colaboração
arquivos de cabeçalho da biblioteca padrão compilador otimizador
assinatura componente
assinatura de função const
biblioteca padrão C++ constante com nome
bloco constante de enumeração
caso básico da recursão cópia de um valor
chamada de função declaração de função
chamada por referência decoração de nome
chamada por valor definição de função
chamada recursiva deslocamento
chamar uma função dividir para conquistar

1

: • A recursão ativa repetidamente o mecanismo, e conseqüentemente o *overhead,* de chamadas de funções. Isso pode custar caro, tanto em termos de tempo do processador como em espaço de memória.
Os programas em C++ não compilam a menos que um protótipo de função seja fornecido para cada função ou que cada função seja definida antes de ser usada.
• Uma função que não retorna um valor é declarada **com um tipo** de retorno void. Uma tentativa de retornar um valor da função ou de usar o resultado da invocação da função na função que chamou é um erro de sintaxe. Uma lista de parâmetros vazia é especificada com parênteses vazios ou void em parênteses.
Funções *mime* eliminam o *overhead* de uma chamada de função. O programador usa a palavra-chave mime para aconselhar o compilador a gerar o código da função em linha (quando possível), para minimizar chamadas de função. O compilador pode escolher optar por ignorar a especificação iriline.
• C++ oferece uma forma direta de chamada por referência usando parâmetros de referência. Para indicar que um parâmetro de função é passado por referência, coloque um & após o tipo do parâmetro, no protótipo da função. Na chamada da função, mencione a variável por seu nome e ela será passada por referência. Na função chamada, mencionar a variável por seu nome local na realidade faz referência à variável original na função que chamou. Deste modo, a variável original pode ser diretamente modificada pela função chamada.
• Parâmetros por referência também podem ser criados para uso local como *aliases* para outras variáveis dentro de uma função. As variáveis de referência devem ser inicializadas em suas declarações e não lhe podem ser atribuídas a *aliases* de outras variáveis. Uma vez que uma variável de referência tenha sido declarada como um nome alternativo para outra variável, todas as operações supostamente executadas sobre o nome alternativo são realmente executadas sobre a variável por ele referida.
• C++ permite ao programador especificar parâmetros default e seus valores default. Se um parâmetro default é omitido em uma chamada de uma função, o valor default daquele parâmetro é usado. Os parâmetros default devem ser os parâmetros mais à direita (finais) na lista de parâmetros de uma função. Os parâmetros default devem ser especificados com a primeira ocorrência do nome da função. Os valores default podem ser constantes, variáveis globais ou chamadas de função.
• O operador unário de resolução de escopo (: :) possibilita a um programa acessar uma variável global quando uma variável local do mesmo nome está no escopo.
• E possível definir várias funções com o mesmo nome, mas com os tipos dos parâmetros diferentes. Isto é chamado de sobrecarregar uma função. Quando é chamada uma função sobrecarregada, o compilador seleciona a função apropriada examinando o número e os tipos de parâmetros na chamada.
• Funções sobrecarregadas podem ter valores de retorno diferentes ou não, mas devem ter listas de parâmetros diferentes. Se duas funções diferem somente pelo tipo de retorno, isto resultará em um erro de compilação.
• Gabaritos de função possibilitam a criação de funções que executam as mesmas operações sobre tipos diferentes de dados, mas o gabarito da função é definido somente uma vez.

Terminologia

ajustar escala classe de armazenamento
argumento em uma chamada de função classe de armazenamento automática
argumentos default de uma função classe de armazenamento estática
armazenamento automático coerção de argumentos
arquivo de cabeçalho colaboração

arquivos de cabeçalho da biblioteca padrão compilador otimizador
assinatura componente
assinatura de função const
biblioteca padrão C++ constante com nome
bloco constante de enumeração
caso básico da recursão cópia de um valor
chamada de função declaração de função
chamada por referência decoração de nome
chamada por valor definição de função
chamada recursiva deslocamento
chamar uma função dividir para conquistar

duração do armazenamento estático efeito colateral
engenharia de software
enum
enumeração

escopo
escopo de arquivo
escopo de bloco
escopo de função
especificação de ligação
especificador de classe de armazenamento
especificador de classe de armazenamento auto especificador de classe de armazenamento mutable especificador de classe de armazenamento register especificador de classe de armazenamento static especificador de classe de armazenamento extern expressão de tipo misto
fator sorte
função
função chamada

função chamadora
função definida pelo programador função fatorial
função mime
função recursiva
funções da biblioteca matemática função gabarito
geração de números aleatórios hierarquia de promoção
invocar uma função

iteração
ligação
ligação segura quanto ao tipo

ocultação de informações
operador unário de resolução de escopo (: :) parâmetro de referência
parâmetro em uma definição de função princípio do mínimo privilégio

programa modular protótipo de função rarid
RANDMAX
randomizar

recursão
recursão infinita referência sem correspondente return
reutilização de software simulação
sobrecarga de função sobrecarregar

srand

sufixo ‘e comercial” (&)

`template`

time

tipo de referência tipo de valor de retorno

typename
unsigned
variável automática variável constante variável global variável local variável somente para leitura variável static

`void`

Terminologia de “Pensando em objetos”

ação
ação de “saída” atividade
atributo

condição de guarda
diagrama de atividades
diagrama de mapa de estados e de atividades em UML elipse no diagrama de atividades da UML

estado

estado inicial
evento
evento “quando”
linha cheia com seta na ponta nos diagramas de mapa de estados e de atividades na UML

Erros comuns de programação

losango no diagrama de atividades da UML nome de atributo em UML palavras descritivas na definição do problema

retângulo com cantos arredondados no diagrama de mapa de estados da UML
rótulo de ação
seta em diagrama UML
símbolo “olho de boi” no diagrama de atividades da UML
símbolo de ponto de partida nos diagramas de mapa de esta-

dos

tipo de atributo em UML transição
valor inicial de atributo em UML valor inicial de um atributo de classe

3.1 Esquecer de incluir o arquivo de cabeçalho <crnath> ao usar funções da biblioteca matemática é um erro de sintaxe. Para cada função da biblioteca padrão usada em um programa deve ser incluído um arquivo de cabeçalho padrão.
3.2 Omitir o tipo do valor de retorno em uma definição de função é um erro de sintaxe.
3.3 Esquecer de retornar um valor de uma função que deve fazê-lo é erro de sintaxe.

3.4 Retornar um valor de uma função cujo tipo de retorno foi declarado **void** causa um erro de sintaxe.
3.5 Declarar parâmetros da função do mesmo tipo como float x, y em vez de float x, float y. A declaração de parâmetros float x, y geraria na realidade um erro de compilação, porque os tipos são obrigatórios para cada parâmetro na lista de parâmetros.
3.6 Colocar um ponto-e-vírgula após o parêntese direito, ao encerrar a lista de parâmetros de uma definição de função, é um erro de sintaxe.
3.7 Definir um parâmetro de função novamente como variável local dentro da função é um erro de sintaxe.
3.8 Os () em uma chamada de função são, na realidade, um operador de C++. Eles fazem com que a função seja chamada. Esquecer os **()** em uma chamada de função que não aceita argumentos não é um erro de sintaxe. A função não é invocada quando você provavelmente pretendia que ela fosse.
3.9 Definir uma função dentro de outra função é um erro de sintaxe.
3.10 E um erro de sintaxe se o protótipo da função, o arquivo de cabeçalho e as chamadas da função não concordarem quanto ao número, tipo e ordem dos argumentos e parâmetros e quanto ao tipo do valor de retomo.
3.11 Esquecer o ponto-e-vírgula no final de um protótipo de função causa um erro de sintaxe.
3.12 Uma chamada de função que não corresponde ao protótipo da função é um erro de sintaxe.
3.13 E um erro de sintaxe se o protótipo da função e a definição da mesma estiverem em desacordo.
3.14 Converter de um tipo superior de dados em uma hierarquia de promoção para um tipo inferior pode modificar o valor dos dados.
3.15 Esquecer um protótipo de função causa um erro de sintaxe se a definição daquela função aparecer depois de sua chamada no programa.
3.16 Usar srand no lugar de rand para gerar números aleatórios é um erro de sintaxe, porque a função srand não retoma um valor.
3.17 Atribuir o inteiro equivalente a uma constante de enumeração a uma variável do tipo da enumeração é um erro de sintaxe.
3.18 Após uma constante de enumeração ter sido definida, tentar atribuir outro valor para a constante de enumeração é um erro de sintaxe.
3.19 Usar diversos especificadores de classe de armazenamento para o mesmo identificador é um erro de sintaxe. Somente um especificador de classe de armazenamento pode ser aplicado a um identificador. Por exemplo, se você incluir regis ter, não inclua também auto.
3.20 Usar acidentalmente, em um bloco interno, o mesmo nome que é usado para um identificador em um bloco mais externo, quando, na realidade, o programador quer que o identificador do bloco externo esteja ativo durante o processamento do bloco interno, geralmente causa um erro de lógica.
3.21 Esquecer-se de retornar um valor de uma função recursiva, quando isto é necessário, fará com que a maioria dos compiladores produza uma mensagem de advertência.
3.22 Omitir o caso básico, ou escrever incorretamente a etapa de recursão, de forma que ela não convirja para o caso básico, ocasionará uma recursão infinita, o que pode eventualmente esgotar a memória. Isso é análogo ao problema de um laço infinito em uma solução iterativa (não-recursiva). A recursão infinita também pode ser causada pelo fornecimento de um dado incorreto de entrada.
3.23 Escrever programas que dependam da ordem de cálculo dos operandos de operadores diferentes de **&&.** 1 ?: e o operador vírgula **(,)** pode levar a erros, porque os compiladores poderiam não calcular necessariamente os operandos na ordem em que o programador espera.
3.24 Fazer uma função não-recursiva chamar acidentalmente a si mesma, tanto direta como indiretamente (através de outra função), é um erro de lógica.
3.25 A menos que sejam fornecidos protótipos de função para todas as funções ou, então, que cada função seja definida antes de ser usada, os programas em C++ não são compilados.
3.26 Como parâmetros por referência são referidos somente por nome no corpo da função chamada, o programador pode, inadvertidamente, tratar parâmetros passados por referência como parâmetros passados por valor. Isto pode ocasionar efeitos colaterais inesperados, se as cópias originais das variáveis são alteradas pela função que está chamando.
3.27 Declarar multiplas referências em um só comando, supondo que o **&** se distribui por uma lista de nomes de variáveis separados por vírgulas. Para declarar as variáveis x, y e z, como referências para inteiros, use a notação irit **& x=a,** &y = b, &z b;,emvezdanotaçãoincorretaint& x = **a,** y = **z** = e; oudeoutranotaçãoincorreta freqüentemente encontrada: int &x, y, z;.
3.28 Não inicializar uma variável de referência quando ela é declarada é um erro de sintaxe.
3.29 Tentar atribuir a uma referência previamente declarada um *alias* para outra variável é um erro de lógica. O valor da outra variável é simplesmente atribuído para a localização para a qual a referência já *é* um *alias.*
3.30 Retornar um ponteiro ou referência para uma variável automática em uma função chamada é um erro de lógica. Alguns compiladores emitirão uma mensagem de advertência quando isto ocorrer em um programa.
3.31 Especificar e tentar usar um argumento default que não é o argumento final (mais à direita) da lista de argumentos (sem, simultaneamente, tornar default os outros argumentos mais à direita) é um erro de sintaxe.
3.32 Tentar acessar uma variável não-global em um bloco externo usando o operador unário de resolução de escopo é um erro de sintaxe se não existir uma variável global com o mesmo nome no bloco externo e um erro de lógica se existir uma.

3.33 Criar funções sobrecarregadas com listas de parâmetros idênticas e tipos diferentes de retomo é um erro de sintaxe.
3.34 Uma função com parâmetros default omitidos pode ser chamada de forma idêntica a outra função sobrecarregada; isto é um erro de sintaxe. Por exemplo, ter em um programa tanto uma função que explicitamente não aceita nenhum argumento e uma função com mesmo nome cujos parâmetros são todos default. provoca um erro de sintaxe quando é feita uma tentativa de usar aquele nome de função em uma chamada sem passar nenhum argumento.

3.35 Não colocar a palavra-chave class ou a palavra-chave typename antes de cada parâmetro de tipo de um gabarito de função é um erro de sintaxe.

Boas práticas de programação

3.1 Familiarize-se com a rica coleção de funções e classes da biblioteca padrão C++.
3.2 Coloque uma linha em branco entre as definições das funções para separá-las e melhorar a legibilidade do programa.
3.3 Embora não seja incorreto fazê-lo, não use os mesmos nomes para os argumentos passados para uma função e os parâmetros correspondentes na definição da função. Isso ajuda a evitar ambigüidades.
3.4 Escolher nomes significativos para funções e parâmetros toma os programas mais legíveis e ajuda a evitar o uso excessivo de comentários.
3.5 Muitos programadores usam nomes de parâmetros nos protótipos de função para fins de documentação. O compilador ignora esses nomes.
3.6 Use maiúscula na primeira letra de um identificador usado como o nome de um tipo definido pelo usuário.
3.7 Use somente letras maiúsculas nos nomes de constantes de enumeração. Isto faz com que estas constantes se destaquem no programa, lembrando ao programador que constantes de enumeração não são variáveis.
3.8 Usar constantes de enumeração em vez de constantes inteiras, toma um programa mais compreensível.
3.9 Evite nomes de variáveis que escondam nomes em escopos mais externos. Isso pode ser conseguido simplesmente evitando-se, em um programa, o uso de identificadores duplicados.
3.10 Sempre inclua protótipos de função, ainda que seja possível omiti-los quando as funções são definidas antes que sejam usadas. Incluir os protótipos evita amarrar o código à ordem na qual as funções são definidas (que pode facilmente mudar
à medida que um programa evolui).
3.11 O qualificador mime deveria ser usado somente com funções pequenas, freqüentemente usadas.
3.12 Usar argumentos default pode simplificar a codificação de chamadas de funções. Contudo, alguns programadores acham que especificar explicitamente todos os argumentos é mais claro.
3.13 Evite usar variáveis com o mesmo nome para propósitos diferentes em um programa. Embora isso seja permitido em várias circunstâncias, pode gerar confusão.
3.14 Sobrecarregar funções que executam tarefas proximamente relacionadas toma um programa mais legível e compreensível.

Dicas de desempenho

3.1 Não tente rescrever rotinas de biblioteca existentes para tomá-las mais eficientes. Você normalmente não será capaz de melhorar o desempenho destas rotinas.
3.2 A função srand necessita ser chamada somente uma vez em um programa para obter o efeito de randomização desejado. Chamá-la mais de uma vez é redundante e, portanto, reduz o desempenho do programa.
3.3 O armazenamento automático é um meio de economizar memória porque as variáveis automáticas são criadas quando o bloco na qual são declaradas é ativado e são destruídas quando o bloco é deixado.
3.4 O especificador de classe de armazenamento register pode ser colocado antes de uma declaração de variável automática para sugerir que o compilador mantenha a variável em um dos registradores de hardware de alta velocidade do computador em vez de na memória. Se as variáveis forem usadas freqüentemente como contadores ou para cálculo de totais, elas podem ser conservadas em registradores de hardware, e o *overhead* de carregar repetidamente as variáveis da memória para os registradores e armazenar os resultados novamente na memória pode ser eliminado.
3.5 Freqüentemente, as declarações regis ter são desnecessárias. Os atuais compiladores otimizadores são capazes de reconhecer as variáveis usadas repetidamente e podem decidir colocá-las em registradores sem que haja a necessidade de
o programador incluir uma declaração register.
3.6 Evite programas recursivos semelhantes ao da função fibonacci que resultem em uma “explosão” exponencial de chamadas.
3.7 Evite usar a recursão em situações que exigem um alto desempenho. Chamadas recursivas consumem tempo e exigem mais memória.
3.8 Um programa muito dividido em funções - em comparação com um programa monolítico, não dividido em funções (i.e., composto de um único módulo) - executa um número potencialmente maior de chamadas de funções e isto consume

tempo de execução e espaço do(s) processador(es) de um computador. Porém, programas monolíticos são difíceis de programar, testar, depurar, manter e de acrescentar novas funcionalidades.
3.9 Usar funções mime pode reduzir o tempo de execução, mas pode aumentar o tamanho do programa.
3.10 Uma desvantagem da chamada por valor é que, se um grande item de dados está sendo passado, copiar esses dados pode
consumir um tempo de processamento considerável.
3.11 Chamada por referência é bom sob o ponto de vista do desempenho porque elimina o *overhead* de copiar grandes

volumes de dados.
3.12 Para passar objetos grandes, use um parâmetro por referência constante para simular a aparência e segurança de uma
chamada por valor e evitar o *overhead* de passar uma cópia do objeto grande.

Dicas de portabilidade

E
3.1 Usar as funções da biblioteca padrão C++ ajuda a criar programas mais portáveis.
3.2 Programas que dependem da ordem de cálculo de operandos de operadores diferentes de **&&,** 1 1 ?: e o operador virgula
(,) podem funcionar de modo distinto em sistemas com compiladores diferentes.
3.3 O significado de uma lista de parâmetros de função vazia em C++ é completamente diferente do que em C. Em C, significa que toda a verificação de argumentos é desativada (i.e., a chamada da função pode passar quaisquer argumentos
que queira). Em C++, significa que a função não recebe argumentos. Assim, a compilação em C++ de programas escritos
em C que usam este recurso pode acusar erros de sintaxe.

Observações de engenharia de software

3.1 Evite reinventar a roda. Quando possível, use as funções da biblioteca padrão C++ em vez de escrever funções novas. Isso
reduz o tempo de desenvolvimento de programas.
3.2 Em programas que contêm muitas funções, main deveria ser implementada como um grupo de chamadas a funções que
executam o "grosso" do trabalho do programa.
3.3 Cada função deve se limitar a realizar uma tarefa simples e bem-definida e o nome da função deve expressar efetivamente
aquela tarefa. Isto promove a reutilização do software.
3.4 Se você não puder escolher um nome conciso que expresse o que a função faz, é possível que sua função esteja tentando
realizar muitas tarefas diferentes. Normalmente, é melhor dividir tal função em várias funções menores.
3.5 Uma função deveria caber inteira na janela de um editor. Independentemente de quão longa seja uma função, ela deveria executar bem uma tarefa. Funções pequenas favorecem a reutilização do software.
3.6 Os programas devem ser escritos como conjuntos de pequenas funções. Isso toma os programas mais fáceis de escrever,
depurar, manter e modificar.
3.7 Uma função que exige um grande número de parâmetros pode estar realizando tarefas demais. Pense na possibilidade de
dividir a função em funções menores, que realizem tarefas separadas. O cabeçalho da função deveria estar contido em uma linha, se possível.
3.8 Protótipos de funções são obrigatórios em C++. Use diretivas #inciude do pré-processador para obter protótipos de todas as funções da biblioteca padrão a partir dos arquivos de cabeçalho das bibliotecas apropriadas. Use também #iriciude para obter os arquivos de cabeçalho que contêm os protótipos de funções usados por você ou pelos membros de sua equipe.
3.9 Um protótipo de função não é obrigatório se a definição da função aparece antes do primeiro uso da função no programa.
Em tal caso, a definição da função também serve como o protótipo da função.
3.10 Um protótipo de função colocado fora de qualquer definição de função se aplica a todas as chamadas daquela função que
aparecem depois de seu protótipo no arquivo. Um protótipo de função colocado dentro de uma função se aplica somente
às chamadas realizadas naquela função.
3.11 O armazenamento automático é mais um exemplo do princípio do menor privilégio. Por que armazenar variáveis na
memória e deixá-las disponíveis, quando na realidade não são mais necessárias?
3.12 Declarar uma variável como global em vez de local permite que ocorram efeitos colaterais indesejáveis quando uma
função que não necessita de acesso à variável modifica, acidental ou intencionalmente, essa variável. Em geral, o uso de
variáveis globais deveria ser evitado, exceto em certas situações com exigências especiais de desempenho.
3.13 As variáveis usadas apenas em uma determinada função devem ser declaradas como variáveis locais naquela função, em
vez de serem declaradas como variáveis globais.

3.14 Qualquer problema que pode ser resolvido recursivamente também pode ser resolvido interativamente (não recursivamente).
Uma abordagem recursiva é escolhida em detrimento de uma abordagem iterativa, quando ela reflete o problema de modo mais natural, resultando em programa mais fácil de compreender e depurar. Outra razão para se optar por uma solução recursiva é quando uma solução iterativa não é fácil de ser encontrada.

3.15 Funcionalizar programas de maneira clara e hierárquica promove a boa engenharia de software. Mas isso tem um preço.
3.16 Qualquer mudança em uma função mime pode exigir que todos os clientes da função sejam recompilados. Isso pode ser relevante em algumas situações de desenvolvimento e manutenção de programas.
3.17 Muitos programadores não se preocupam em declarar parâmetros passados por valor como const. ainda que a função chamada não deva modificar o argumento passado. A palavra-chave const está somente protegendo uma cópia do argumento original, não o próprio argumento original.
3.I Chamada por referência pode diminuir a segurança porque a função chamada pode corromper os dados da função que chamou.
3.19 Tanto para fins de clareza como de desempenho, muitos programadores de C++ preferem que argumentos modificáveis sejam passados para funções por meio de ponteiros, argumentos pequenos não-modificáveis sejam passados por chamadas por valor e argumentos grandes não-modificáveis sejam passados para funções usando referências para constantes.

Dica de teste e depura ção

3.1 Inclua um caso default em uma estrutura switch para detectar erros mesmo que você esteja absolutamente seguro de que não há erros.

Exercícios de auto-revisão

3.1 Complete as frases abaixo:
a) Os componentes de programa em C++ são chamados de _____ e _____
b) Uma função é invocada com uma _____
c) Uma variável que é conhecida só dentro da função em que é definida é chamada de _____
d) O comando _____ em uma função chamada é usado para passar de volta o valor de uma expressão para a função que chamou.
e) A palavra-chave é usada em um cabeçalho de função para indicar que uma função não retorna um valor, ou para indicar que uma função não recebe nenhum parâmetro.
f) O _____ de um identificador é a parte do programa em que o identificador pode ser usado.
g) Os três meios para retornar o controle de uma função chamada para que chamou são _____ _____ e _____
h) Um permite ao compilador conferir a quantidade, os tipos e a ordem dos parâmetros passados para uma função.
i) A função é usada para produzir números aleatórios.
J) A função é usada para inicializar a semente dos números aleatórios usados para randomizar um programa.
k) Os especificadores de classes de armazenamento são mutabie. _____, _____, _____ e _____
1) Variáveis declaradas em um bloco ou na lista de parâmetros de uma função são assumidas como sendo da classe de armazenamento _____ 'a menos que especificado de forma diversa.
m) O especificador de classe de armazenamento _____ é uma recomendação para o compilador armazenar uma variável em um dos registradores do computador.
n) Uma variável declarada fora de qualquer bloco ou função é uma variável _____
o) Para uma variável local em uma função reter seu valor entre chamadas à função,ela deve ser declarada com o especificador de classe de armazenamento _____
p) Os quatro escopos possíveis de um identificador são ,, e _____
q) Unia função que chama a si mesma, direta ou indiretamente, é uma função
r) Uma função recursiva tem tipicamente dois componentes: um que fornece um meio para a recursão terminar testando a ocorrência de um caso _____ e um que expressa o problema como uma chamada recursiva para um problema ligeiramente mais simples que a chamada original.
s) Em C ++, é possível se ter várias funções com o mesmo nome, cada uma operando sobre tipos e/ou quantidades de parâmetros diferentes. Isto é chamado de _____ a função.
t) O possibilita o acesso a uma variável global com o mesmo nome que uma variável no escopo atual.
u) O qualificador é usado para declarar variáveis somente para leitura.
v) Uma função possibilita que uma única função seja definida para executar uma tarefa em muitos tipos de dados diferentes.
3.2 Para o programa seguinte, indique o escopo (escopo de função, escopo de arquivo, escopo de bloco ou escopo de protótipo função) de cada dos elementos seguintes.
a) A variável x em main.
b) A variável y em cube.
e) A função cube.

d) A função fiam.

e) O protótipo da função para cube.
1') O identificador y no protótipo de função para cube.

3.3 Escreva um programa que testa se os exemplos de chamadas à função da biblioteca de matemática mostrados na Fig. 3.2 produzem realmente os resultados indicados.

Escreva o cabeçalho da função para cada uma das seguintes funções:
a) Função hypotenuse. que recebe dois parâmetros em ponto flutuante com precisão dupla, sidel e **side2. e** retoma um resultado em ponto flutuante com precisão dupla.
b) Função smallest, que recebe três inteiros, x, y e z, e retoma um inteiro.
c) Função instructions, que não recebe quaisquer parâmetros e não retoma um valor. (Nota: tais funções são comumente usadas para exibir instruções para o usuário).
d) Função íntToDouble que recebe um argumento do tipo inteiro, number, e devolve um resultado em ponto flutuante com precisão simples.

Escreva o protótipo de função para cada uma das seguintes funções:
a) A função descrita no Exercício 3.4a.
b) A função descrita no Exercício 34b.
c) A função descrita no Exercício 3.4c.
d) A função descrita no Exercício 3.4d.

Escreva uma declaração para cada um dos seguintes casos:
a) O inteiro *counter,* que deve ser mantido em um registrador. lnicialjze count com 0.
b) Variável em ponto flutuante com precisão dupla lastVal, que deve manter seu valor entre chamadas à função em que está definida.
c) Inteiro externo nuinber, cujo escopo deve ser restringido ao restante do arquivo em que é definido.

3.7 Ache o erro em cada um dos segmentos seguintes de programa e explique como o erro pode ser corrigido (veja também Exercício *3.53):*

a) int g(void) (
cout « "Dentro da função g" « endi;

int h(vojd

{

1
2

3

// **ex0302.cpp #include <iostream>**

4
5
6

using std::cout; usinq std: :endl

7
8

int cube (int y

int main O

int x

9
10
11
12
13
14
15
16
17
18
19
20
21
22

```
for ( x 1; x <= 10; x++
cout « cube( x ) « endi;
```

return 0;

}

{

int cube (int y

}

return y * y * y

3.4

3.5

3.6

cout « "Dentro da função h" « endl;

)


```
b) int sum( int x, int y ){
int result;
result = x + y;
c) int sum( int n
if (n == 0)
return 0;
else
```

```
n + sum( n - 1 );
d) void f ( double a );
float a;
cout « a « endl;
e) void product ( void
int a, b, c, result;
cout « "Digite três inteiros:
cm » a » b » c;
result = a * b * c;
cout « "Resultado é « result;
return result;
```

3.8 Por que um protótipo de função conteria **uma declaração de tipo de parâmetro tal como double &?**
3.9 (Verdadeiro/Falso) Todas as chamadas em C++ são chamadas por valor.
3.10 Escreva um programa completo em C++ que usa uma função mime sphereVolume para solicitar ao usuário o raio de uma esfera e calcular e imprimir o volume dessa esfera usando a atribuição volume = (4 .0 / 3) * 3.14159 * pow (radius, 3).

Respostas aos exercícios de auto-revisão

3.1 a) Funções e classes. b) Chamada de função. c) Variável local. d) return. e) void. f) Escopo. g) return; ou return expression; ou encontrando a chave à direita O) de término de uma função. h) Protótipo de função. i) rand. j) srand. k) auto. register. extern e static. 1) auto. m) register. n) Externa, global, o) static. p) Escopo de função, escopo de arquivo, escopo de bloco e escopo de protótipo de função. q) Recursiva. r) Básico. s) Sobrecarregar. t) Operador unário de resolução de escopo (: :). o) const. v) Gabarito.
3.2 a) Escopo de bloco. b) Escopo de bloco. c) Escopo de arquivo. d) Escopo de arquivo, e) Escopo de arquivo. f) Escopo de protótipo de função.
3.3 Veja abaixo.

```
1 // ex03_03.cpp
2 // Testando as funções da biblioteca de matemática
3 #include <iostream>
4
5 using std: :cout;
6 using std: :endl;
7 using std: :ios;
8
```



```
9 #include <iomanip>
lo
11 using std: :setiosflags;
12 using std: :fixed;
13 using std: :setprecision;
14
15 #include <cmath>
16
17 int main()
18 {
19 cout« setiosflags( ios::fixed l ios::showpoint
20 « setprecision( 1
21 « "sqrt(" « 900.0 « )= « sqrt( 900.0
22 « "\nsqrt(' « 9.0 « ")= « sqrt( 9.0
23 « "\nexp(" « 1.0 « ")= « setprecision( 6
24 « exp( 1.0 )« "\nexp(" « setprecision( 1 )« 2.0
25 « ')= « setprecision( 6 )« exp( 2.0
26 « "\nlog(" « 2.718282 « ")= « setprecision( 1
27 « log( 2.718282 )« '\nlog(" « setprecision( 6
28 « 7.389056 « ")= « setprecision( 1
29 « log( 7.389056 )« endi;
30 cout« loglO(" « 1.0 « ")= « loglO( 1.0
31 « '\nloglO(" « 10.0 « )= « loglO( 10.0
32 « "\nloglO( « 100.0 « ")= « loglO( 100.0
33 « °\nfabs( « 13.5 « ")« fabs( 13.5
34 « '\nfabs(" « 0.0 « ")= « fabs( 0.0
35 « "\nfabs(" « -13.5 « ')= « fabs( -13.5 )« endi;
36 cout « "ceil(" « 9.2 « ')= « ceil( 9.2
37 « "\nceil(" « -9.8 « ")= « ceil( -9.8
38 « "\nfloor( « 9.2 « ")= « floor( 9.2
39 « \nfloor(" « -9.8 « ")= « floor( -9.8 )« endi;
40 cout« pow(" « 2.0 « ,« 7.0 « ")
41 « "pow( 2.0, 7.0 )« \npow(" « 9.0 « ,
42 « 0.5 « ")= « pow( 9.0, 0.5
43 « setprecision( 3 )« "\nfmod("
```

```
44 « 13.675 « ", « 2.333 « ")
45 « fmod( 13.675, 2.333 )« setprecision( 1
46 « "\nsin(" « 0.0 « ") = « sin( 0.0
47 « "\ncos(" « 0.0 « ") = « cos( 0.0
48 « \ntan(" « 0.0 « ) = « tan( 0.0 )« endi;
49 return 0;
50 }
sqrt(900.0) = 30.0
sqrt(9.0) = 3.0
exp(1.0) = 2.718282
exp(2.0) = 7.389056
log(2.718282) = 1.0
log(7.389056) = 2.0
loglO(1.0) = 0.0
loglO(10.0)= 1.0
loglO(l00.0) 2.0
fabs(13.5) = 13.5
fabs(0.0) = 0.0


fabs(-13.5) = 13.5
ceil(9.2) = 10.0
ceil(-9.8) = -9.0
floor(9.2) = 9.0
floor(-9.8) = -10.0
pow(2.0, 7.0) = 128.0
pow(9.0, 0.5) = 3.0
fmod(13.675, 2.333) = 2.010
sirz(0.0) = 0.0
cos(0.0) = 1.0
tan(0.0) = 0.0
```


3.4 a) double hypotenuse (double sidel, double side2
b) int smallest (int x, int y, int z)
c) void instructions (void) // em C++ (void) pode
d) float intToDouble(int number

3.5 a) double hypotenuse (double, double);
b) int smallest(int, int, int);
c) void instructions (void); // em C-I-+ **(void) pode**
d) float intToDouble (int);

3.6 a) register int count = 0; b) static double lastVal;

c) static **int nunber;**
Nota: isto apareceria fora de qualquer definição de função.

3.7 a) Erro: a função h é definida dentro da função g.
Correção: mova a definição de h para fora da definição de g.
b) Erro: a função deveria retomar um inteiro, mas não o faz.
Correção: delete a variável result e coloque o seguinte comando na função:

```
return x + y;
```

c) Erro: o resultado de n **+** sum **(n** - 1) não é retornado: sum retoma um resultado impróprio. Correção: rescreva o comando na cláusula else como segue

```
return n + sum( n - 1 );
```

3.10 Veja abaixo.

```
// exO3_10.cpp
// Função mime que calcula o volume de uma esfera #include <iostream>

using std::cout;

ser escrito O

ser escrito O
```

d) Erros: o ponto-e-vírgula depois do parêntese direito que fecha a lista de parâmetros e está redefinindo o parâmetro a na definição da função.
Correções: apague o ponto-e-vírgula depois do parêntese à direita da lista de parâmetros e apague a declaração float a:.
e) Erro: a função retoma um valor quando não deve fazê-lo.
Correção: elimine o comando return.
3.8 Porque o programador está declarando um parâmetro por referência do tipo "referência para" double para obter acesso através de chamada por referência à variável argumento original.
3.9 Falso. C++ permite chamada por referência direta através do uso de parâmetros de referência, além do uso de ponteiros.

```
1
2
3
4
5
6
7

using std: :cin;
using std: :endl;
```



```
8
9 const double P1 = 3.14159;
10
11 mime doubie sphereVoiume( const double r
12 { return 4.0 / 3.0 * P1 * r * r * r;
13
14 int main Q
15
16 double radius;
17
18 cout« "Forneça o raio de sua esfera:
19 cm » radius;
20 cout « 'Volume da esfera com raio « radius «
21 é « sphereVolume ( radius ) « endi;
22 return 0;
23
```

Exercícios

3.11 Mostre o valor de x após cada um dos comandos seguintes ser executado:

a) x = fabs(7.5)
b) x = fioor(7.5)
c) x = fabs(0.0)
d) x = ceil(0.0)
e) x = fabs(-6.4
f) x = ceii(-6.4
g) x = ceil(-fabs(-8 + fioor(-5.5
3.12 Uma garagem de estacionamento cobra $2,00 de taxa mínima para estacionar até três horas. A garagem cobra um adicional de $0,50 por hora ou fração de hora, caso sejam excedidas as três horas. A taxa máxima para qualquer período determinado de 24 horas é

$10,00. Suponha que nenhum carro fique estacionado por mais de 24 horas. Escreva um programa que calcule e imprima as taxas de estacionamento para três clientes que ontem estacionaram seus carros nessa garagem. Você deve digitar a quantidade de horas durante as quais cada cliente ficou estacionado. Seu programa deve imprimir os resultados organizados em um formato de tabela e deve calcular e imprimir o total das receitas do dia de ontem. O programa deve usar a função calculateCharges para determinar o valor a ser cobrado de cada cliente. A saída de seu programa deve ter o seguinte formato:

Carro **Horas Taxa Cobrada**
1 1.5 2.00
2 4.0 2.50
3 24.0 10.00
TOTAL 29.5 14.50

3.13 A função floor pode ser usada para arredondar um valor para o inteiro mais próximo. O comando

y = fioor(x + **.5);**

arredonda o valor numérico x para o inteiro mais próximo e atribui o resultado a y. Escreva um programa que leia vários números e use o comando precedente para arredondar cada um desses números para o inteiro mais próximo. Para cada número processado, imprima o número original e o número arredondado.

3.14 A função floor pode ser usada para arredondar um número para uma determinada casa decimal. O comando **y=fioor(x * 10 + .5) / 10;**

arredonda x para décimos (a primeira posição à direita da casa decimal). O comando

y floor(x *100 + .5) / 100;

arredonda x para centésimos (a segunda posição à direita da casa decimal). Escreva um programa que defina quatro funções para
arredondar um número x de várias maneiras:

a) roundTolnteger(number)
b) roundToTenths (number)
c) roundToHundredths (number
d) roundToThousandths(nuinber

Para cada valor lido, seu programa deve imprimir o valor original, o número arredondado para o inteiro mais próximo, o número arredondado para o décimo mais próximo, o número arredondado para o centésimo mais próximo e o número arredondado para o milésimo mais próximo.

3.15 Responda a cada uma das seguintes perguntas:
a) O que significa escolher números aleatoriamente"?
b) Por que a função rand é útil para simular jogos de azar?
c) Por que randomizar um programa usando srand? Sob que circunstâncias não é desejável randomizar?
d) Por que é necessário ajustar e dimensionar a escala dos valores produzidos por rand?
e) Por que a simulação computadorizada de situações do mundo real é uma técnica útil?

3.16 Escreva comandos que atribuam inteiros aleatórios à variável *n* nos seguintes intervalos:
a) *ln2*
b) *1nl0O*
c) *0n9*
d) *l000n* 1112
e) -Ln1
1) -3nll

3.17 Para cada um dos seguintes conjuntos de inteiros, escreva um comando simples que imprima um número aleatório do conjunto
a) 2,4,6, 8, 10.
b) 3,5,7,9, *11.*
c) 6, 10, 14, 18,22.

3.18 Escreva uma função integerPower (base, exponent) que retorne o valor de
exponent
base

Por exemplo, integerPower (3, 4) = 3 * 3 * 3 * 3. Suponha que exponent seja um inteiro positivo, diferente de zero, e base seja um inteiro. A função integerPower deve usar for ou while para controlar o cálculo. Não use nenhuma das funções da biblioteca matemática.

3.19 Defina uma função hypotenuse que calcule o comprimento da hipotenusa de um triângulo retângulo quando são fornecidos os catetos. Use essa função em um programa para determinar o comprimento da hipotenusa de cada um dos seguintes triângulos. A função deve utilizar dois argumentos do tipo double e retornar a hipotenusa com o tipo double.

Triângulo Lado **1 Lado 2**

1 3,0 4,0

2 *5,0* 12,0
3 8,0 *15,0*
3.20 Escreva uma função multiple que determine, para um par de números inteiros, se o segundo número é múltiplo do primeiro. A função deve ter dois argumentos inteiros e retornar true, se o segundo número for múltiplo do primeiro, ou false caso contrário. Use essa função em um programa que leia uma série de números inteiros.

3.21 Escreva um programa que receba uma série de inteiros e passe cada um deles por vez para a função par, que usa o operador módulo para determinar se um inteiro é par. A função deve utilizar um argumento inteiro e retornar true se o inteiro for par e false caso contrário.
3.22 Escreva uma função que mostre, na margem esquerda da tela, um quadrado de asteriscos cujo lado é especificado por um parâmetro inteiro lado. Por exemplo, se lado for igual a 4, a função exibe
3.31
núm
3.32 Escr
3.23 Modifique a função criada no Exercício 3.22 para formar um quadrado com qualquer caractere que esteja contido no parâmetro 3 33
do tipo char caractereDePreenchimento. Dessa forma, se lado for igual a **5** e caractereDePreenchimento for 90-lI
a função deve imprimir e O s
3.34
mir
Impr
e I
dam
3.24 Use técnicas similares às descritas nos Exercícios 3.22 e 3.23 para produzir um programa que gere gráficos com uma 3.35 grande variedade de formas. alum
progi
3.25 Escreva segmentos de programas que realizem cada uma das tarefas seguintes:
a) Calcule a parte inteira do quociente quando um inteiro a é dividido por um inteiro b.
b) Calcule o resto inteiro quando um inteiro a é dividido por um inteiro b.
e) Use os segmentos de programa desenvolvidos ema) e b) para escrever uma função que receba um inteiro entrei e 32767 O ah e o imprima como uma série de dígitos separados por dois espaços. Por exemplo, o inteiro 4562 deve ser impresso como
nov

1 4562

3.36
3.26 Escreva uma função que obtenha a hora como três argumentos inteiros (para horas, minutos e segundos) e retorne o *assis* número de segundos desde a última vez que o relógio "bateu 12 horas". Use essa função para calcular o intervalo de tempo em do-s segundos, entre duas horas, ambas dentro de um ciclo de doze horas do relógio. sejar
3.27 Implemente as seguintes funções inteiras: Com
a) A função celsius retorna a temperatura no equivalente em graus Celsius a uma temperatura em graus Fahrenheit.
b) A função fahrenheit retorna a temperatura em graus Fahrenheit equivalente a uma temperatura em graus Celsius.
e) Use essas funções para escrever um programa que imprima gráficos mostrando as temperaturas Fahrenheit equivalen te a temperaturas Celsius de O a 100 graus e temperaturas Celsius equivalentes a temperaturas Fahrenheit de 32 a 212 graus. Imprima as saídas organizadas em um formato de tabela que minimize o número de linhas de saída ao mesmo tempo em que permanece legivel. Con
3.28 Escreva uma função que retorna o menor número entre três números do tipo ponto flutuante com precisão dupla.
3.29 Diz-se que um número inteiro é um *número perfeito* se a soma de seus fatores, incluindo 1 (mas não o próprio número), é igual ao próprio número. Por exemplo, 6 é um número perfeito porque 6 1 + 2 + 3. Escreva uma função perfect que determine se o parâmetro riumber é um número perfeito. Use essa função em um programa que determine e imprima todos os números perfeitos entre 1 e 1000. Imprima os fatores de cada número encontrado para confirmar que ele é realmente perfeito. Use Desafie o poder de seu computador testando números muito maiores do que 1000. resp
3.30 Diz-se que um inteiro é *primo* se for divisível apenas por 1 e por si mesmo. Por exemplo, 2, 3,5 e 7 são primos, mas 4,6, 3.37 8e9nãoosão. det
pro
digi

****
****

a) Escreva uma função que determine se um número é primo.

seu

b) Use essa função em um programa que determine e imprima todos os números primos entre 1 e 10.000. Quantos desses 10.000 números deverão ser realmente testados, antes de se ter a certeza de que todos os números primos foram encontrados?

c) lnicialmente você poderia pensar que *nl2* é o limite superior que deve ser testado para verificar se um número é primo, mas você só precisa ir até a raiz quadrada de *n.* Por quê? Rescreva o programa e execute-o das duas maneiras. Faça uma estimativa da melhora no desempenho.

3.31 Escreva uma função que receba um valor inteiro e retorne o número com seus dígitos invertidos. Por exemplo, dado o número 7631, a função deve retornar 1367.

3.32 O *máximo divisor comum* (MDC) de dois inteiros é o maior inteiro que divide exatamente cada um dos dois números. Escreva uma função mdc que retorna o máximo divisor comum de dois inteiros.

3.33 Escreva uma função pontosDeQualificacao que receba a média de um aluno e retome 4, se ela estiver no intervalo 90-100,3 se a média estiver no intervalo 80-89,2 se a média estiver no intervalo 70-79, 1 se a média estiver no intervalo 60-69, e O se a média for menor que 60.

3.34 Escreva um programa que simule o lançamento de uma moeda. Para cada lançamento da moeda, o programa deve imprimir Cara ou Coroa. Deixe o programa lançar a moeda 100 vezes e conte o número de vezes que cada lado da moeda aparece. Imprima os resultados. O programa deve chamar uma função separada flip que não utiliza argumentos e retorna O para coroa e 1 para cara. *Nota:* se o programa simular realisticamente o lançamento da moeda, cada lado da moeda deve aparecer aproximadamente metade das vezes.

3.35 Os computadores estão desempenhando um papel cada vez maior na educação. Escreva um programa que ajudará os alunos da escola do primeiro grau a aprender a multiplicar. Use rand para produzir dois inteiros positivos de um dígito. O programa deve então imprimir uma pergunta do tipo

Quanto **é 6 vezes 7?**

O aluno então digita a resposta. Seu programa deve examinar a resposta do aluno. Se ela estiver correta, o programa deve imprimir "Muito bem!" e fazer outra pergunta de multiplicação. Se a resposta estiver errada, o programa deve imprimir "Não. **Tente** novamente." e então deixar que o aluno fique tentando acertar a mesma pergunta repetidamente, até conseguir.

3.36 O uso dos computadores na educação é conhecido como *instrução assistida por computador* **(CAI,** *computerassisted instruction).* Um problema que ocorre em ambientes CAI é a fadiga dos alunos.. Isso pode ser eliminado variando-se o diálogo do computador para atrair a atenção do aluno. Modifique o programa do Exercício 3.35 de forma que sejam impressos vários comentários para cada resposta correta e incorreta, como se segue:

Comentários para uma resposta correta

Muito bem!
Excelente!
Bom trabalho!
Certo! Continue assim!

Comentários para uma resposta incorreta

Não. Tente novamente.
Errado. Tente outra vez.
Não desista!
Não. Continue tentando.

Use o gerador de números aleatórios para escolher um número de **1 a** 4 para selecionar o comentário apropriado para cada resposta. Use uma estrutura switch para as respostas.

3.37 Sistemas mais sofisticados de instrução assistida por computador controlam o desempenho do aluno durante um período de tempo. Freqüentemente, a decisão de iniciar um novo tópico é baseada no sucesso do aluno no tópico anterior. Modifique o programa do Exercício 3.36 para contar o número de respostas corretas e incorretas digitadas pelo aluno. Depois de o aluno digitar 10 respostas, seu programa deve calcular a porcentagem de respostas corretas. Se a porcentagem for menor do que 75%, seu programa deve imprimir "Solicite ajuda extra ao professor' e então encerrar a execução.

3.38 Escreva um programa em C++ que faça o jogo de "adivinhar um número" da maneira que se segue: seu programa escolhe a) Mc
um número para ser adivinhado, sorteando aleatoriamente um inteiro no intervalo de 1 a 1000. O programa então escreve: b) Mc
c) Mc

Se o palpite do jogador estiver incorreto, seu programa deve repetir as leituras até que o jogador finalmente acerte o número. O programa deve continuar dizendo Muito alto ou Muito baixo para ajudar o jogador a "aproximar-se" da resposta correta.

3.39 Modifique o problema do Exercício 3.38 para contar o número de palpites que o jogador fez. Se o número for 10 ou menor, imprima Ou você sabe o segredo ou está com sorte! Se o jogador acertar o número em 10 tentativas, **Fig. 3.33**

imprima Ah! Você sabe o segredo! Se o jogador fizer mais de 10 tentativas, imprima Você deve ser capaz de O processot

fazer melhor! Por que não deve demorar mais do que 10 tentativas? Bem, com cada “bom palpite” ojogador deve ser capaz te o disco pa
de eliminar metade dos números. Agora mostre por que qualquer número entre 1 e 1000 pode ser adivinhado, com 10 palpites ou Escre
menos, a) o
b)o
3.40 Escreva uma função recursiva potencia (base, expoente) que, quando chamada, retoma c) o
d) o Seu
no. Por exe
Por exemplo, potencia (3, 4) = 3 * 3 * 3 * 3. Assuma que expoente é um inteiro maior ou igual a 1. *Sugestão:* seqüência d
a etapa de recursão usaria o relacionamento
1.
base = *base. base* 1 -
3-
e a condição de término ocorre quando expoente for igual a 1 porque 1 -
2-
base' base 2+
l-.
3.41 A série de Fibonacci 3.43 Quah
vezes com
0,1,1,2,3,5,8,13,21,... conseguir,cc
clareza e sua
começa com os termos O e 1 e tem a propriedade de que cada termo subseqüente *é* a soma dos dois termos precedentes. a)
Escreva uma função *não-recursiva* fibonacci (ri) que calcula o n-ésimo número de Fibonacci. b) Determine o maior 3.44 (Visu
número de Fibonacci que pode ser impresso em seu sistema. c) Modifique o programa da parte a) para usar double em vez para imprim
de int para calcular e retornar os números de Fibonacci e use este programa modificado para repetir a parte (b). separadas e objetivo aqu
3.42 *(Torres de Hanói)* Todo principiante em Ciência da Computação deve lidar com determinados problemas clássicos, e o desejar adiei
problema das Torres de Hanói (veja a Fig. 3.28) é um dos mais famosos. Diz a lenda que, em um templo do Extremo Oriente, os monges estão tentando mover uma pilha de discos de um pino para outro. A pilha inicial tinha 64 discos colocados em um pino 3.45 O má
e organizados na ordem decrescente de diâmetro, da base para o topo. Os monges estão tentando mover a pilha desse pino para mdc que retc
um segundo pino com a restrição de que exatamente um disco deve ser movido de cada vez e em nenhum momento um disco se y for iguai
maior pode ser colocado sobre um disco menor. Há um terceiro pino disponível para colocação temporária de discos. Teoricamente, o mundo terminará quando os monges completarem sua tarefa, portanto há pouco incentivo para nós facilitarmos seus esforços. **3.46 A fur**
Assumiremos que os monges estão tentando mover os discos do pino 1 para o pino 3. Desejamos desenvolver um algoritmo static loc
que imprimirá a seqüência exata de transferências de discos de um pino para outro. da. Execute
Se fôssemos solucionar este problema com os métodos convencionais, rapidamente nos encontraríamos perdidos lidando
com os discos. Em vez disso, se abordarmos o problema com a recursão em mente, ele se toma imediatamente tratável. Mover *n* discos pode ser considerado em termos de mover apenas *n* - 1 discos (daí a recursão), como segue:

3.47 Os E:
plicação a u

Tenho um número entre 1 e 1000.
Você pode adivinhar meu número?
Digite seu primeiro palpite.
O jogador digita então o primeiro palpite. O programa escreve uma das respostas seguintes:
1.Excelente! Você adivinhou o número!
Você gostaria de jogar novamente (s ou n)?
2.Muito baixo. Tente novamente.
3.Muito alto. Tente novamente.

a) Mova *n* - 1 discos do pino 1 para o pino 2, usando o pino 3 como área de armazenamento temporário.
b) Mova o último disco (o maior) do pino 1 para o pino 3.
c) Mova os *n - 1* discos do pino 2 para o pino 3, usando o pino 1 como área de armazenamento temporário.

Fig. 3.33 O problema das Torres de Hanói para o caso com quatro discos.

O processo termina quando a última tarefa envolver mover *n = 1* disco, i.e., o caso básico. Isso é realizado movendo-se simplesmente o disco para o seu destino final, sem a necessidade de uma área de armazenamento temporário.

Escreva um programa para resolver o problema das Torres de Hanói. Use uma função recursiva com quatro parâmetros:

a) o número de discos a serem movidos
b) o pino no qual esses discos estão inicialmente colocados
c) o pino para o qual essa pilha de discos deve ser movida
d) o pino a ser usado como área de armazenamento temporário

Seu programa deve imprimir instruções precisas necessárias para mover os discos do pino inicial para o pino de destino. Por exemplo, para mover uma pilha de três discos do pino 1 para o pino 3. seu programa deve imprimir a seguinte seqüência de movimentos:

1 -3 (Isso significa mover o disco do pino 1 para o pino 3.)
1 .-,2
3-*2
1-s3
2- 1
2-3
l-3

3.43 Qualquer programa que pode ser implementado recursivamente pode ser implementado iterativamente, embora algumas vezes com dificuldade maior e clareza menor. Tente escrever uma versão iterativa do problema das Torres de Hanói. Se você conseguir, compare sua versão iterativa com a versão recursiva desenvolvida no Exercício 3.42. Investigue questões de desempenho, clareza e sua habilidade de demonstrar a correção do programa.

3.44 (Visualizando a redursão) É interessante observar a recursão "em ação". Modifique a função factorial da Fig. 3.14 para imprimir sua variável local e o parâmetro da chamada recursiva. Para cada chamada recursiva, mostre as saídas em linhas separadas e adicione um nível de recuo. Faça o melhor possível para tornar a saída clara, interessante e com significado. Seu objetivo aqui é desenvolver e implementar um formato de saída que ajude uma pessoa a entender melhor a recursão. Você pode desejar adicionar tais capacidades de exibição a muitos outros exemplos e exercícios de recursão ao longo do texto.

3.45 O máximo divisor comum dos inteiros x e y é o maior inteiro que divide exatamente x e y. Escreva uma função recursiva mdc que retoma o máximo divisor comum de x e y. O máximo divisor comum de x e y é definido recursivamente como se segue:

se y for igual a O, então mdc (**x,** y) é x: caso contrário, mdc (**x,** y) é mdc (y, x % y) onde % é o operador módulo.

3.46 A função main pode ser chamada redursivamente? Escreva um programa contendo a função main. Inclua a variável static local contador inicializada com 1. Pós-incremente e imprima o valor de contador cada vez que main for chamada. Execute seu programa. O que acontece?

3.47 Os Exercícios *3.35* a 3.37 desenvolveram um programa de instrução assistido por computador para ensinar multiplicação a um aluno de uma escola do primeiro grau. Este exercício sugere algumas melhorias para esse programa.

a) Modifique o programa para que ele permita ao usuário especificar o nível de dificuldade dos testes. Um nível de dificuldade 1 significa usar somente números de um único dígito nos problemas, um nível de dificuldade dois significa usar números de até dois dígitos, etc.

b) Modifique o programa para que ele permita ao usuário selecionar o tipo de problemas aritméticos que deseja estudar. Uma opção igual a 1 significa apenas problemas de adição, 2 significa apenas problemas de subtração, 3 significa apenas problemas de multiplicação, 4 significa apenas problemas de divisão e 5 significa misturar aleatoriamente os problemas de todos os tipos.

3.48 Escreva uma função distancia que calcule a distância entre dois pontos *(xl,* yl) e *(x2, y2).* Todos os números e valores de retorno devem ser do tipo double.

3.49 O que faz o programa seguinte?

```
1 // ex3_49.cpp
2 #include <iostream>
3
4 using std::cin;
5 using std::cout;
6
7 int main()
8
9 intc;
10
11 if ( ( c = cmn.getO) ) != EOF
```

```
12 main();
13 cout « c;
14 }
15
16 return 0;
17 }
```

3.50 O que faz o programa seguinte?

```
1 // ex0350.cpp
2 #include <iostream>
3
4 using std: :cout;
5 using std::cin;
6 using std: :endl;
7
8 int mystery ( int, int );
9
10 int main()
8 {
12 int x, y;
13
14 cout « "Digite dois inteiros: ";
15 cin»x»y;
16 cout « 'O resultado é ' « mystery( x, y ) « endi;
17 return 0;
18 }
19
20 // O parâmetro b deve ser um inteiro
```

1

```
21 // positivo para evitar uma recursão infinita
22 int mystery ( int a, int b
23 {
24 if(b==1)

26 else

return a + mystery ( a, b - 1
```


3.51 Depois de determinar o que faz o programa do Exercício 3.50, modifique-o para funcionar adequadamente após remover a restrição do segundo argumento não ser negativo.

3.52 Escreva um programa que teste o máximo possível das funções da biblioteca matemática listadas na Fig. 3.2. Experimente cada uma das funções, fazendo com que seu programa imprima tabelas com os valores de retomo para diversos valores de argumentos.

3.53 Encontre o erro em cada um dos seguintes segmentos de programas e explique como corrigi-lo:

a) float cube (float);

// **protótipo da função**

double cube (float nuniber) // **definição da função**

return number * nurnber * number;

b) register auto int x = 7;

e) int randomNuniber =srand d) float y =123.45678;

int x;

x =

cout « static_cast< float >(x) « endi;

e) double square (double number

(

}

double nuniber;

return nuxnber * number;

f) **int** sum(**int n**

if (n = O

return O;

else

return n + sum(n **);**

3.54 Modifique o programa do jogo de *craps* da Fig. 3.10 para permitir apostas. Transforme em uma função a parte do programa que executa um jogo de *craps.* Inicialize a variável bankBalance com 1.000 dólares. Peça ao jogador para digitar uma aposta e coloque este valor na variável wager. Use um laço while para verificar se wager é menor ou igual a bankBalance e, se não for, peça ao jogador para fornecer um novo valor de aposta, até que um valor válido para wager seja fornecido. Depois de ser informado um valor correto para wager. execute um jogo de *craps.* Se o jogador vencer, aumente bankBalance, com o valor de wager e imprima o novo **bankBalance.** Se o jogador perder, subtraia o valor de wager do total em bankBalance, imprima o novo bankBalance. verifique se bankBalance se tomou zero e, em caso positivo, imprima a mensagem "Sinto muito. Você está arruinado! - Com o passar do jogo. imprima várias mensagens para criar um "interlocutor estilo 'crupiê' - tais como: "Oh!, você vai perder tudo, hein?" ou "Isso mesmo, teste sua sorte!" ou **"Você é o máximo. Chegou a hora de trocar suas fichas por 'grana'!.**

25

return a;

27
28 }

3.55 Escreva um programa em C++ que usa um função iriline areaDoCirculo para solicitar ao usuário o raio de um círculo e calcular e imprimir a área desse círculo.

3.56 Escreva um programa completo em com as duas funções alternadas especificadas abaixo, sendo que cada uma delas simplesmente triplica a variável contador definida em main. Então compare e contraste as duas abordagens. Estas duas funções são
a) Função triplaChamadaPorValor que passa uma cópia de contador por valor, triplica a cópia, e retoma o novo valor.
b) Função triplaChamadaPorReferencia que passa contador com uma verdadeira chamada por referência, via um parâmetro de referência, e triplica a cópia original de contador através de seu nome alternativo (i.e., o parâmetro de referência).
3.57 Qual é a finalidade do operador unário de resolução de escopo?
3.58 Escreva um programa que usa um gabarito de função chamado mm para determinar o menor de dois parâmetros. Teste o programa usando pares de números inteiros, de caracteres e de números em ponto flutuante.
3.59 Escreva um programa que usa um gabarito de função chamado **max** para determinar o maior de três parâmetros. Teste o programa usando triplas de números inteiros, de caracteres e de números em ponto flutuante.
3.60 Determine se os segmentos de programa seguinte contêm erros. Para cada erro, explique como ele pode ser corrigido. Nota: para um segmento de programa particular, é possível que nenhum erro exista no segmento.
a) template < class A >

int sum(int numi, int num2, int num3)

return numl + num2 + num3;

```
b) void printResults( int x, int y
cout « "A soma é " « x + y « '\n';
return x + y;
e) template < A >
A product( A numi, A num2, A num3
return numi * num2 * num3;
```

d) double cube (int);

```
int cube ( int );
```

Arrays

Objetivos

• Apresentar a estrutura de dados array.
• Entender o uso de arrays para armazenar, classificar e pesquisar listas e tabelas de valores.
• Entender como declarar um array, inicializar um array e fazer referência aos elementos de um array.
• Ser capaz de passar arrays a funções.
• Entender técnicas básicas de classificação. __·Ser capaz de declarar e manipular arrays com vários subscritos.

Com soluços e lágrimas ele selecionou
Aquelas de maior tamanho
Lewis Carroli

Attempt the end, and never stand to doubt;
Nothing 's so hard, but search willfind it out.
Robert Herrick

Vai agora, escreve isso em uma tábua perante eles; vai e registra tudo em um livro Isaías 30:8

Fechei vossas palavras na memória;
a chave vós mesmo a guardareis.
William Shakespeare

4

Visão Geral

4.1 Introdução
4.2 Arrays
4.3 Declarando arrays
4.4 Exemplos usando arrays
4.5 Passando arrays a funções
4.6 Ordenando arrays
4.7 Estudo de caso: calculando média, mediana e moda usando arrays
4.8 Pesquisando arrays: pesquisa linear e pesquisa binária
4.9 Arrays multidimensionais
4.10 (Estudo de caso opcional) Pensando em objetos: identificando as operações de uma classe
Resumo Terminologia Erros comuns de programação Boas práticas de programação • Dicas de desempenho Dicas de portabilidade. Observações de engenharia de software • Dicas de teste e depuração . Exercícios de auto-revisão . Respostas dos exercícios de auto-revisão • Exercícios • Exercícios de recursão

4.1 Introdução

Este capítulo serve como uma introdução para o importante tópico de estruturas de dados. Os *arrays* são estruturas de dados que consistem em itens de dados do mesmo tipo, relacionados entre si. No Capítulo 6, discutimos a noção de *estruturas* e *classes* . cada uma capaz de manter itens de dados relacionados entre si, provavelmente de tipos diferentes. Os arrays e as estruturas são entidades "estáticas", **já** que permanecem do mesmo tamanho ao longo de toda a execução do programa (obviamente, eles podem ser da classe de armazenamento automático e, assim, podem ser criados e

destruídos cada vez que o programa entra e sai dos blocos nos quais são definidos). No Capítulo 15, apresentamos estruturas dinâmicas de dados, tais como listas, filas, pilhas e árvores, que podem crescer ou reduzir de tamanho à medida que o programa é executado. O estilo de arrays que utilizamos neste capítulo é o de arrays baseados em ponteiros, ao estilo de C (estudaremos ponteiros no Capítulo 5). Mais à frente no livro, no Capítulo 8, sobre "Sobrecarga de operadores", e no Capítulo 20, sobre "A biblioteca padrão de gabaritos (STL)", cobriremos arrays como objetos no sentido pleno da palavra, usando as técnicas da programação orientada a objetos. Descobriremos que estes arrays baseados em objetos são mais seguros e versáteis do que aqueles baseados em ponteiros, usados em C, que discutimos aqui no Capítulo 4.

4.2 Arrays

Um array é um grupo de posições de memória consecutivas, todas de mesmo nome e mesmo tipo. Para fazer referência a uma posição particular ou elemento no array, especificamos o nome do array e o *número da posição* daquele elemento no array.

A Fig. 4.1 mostra um array de valores inteiros chamado c. Esse array contém 12 *elementos.* Qualquer um destes elementos pode ser referenciado fornecendo-se o nome do array seguido do número da posição do elemento particular, entre colchetes **([1).** O primeiro elemento em qualquer array é o *elemento zero.* Dessa forma, o primeiro elemento do array c é referenciado como c [O 1 , o segundo elemento do array e é referenciado como c [1
(1 elemento a partir do início do array), o sétimo elemento do array e é referenciado como e [6] (6 elementos a partir do início do array) e, em geral, o elemento *i* do array e é referenciado como e [i - 1 1 **.** Os nomes de arrays seguem as mesmas convenções que os nomes de outras variáveis.

Nome do array (Observe que todos os
elementos desse array possuem o mesmo
nome, c)
c[O] -45
c[1] 6
c[2] O
c[3) 72
c[43 1543
c[5] -89
c[6] O
c[7) 62
c[8] -3
c[9] 1
c[1O] 6453
c[11] 78
Número de posição do elemento dentro
do array c

Fig. 4.1 Um array com 12 elementos.

O número da posição contido entre colchetes é chamado mais formalmente de *subscrito* (este número especifica a quantidade de elementos a partir do início do array). Um subscrito deve ser um inteiro ou uma expressão inteira (usando qualquer tipo integral). Se um programa usar uma expressão inteira como subscrito, a expressão será calculada para determinar o valor do subscrito. Por exemplo, se a é igual a 5 e b é igual a 6, o comando

```
c[ a + b ] += 2;
```

adiciona 2 ao elemento de array c [11] . Observe que o nome de um array com subscritos é um *Ivalue* - ele só pode ser usado no lado esquerdo de uma atribuição.

Vamos examinar o array c da Fig. 4.1 mais detalhadamente. O *nome* do array em sua totalidade é c. Seus 12 elementos são denominados c [0], c [1], c [2], ... c [11]. O *valor* de c [0] *é* -45. o valor de c [1] **6,0** valor de c[2] é 0,o valorde c[7] é 62 e o valor de c [11] é 78. Para imprimir a soma dos valores contidos nos três primeiros elementos do array c, escreveríamos

```
cout «c[ O ] + c[ 1 1 + c[ 2 1 « endi:
```

Para dividir por **2** o valor do sétimo elemento do array e e atribuir o resultado à variável **x.** escreveríamos

```
x = c[ 6 1 / 2;
```

Erro comum de programação 4.1

É *importante observar a diferença entre o "sétimo elemento do* ***array" e o "elemento número sete do*** *array". Como os subscritos dos arrays iniciam em O, o "sétimo elemento do array" tem subscrito 6, ao*

passo que o "elemento número sete do array" tem subscrito 7 e, na realidade, é o oitavo elemento do arra' Esse fato é uma fonte de "erros por um".

Na verdade, os colchetes usados para conter o subscrito de um array são considerados um operador em C++. Eles possuem o mesmo nível de precedência que os parênteses. A tabela da Fig. 4.2 mostra a precedência e a associatividade dos operadores apresentados até agora no texto. Eles são mostrados de cima para baixo, em ordem decrescente de precedência.

Fig. 4.2 Precedência e associatividade dos operadores.

4.3 Declarando arrays

Os arrays ocupam espaço na memória. O programador especifica o tipo de cada elemento e o número de elementos exigidos pelo array, de forma que o compilador possa reservar a quantidade apropriada de memória. Para dizer ao compilador para reservar 12 elementos para o array inteiro c, use a declaração

int c[**12);**

Pode-se reservar memória para vários arrays com uma única declaração. A seguinte declaração reserva 100 elementos para o array inteiro b e 27 elementos para o array inteiro x.

intb[**100** *]* . **x[27** *]*

Podem ser declarados arrays para conter outros tipos de dados. Por exemplo, pode ser usado um array do tipo char para armazenar um *string* de caracteres. Os *strings* de caracteres e seu relacionamento com os arrays (um relacionamento que C++ herdou de C) e o relacionamento entre ponteiros e arrays são discutidos no Capítulo 5. Depois, então, discutiremos *strings* como objetos no pleno sentido da palavra.

4Ã Exemplos usando arrays

O programa da Fig. 4.3 usa uma estrutura de repetição for para inicialiLar com zeros os elementos de um array inteiro a de dez elementos e imprime o array em formato de tabela. O primeiro comando de saída exibe os cabeçalhos das colunas para as colunas impressas na estrutura for subsequente. Lembre-se de que setw especifica o comprimento do campo no qual o *próximo* valor será exibido.

Operadores			Associatividade	Tipo
O	[]		da esquerda para a direita	precedência mais alta
++	¨	static_cast<tipo> ()	da esquerda para a direita	pós-fixo
++	¨	+ -	da direita para a esquerda	unário
*	*1*	%	da esquerda para a direita	multiplicativo
+	-		da esquerda para a direita	aditivo
«	»		da esquerda para a direita	inserção/extração
<	<=	> >	da esquerda para a direita	relacional
==	! =		da esquerda para a direita	igualdade
&&			da esquerda para a direita	E lógico
1			da esquerda para a direita	OU lógico
?:			da direita para a esquerda	condicional
	+=	-= *= *1* = %=	da direita para a esquerda	atribuição
,			da esquerda para a direita	vírgula


```
1 II Fig. 4.3: figO4_03.cpp
2 II inicializando um array
3 #include <iostream>
4
5 usirig std::cout;
6 using std: :endl;
7
8 #include <iomanip>
```

Os elementos de um array também podem ser inicializados na declaração do array com um sinal de igual e uma lista de *inicializadores* separados por vírgulas (entre chaves). O programa da Fig. 4.4 inicializa um array inteiro com dez valores e o imprime em forma de tabela.

```
9
10 using std::setw;
11
12 int main()
13
14 int i, n[ 10 ];
15
16 for ( i =O; i < 10; i++ ) II inicializa o array
17 n[ i ]=0;
18
19 cout « "Elemento" « setw( 13 ) « "Valor" « endi;
20
21 for ( i =O; i < 10; i++ ) II imprime o array
22 cout « setw( 7 ) « i « setw( 13 ) « n[ i ] « endl;
23
24 return O;
25 )
```

Fig. 4.3 Inicializando os elementos de um array com zeros.

Fig. 4.4 Inicializando os elementos de um array com uma declaração (parte 1 de 2).

Elemento	Valor
O	O
1	O
2	O
3	O
4	O
5	O
6	O
7	O
8	O
9	O

1	II Fig. 4.4: figO4O	4.cpp
2	1/ Inicializando um	array com uma declaração
3	#include <iostreain>	
4		
5	using std::cout;	
6	using std: :endl;	
7		
8	#include <iomanip>	
9		
10	using std::setw;	

20

```
21 returri 0;
22 )
```

```
Elemento Valor

< 10; i++)
i « setw( 13 ) « n[ i J « endl;
```

Se houver menos inicializadores do que o número de elementos do array, os elementos restantes são inicializados automaticamente com zero. Por exemplo, os elementos do array n da Fig. 4.3 poderiam ser inicializados com o valor

zero por meio da declaração

```
int n[ 10 ] = { O };
```

que inicializa explicitamente o primeiro elemento com zero e automaticamente inicializa os nove elementos restantes também com zero, porque há menos inicializadores do que elementos do array. Lembre-se de que os arrays automáticos não são inicializados implicitamente com zero, O programador deve inicializar pelo menos o primeiro elemento com zero para que os elementos restantes sejam automaticamente zerados. O método usado na Fig. 4.3 pode ser usado repetidamente durante a execução do programa.
A declaração de array

```
int n[ 5 ] { 32, 27, 64, 18, 95, 14 };
```

causaria um erro de sintaxe, porque há seis inicializadores e apenas 5 elementos no array.

IErro comum de programação 4.2
Esquecer de inicializar os elementos de um array cujos elementos precisam ser inicializados é um erro de lógica.

Erro comum de programação 4.3
Fornecer mais inicializadores para um array do que o número de seus elementos é um erro de sintaxe.

```
11
12
13
14


15
16


17


int main O
int n[ 10 3 { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 }; cout « "Elemento « setw( 13 ) « 'Valor" « endl;


18
19


for ( int i = 0; i cout « setw( 7
```

Fig. 4.4 Inicializando os elementos de um array com uma declaração (parte 2 de 2).

1	27
2	64
3	18
4	95
5	14
6	90
7	70
8	60
9	37
	1

Se o tamanho do array for omitido em uma declaração com uma lista de inicializadores, o número de elementos do array será o número de elementos da lista de inicializadores. Por exemplo,
criaria um array com cinco elementos.

Dica de desempenho 4.1

Se, em vez de inicializar um array com comandos de atribuição, durante a execução, você inicializar o array durante a compilação com uma lista de inicializadores de array, seu programa será executado mais rapidamente.

O programa da Fig. 4.5 inicializa os dez elementos de um array s com os valores 2, 4, 6 **20** e imprime o array em formato de tabela. Os valores são gerados multiplicando-se o contador de controle do laço por 2 e somando-se 2.

```
1 // Fig. 4.5 figO4O5.cpp
2 // Inicializa os elementos do array com os inteiros pares de 2 a 20
3 #include <iostream>

4

5 using std::cout;
6 using std: :endl;
7
8 #include<iomanip>
9
10 using std::setw;
11

24

int main

const int arraySize = 10;
int j, s[ arraySize ];

for ( j = 0; j < arraySize; ++ ) // define os valores s[ j ] = 2 + 2 * j;
cout « "Elemento" « setw( 13 ) « "Valor" « endl;
for ( j = 0; j < arraySize; j++ ) // imprime os valores
cout « setw( 7 ) « j « setw( 13 ) « s[ j ] « endi;

25 return 0;

26 )

intn[] {1, 2,3,4,5);
```

iados

valor

```
12
13
14
15
16
17
18
19
20
21
22
23
```

stanrrays eiro
;. 4.3

rode

Fig. 4.5 Gerando os valores para serem colocados nos elementos de um array.

Elemento	Valor
0	2
1	4
2	6
3	8
4	10
5	12
6	14
7	16
8	18
9	20

A linha 14
`const int arraySize` = 10;
usa o qualificador const para declarar arraySize. denominada uma *variável constante,* cujo valor é 10. Variáveis constantes devem ser inicializadas com uma expressão constante quando de sua declaração e não podem mais ser modificadas (Fig. 4.6 e Fig. 4.7). Variáveis constantes são também chamadas de *constantes com nome* ou *variáveis somente leitura.* Note que o termo "variável constante" é um oxímoro . uma contradição de linguagem, como "camarão gigante" ou "queimadura de frio". (Favor mandar seus oxímoros favoritos para nosso e-mail listado no prefácio. Obrigado!).

```
1 // Fíg. 4.6: fig04_06.cpp
2 // Usando uma variável constante inicializada apropriadamente
3 #include <iostream>
4
5 usíng std::cout;
6 usíng std::endl;
7
8 int main()
9{
10 const int x = 7; // variável constante inicializada
11
12 cout « "O valor da variável constante x é:
13 « x « endl;
14
15 return 0;
16 }
```

O valor da variável constante x **é: 7**

Fig. **4.6** Inicializando e usando corretamente uma variável constante.

```
1 // Fig. 4.7: fig04_07.cpp
2 // Um objeto const deve ser inicializado
3
4 int main()
5{
6 const int x; // Erro: x deve ser inicializado
7
8 x = 7; // Erro: não se pode modificar uma variável const
9
10 return 0;
11 }
```

Mensagens de erro do compilador Borland C++ com linha de comando:

```
Fig04_07.cpp:
Error E2304 Fig04_07.cpp 6: Constant variable 'x' must be
initialized in function main()
Error E2024 Fig04_07.cpp 8: Cannot modify a const object in
function main()
*** 2 errors in Compile
```

Fig. 4.7 Um objeto const deve ser inicializado (parte 1 de 2).

Fig. 4.7 Um objeto **const** deve ser inicializado (parte 2 de 2).

Erro comum de programação 4.4

Atribuir um valor a uma variável constante em um coínando executável é um erro de sintaxe.

Variáveis constantes podem ser colocadas em qualquer lugar onde é permitida uma expressão constante. Na Fig. 4.5, a variável constante arraySize é usada para especificar o tamanho do array s na declaração

```
int j, s [ arraySize ];
```

Erro comum de programação 4.5

Somente constantes devem ser usadas para declarar arrays automáticos e estáticos. Não usar uma constante para esta finalidade é um erro de sintaxe.

Usar variáveis constantes para especificar tamanhos de arrays torna os programas *mais flexíveis.* Na Fig. 4.5, o primeiro laço for preencheria um array de 1000 elementos, simplesmente modificando-se de 10 para 1000 o valor de arraySize. Se a variável constante arraySize não tivesse sido utilizada, teríamos de modificar o programa em três lugares diferentes para adaptá-lo ao array de 1000 elementos. A medida que os programas se tornam maiores, essa técnica se torna mais útil para escrever programas mais claros.

Observação de engenharia de software 4.1

___Definir o tamanho de cada array com uma variável constante torna os programas mais flexíveis.

Boa prática de programação 4.1

Definir o tamanho do array como uma variável constante, em vez de se usar uma constante, torna os programas mais claros. Esta técnica é usada para se eliminar os chamados números mágicos; *ou seja, mencionar repetidamente o número 10 no código que processa um array de 10 elementos dá ao número 10 um significado artificial, podendo infelizmente confundir o leitor quando também existirem no programa outros números 10 que nada têm a ver com o tamanho do array.*

O programa da Fig. 4.8 soma os valores contidos em um array a, que possui 12 elementos inteiros. O comando no corpo do laço for faz a totalização. E importante lembrar que os valores que estão sendo fornecidos como inicializadores para o array a normalmente seriam digitados pelo usuário através do teclado para o programa ler. Por exemplo, a seguinte estrutura for

```
for ( int = O; j < arraySize; j++
cm >> a[ j ] 1;
```

Mensagens de erro do compilador Microsoft Visual C++

riaais
mo no

lê um valor digitado no teclado de cada vez e armazena o valor no elemento a [j

Compiling...				
FigO4_07 cpp				
d:\FigOd_07.cpp(6) error	C2734:			
const object must	be initialized	if	not	extern
d:\figO4_07.cpp(8) : error	C2166:			
1-value specifies const	object			
Error executing ciexe.				
test.exe - 2 error(s), O warning(s)				


```
1 // Fig. 4.8: figO4O8.cpp
2 // Calcula a soma dos elementos de um array
3 #include <iostream>
4
5 using std::cout;
6 using std: :endl;
7
8 int main ()
9{
10 const int arraySize = 12;
11 int a[ arraySize ] = { 1, 3, 5, 4, 7, 2, 99,
12 16, 45, 67, 89, 45 };
13 int total = O;
14
15 for ( nt i = 0; i <= arraySize; i++
16 total += a[ i ];
17
18 cout « "A soma dos valores dos elementos do array e · « total « enl;
19 return 0;
20 }
```

```
A soma dos valores dos elementos do array é 383
```

Fig. 4.8 Calculando a soma dos elementos de um array.

Nosso próximo exemplo usa arrays para resumir os resultados dos dados coletados em uma pesquisa. Considere o seguinte enunciado de problema:

h)iperguntado a quarenta alunos o nível de qualidade da comida na cantina estudantil, em uma escala de O a 10 (1 significando horrorosa e 10 significando excelente). Coloque as quarenta respostas em um array inteiro e resuma os resultados da pesquisa.

Essa é uma aplicação típica de arrays (veja a Fig. 4.9). Queremos resumir o número de respostas de cada tipo (i.e., 1 a 10). O array responses possui 40 elementos, com as respostas dos alunos. Usamos um array frequency. com li elementos, para contar o número de ocorrências de cada resposta. Ignoramos o primeiro elemento, frequency [O 1 porque é mais lógico fazer com que a resposta 1 incremente frequency [1] do que frequency [O]. Isso permite usar diretamente cada resposta como subscrito do array frequency.

```
1 // Fig 4.9: fig04_09.cpp
2 // Programa de resumo da pesquisa com os estudantes
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <iomanip>
9
10 using std::setw;
11
12 int main()
13
```

```
14 const int responseSize = 40, frequencySize = 11;
15 int responses[ responseSize ] = { 1, 2, 6, 4, 8, 5, 9, 7, 8,
16 10,1,6,3,8, 6,10,3,8,2,7, 6,5,7,6,8, 6,7,
```

Fig. 4.9 Um programa de análise de uma votação de estudantes (parte 1 de 2).


```
17 5, 6, 6, 5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
18 int frepiency[ frequencySize 1 = { 0 };
19
20 for ( int answer 0; answer < responseSize; answer++
21 ++frequency[ responses[answer] ];
22
23 cout « "Avaliação « setw( 17 ) « "Freqüência" « endi;
24
25 for ( int rating =1; rating < frequencySize; rating++
26 cout « setw( 6 ) « rating
27 « setw( 17 ) « frequency[ rating ] « endi;
28
29 return 0;
30 }
Avaliação Freqüência
1 2
2 2
3 2
4 2
5 5
6 11
7 5
8 7
9 1
10 3
lere o
```

Fig. 4.9 Um programa de análise de uma votação de estudantes (parte 2 de 2).

Boa prática de programação 4.2

Procure obter clareza nos programas. Algumas vezes, vale a pena trocar um uso mais eficiente da memória e do processador por programas escritos com maior clareza.

(i.e.,

flCY. 1 *Dica de desempenho 4.2*

ento, _____

que *Algumas vezes, as considerações de desempenho são muito mais importantes do que as considerações de clareza.*

O primeiro laço for (linhas 20 e 21) obtém uma resposta por vez do array responses e incrementa 1 dos 10 contadores (frequency [1 1 a frequency 10 1) no array frequency. O comando principal do laço e

```
++frequency[ responses[ answer ] ];
```

Esse comando incrementa o contador frequency apropriado, dependendo do valor de responses [answer 1. Por exemplo, quando a variável do contador answer for O, responses [answer] é 1, portanto ++frequency [responses [answer 1 1 ; é na verdade interpretado como

```
++frequency[ 1 ];
```

que incrementa o elemento um do array. Quando answer for 1, responses [answer] será 2, portanto ++frequency [responses [answer 1 1 ; é interpretado como

```
++frequency[ 2 1;
```


que incrementa o elemento dois do array. Quando answer for 2, responses *[* answer J será 6, portanto ++frequency *[* responses *[* answer j *1* ; é interpretado como

```
++frequency[ 6 ];
```

que incrementa o elemento seis do array, e assim por diante. Observe que, independentemente do número de respostas processadas, apenas um array de 11 elementos é necessário (ignorando o elemento zero) para resumir os resultados. Se os dados tivessem valores inválidos, como 13, o programa tentaria adicionar 1 a frequency *[* 13]. Isso estaria além dos

limites do array. *C++ não verifica os limites do array para evitar que o computadorfaça referência a uni elemento não-existente.* Assim, um programa pode ultrapassar os limites de um array sem emitir qualquer aviso. O programador deve ter certeza de que todas as referências ao array permanecem dentro de seus limites. No Capítulo 8, estenderemos C++ implementando um array como um tipo definido pelo programador através do emprego de uma classe. Nossa nova definição de array nos permitirá executar muitas operações que não são operações padrão dos arrays primitivos disponíveis em C++. Por exemplo, seremos capazes de comparar arrays diretamente, atribuir um array a outro array, exibir ou ler arrays completos usando cm e cout, inicializar arrays automaticamente, evitar o acesso a elementos fora do array e mudar o intervalo dos subscritos (e mesmo o tipo de seus subscritos) de maneira que o primeiro elemento de um array não precise ser o elemento 0.

Erro comum de programação ***4.6***

Fazer referência a um elemento além dos limites de um array é um erro de lógica que ocorre durante a execução. Não é um erro de sintaxe.

Dica de teste e depura ção ***4.1***

Ao usar um laço para percorrer um array, os subscritos nunca devem ser menores do que O e sempre devem ser menores do que o número total de elementos do array (uma menos que o tamanho do array). Certifique-se de que a condição de término do laço evite o acesso a elementos fora desses limites.

Dica de teste e depura ção 4.2

Os programas devem verificar a exatidão de todos os valores de entrada para evitar que informações erradas afetem os cálculos executados pelo programa.

Dica de portabilidade ***4.1***

Os efeitos (geralmente graves) da referência a elementos situados fora dos limites de um array dependem dos sistemas. Freqüentemente isso resulta em alteração do valor de uma variável não-relacionada.

Dica de teste e depura ção ***4.3***

Quando estudarmos classes (a partir do Capítulo 6), veremos como desenvolver um "array inteligente" que verifica automaticamente se os subscritos em todas as referências estão dentro dos limites durante a execução. O uso de tais tipos de dados inteligentes ajuda a eliminar erros.

Nosso próximo exemplo (Fig. 4.10) lê os números de um array e põe em gráfico as informações na forma de um gráfico de barras ou histograma - cada número é impresso e então uma barra constituída do mesmo número de asteriscos é impressa ao seu lado. O laço for aninhado na verdade desenha as barras. Observe o uso de endl para finalizar uma barra do histograma.

```
1 // Fig. 4.10: fig04_10.cpp
2 // Programa para imprimir histograma
3 #include <iostream>
4
```

Fig. 4.10 Um programa que imprime histogramas (parte 1 de 2).


```
5 using std::cout;
6 using std: :endl;
7
8 #include <iomanip>
9
10 using std::setw;
11
12 int main()
13
14 const int arraySize = 10;
15 int n[ arraySize ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
16
17 cout « 'Elemento" « setw( 13 ) « "Valor"
18 « setw( 17 ) « "Histograma" « endi;
19
20 for ( int i = i < arraySize; i++ ) {
21 cout « setw( 7 ) « i « setw( 13
22 «n[i)«setw(9);
23
24 for ( int = 0; j < n[ i ); j++ ) // imprime urna barra
25 cout « "*";
26
27 cout « endl;
28 }
29
```

```
30 return 0;
31 }
Elemento Valor Histograrna
0 19
1 3
2 15
3 7 *******
4 11
5 9
6 13
7 5 *****
8 17
9 1 *
```

Fig. 4.10 Um programa que imprime histogramas (parte 2 de 2).

Erro comum de programação 4.7

Embora seja possível usar a mesma variável contadora de um laço for *em um segundo laço* for *aninhado no interior do primeiro, isto é normalmente um erro de lógica.*



Dica de teste e depura ção 4.4

Embora seja possível modificar um contador de laço no corpo de um for, *evite fazer isso porque essa prática freqüentemente leva à ocorrência de erros sutis.*

No Capítulo 3, afirmamos que mostraríamos um método mais elegante de escrever o programa do jogo de dados da Fig. 3.8. O problema era lançar um dado de seis faces 6000 vezes para verificar se o gerador de números aleatórios produzia realmente números aleatórios. Uma versão desse programa utilizando arrays é mostrada na Fig. 4.11.


```
1 // Fig. 4.11: fig04ll.cpp
2 // Lançar um dado de seis lados 6000 vezes
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <iomanip>
9
10 using std::setw;
11
12 #include <cstdlib>
13 #include <ctime>
14
15 int main ()
16
17 const int arraySize =7;
18 int face, frequency[ arraySize ] ={ 0 };
19
20 srand( time( 0 ) );
21
22 for ( int roll =1; roll <= 6000; roll++
23 ++frequency[ 1 + rand() % 6 ]; // substitui o switch de
24 // 20 linhas da Fig. 3.8
25
26 cout « "Face" « setw( 13 ) « "Freqüência" « endi;
27
28 // ignora o elemento 0 no array frequency
29 for ( face =1; face < arraySize ; face++
30 cout « setw( 4 ) « face
31 « setw( 13 ) « frequency[ face 1 « endl;
32
33 return 0;
```

```
34 }
```

Fig. 4.11 Programa de lançamento de dados usando arrays em vez de switch.

Até este ponto, analisamos apenas arrays de inteiros. Entretanto, os arrays são capazes de conter dados de qualquer tipo. Analisamos, agora, o armazenamento de *strings* em arrays de caracteres. Até agora, o único recurso de processamento de *strings* que utilizamos foi imprimir um *string* com cout e «. Um *string* como "alô" é, na realidade, um array de caracteres em C++.

Os arrays de caracteres possuem várias características exclusivas. Um array de caracteres pode ser inicializado

usando um *string* literal. Por exemplo, a declaração

char stringl[ ] = "primeiro";

inicializa os elementos do array stringl com os valores de cada um dos caracteres do *string* literal "primeiro". O tamanho do array stringl na declaração anterior é determinado pelo compilador, com base no comprimento do *string.* E importante notar que o *string* "primeiro" contém oito caracteres *mais* um caractere especial

Face	Freqüência
1	1037
2	987
3	1013
4	1028
5	952
6	983

de término de *string,* chamado *caractere nulo.* Dessa forma, o array stringl contém, na verdade, nove elementos. A representação como caractere constante do caractere nulo é \O' .Todos os *strings* em C++ terminam com esse caractere. Um array de caracteres representando um *string* deve sempre ser declarado com tamanho suficiente para conter o número de caracteres do *string* mais o caractere nulo de término.

Os arrays de caracteres também podem ser inicializados com constantes de caracteres individuais em uma lista de inicializadores. A declaração anterior é equivalente a

char stringl[ ] = { 'p', 'r', 'i', 'm', 'e', **'i',** 'r', 'o', '\O' };

Como os *strings* são, na verdade, arrays de caracteres, podemos acessar diretamente os caracteres individuais de um *string* usando a notação de subscritos de arrays. Por exemplo, stringl O *]* é o caractere p' e stringl *[* 3 é o caractere 'm'.

Também podemos ler um *string* diretamente para um array de caracteres, a partir do teclado, usando cm e».

Por exemplo, a declaração

char string2 [20);

cria um array de caracteres capaz de armazenar um *string* de 19 caracteres e um caractere nulo de término. O comando

cm » string2;

lê um *string* do teclado para string2 e automaticamente acrescenta o caractere nulo ao fim de string2. Observe que, no comando precedente, somente o nome do array é passado; nenhuma informação sobre o tamanho do array é fornecida. E responsabilidade do programador assegurar que o array para o qual o *string* é lido pode conter qualquer string que o usuário digite no teclado. cm lê caracteres do teclado, até que o primeiro caractere de espaço em branco seja encontrado .para cm o tamanho do array não tem importância. Dessa forma, ler dados com ciri e » pode inserir dados além do final do array (veja a Seção *5.12* para obter informações de como evitar a inserção de dados além do final de um array char).

Erro comum de programação 4.8

Não fornecer a cm » *um array de caracteres com comprimento suficiente para armazenar um* string *digitado no teclado pode resultar na perda de dados em um programa e em outros erros sérios durante a execução.*

Um array de caracteres representando um *string* terminado com o caractere nulo pode ser impresso com cout e «. O array **string2** é impresso com o comando

```
cout « string2 « endi;
```

Observe que para cout «, assim como para cm», não importa o tamanho do array de caracteres. Os caracteres lquer do *string* são impressos até que seja encontrado um caractere nulo de término.

so de A Fig. 4.12 mostra como inicializar um array de caracteres com um *string* literal, ler um *string* e armazená-lo é, na em um array de caracteres, imprimir um array de caracteres como *string* e ter acesso a caracteres individuais de um *string.*

izado

```
1 // Fig. 4.12: fig04l2.cpp
2 // Tratando arrays de caracteres como strings
3 #include <iostream>
4
aei- 5 using std::cout;
```

Tipri ecia

Fig. 4.12 Tratando `arrays` de caracteres como *strings* (parte 1 de 2).


```
6 using std::cin;
7 using std: :endl;
8
9 int main O
10
11 char stringl[ 20 ], string2[] = "string constante";
12
13 cout « "Digite um string:
14 cm » stringl;
15 cout « "stringl é: -« stringl
16 « '\nstring2 é: -« string2
17 « "\nstringl com espaços entre os caracteres é:\n';
18
19 for ( int i = 0; stringl[ i ] != 'O'; i++
20 cout « stringl [ i ] «
21
22 cm » stringl; II lê "bem?"
23 cout « "\nstringl é: -« stringl « endi;
24
25 cout « endi;
26 return 0;
27 }
Digite um string: Tudo bem?
stringl é: Tudo
string2 é: string constante
stringl com espacos entre os caracteres é:
Tudo
Stringl é: bem?
```

Fig. 4.12 Tratando arrays de caracteres como *strings* (parte 2 de 2).

A Fig. 4.12 usa uma estrutura for (linhas 19 e 20) para fazer um laço através do array stringl e imprimir os caracteres individuais, separados por espaços. A condição na estrutura for, stringl [i] ! = '\O',é verdadeira enquanto o caractere nulo não for encontrado no *string.*

O Capítulo 3 analisou o especiticador de classe de armazenamento static . Uma variável local static em uma definição de função existe durante toda a execução do programa, mas só é visível no corpo da função.

Dica de desempenho 4.3

f *Podemos aplicar* **static** *a uma declaração de um array local de modo que o array não seja criado e inicializado, cada vez que a função é chamada, e destruído cada vez que a execução deixa afunção. Isto melhora o desempenho.*

Os arrays declarados como estáticos (static) são inicializados automaticamente uma única vez, durante a compilação. Se um array static não for inicializado explicitamente pelo programador, será inicializado com zeros pelo compilador quando o array for criado.

A Fig. 4.13 mostra a função staticArraylnit com um array local declarado static e a função automaticArraylnit com um array local automático. A função staticArraylnit é chamada duas vezes. O array local static na função é inicializado com zeros pelo compilador. A função imprime o array, adiciona 5 a cada elemento e imprime o array novamente. Na segunda vez em que a função é chamada, o array static contém os valores armazenados durante a primeira execução da função. A função automaticArraylnit também é chamada duas vezes. Os elementos do array local automático na função são inicializados com os valores 1, 2 e 3. A função imprime o array, adiciona *5* a cada elemento e imprime o array novamente. Na segunda vez em que a função é chamada, os elementos do array são inicializados com 1, 2 e 3 novamente porque o array tem classe de armazenamento automática.

```
1 // Fig. 4.13: fig04_13.cpp
2 // Arrays static são inicializados com zeros
3 #include <iostream>
4
5 using std::cout;
6 using std: :endl;
7
8 void staticArraylnit( void );
9 void automaticArraylnit( void );
10
11 int main()
12
13 cout « Primeira chamada para cada função:\n";
14 staticArraylnitQ;
15 automaticArraylnitO;
16
17 cout « \n\nSegunda chamada para cada função:\n';
18 staticArraylnitO;
19 automaticArraylnitO;
20 cout « endl;
21
22 return 0;
23
24
25 // função para demonstrar um array local static
26 void staticArraylnit( void
27
28 static int arrayl[ 3 ];
29 int i;
30
31 cout « \nValores ao entrar em staticArraylnit:\n";
32
iprimiros 33 for ( i 0; i < 3; i++
verdadei- 34 cout « "arrayl[" « i « "] = " « arrayl [ i ] «
35
static 36 cout « "\nValores ao sair de staticArraylnit:\n";
inção. 37
38 for (i=0; i<3; i++)
39 cout « "arrayl[ « i « ") =
40 « ( arrayl[ i ] += 5 ) «
icriadoe 41 }
ição. Isto 42
43 // função para demonstrar um array local automático
44 void automaticArraylnit( void
acompi- 5 {
erospe1o 46 int i, array2[ 3 ] { 1, 2, 3 };
47
a função 48 cout « "\n\nValores ao entrar em automaticArraylnit:\n";
ias vezes.
50 for(i=O;i<3;.++)
iciona5a 51 cout « array2[" « i « "] = « array2[ i ] «
contem 52
ambémé 53 cout « "\nValores ao sair de automaticArraylnit:\n";
,2e3,A 54
afunção 55 for ( i = i < 3; i++ )
Diasse de
```

Fig. 4.13 Comparando a inicialização de arrays static com a inicialização automática (parte 1 de 2).


```
56 cout « "array2 [" « i « '1 =
57 « ( array2[ i ] += 5 ) «
58 }
Primeira chamada para cada função:
Valores ao entrar em staticArraylnit:
arrayl[O) = O arrayl[l] = O arrayl[2] = O
Valores ao sair de staticArraylnit:
```

```
arrayl[O] = 5 arrayl[l] 5 arrayl[2] = 5
Valores ao entrar em automaticArraylnit:
array2[O] = 1 array2[l] = 2 array2[2] = 3
Valores ao sair de automaticArraylnit:
array2[O) = 6 array2[l] = 7 array2[2] = 8
Segunda chamada para cada função:
Valores ao entrar em staticArraylnit:
arrayl[O] = 5 arrayl[l] = 5 arrayl[2] = 5
Valores ao sair de staticArraylnit:
arrayl[O] = 10 arrayl[1) = 10 arrayl[2] = 10
Valores ao entrar em automaticArraylnit:
array2[OJ = 1 array2[l] = 2 array2[2) = 3
Valores ao sair de automaticArraylnit:
array2[O] = 6 array2[l] = 7 array2[2] = 8
```

Fig. 4.13 Comparando a inícialização de arrays static com a inicialização automática (parte 2 de 2).

Erro comum de programação 4.9

Supor que os elementos de um array local declarado static *são inicializados com zeros sempre que a função na qual o array* é *declarado for chamada, pode levar a erros em um programa.*

4.5 Passando arrays a funções

Para passar um array como argumento para uma função. especifique o nome do array sem os colchetes. Por exemplo, se o array temperaturaPorHora for declarado como

```
int temperaturaPorHora[ 24 );
```

o comando de chamada de função

```
modifyArray( temperaturaPorHora, 24 );
```

passa o array temperaturaPorHora e seu tamanho para a função modifyArray. Freqüentemente, ao passar um array para uma função, o tamanho do array também é passado, para que ela possa processar o número total de elementos do aay (caso contrário, teríamos de embutir este conhecimento na própria função ou, pior ainda, colocar o tamanho do array em uma variável global). No Capítulo 8, quando introduzirmos a classe Array. incorporaremos o tamanho do array no tipo definido pelo usuário - cada objeto Array que criarmos "saberá" seu próprio tamanho. Assim, quando passarmos um objeto Array para uma função, não precisaremos mais passar seu próprio tamanho como argumento.

A linguagem C++ automaticamente passa os arrays às funções usando chamadas por referência simulada as funções chamadas podem modificar os valores dos elementos nos arrays originais do chamador. O nome do array é o endereço do primeiro e'emento do array. Por ser passado o endereço inicial do array, a função chamada sabe precisamente onde o array está armazenado. Portanto, quando a função chamada modifica os elementos do array em seu corpo de função, os elementos reais estão sendo modificados em suas posições originais da memória.

1 *Dica de desempenho 4.4*

t *Faz sentido passar arrays através de chamadas por referência simuladas, por motivos de desempenho. Se os arrays fossem passados por chamadas por valor seria passada uma cópia de cada elemento. Para arrays grandes, passadosfreqüentemente, isso seria demorado e consumiria um espaço considerável para armazenar as cópias dos arrays.*

Observação de engenharia de software 4.2

___É possível passar um array por valor (usando um truque simples explicado no Capítulo 16) - isso rara- mente éfeito.

Embora arrays inteiros sejam passados simultaneamente por meio de chamadas por referência simuladas, os elementos individuais dos arrays são passados por meio de chamadas por valor, exatamente como as variáveis simples. Tais porções simples de dados são chamadas de *escalares* ou *quantidades escalares.* Para passar um elemento de um array para uma função, use o nome do elemento do array, com o subscrito, como argumento na chamada da função. No Capítulo 5, mostramos como simular chamadas por referência para escalares (i.e., variáveis individuais e elementos específicos de arrays).

Para uma função receber um array por meio de uma chamada de função, sua lista de parâmetros deve especificar que um array será recebido. Por exemplo, o cabeçalho da função modifyArray pode ser escrito como

```
void modifyArray( int b[|, int arraySize );
```

indicando que modifyArray espera receber o endereço de um array de inteiros no parâmetro b e o número de elementos do array no parâmetro **arraySize.** Não é exigido o tamanho do array entre colchetes. Se ele for incluído, o compilador o ignorará. Como os arrays são passados através de chamadas por referência simuladas, quando a função chamada usa o nome do array b, na realidade ela está se referindo ao array real no local que faz a chamada (array temperaturaPorHora na chamada anterior). No Capítulo 5, apresentamos outras notações para indicar que um array está sendo recebido por uma função. Como veremos, essas notações se baseiam no estreito relacionamento entre arrays e ponteiros.

Observe o aspecto estranho do protótipo de função para modifyArray
`void modifyArray( int[], int );`
Esse protótipo poderia ter sido escrito como
void modifyArray(int anyArrayName[], int anyVariableName
mas, como aprendemos no Capítulo 3, o compilador C++ ignora os nomes de variáveis em protótipos.
r
Boa prática de programação 4.3
r *Alguns programadores incluem nomes de variáveis em protótipos de funções para tornar os programas mais claros, O compilador ignora esses nomes.*
Lembre-se de que o protótipo informa ao compilador o número de argumentos e o tipo de cada argumento (na ordem em que se espera que os argumentos se apresentem).

O programa da Fig. 4.14 demonstra a diferença entre passar um array inteiro e passar um elemento de um array. O programa imprime inicialmente os cinco elementos de um array inteiro a. A seguir, a e seu tamanho são passados para a função modifyArray. na qual cada elemento de a é multiplicado por 2. Então, a é impresso novamente em main. Como mostra a saída do programa, os elementos de a são verdadeiramente modificados por modifyArray. Depois disso, o programa imprime o valor de a [3 1 e o passa para a função modifyElement. A função modifyElement multiplica seu argumento por 2 e imprime o novo valor. Observe que, ao ser impresso novamente em main, a [3] não é modificado porque os valores individuais dos elementos do array são passados por meio de uma chamada por valor.

```
1 // Fig. 4.14: fig04_14.cpp
2 // Passando arrays e elementos individuais de arrays para funções
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <iomanip>
9
10 using std::setw;
11
12 void niodifyArray( int [], int ); // parece estranho
13 void modifyElement( int );
14
15 int main()
16
17 const int arraySize 5;
18 inti, a[arraySize]={0, 1,2,3,4);
19
20 cout « 'Efeitos de passar o array inteiro usando chamada por rexerencia:
21 « "\n\nOs valores do array original são:\n';
22
23 for ( i =0; i < arraySize; i++
24 cout « setw( 3 ) « a[ i ];
25
26 cout « endl;
27
28 // array a passado usando chamada por referência
29 modifyArray( a, arraySize );
30
31 cout « "Os valores do array modificado são:\n";
32
33 for ( i =0; i < arraySize; i++
34 cout « setw( 3 ) « a[ i 1;
35
36 cout « "\n\n\n"
37 « "Efeitos de passar elemento do array usando chamada por valor:"
38 « "\n\nO valor de a[3] é -« a[ 3 ] « '\n';
39
40 modifyElement( a[ 3 J );
41
```

```
42 cout « "O valor de a[3] é -« a[ 3 ] « endl;
43
44 return 0;
45 }
46
47 // Na função modifyArray, "b" aponta para
```

Fig. 4.14 Passando arrays e elementos individuais de arrays para funções (parte 1 de 2).


```
48 // o array "a" original na memória.
49 void modifyArray( int b[], int sizeOfArray
50
51 for ( int j = 0; j < sizeOfArray; j++
52 b[j] *=2;
53 }
// Na função modifyElement, "e" é uma cópia local
// do elemento de array a[ 3 ] recebido de main.
void modifyElement( int e
cout « "Valor em modifyElement é
« ( e *= 2 ) « endl;
```

Fig. 4.14 Passando arrays e elementos individuais de arrays para funções (parte 2 de 2).

Podem existir situações em seu programa nas quais não se deve permitir que uma função modifique os elementos de um array. Como os arrays são sempre passados por chamadas por referência simuladas, as modificações nos valores de um array são difíceis de controlar. A linguagem C++ fornece o qualificador de tipo const para evitar a modificação dos valores de um array em uma função. Quando uma função especifica um parâmetro array que é precedido pelo qualificador const, os elementos do array se tornam constantes no corpo da função e qualquer tentativa de modificar ali um elemento do array resulta em um erro durante a compilação. Isso permite que o programador corrija um programa para que não se tente modificar o valor dos elementos do array.

A Fig. 4.15 demonstra o qualificador const. A função tryToModifyArray é definida com o parâmetro cons t int b **[ ]** que especifica que o array b é constante e não pode ser modificado. Cada uma das três tentativas da função de modificar os elementos do array resulta no erro de compilação 'Cannot modify a const obj ect." (Não é possível modificar um objeto const.). O qualificador const será analisado novamente no Capítulo 7.

```
54
55
56
57
58
59
60
61
```

1

```
L.
```

Fig. 4.15 Demonstrando o qualificador de tipo const (parte 1 de 2).

1	// Fig. 4.15: figO4l5.cpp // Demonstrando o qualificador de tipo

2 3	const #include <iostream>		
4			
5 6 7	using std: :cout; using std: :endl;		
8 9	void tryToModifyArray(const	int [])	;
10	int main O		
11	{		

Efeitos de	passar o array inteiro usando chamada por referência:			
Os valores 012 Os valores 024	do array original são: 34 do array modificado são: 68			
Efeitos de	passar elemento do array usando	chamada	por	valor:
O valor de Valor em O valor de	a[3] é 6 modifyElement é 12 a[3] é 6			


```
12 int a[] ={ 10, 20, 30 };
13
14 tryToModifyArray( a );
15 cout«a[ 0] « «a[ 1] « «a[ 2] « '\n';
16 return 0;
17
18
19 II Na função tryToModifyArray, "b" não pode ser
20 // usado para modificar o array "a" original em main.
21 void tryToModifyArray(const int b[])
22
23 b[ O ] /=2; // erro
24 b[ 1 ] /=2; // erro
25 b[ 2 ] /=2; II erro
26
```

Mensagens de erro do compilador Borland C++ com linha de comando

Observação de engenharia de software 4.3

O qualificador de tipo cons *t pode ser aplicado a um parâmetro de um array em uma definição de função para evitar que o array original seja modificado no corpo da função. Esse é outro exemplo do princípio do mínimo privilégio. As funções não devem ter a capacidade de modificar um array, a menos que isso seja absolutamente necessário.*

```
FigO4_15 .cpp:
Error E2024 FigO4_15.cpp 23: Cannot modify a const object in
function tryToModifyArray(const int *const)
Error E2024 FigO4_15.cpp 24: Cannot modify a const object in
function tryToModifyArray(const int *const)
Error E2024 FigO4 15.cpp 25: Cannot modify a const object in
function tryToModifyArray(const int *const)
Warning W8057 FigO4_15.cpp 26: Paraxneter 'b' is never used in
function tryToModifyArray(const int *const)
3 errors in Compile ***
```

Mensagens de erro do compilador Microsofi Visual C++

```
Compiling...
FigO 415. cpp
D:\FigO4_15.cpp(23) : error C2166:
1-value specifies const object
D:\FigOd_15.cpp(24) : error C2166:
l-value specifies const object
D:\FigO4_15.cpp(25) : error C2166:
```

```
1-value specifies const object
Error executing cl.exe.
test.exe - 3 error(s), O warning(s)
```

Fig. 4.15 Demonstrando o qualificador de tipo const (parte 2 de 2). *Erro comum de programação 4.10*

Esquecer que arrays são passados por referência e, portanto, podem ser modificados, pode resultar em um erro de lógica.

4.6 Ordenando arrays

Ordenar dados (i.e., colocar os dados segundo uma determinada ordem, como ascendente ou descendente) é uma das aplicações computacionais mais importantes. Um banco classifica todos os cheques pelos números das contas para que possa preparar os extratos de cada conta no final do mês. As companhias telefônicas classificam suas listas pelo á 1 timo nome e, dentro disso, pelo primeiro nome, para que fique fácil encontrar números de telefone. Pratica- mente todas as organizações devem classificar algum dado e, em muitos casos, quantidades muito grandes de dados. Classificar dados é um problema fascinante, que tem atraído alguns dos esforços mais intensos de pesquisa no campo da Ciência da Computação. Neste capítulo, analisamos o que talvez seja o esquema mais simples de classificação. Nos exercícios e no Capítulo 15, investigamos esquemas mais complexos, que levam a um desempenho muito superior.

1 *Dica de desempenho 4.5*

f *Freqüentemente, os algoritmos mais simples apresentam um desempenho muito ruim. Sua virtude reside no fato de que são fáceis de escrever testar e depurar Algumas vezes, porém, são necessários algoritmos mais complexos para se atingir o melhor desempenho possível.*

O programa na Fig. 4.16 classifica os valores dos elementos de um array a de dez elementos em ordem ascendente. A técnica utilizada é chamada de *bubble sort,* ou *sinking sort,* porque os valores menores "sobem" gradualmente para o topo do array, da mesma forma que bolhas de ar sobem na água, enquanto os valores maiores afundam (submergem) para a parte de baixo do array. A técnica faz diversas passagens pelo array. Em cada passagem, são comparados pares sucessivos de elementos. Se um par estiver na ordem crescente (ou se os valores forem iguais), os valores são deixados como estão. Se um par estiver na ordem decrescente, seus valores são permutados no array.

```
1 // Fig.4.16: figo4_16.cpp
2 // Este programa classifica os valores de
3 // um array em ordem ascendente
4 #include <iostream>
5
6 using std::cout;
7 using std::endl;
8
9 #include <iomanip>
10
11 using std::setw;
12
13 int main ()
14
15 const int arraySize = 10;
16 int a[ arraySize ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
17 int i, hold;
18
19 cout « "Itens de dados na ordem original\n";
20
21 for ( i = 0; i < arraySize; i++
22 cout « setw( 4 ) « a[ i 3;
23
24 for ( int pass = 0; pass < arraySize - 1; pass++ ) // passagens
25
26 for ( i = 0; i < arraySize - 1; i++ ) // uma passagem
27
28 if ( a[ i 3 > a[ i + 1 3 ) { // uma comparação
29 hold = a[ i 3; // uma permuta
30 a[ i 1 = a[ i + 1 3;
```

Fig. 4.16 Ordenando um array com o *bubble sort* (parte 1 de 2).

```
31 a[ i + 1 ] hoJ.d;
32 }
33
34 cout « "\nltens de dados em ordem ascendente\n";
35
36 for ( i = O; i < arraySize; i++
37 cout « setw( 4 ) « a[ i ];
38
39 cout « endi;
40 return O;
4' }
Itens de dados na ordem original
2 6 4 8 10 12 89 68 45 37
Itens de dados em ordem ascendente
2 4 6 8 10 12 37 45 68 89
```

Fig. 4.16 Ordenando um array com o *bubble sort(parte* 2 de 2)

Inicialmente, o programa compara a [O) com a [**1** 1 . depois a [1 1 com a [**2**] . depois a [2 1 com a [3 J e assim por diante, até completar a passagem, comparando a [8 1 com a [9] . Observe que, embora existam 10 elementos, são realizadas apenas nove comparações. Tendo em vista o modo como são feitas as comparações, um valor grande pode se mover para baixo várias posições, mas um valor pequeno só pode se mover para cima uma única posição. Na primeira passagem, garante-se que o maior valor "submergirá" para o elemento mais baixo (para o "fundo") do array, a [9] . Na segunda passagem, garante-se que o segundo maior valor submergirá para a [8 1 . Na nona passagem, o nono maior valor submerge para a [1]. Isso deixa o menor valor em a [O 1, sendo necessárias apenas nove passadas no array para classificá-lo, embora ele tenha dez elementos.

A classificação é realizada pelo laço for aninhado. Se for necessária uma permuta, ela é realizada pelas três atribuições seguintes

```
hold = a[ i ];
a[ i  1 = a[ i  + ı 1;
a[ i + 1  1 = hold;
```

hold armazena temporariamente um dos valores que está sendo permutado. A permuta não pode ser realizada somente com as duas atribuições seguintes

```
a[ i  J = a[ i + 1  1;
a[ i + 1  ] = a[ i  1;
```

Se, por exemplo, a [i] é 7 e a [i + 1 1 é 5, depois da primeira atribuição ambos os valores serão iguais a 5 e o valor 7 será perdido. Daí necessitarmos da variável extra hold.

A principal virtude do *bubble sort* reside no fato de que ele é fácil de programar. Entretanto, o *bubble sort é* lento. Isso se toma perceptível durante a classificação de arrays grandes. Nos exercícios, desenvolveremos versões mais eficientes do *bubble sort* e investigaremos algoritmos de classificação muito mais eficientes do que o *bubble sort.* Cursos mais avançados investigam classificação e pesquisa em arrays com maior profundidade.

4.7 Estudo de caso: calculando média, mediana e moda usando arrays

Veremos agora um exemplo maior. Os computadores são usados normalmente para compilar e analisar os resultados de pesquisas de opinião e levantamentos. O programa da Fig. 4.17 usa o array response inicializado com 99

(representado pela variável constante responseSize) respostas a uma pesquisa de opinião. Cada uma das respostas é um número de 1 a 9. O programa calcula a média, a mediana e a moda dos 99 valores.

```
1
2
3
4
5
6
7
8
9
10
```

```
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53


using std::cout;
using std::endl;
using std::ios;
#include <iomanip>
using std::setw;
using std: :setiosflags; using std: :setprecision;


{ 6, 7, 8, 9, 8,
7, 8, 9, 5, 9,
6, 7, 8, 9, 3,
7, 8, 9, 8, 9,
6, 7, 8, 7, 8,
7, 8, 9, 8, 9,
5, 6, 7, 2, 5,
7, 8, 9, 6, 8,
7, 4, 4, 2, 5,
4, 5, 6, 1, 6,


7, 8, 9, 8, 9,
8, 7, 8, 7, 8,
9, 8, 7, 8, 7,
8, 9, 7, 8, 9,
7, 9, 8, 9, 2,
8, 9, 7, 5, 3,
```

```
3, 9, 4, 6, 4,
7, 8, 9, 7, 8,
3, 8, 7, 5, 6,
5, 7, 8, 7 };
```

```
// Fig. 4.17: fig04_17.cpp
// Este programa apresenta o tópico de análise de dados de pesquisas.
// Ele calcula a média, a mediana e a moda dos dados
#include <iostream>
```

j com mbora itas as ode se para o maior leixa o tenha

```
void mean( const int [], int ); void median( int [], int );
voidmode( int [), int [1, int ); void bubbleSort( int[], int ); void printArray( const
int[J, int );
int main Q
const int responseSize =99;
int frequency[ 10 ] = { O },
response[ responseSize ] =
```

as três

pode

isa5

sorté
Tsões
ubble

```
mean( response, responseSize );
median( response, responseSize ); mode( frequency, response, responseSize );
return 0;
void mean( const int answer[), int arraySize int total =0;
cout « "********\n Média\n********\n";
for (int j =0; j <arraySize; j++ total +=answer[ j ];
```

Lados rn 99

Fig. 4.17 Programa de análise de uma pesquisa de opinião (parte 1 de 3).

]


```
54 cout « "A média é o valor médio dos itens de\n"
55 « "dados. A média é igual à soma de todos os\n"
56 « "itens de dados dividido pelo número de\n"
```

```
57 « “itens de dados (“ « arraySize
58 « “) O valor médio para\n este caso é:
59 « total « -/ -« arraySize « -=
60 « setiosflags ( ios: : fixed l ios: : showpoint
61 « setprecision( 4
62 « static_cast< double >( total ) / arraySize« ‘\n\n”;
63
64
65 void median( int answer[], int size
66
67 cout « “\n********\n Mediana\n********\n”
68 « “0 array de respostas não ordenado é”;
69
70 printArray( answer, size );
71 bubbleSort( answer, size );
72 cout « “\n\nO array ordenado é”;
73 printArray( answer, size );
74 cout « ‘\n\nA mediana é o elemento -« size / 2
75 « -do\narray ordenado de -« size
76 « -elementos.\npara este caso a mediana é
77 « answer[ size / 2 l « “\n\n”;
78
79
80 void mode( int freq[], int answer[), int size
81 {
82 int rating, largest =0, modeValue =0;
83
84 cout « “\n********\n Moda\n********\n;
85
86 for ( rating =1; rating <= 9; rating++
87 freq[ rating ] = 0;
88
89 for ( int j = 0; j < size; j++
90 ++freq[ answer[ j ) ];
91
] 92 cout « “Resposta” « setw( 11 ) « Freqüência”
93 « setw( 19 )« “Histograma\n\n” « setw( 55
94 « “1 1 2 2\n” « setw( 56
95 « “5 O 5 O 5\n\n”;
96
97 for ( rating =1; rating <= 9; rating+-I- ) {
98 cout « setw( 8 )« rating « setw( 11
99 « freq[ rating l « -
100
101 if ( freq[ rating ] > largest ) {
102 largest = freq[ rating );
103 modeValue = rating;
104 }
105
106 for ( int h =1; h <= freq[ rating J; h++
107 cout « -:
108
109 cout « ‘\n’;
110 }
```

Fig. 4.17 Programa de análise de uma pesquisa de opinião (parte 2 de 3).


```
112 cout « ‘A moda é o valor mais freqüente.\n”
113 « “Para este caso a moda é -« modeValue
```

```
114 « que ocorreu « largest « vezes.” « endl;
115
116
117 void bubbleSort( int a[], int size
118 {
119 int hold;
120
121 for ( int pass =1; pass < size; pass++
122
123 for ( int j =j < size -1; j++
124
if ( a[ j 3 > a[ j + 1 3 ) {
hold= a[ j ];
a[ j 3 =a[ j + 1 ];
a[ j + 1 3 =hold;
132 void printArray( const int a[], int size
133
134 for ( int j =0; j < size; j++
135
if ( j % 20 == O
cout « endl;
138
139 cout « setw( 2 ) « a[ j 1;
140 }
141 )
```

Fig. 4.17 Programa de análise de uma pesquisa de opinião (parte 3 de 3).

A média é a média aritmética dos 99 valores. A função mean calcula a média somando os 99 elementos e dividindo o resultado por 99.

A mediana é o “valor do meio”. A função median determina a mediana chamando bubbleSort para ordenar o array answer e apanha o elemento situado no meio do array ordenado, answer [size / 2]. Observe que, quando houver um número par de elementos, a mediana deve ser calculada como a média dos dois elementos do meio. A função median não possui atualmente a capacidade de fazer isso. A função printArray é chamada para imprimir o array answer.

Fig. 4.18 Exemplo de execução do programa de análise de dados de uma pesquisa de opinião (parte 1 de 2).

```
111


125
126
127
128
129
130
131


136
137
```



********			
Média			
********			
A média é o valor médio		**dos itens de**	
dados. A média é igual itens de dados dividida itens de dados (99). O este caso é: 681 / 99 =		**à soma de todos pelo número de valor médio para 6.8788**	**os**


```
Mediana
********
O array de respostas não ordenado é
67898789897895987878
67893987877898989789
67878798927898989753
56725394647896878978
7442538756456165787
O array ordenado é
12223333444445555555
56666666667777777777
77777777777788888888
88888888888888888888
99999999999999999999
A mediana é o elemento 49 do
array ordenado de 99 elementos.
Para este caso a mediana é 7
********
Moda
********
Resposta Freqüência Histograma
1 1 2 2
5 O 5 O 5
1 1 *
2 3
3 4 ****
4 5 ******
5 8 *********
6 9 **********
7 23 ***********************
8 27 ***************************
9 19
A moda é o valor mais freqüente.
Para este caso a moda é 8 que ocorreu 27 vezes.
```

Fig. 4.1 8 Exemplo de execução do programa de análise de dados de uma pesquisa de opinião (parte 2 de 2).

A moda o valor que ocorre mais freqüentemente entre as 99 respostas. A função mode determina a moda contando o número de respostas de cada tipo e depois selecionando o valor que obteve maior contagem. Essa versão da função mode não manipula um empate na contagem (veja o Exercício 4.14). A função mode produz ainda um histograma para ajudar na determinação da moda graficamente. A Fig. 4.18 contém um exemplo de execução desse programa. Esse exemplo inclui a maioria das manipulações comuns exigidas normalmente em problemas de arrays, incluindo a passagem de arrays a funções.

4.8 Pesquisando arrays: pesquisa linear e pesquisa binária

Freqüentemente, um programador trabalhará com grandes quantidades de dados armazenadas em arrays. Pode ser necessário determinar se um array contém um valor que corresponde a um determinado *valor-chave.* O processo de encontrar um determinado elemento de um array é chamado de *pesquisa.* Nesta seção, analisaremos duas técnicas de pesquisa - a técnica simples de *pesquisa linear* e a técnica mais eficiente de *pesquisa binária.* Os Exercícios 4.33 e 4.34, no final do capítulo, pedem a implementação de versões recursivas da pesquisa linear e da pesquisa binária.

A pesquisa linear (Fig. 4.19) compara cada elemento do array com a *chave de pesquisa.* Como o array não está em uma ordem específica, as probabilidades de o valor ser encontrado no primeiro elemento ou no último são

as mesmas. Entretanto, na média, o programa precisará comparar a chave de pesquisa com metade dos

elementos do array. Para determinar que um elemento não está no array, o programa deve comparar a chave de pesquisa com cada elemento do array.

```
1 II Fig. 4.19: figO4l9.cpp
2 /1 Pesquisa linear em um array
3 #include <iostreaxn>
4
5 using std::cout;
6 using std::cin;
7 using std::endl;
8
9 int linearSearch ( const int [] , int, int );
lo
11 int main()
12
13 const int arraySize = 100;
14 int a[ arraySize ], searchKey, element;
15
16 for ( int x = 0; x < arraySize; x++ ) // cria alguns dados
17 a[ x ] = 2 * x;
18
19 cout « "Digite a chave de pesquisa, valor inteiro:" « endl;
20 cm « searchKey;
21 element = linearSearch ( a, searchKey, arraySize
22
23 if ( element ! -1
24 cout « "Valor encontrado no elemento - « element « endl;
25 else
26 cout « "Valor não encontrado" « endl;
27
28 return 0;
29 )
30
31 int linearSearch( const int array[], mnt key, int smzeOfArray
32 {
33 for O int n = o; n < sizeOfArray; n++
34 if ( array[ n ] == key
35 return n;
36
37 return -1;
38 }
Digite a chave de pesquisa, valor inteiro:
36
Valor encontrado no elemento 18
Digite a chave de pesquisa, valor inteiro:
37
Valor não encontrado
```

Fig. 4.19 Pesquisa linear de um array.

O método de pesquisa linear funciona bem com arrays pe9uciios ou com arrays não-ordenados. Entretanto, para arrays grandes, a pesquisa linear é ineficiente. Se o array estiver ordenado, a técnica veloz de pesquisa binária pode ser utilizada.

O algoritmo de pesquisa binária elimina metade dos elementos do array que está sendo pesquisado após cada comparação. O algoritmo localiza o elemento do meio do array e o compara com a chave de pesquisa. Se forem

iguais, a chave de pesquisa foi encontrada e o subscrito daquele elemento do array é retornado. Se não forem iguais, o problema fica reduzido a pesquisar uma metade do array. Se a chave de busca for menor que o elemento do meio do array, a primeira metade do array será pesquisada; caso contrário, a segunda metade do array será pesquisada. Se a chave de busca não for encontrada no subarray (parte do array original), o algoritmo é repetido na quarta parte do array original. A pesquisa continua até que a chave de busca seja igual ao elemento situado no meio de um subarray, ou até que o subarray consista em um elemento que não seja igual à chave de busca (i.e., a chave de busca não é encontrada).

No pior caso, pesquisar um array de 1024 elementos precisará de apenas 10 comparações utilizando pesquisa binária. Dividir repetidamente 1 024 por 2 leva aos valores 5 1 2, *256,* 128, 64, 32, 16, 8, 4, 2 e 1 . **O** número 1 024 **(2b0)** é dividido por 2 apenas 10 vezes para que seja obtido o valor 1. Dividir por 2 é equivalente a uma comparação no algoritmo de pesquisa binária. Um array com 1048576 **(220)** elementos precisa de um máximo de 20 comparações para que a chave de pesquisa seja encontrada. Um array com um bilhão de elementos precisa de um máximo de 30 comparações para que a chave de pesquisa seja encontrada. Isso significa uma grande melhoria no desempenho, comparado com a pesquisa linear, que exigia, em média, a comparação da chave de pesquisa com metade dos elementos do array. Para um array de um bilhão de elementos, isso significa uma diferença entre 500 milhões de comparações e um máximo de 30 comparações! A quantidade máxima de comparações para qualquer array pode ser determinada encontrando-se a primeira potência de 2 maior do que o número de elementos do array.

1-Dica de desempenho 4.6

-F *Os tremendos ganhos em desempenho da pesquisa binária em relação à linear não são obtidos sem um preço. Classificar um array é uma operação cara, comparada com pesquisar todo um array em busca de um único item, uma vez só. O overhead de classificar um array passa a valer a pena quando o array necessitará ser pesquisado muitas vezes em alta velocidade.*

A Fig. 4.20 apresenta uma versão interativa da função binarySearch . A função recebe quatro argumentos - um array inteiro b, um inteiro searchKey. o subscrito do array low e o subscrito do array high. Se a chave de pesquisa não for igual ao elemento situado no meio de um subarray, o subscrito low ou high é modificado, de forma que um subarray menor possa ser examinado. Se a chave de pesquisa for menor que o elemento situado no meio, o subscrito high é definido como middle - 1, e a pesquisa continua nos elementos de **low a middle - 1** . Se a chave de pesquisa for maior do que o elemento situado no meio, o subscrito low é definido como middle + **1,** e a pesquisa continua nos elementos de middle + **1** a high . **O** programa usa um array de 15 elementos.

A primeira potência de 2 maior do que o número de elementos no array é 16 (2); portanto, exige-se um máximo de *4* comparações para encontrar a chave de pesquisa. A função printHeader exibe os subscritos do array e a função **printRow exibe cada** subarray durante o processo de pesquisa binária. O elemento do meio de cada subarray é marcado com um asterisco **(*),** para indicar o elemento com o qual a chave de pesquisa é comparada.

```
1 // Fig. 4.20: fig04_20.cpp
2 // Pesquisa binária em um array
3 #include <iostream>
4
5 using std::cout;
6 using std::cin;
7 using std: :endl;
8
9 #include <iomanip>
10
11 using std::setw;
12
13 int binarySearch ( const int [] int, int, int, int );
14 void printHeader ( int );
15 void printRow ( const int [], int, int, int, int);
16
```

1

Fig. 4.20 Pesquisa binária em um array ordenado (parte 1 de 3).

uais, 17 int main ()

```
meio 18
la.Se 19 const int arraySize =15;
rtedo 20 int a[ arraySize ), key, result;
array, 1 21
nãoé 22 for ( int i =O; i <arraySize; i++ )
23 a[ i ]=2 *j; // coloca alguns dados no array
24
quisa ...
11O\ 25 cout « Digite um numero entre O e 28:
1 26 cm » key;
ElOflo 27
Elções 28 printHeader( arraySize );
E1e30 29 result =binarySearch( a, key, 0, arraySize -1, arraySize );
nho, 30
dos 31 if ( result != -1 )
es de 32 cout « \n' « key « ' encontrado no elenento do array
leser 33 « result « endl;
34 else
35 cout « '\n' « key «" não encontrado" « endl;
36
-37 return 0;
num 38 }
fade 39
rray 40 II Pesquisa binária
41 int binarySearch ( const int b[], int searchKey, int 10w, int high,
42 int size
-um 43 {
iede 44 int middle;
o,de 45
[ono 46 while ( low <= high ){
e 47 middle =( low + high ) / 2;
dle 48
ntos. 49 printRow( b, low, middle, high, size );
iode 50
nção 51 if (searchKey == b[ middle ]) 1/ encontrou elemento igual
ayé 52 return middle;
53 else if (searchKey < b[ middle
54 high =middle -1; /1 pesquisa metade do array c/valores menores
55 else
56 low =middle + 1; 1/ pesquisa metade do array c/valores maiores
57
58
59 return -1; 1/ chave de pesquisa não encontrada
60 }
61
62 1/ Imprime um cabeçalho para a saida
63 void prmntHeader( int size
64 {
65 int i;
66
67 cout « "\nSubscritos:\n';
68
69 for ( i =0; i <size; i++
70 cout « setw( 3 ) « i « '
```

Fig. 4.20 Pesquisa binária em um array ordenado (parte 2 de 3).


```
71
72 cout « '\n;
73
74 for ( i =1; i <4 *size; i++
75 cout«
```

```
76
77 cout « endi;
78
79
80 // Imprime uma linha da saída mostrando a parte
81 // do array que está sendo processada.
82 void printRow( const int b[], int low, int mid, int high, int size
83
84 for ( int i = 0; i < size; i++
85 if ( i < low 1 1 i > high
86 cout «
87 else if ( i == mid ) // marca valor do meio
88 cout « setw( 3 ) « b[ i ] « "
89 else
90 cout « setw( 3 ) « b[ i ] « '
91
92 cout « endl;
```

1

```
Digite um número entre O e 28: 25
Subscritos:
O i 2 3 4 5 6 7 8 9 10 11 12 13 14
O 2 4 6 8 10 12 14* 16 18 20 22 24 26 28
16 18 20 22* 24 26 28
24 26* 28
24*
25 não encontrado
Digite um número entre O e 28: 8
Subscritos:
O i 2 3 4 5 6 7 8 9 10 11 12 13 14
O 2 4 6 8 10 12 14* 16 18 20 22 24 26 28
O 2 4 6* 8 10 12
8 10* 12
8*
8 encontrado no elemento 4 do array
Digite um número entre O e 28: 6
Subscritos:
O i 2 3 4 5 6 7 8 9 10 11 12 13 14
O 2 4 6 8 10 12 14* 16 18 20 22 24 26 28
O 2 4 6* 8 10 12
6 encontrado no elemento 3 do array
```

Fig. 4.20 Pesquisa binária em um array ordenado (parte 3 de 3).

Linha O
Linha 1
Linha 2

Subscrito da coluna
Subscrito da linha
Nome do array

Fig. 4.21 Um array bidimensional com três linhas e quatro colunas.

Erro comum deprogramação 4.11

Referenciar um elemento de array a **[** ***x*** *1 [y]* .*incorretamente, com subscritos como a [x* .***y*** *1. Na realidade, a [x* .***y*** *1 é tratado como a [y] porque C+* **+** *calcula o valor da expressão* .*y (contendo um operador vírgula) simplesmente como y (a última das expressões separadas por vírgulas).*

Um array com vários subscritos pode ser inicializado em sua declaração da mesma forma que um array de apenas um subscrito [array unidimensional]. Por exemplo, um array b *[* **2** *] [* **2** *]* com dois subscritos poderia ser declarado e inicializado com

```
intb[ 2 // 2 ] ={{ 1, 2 }, { 3, 4 } };
```

Os valores são agrupados por linha e colocados entre chaves. Assim, 1 e 2 inicializam b */* O 1 [O */* e b [O 1 *1* e 3 e 4 inicializam b */* 1 */ /* O */* e b */* 1 *1 /* 1 *1.* Se não houver inicializadores suficientes para uma determinada linha, os elementos restantes daquela linha são inicializados com O. Desta forma, a declaração intb[`2 ]/[ 2 ]` =*{ {* `1 ),{ 3, 4 } );` inicializariab[O */ /* O */* comi, b[O *1 /* 1 *1* com O, b[1 *1 /* O) com 3eb[1 *1 / 1]* com 4. A Fig. 4.22 demonstra a inicialização de arrays bidimensionais em declarações. O programa declara três arrays de duas linhas e três colunas. A declaração de arrayl fornece seis inicializadores em duas sublistas. A primeira

4.9 Arrays multidimensionais

Os arrays em C++ podem ter vários subscritos. Um uso comum de arrays com vários subscritos é representar *tabelas* de valores que consistem em informações organizadas em *linhas* e *colunas.* Para identificar um elemento específico de uma tabela, devemos especificar dois subscritos: o primeiro (por convenção) identifica a linha do elemento e o segundo (por convenção) identifica a coluna do elemento.

As tabelas, ou arrays, que exigem dois subscritos são chamados *arrays bidimensionais.* Observe que arrays podem ter mais de dois subscritos. Os compiladores de C++ suportam no mínimo 12 subscritos para um array. A Fig. 4.21 ilustra um array ***a*** com dois subscritos. O array contém três linhas e quatro colunas; portanto, diz-se que ele é **um** array 3 por 4. Em geral, diz-se que um array com *m* linhas e *n* colunas é um *array m por n.*

Cada elemento de a é identificado na Fig. 4.21 por um nome na forma ***a /*** i **J [j];** ***a é*** o nome do array e i e j são subscritos que identificam um elemento específico em a. Observe que os nomes de todos os elementos da primeira linha têm um primeiro subscrito igual a O; todos os nomes dos elementos da quarta coluna possuem um segundo subscrito igual a 3.

Coluna O Coluna 1 Coluna 2 Coluna 3

`a[0) [0]` a[O) `[1] a[0] [2]` a[O] `[3]`

a[1) [O) a[1] [1] a[1] [2] a[1] [3)
a[2] [O] a[2] [1) a[2] [2] a[2] [3) -

```
1
2
3
4


5
6
7


8
9


10
11
12
13
14
15
16
17
18
19
20
21


22
23
```

```
24


25
26
27


28
29
30
31


32
33


34


35
36
37


using std::cout;
using std::endl;
```


sublista inicializa a primeira linha do array com os valores 1. 2 e 3; e a segunda sublista inicializa a segunda linha do array com os valores 4, 5 e 6. Se as chaves em torno de cada sublista fossem removidas da lista de inicializadores de arrayl, o compilador inicializaria automaticamente os elementos da primeira linha e, a seguir, inicializaria os elementos da segunda linha.

```
// Fig. 4.22: figo4_22.cpp
// Inicializando arrays multidimensionais #include <iostream>


6 } ),


void printArray( int [] [ 3 ] );
int main O
intarrayl[2][3]={{1,2,3},{4,5, array2[2][3]{1,2,3,4,5}, array3[2][3]({1,2},{4}};
cout « "Valores em arrayl por linha: -« endl; printArray ( arrayl );
cout « "Valores em array2 por linha:" « endl;
printArray( array2 );
cout « "Valores em array3 por linha:" « endl; printArray( array3 ):
return 0;


}


void printArray( int a[] [ 3 ]
for ( int i =0; i < 2; i++ )
for (intjo; j<3; j++) cout«a[ i )[ j ] « ·
cout « endl;


}


}
```

```
Valores em arrayl por linha:
123
456
Valores em array2 por linha:
123
450
Valores em array3 por linha.
120
400
```

Fig. 4.22 Inicializando arrays multidimensionais.

A declaração de array2 fornece cinco inicializadores. Os inicializadores são atribuídos à primeira linha e, depois, à segunda linha. Quaisquer elementos que não tenham um inicializador explícito são inicializados com zero automaticamente; portanto, array2 [1 1 [2 J é inicializado com O.

L do A declaração de array3 fornece três inicializadores em duas sublistas. A sublista da primeira linha inicializa, **;** de explicitamente, os dois primeiros elementos da primeira linha com os valores 1 e 2. O terceiro elemento é inicializado flos automaticamente com o valor zero. A sublista da segunda linha inicializa explicitamente o primeiro elemento com 4.

Os dois últimos elementos são automaticamente inicializados com zero.

O programa chama a função printArray para criar a saída dos elementos de cada array. Observe que a definição da função especifica o parâmetro do aay como int a ***[J [*** 3 ***]*** . Ao recebermos um array unidimensional como argumento de uma função, os colchetes ficam vazios na lista de parâmetros da função. O primeiro subscrito de um array multidimensional também não é exigido, mas todos os outros subscritos o são. O compilador usa esses subscritos para determinar as posições na memória dos elementos de arrays multidimensionais. Todos os elementos do array são armazenados seqüencialmente na memória, independentemente do número de subscritos. Em um array bidimensional, a primeira linha é armazenada na memória, seguida da segunda linha.

Fornecer os valores dos subscritos na declaração de parâmetros permite ao compilador informar à função como localizar os elementos do array. Em um array bidimensional, cada linha é um array com um único subscrito. Para localizar um elemento de uma determinada linha, o compilador deve saber exatamente quantos elementos existem em cada linha, para que possa saltar o número adequado de locais da memória quando tiver acesso ao array. Desta forma, ao ter acesso a a ***[*** 1 ***] [*** 2 ***1*** em nosso exemplo, o compilador sabe que deve saltar os três elementos da primeira linha na memória para obter a segunda linha (linha 1). A seguir, o compilador tem acesso ao terceiro elemento daquela linha (elemento 2).

Muitas manipulações comuns de arrays usam estruturas de repetição for. Por exemplo, a estrutura a seguir atribui zero a todos os elementos da terceira linha do array a da Fig. 4.21:

```
for ( colunin = O; column < 4; column++ )
a[ 2 1 [ column ] = O;
```

Especificamos a *terceira* linha e, portanto, sabemos que o primeiro subscrito sempre será **2 (O** é o subscrito da primeira linha e 1 é o da segunda linha). O laço for faz variar apenas o segundo subscrito (i.e., o subscrito da coluna). A estrutura for anterior é equivalente aos comandos de atribuição:

```
a[ 2 ] [ O 3 O;
a[ 2 ] [ 1 ] = O;
a[ 2 ] [ 2 1 = O;
a[ 2 ) [ 3 ] = O;
```

A estrutura for aninhada a seguir determina o total de elementos no array a.

```
total = O;
for ( row = O; row < 3; row++
for ( column = O; column < 4; column++
total += a[ row ][ column 1;
```

A estrutura for calcula o total de elementos do array, passando por uma linha de cada vez. A estrutura for externa começa definindo row (i.e., o subscrito da linha) como O para que os elementos da primeira linha possam ser contados pela estrutura for interna. A estrutura for externa incrementa row de 1, para que os elementos da segunda linha possam ser contados. A seguir, a estrutura for externa incrementa row de **2,** para que os elementos da terceira linha possam ser contados. O resultado é impresso quando a estrutura for aninhada chega ao fim.

O programa da Fig. 4.23 realiza várias outras manipulações de arrays no array studentGrades usando estruturas for. Cada linha do anay representa um aluno e cada coluna representa uma nota em um dos quatro exames que os alunos fizeram durante o semestre. As manipulações de arrays são realizadas por quatro funções. A função minimum determina a menor nota de qualquer aluno naquele semestre. A função maximum determina a maior nota de qualquer aluno no semestre. A função ave rage determina a média semestral de um determinado aluno. A função printArray

imprime a saída do array de dois subscritos em um formato elegante de tabela.


```
1 II Fig. 4.23: figO4_23.cpp
2 /1 Exemplo de uso de array bidimensional
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7 using std: :ios;
8
9 #include <iomanip>
lo
11 using std::setw;
12 using std::setiosflags;
13 using std::setprecision;
14
15 const int students 3; /1 número de alunos
16 const int exaxns 4; /1 número de exames
17
18 int minimum( int [][ exams ) int, int );
19 int maximum( int [][ exains ] int, int );
20 double average( int [3 int );
21 void printArray( int [)[ exams J int, int );
22
23 int main O
24 {
25 int studentGrades[ students ][ exams )
26 { { 77, 68, 86, 73 },
27 { 96, 87, 89, 78 },
28 { 70, 90, 86, 81 } };
29
30 cout « "O array é:\n';
31 printArray( studentGrades, students, exams );
32 cout « "\n\nNenor nota:
33 « minimum( studentGrades, students, exams
34 « "\n\nMaior nota:
35 « maximum( studentGrades, students, exams ) « '\n';
36
37 for ( int person =0; person < students; person++
38 cout « "A nota média do aluno -« person « -é
39 « setiosflags( ios::fixed 1 ios::showpoint
40 « setprecision( 2
41 « average( studentGrades[ person ), exams ) « endl;
42
43 return 0;
44 }
45
46 // Encontra a nota mínima
47 int minimum( int grades[J [ exams ], int pupils, int tests
48 {
49 int lowGrade =100;
50
51 for ( int i =O; i <pupils ; i++
52
53 for ( int j=0; < tests ; j++
54
55 if ( grades[ i ][ j ]< lowGrade
56 lowGrade =grades ( i ][ j ];
57
```

Fig. 4.23 Exemplo de uso de arrays bidimensionais (parte 1 de 2).

```
58 return lowGrade;
59 )
60
61 1/ Encontra a nota máxima
62 int maximuxn( int gradest] [exaxns ) int pupils, int tests
63 {
64 int highGrade =0;
65
66 for (int i 0; i < pupils ; i++
67
68 for (int j =0; j <tests ; j++
69
70 if (grades[ i 3[ j ]>highGrade
71 highGrade =grades[ i ][j 1;
72
73 return highGrade;
74 )
75
76 II Determina a nota média de um determinado aluno
77 double average( int setOfGrades[), int tests
78 {
4_79 int total =0;
80
81 for (int i 0; i < tests; i++
82 total +setOfGrades [i ];
83
84 return staticcast< double >( total )/tests;
85
86
87 1/ Imprime o array
88 void printArray( int grades[) [exaxns ) int pupils, int tests
89
90 cout « 10) [1] [2] [3)';
91
92 for (int i 0; i < pupils; i++ ){
93 cout « '\nstudentGrades[" « i « ") ";
94
95 for (int j =0; j < tests; j++
96 cout« setiosflags( ios::left ) « setw( 5
97 « grades[ i )[j];
98 )
99 }
```

```
O array é:
[0] [1] [2] [3)
studentGrades[0] 77 68 86 73
studentGrades[l) 96 87 89 78
studentGrades[2] 70 90 86 81
Menor nota: 68
Maior nota: 96
A nota média do aluno O é 76.00
A nota média do aluno 1 é 87.50
A nota média do aluno 2 é 81.75
```

Fig. 4.23 Exemplo de uso de arrays bidimensionais (parte 2 de 2).

Cada uma das funções minimum, maximum e printArray recebe três argumentos - o array studentGrades (chamado grades em cada função), o número de alunos (linhas do array) e o número de exames (colunas do array).

E

Cada função executa um laço pelo array grades usando estruturas for aninhadas. A estrutura for aninhada a seguir é da definição da função minimum:

```
for ( i =0; i < pupils; i++
for ( j =0; j < tests; j++
if ( grades [ i ] [ :i ] < lowGrade
lowGrade =grades[ i ][ j ];
```

A estrutura for externa começa definindo i (i.e., o subscrito da linha) como O para que os elementos da primeira linha possam ser comparados com **lowGrade no corpo da estrutura for interna.** A estrutura for interna percorre as quatro notas de uma determinada linha e compara cada nota com lowGrade. Se uma nota for menor do que **lowGrade, lowGrade** é definido como essa nota. A seguir, a estrutura for externa incrementa o subscrito da linha em 1. Os elementos da segunda linha são comparados com a variável lowGrade. A estrutura for externa incrementa então o subscrito da linha para 2. Os elementos da terceira linha são comparados com a variável lowGrade. Quando a execução da estrutura aninhada for concluída. lowGrade conterá a menor nota do array bidimensional. A função maximum funciona de maneira similar à função minimum.

A função average recebe dois argumentos - um array unidimensional de resultados de testes para um estudante em particular e o número de resultados de testes no array. Quando a função average é chamada, o **primeiro argumento passado** é studentGrades [**student** J , que especifica que uma linha particular do array bidimensional studentGrades seja passada para average . Por exemplo, o argumento studentGrades [1] representa os quatro valores (um array unidimensional de notas) armazenados na segunda linha do array bidimensional studentGrades. Um array bidimensional pode ser considerado um array com elementos que são arrays unidimensionais. A função average calcula a soma dos elementos do array, divide o total pelo número de resultados de testes e retorna o valor de ponto flutuante obtido.

4.10 (Estudo de caso opcional) Pensando em objetos: identificando as operações de uma classe

Nas seções "Pensando em objetos" nos finais dos Capítulos 2 e 3, executamos algumas etapas iniciais de um projeto orientado a objetos para nosso simulador de elevador. No Capítulo 2, identificamos as classes que precisamos implementar e criamos um diagrama de classes que modela a estrutura do nosso sistema. No Capítulo 3, determinamos muitos dos atributos de nossas classes, investigamos os estados possíveis da classe Elevator e os representamos em um diagrama de mapa de estados, modelando em um diagrama de atividades a lógica que o elevador utiliza para responder ao pressionamento de botões.

Nesta seção, concentramos-nos na determinação das *operações* (ou comportamentos) das classes necessárias para implementar o simulador de elevador. No Capítulo 5, concentrar-nos-emos nas colaborações (interações) entre os objetos de nossas classes.

Uma operação de uma classe é um serviço que a classe presta a "clientes" (usuários) daquela classe. Consideremos as operações de algumas classes do mundo real. As operações de um rádio incluem ajustar sua estação e volume (normalmente invocadas por um ouvinte que está ajustando os controles do rádio). As operações de um carro incluem acelerar (invocada apertando-se o pedal do acelerador), desacelerar (invocada pressionando-se o pedal do freio), girar o volante e trocar de marchas.

Objetos normalmente não executam suas operações expontaneamente. Em vez disso, uma operação específica é normalmente invocada quando um objeto remetente (freqüentemente chamado de *objeto cliente)* envia uma mensagem para um objeto destinatário (freqüentemente chamado de *objeto servidor),* solicitando que o objeto destinatário execute aquela operação específica. Isto se parece com uma chamada de função membro - **exatamente** como as mensagens são enviadas para objetos em C++. Nesta seção, identificaremos muitas das operações que nossas classes precisam oferecer a seus clientes em nosso sistema.

Podemos derivar muitas das operações de cada classe diretamente da definição do problema. Para fazer isso, examinamos os verbos e frases com verbos da definição do problema. Relacionamos, então, cada uma destas frases a uma classe particular em nosso sistema (ver Fig. 4.24). Muitas das frases com verbos na tabela da Fig. 4.24 ajudarão a determinar as operações de nossas classes.

Fig. 4.24 Frases com verbos para cada classe do simulador.

Para criar operações a partir destas frases com verbos, examinamos as frases com verbos listadas com cada classe. O verbo “move-se”, listado com a classe Elevator. refere-se à atividade na qual o elevador se move entre andares. Deve “move-se” ser uma operação da classe Elevator? Nenhuma mensagem diz ao elevador para se mover; em vez disso, o elevador decide se mover em resposta ao pressionamento de um botão, com base na condição da porta estar fechada. Assim, “move-se” não corresponde a uma operação. A frase “chega em um andar” também não é uma operação porque o elevador decide sozinho quando chegou em um andar, baseado no tempo.
A frase “desliga botão do elevador” implica que o elevador envie uma mensagem para o botão do elevador, dizendo ao botão para se desligar. Portanto, a classe ElevatorButton precisa de uma operação para oferecer este serviço para o elevador. Colocamos esta operação no compartimento inferior da classe ElevatorButton em nosso diagrama de classes (Fig. 4.25). Representamos os nomes das operações como nomes de funções e incluímos as informações sobre o tipo de valor devolvido:
resetButton() : void
O nome da operação é escrito primeiro, seguido por parênteses contendo uma lista de parâmetros, separados por vírgulas, que a operação recebe (neste caso, nenhum). Um dois-pontos segue a lista de parâmetros, seguido pelo tipo de valor devolvido pela operação (neste caso, void). Observe que a maioria de nossas operações parece não ter parâmetros e ter um tipo devolvido void; isto pode mudar na medida em que nossos processos de projeto e implementação continuarem.
Da frase “soa a campainha do elevador” listada com a classe Elevator. concluímos que a classe Beil deveria ter uma operação que presta um serviço . soar. Listamos a operação ringBell sob a classe Beil.
Quando o elevador chega em um andar, ele “sinaliza sua chegada em um andar” e o andar responde executando suas diversas atividades (i.e., desligando o botão do andar e ligando a luz). Portanto, a classe Floor precisa de

‘ada a

meira rcorre
o que ito da tema cade.
ional.

a um ida, o array ides array ie são ro de

ojeto amos iinaesenvador

árias entre

sideção e e um o pe ecífium

bjeto lente ; que

isso, rases
4.24

Classe	Frases com verbos 1
E].evator	move-se, chega em um andar, desliga o botão do elevador, soa a campainha do elevador, sinaliza sua chegada em um andar, abre a porta, fecha a porta
Clock	bate a cada segundo
Scheduler	escalona momentos aleatoriamente, cria uma pessoa, diz a uma pessoa para entrar em um andar, verifica se um andar está desocupado, retarda a criação de uma pessoa em um segundo
Person	entra em um andar, aperta o botão do andar, aperta o botão do elevador, entra no elevador, sai do elevador
Floor	desliga o botão do andar, apaga a luz, acende a luz

FloorButton	chama o elevador
ElevatorButton	avisa o elevador para se mover
Door	(abertura da porta) avisa a pessoa para sair do elevador, (abertura da porta) avisa a pessoa para entrar no elevador
Beli	nenhuma na definição do problema
Light	nenhuma na definição do problema
Building	nenhuma na definição do problema

Elevator Clock Door
currentFloor : int = 1 time : int = O open : bool = false
direction : enum up getTime ( ) : int openDoor( ) : void
capacity : int = 1 tick( ) : void closeDoor( ) : void
arrivalTime : int
moving : bool = false
processTime (time : int) : void Floor
personEnters( ) : void occupied : bool = false Beli
personExits( ) : void <nenhuma ainda>
summonElevator( ) : void elevatorArrived( ) : void B
prepareToLeave( ) : void isOccupied( ) : bool ring e voi
Scheduler FloorButton Light
floorlArrivalTime : int pressed : bool = false
floor2ArrivalTime : int (bool = false
pressButton( ) : void
processTime(time : int) : void resetButton( ) : void turnOn( ) : void
Person 4
ID : int ElevatorButton _____
stepOntoFloor( ) : void pressed : bool = false Building
exitElevator( ) : void resetButton( ) : void <nenhuma ainda>
enterElevator( ) : void pressButton( ) : void runSimulation( ) : void

Fig. 4.25 Diagrama de classes com atributos e operações.

uma operação que preste este serviço. Chamamos esta operação de elevatorArrived e colocamos o nome da operação no compartimento inferior da classe Floor na Fig. 4.25.

As duas frases com verbos restantes, listadas com a classe Elevator, afirmam que o elevador precisa abrir e fechar sua porta. Portanto, a classe Door precisa oferecer estas operações. Colocamos as operações openDoor e closeDoor no compartimento inferior da classe Door.

A classe Clock lista a frase "bate a cada segundo". Esta frase chama a atenção para um ponto interessante. Certamente, "obter a hora" é uma operação que o relógio oferece, mas será a batida do relógio também uma operação? Para responder a esta questão, concentramo-nos em como funcionará nossa simulação.

A definição do problema indica que o *scheduler* precisa saber a hora atual para decidir se ele deve criar uma nova pessoa para entrar em um andar. O elevador precisa da hora para decidir se é hora de chegar em um andar. Também decidimos que cabe ao edifício a responsabilidade de executar a simulação e passar a hora para o *scheduler* e para o elevador. Começamos agora a ver como vai funcionar nossa simulação. O edifício repete as seguintes etapas, uma vez por segundo, durante toda a duração da simulação:

1. Obter a hora do relógio.
2. Passar a hora ao *scheduler* para que ele possa criar uma nova pessoa, se necessário.
3. Passar a hora para o elevador para que ele possa decidir chegar a um andar, se o elevador estiver se movendo.

Decidimos que o edifício tem responsabilidade total para executar todas as partes da simulação. Portanto, o edifício também deve incrementar o relógio. O relógio deve ser incrementado uma vez por segundo; então, a hora deve ser passada para o *scheduler* e para o elevador.

Isto nos leva a criar duas operações getTime e tick e listá-las sob a classe Clock. A operação getTime devolve um inteiro como o valor do atributo hora do relógio. Nos itens 2 e 3 anteriores, vemos as frases "passar a hora para o

scheduler" e "passar a hora para o elevador". Assim, podemos adicionar a operação processTime às classes Scheduler e Elevator. Podemos também adicionar a operação runSimulation à classe Bui].ding.
A classe Scheduler lista as frases com verbos "escalona momentos aleatoriamente" e "retarda a criação de uma pessoa em um segundo". O *scheduler* decide executar estas ações ele mesmo e não oferece estes serviços para clientes. Portanto, estas duas frases não correspondem a operações.
A frase "cria pessoa", listada com a classe Scheduler. representa um caso especial. Embora possamos modelar um objeto da classe Scheduler enviando uma mensagem "criar", um objeto da classe Person não pode responder a uma mensagem "criar" porque este objeto ainda não existe. A criação de objetos é deixada para os detalhes da implementação e não é representada como uma operação de uma classe. Discutimos a criação de novos objetos quando discutimos implementação, no Capítulo 7.
A frase "diz a uma pessoa para entrar em um andar", listada na Fig. 4.24, significa que a classe Person deve ter uma operação que o *scheduler* possa invocar para dizer à pessoa para entrar em um andar. Chamamos esta operação de stepOntoFloor e a listamos sob a classe Person.
A frase "verifica se um andar está desocupado" implica que a classe Floor precisa prestar um serviço que permita aos objetos do sistema saber se o andar está ocupado ou desocupado. A operação que criamos para este serviço deve devolver true se o andar está ocupado e false caso contrário. Colocamos a operação
isOccupied( ) : bool
no compartimento inferior da classe Floor.
A classe Person lista as frases "aperta o botão do andar" e "aperta o botão do elevador". Portanto, colocamos a operação pressButton sob as classes FloorButton e ElevatorButton em nosso diagrama de classes da UML (Fig. 4.25). Note que já lidamos com o fato de que uma pessoa "entra em um andar" quando analisamos as frases com verbos para a classe Scheduler, de modo que não precisamos criar nenhuma operação com base na frase "entra em um andar", listada com a classe Person. As frases "entra no elevador" e "sai do elevador", listadas com a classe Person, sugerem que a classe Elevator precisa operações que correspondam a estas ações.1
A classe Floor também lista "desliga o botão do andar" em sua coluna de frases com verbos, de modo que listamos a operação resetButton apropriada sob a classe FloorButton. A classe Floor também lista "apaga a nome da luz" e "acende a luz", de modo que criamos as operações turnOff e turnOn e as listamos sob a classe Light.
A frase "chama o elevador", listada sob a classe FloorButton, implica que a classe Elevator precisa de cisa abrir uma operação sununonElevator. A frase "avisa o elevador para se mover", listada com a classe ElevatorButton, nDoor e implica que a classe Elevator precisa prestar um serviço "mover-se". Antes que o elevador possa se mover,
entretanto, o elevador precisa fechar aporta. Portanto, listar uma operação prepareToLeave. na qual o elevador ressante. executa as ações necessárias antes de se mover, sob a classe Elevator. parece uma escolha mais apropriada. na opera- As frases listadas com a classe Door implicam que a porta envie uma mensagem para uma pessoa para dizer a ela que saia do elevador ou entre no elevador. Criamos duas operações para a classe Person para cobrir estes cnar uma comportamentos - **exitElevator e enterElevator.**
im andar. Por enquanto, não nos preocupamos demais com os parâmetros ou tipos de valores devolvidos; estamos *tcheduler* tentando somente obter um entendimento básico das operações de cada classe. A medida que continuarmos em seguintes nosso processo de projeto, o número de operações pertencentes a cada classe pode variar - podemos descobrir que
são necessárias novas operações ou que algumas das operações atuais são desnecessárias.

Diagramas de seqüência

Podemos usar o diagrama de seqüência da UML (ver Fig. 4.26) para modelar nosso "laço de simulação" - as etapas da discussão precedente que o edifício repete enquanto durar a simulação. O diagrama de seqüência enfoca como as mensagens são enviadas entre objetos ao longo do tempo.
e moven
edifício
deve ser Neste ponto, podemos apenas adivinhar o que fazem estas operações. Por exemplo, talvez estas operações modelem elevadores do mundo real com um sensor que detecta quando entram e saem passageiros. Por enquanto, simplesmente listamos estas operações. Descobriremos quais ações esta classe executa, se for o caso, quando nos concentrarmos na implementação de nosso simulador em C++.

: Building : Clock : Scheduler : Elevator
tick() getTime( )
processTime (cLirrentTime : int)



Fig. 4.26 Diagrama de seqüência modelando um laço de simulação.

Cada objeto é representado por um retângulo no topo do diagrama. O nome do objeto é colocado dentro do retângulo. Escrevemos nomes de objetos no diagrama de seqüência usando a convenção que apresentamos com o diagrama de objetos, na seção “Pensando em objetos” no fim do Capítulo 2 (Fig. 2.45). A linha tracejada que se estende para baixo, a partir do retângulo de um objeto, é a *linha de vida* deste objeto. A linha de vida representa a progressão do tempo. Ações acontecem ao longo da linha de vida de um objeto em ordem cronológica, de cima para baixo - uma ação próxima ao topo de uma linha de vida acontece antes de uma ação próxima à base.

Uma mensagem entre dois objetos em um diagrama de seqüência é representada como uma linha cheia com uma seta na ponta, que parte do objeto que envia aquela mensagem e chega no objeto que recebe a mensagem. O 1 nome da mensagem aparece acima da linha da mensagem e deve incluir quaisquer parâmetros que sejam passa- dos. Por exemplo, o objeto da classe Building envia a mensagem processTime para o objeto da classe **Elevator. O nome da** mensagem aparece acima da linha da mensagem e o nome do parâmetro (currentTime) aparece entre parênteses à direita da mensagem; cada nome de parâmetro é seguido por dois-pontos e o tipo do parâmetro.

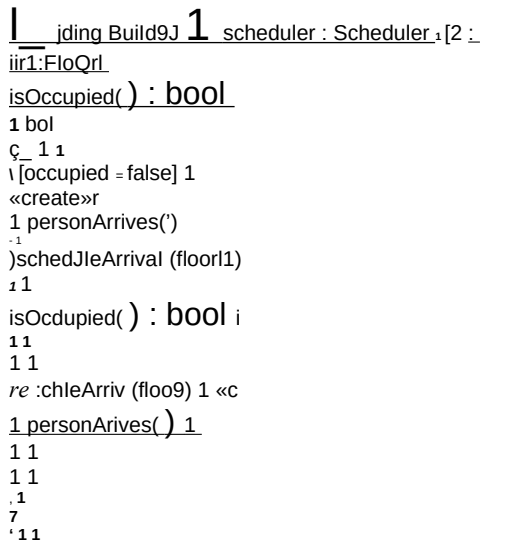
Se um objeto devolve o fluxo de controle ou se um objeto devolve um valor, uma mensagem de retorno (representada como uma linha tracejada com uma seta na ponta) parte do objeto que está devolvendo o controle para o objeto que inicialmente enviou a mensagem. Por exemplo, um objeto da classe Clock devolve time em resposta à mensagem getTime recebida de um objeto da classe Building.

Cada um dos retângulos ao longo das linhas de vida dos objetos - chamados de *ativações* - representa a duração de uma atividade. Uma ativação é iniciada quando um objeto recebe uma mensagem e é indicada por um ,f1 retângulo na linha de vida deste objeto. A altura do retângulo corresponde à duração da atividade ou atividades : iniciadas pela mensagem - quanto mais longa a duração da atividade, mais alto é o retângulo.

O texto na extremidade esquerda do diagrama na Fig. 4.26 indica uma restrição de temporização. Enquanto a hora atual é menor do que o tempo total de simulação (currentTime **< totalTime),** os objetos continuam enviando mensagens uns para os outros, na seqüência modelada no diagrama.

A Fig. 4.27 modela como o *scheduler* manipula o tempo e cria novas pessoas para entrar nos andares. Para este diagrama, supomos que o *scheduler* escalonou uma pessoa para entrar em cada um dos dois andares em um momento que coincide com a hora fornecida pelo edifício. Sigamos o fluxo de mensagens através deste diagrama de seqüência.

{currentTime < totalTime}



processTime (time)

[occupied = true]

delayArrival (floc

retânguliagrama nde para essão do o . uma

[occupied = false]

[occupied = true]

leia com agem. O II passa- [a classe tTime)
)tipodo

delayArrival (fio

retorno role para resposta

esenta a 1 por um ividades

Fig. 4.27 Diagrama de seqüência para o processo de agendamento.

quanto a ntinuam

es. Para em um rama de

Inicialmente, o objeto Building envia a mensagem processTime para o scheduler. passando a hora atual. O objeto scheduler precisa então decidir se deve criar uma pessoa para entrar no primeiro andar (representado pelo objeto floori da classe Floor). A definição do problema nos diz que o scheduler precisa primeiro verificar se o andar está desocupado antes que possa criar uma nova pessoa para entrar naquele andar. Portanto, o objeto scheduler envia uma mensagem isOccupied para o objeto floori para realizar esta tarefa.
O objeto floori devolve true ou false (indicado pela linha tracejada de mensagem de retomo e pelo tipo boo1. Neste ponto. a linha de vida do objeto scheduler se divide em duas linhas de vida paralelas para representar cada seqüência possível de mensagens que o objeto pode enviar, com base no valor devolvido pelo objeto floori. A linha de vida de um objeto pode se dividir em duas ou mais linhas de vida para indicar a *execução condicional de atividades.* Para cada linha de vida, deve ser especificada uma condição. A(s) nova(s) linha(s) de vida corre(m) em paralelo com a linha de vida principal e as linhas de vida podem convergir em algum ponto mais adiante.

Se o objeto floori devolve true (i.e., o andar está ocupado), o scheduler chama sua própria função delayArrival, passando um parâmetro que indica que o tempo de chegada no andar 1 precisa ser reescalonado. Esta função não é uma operação da classe Schedu].er porque não é invocada por outro objeto. A função delayArrival é simplesmente uma atividade que a classe Scheduler executa dentro de uma operação. Note que, quando o objeto scheduler envia uma mensagem para si mesmo (i.e., invoca uma de suas próprias funções membro), a barra de ativação para esta mensagem está alinhada sobre a borda direita da barra de ativação corrente.
Se o objeto floori devolve false (i.e., o andar está desocupado), o scheduler cria um novo objeto da classe Person. Em um diagrama de seqüência, quando é criado um novo objeto, o retângulo do novo objeto é colocado em uma posição vertical que corresponde ao momento em que o objeto é criado. Um objeto que cria outro objeto envia uma mensagem com a palavra "create" colocada entre os sinais (« **»).** A seta na ponta desta mensagem aponta para o retângulo do novo objeto. Um "X"grande no fim da linha de vida de um objeto indica a destruição daquele objeto. [Nota: nosso diagrama de seqüência não modela a destruição de nenhum objeto da classe Person portanto, não aparece nenhum "X" no diagrama. A criação e a destruição de objetos de forma dinâmica, usando os operadores new e delete são discutidas no Capítulo 71.
Depois que um novo objeto da classe Person é criado, a pessoa precisa, a seguir, entrar no primeiro andar.
Portanto, o novo objeto Person envia uma mensagem personArrives para o objeto f].oorl. Esta mensagem avisa o objeto floori de que uma pessoa está entrando nele.
Depois que o objeto scheduler criou um novo objeto da classe Person, ele escalona uma nova chegada para floori. O objeto scheduler invoca sua própria função scheduleArrival e a barra de ativação para esta chamada é centrada na borda direita da barra de ativação corrente. A função scheduleArrival não é uma operação; é uma atividade que a classe Scheduler executa dentro de uma operação. Neste ponto, as duas linhas de vida convergem. O objeto scheduler, então, trata do segundo andar da mesma maneira que o primeiro.
Quando o *scheduler* terminou com floor2, o objeto scheduler devolve o controle para o objeto building.
Nesta seção, discutimos as operações de classes e apresentamos o diagrama de seqüência da UML para ilustrar estas operações. Na seção "Pensando em objetos" no fim do Capítulo 5, examinamos como os objetos em um sistema interagem uns com os outros para executar uma tarefa específica e começamos a implementar nosso simulador de elevador em C++.

Resumo

. C++ armazena listas de valores em arrays. Um array é um grupo de posições de memória consecutivas relacionadas. Estas posições são relacionadas pelo fato de que todas elas têm o mesmo nome e o mesmo tipo. Para se referir a uma posição ou elemento particular no array, especificamos o nome do array e o subscrito. O subscrito indica o número de elementos a partir do início do array.
. Um subscrito pode ser um inteiro ou uma expressão inteira. As expressões de subscritos são avaliadas para determinar o elemento particular do array.
. É importante notar a diferença ao se referir ao sétimo elemento do array em vez do elemento sete do array. O sétimo elemento tem um subscrito 6, enquanto o elemento sete do array tem um subscrito 7 (na realidade, é o oitavo elemento do array). Isso é fonte de "erros por um".
. os arrays ocupam espaço na memória. Para reservar 100 elementos para o array inteiro b e 27 elementos para o array inteiro **x,** o programador deve escrever

```
int b[ 100 ], x[ 27 1;
```

. Pode ser usado um array do tipo char para armazenar um *string* de caracteres.
. Os elementos de um array podem ser inicializados de três maneiras: por declaração, por atribuição e por entrada de dados.
. Se houver menos inicializadores do que elementos do array, C++ inicializa os elementos restantes com o valor zero.
. A linguagem C++ não evita a referência a elementos além dos limites do array.
. Um array de caracteres pode ser inicializado por meio de um *string* literal.
. Todos os *strings* em C++ terminam com o caractere nulo **(. ** O **.)**
. Os arrays de caracteres podem ser inicializados com constantes de caracteres em uma lista de inicializadores.

. É possível ter acesso direto a cada caractere de um string armazenado em um array utilizando a notação de subscritos de array.
. Para passar um array a uma função, o nome do array é passado. Para passar um elemento específico de um array para uma função, simplesmente passe o nome do array seguido do subscrito (entre colchetes) do elemento em particular.
. A linguagem C++ passa arrays para funções usando chamada por referência simulada . as funções chamadas podem modificar os valores dos elementos nos arrays originais dos chamadores. O nome do array é o endereço de seu primeiro elemento. Por ser passado o endereço inicial do array, a função chamada sabe precisamente onde o array está armazenado.

. Para receber um array como argumento, a lista de parâmetros da função deve especificar que um array será recebido. Para um array unidimensional passado como parâmetro, não é exigido o tamanho do array entre colchetes.

. C++ fornece o qualificador de tipo const para evitar a modificação dos valores dos arrays em uma função. Quando um parâmetro array é precedido pelo qualificador const, os elementos do array se tornam constantes no corpo da função, e qualquer tentativa de modificar um elemento ali situado resulta em um erro durante a compilação.

. Um array pode ser ordenado usando o algoritmo de *bubble sort.* São feitas várias passagens pelo array. Em cada passagem, são comparados pares sucessivos de elementos. Se um par estiver em ordem (ou se os valores forem idênticos), tudo fica como estava. Se um par estiver fora de ordem, os valores são permutados. Para arrays pequenos, o *bubble sort é* aceitável, mas para arrays grandes ele é ineficiente comparado com outros algoritmos de classificação mais sofisticados.

. A pesquisa linear compara cada elemento do array com a chave de busca. Como o array não se encontra em nenhuma ordem determinada, as probabilidades de o elemento ser o primeiro ou o último são as mesmas. Na média, portanto, o programa precisará comparar a chave de busca com metade dos elementos do array. O método de pesquisa linear funciona bem com arrays pequenos e é aceitável para arrays que não estejam ordenados.

. A pesquisa binária elimina metade dos elementos do array que está sendo pesquisado após cada comparação. O algoritmo localiza o elemento no meio do array e o compara com a chave de busca. Se forem iguais, a chave de busca foi encontrada e o subscrito do array daquele elemento é retornado. Se não forem iguais, o problema é reduzido para a pesquisa em metade do array.

. Na pior hipótese, pesquisar um array de 1024 elementos, utilizando pesquisa binária, precisará de apenas lO comparações.

. Os arrays podem ser usados para representar tabelas de valores que consistem em informações organizadas em linhas e colunas. Para identificar um elemento específico de uma tabela, são especificados dois subscritos: o primeiro (por convenção) identifica a linha, e o segundo (por convenção) identifica a coluna na qual o elemento está contido. As tabelas ou arrays que exigem dois subscritos para identificar um determinado elemento são chamados de arrays bidimensionais.

. Quando recebemos um array de um único subscrito como argumento de uma função, os colchetes estão vazios na lista de parâmetros da função. O primeiro subscrito de um array multidimensional também não é exigido, mas todos os subscritos subseqüentes o são. O compilador usa esses subscritos para determinar, na memória, os locais dos elementos de um array multidimensional.

. Para passar uma linha de um array bidimensional para uma função que recebe um array de um subscrito, simplesmente passe o nome do array seguido do primeiro subscrito.

Terminologia

a[i *]* "erroporum"

a[i *][* j] escalar

área temporária para permuta de valores escalabilidade

array formato de tabela

array bidimensional inicializar um array

array tridimensional inicializador

array multidimensional nome de um array

array unidimensional número da posição

array *m* por *n* número mágico

lista de inicializadores de um array passagem do *bubble sort*

variável constante passagem por referência

bubble .sort passando arrays para funções

pesquisa linear em um array pesquisa binária em um array

caractere nulo (\O') pesquisar um array

chave de pesquisa "sair de" um array

colchetes [1 chamada por referência simulada

Terminologia de "Pensando em objetos"

Erros comuns de programação

4.1 É importante observar a diferença entre o "sétimo elemento do array" e o "elemento número sete do array". Como os subscritos dos arrays iniciam em O, o "sétimo elemento do array" tem subscrito 6, ao passo que o "elemento número sete do array" tem subscrito 7 e, na realidade, é o oitavo elemento do array. Esse fato é uma fonte de "erros por um".

4.2 Esquecer de inicializar os elementos de um array cujos elementos precisam ser inicializados é um erro de lógica.

4.3 Fornecer mais inicializadores para um array do que o número de seus elementos é um erro de sintaxe.

4.4 Atribuir um valor a uma variável constante em um comando executável é um erro de sintaxe.

4.5 Somente constantes devem ser usadas para declarar arrays automáticos e estáticos. Não usar uma constante para esta finalidade é um eno de sintaxe.

4.6 Fazer referência a um elemento além dos limites de um array é um erro de lógica que ocorre durante a execução. Não é um erro de sintaxe.

4.7 Embora seja possível usar a mesma variável contadora de um Jaço for em um segundo laço for aninhado no interior do primeiro,

isto é normalmente um erro de lógica.

4. Não fornecer a cm » um array de caracteres com comprimento suficiente para armazenar um *string* digitado no teclado pode resultar na perda de dados em um programa e em outros erros sérios durante a execução.

4.9 Supor que os elementos de um array local declarado static são inicializados com zeros sempre que a função na qual o array é declarado for chamada, pode levar a erros em um programa.

4. 1 O Esquecer que arrays são passados por referência e, portanto, podem ser modificados, pode resultar em um erro de lógica.

4. 1 1 Referenciar um elemento de array a [x 1 [y] . incorretamente, com subscritos como a [x .*y* **]** . Na realidade. a **x** .*y* **]** é tratado como a [y] porque C÷÷ calcula o valor da expressão x .*y* (contendo um operador vírgula) simplesmente como y (a última das expressões separadas por vírgulas).

Boas práticas de programação

4.1 Definir o tamanho do array como uma variável constante. em vez de se usar uma constante, toma os programas mais claros. Esta técnica é usada para se eliminar os chamados *números mágicos;* ou seja, mencionar repetidamente o número lo no código que processa um array de 10 elementos dá ao número 10 um significado artificial, podendo infelizmente confundir o leitor quando também existirem no programa outros números 1 O que nada têm a ver com o tamanho do array.

4.2 Procure obter clareza nos programas. Algumas vezes, vale a pena trocar um uso mais eficiente da memória e do processador por programas escritos com maior clareza.

4.3 Alguns programadores incluem nomes de variáveis em protótipos de funções para tomar os programas mais claros. O compilador ignora esses nomes.

qualificador de tipo const .*sinking sort*
classificar um array constante com nome declarar um array elemento de um array elemento de ordem zero

string
subscrito subscrito de coluna subscrito de linha tabela de valores valor de um elemento verificação dos limites

colaboração
comportamento
diagrama de seqüência
dividindo a linha de vida de um objeto
duração de atividade
execução condicional de atividades
fluxo de mensagens em diagrama de seqüência UML laço de simulação
linha de vida de um objeto em diagrama de seqüência UML linha sólida com seta na ponta em diagrama de seqüência mensagem

objeto cliente
objeto servidor
operação
serviço que um objeto presta
símbolo retângulo de ativação em diagrama de seqüência
UML
símbolo de devolução de mensagem em diagrama de seqüência UML
símbolo de objeto em um diagrama de seqüência UML tipo de retomo de uma operacão
verbos em uma definição de problema

Dicas de desempenho

4.1 Se, em vez de inicializar um array com comandos de atribuição durante a execução, você inicializar o array durante a compilação com uma lista de inicializadores de array, seu programa será executado mais rapidamente.

4.2 Algumas vezes, as considerações de desempenho são muito mais importantes do que as considerações de clareza.

4.3 Podemos aplicar static a uma declaração de um array local de modo que o array não seja criado e inicializado. cada vez que a função é chamada, e destruído cada vez que a execução deixa a função. Isto melhora o desempenho.

4.4 Faz sentido passar arrays através de chamadas por referência simuladas, por motivos de desempenho. Se os arrays fossem passados por chamadas por valor, seria passada uma cópia de cada elemento. Para arrays grandes, passados freqüentemente, isso seria demorado e consumiria um espaço considerável para armazenar as cópias dos arrays.

4.5 Freqüentemente, os algoritmos mais simples apresentam um desempenho muito ruim. Sua virtude reside no fato de que são fáceis de escrever, testar e depurar. Algumas vezes, porém, são necessários algoritmos mais complexos para se atingir o melhor desempenho possível.
4.6 Os tremendos ganhos em desempenho da pesquisa binária em relação à linear não são obtidos sem um preço. Classificar um array é uma operação cara, comparada com pesquisar todo um array em busca de um único item, uma vez só. O *overhead* de classificar um array passa a valer a pena quando o array necessitará ser pesquisado muitas vezes em alta velocidade.

Dica de portabilidade

4.1 Os efeitos (geralmente graves) da referência a elementos situados fora dos limites de um array dependem dos sistemas. Freqüentemente, isso resulta em alteração do valor de uma variável não-relacionada.

Observações de engenharia de software

4.1 Definir o tamanho de cada array com uma variável constante toma os programas mais flexíveis.
4.2 E possível passar um array por valor (usando um truque simples explicado no Capítulo 6) - isso raramente é feito.
4.3 O qualificador de tipo const pode ser aplicado a um parâmetro de um arras em uma definição de função para evitar que o array original seja modificado no corpo da função. Esse é outro exemplo do princípio do mínimo privilégio. As funções não devem ter a capacidade de modificar um array, a menos que isso seja absolutamente necessário.

Dicas de teste e depura ção

4.1 Ao usar um laço para percorrer um array, os subscritos nunca devem ser menores do que O e sempre devem ser menores do que o número total de elementos do array (um a menos que o tamanho do array). Certifique-se de que a condição de término do laço evita o acesso a elementos fora desses limites.
4.2 Os programas devem verificar a exatidão de todos os valores de entrada para evitar que informações erradas afetem os cálculos executados pelo programa.
4.3 Quando estudarmos classes (a partir do Capítulo 6), veremos como desenvolver um "array inteligente", que verifica automaticamente se os subscritos em todas as referências estão dentro dos limites durante a execução. O uso de tais tipos de dados inteligentes ajuda a eliminar erros.
4.4 Embora seja possível modificar um contador de laço no corpo de um for, evite fazer isso porque essa prática freqüentemente leva à ocorrência de erros sutis.

Exercícios de auto-revisão

4.1 Complete os espaços em branco:
a) Listas e tabelas de valores são armazenadas em _____
b) Os elementos de um array são relacionados entre si pelo fato de que possuem o mesmo ___ e ___
c O número usado para fazer referência a um elemento específico de um array é chamado de _____
d) Deve ser usado um(a) para declarar o tamanho de um array porque isso torna os programas mais flexíveis.
e) O processo de colocar em ordem os elementos de um array é chamado de ___ de um array.
f) O processo de determinar se um array contém um valor-chave específico é chamado de _____ o array.
g) Um array que usa dois subscritos é chamado um array

4.2 Diga se cada uma das sentenças a seguir é *verdadeira oufalsa.* Se a resposta *forfalsa,* explique o motivo.
a) Um array pode armazenar muitos tipos diferentes de valores.
b) Um subscrito de array pode ser do tipo de dado float.
c) Se houver menos inicializadores em uma lista do que o número de elementos no array, C++ inicializa automaticament,e os elementos restantes com o último valor da lista de inicializadores.
d) E um erro se uma lista de inicializadores possuir mais inicializadores do que o número de elementos do array.
e) Um elemento individual de um array que for passado a uma função e modificado pela função chamada manterá o valor modificado quando a função chamada terminar sua execução.
4.3 Responda às seguintes perguntas a respeito de um array chamado fracoes.
a) Defina uma variável constante tamanhoDoArray inicializada com 10.
b) Declare um array com um número de elementos tamanhoDoArray, do tipo double. e inicialize os elementos com o valor O.
e) Dê o nome do quarto elemento a partir do início do array.
d) Referencie o elemento 4 do array.
e) Atribua o valor 1 . 667 ao elemento nove do array.
h) Atribua o valor 3 . 333 ao sétimo elemento do array.
g) Imprima os elementos 6 e 9 do array com dois dígitos de precisão à direita da casa decimal e mostre a saída que é realmente apresentada na tela.
h) Imprima todos os elementos do array usando uma estrutura de repetição for. Defina a variável inteira x como uma variável de controle para o laço. Mostre a saída do programa.
4.4 Responda às seguintes perguntas a respeito de um array chamado tabela.
a) Declare o array como um array inteiro, com 3 linhas e 3 colunas. Considere que a variável constante tamarihoDoArray foi definida como 3.
b) Quantos elementos contém o array?
c) Use uma estrutura de repetição for para inicializar cada elemento do array com a soma de seus subscritos. Assuma que as variáveis

inteiras x e y são declaradas como variáveis de controle.
d) Imprima os valores de cada elemento do array tabela. Assuma que o array foi inicializado com a declaração
int tabela[tamanhoDoArray) [tamanhoDoArray]
{{ 1, 8 }, { 2, 4, 6 },{ 5) };
e as variáveis inteiras x e y são declaradas como variáveis de controle. Mostre a saída.
4.5 Encontre o erro em cada um dos segmentos de programa a seguir e o corrija.
a) #include <iostreain>;
b) tamanhoDoArray =10; II taiuanhoDoArray foi declarado const
c) Pressuponha que int b [10] ={ O };
for (int i =O; i <=10; i++
b[i] =1;
d) Pressuponhaque irit a[2][2] ={{ 1, 2 }, { 3, 4} };
a[1, 1] =5;

Respostas dos exercícios de auto-revisão

4.1 a) Arrays. b) Nome, tipo. c) Subscrito. d) Variável constante.
e) Classificação. t) Pesquisa. g) Bidimensional.
4.2 a) Falsa. Um array pode armazenar apenas valores do mesmo tipo.
b) Falsa. Um subscrito de array deve ser um inteiro ou uma expressão inteira.
c) Falsa. C++ inicializa automaticamente os elementos restantes com zeros.
d) Verdadeira.
e) Falsa. Os elementos individuais de um array são passados por meio de chamadas por valor. Se o array inteiro for passado a uma função, então quaisquer modificações se refletirão no original.
4.3 a) const int tamanhoDoArray =10;
b) double fracoes [tamanhoDoArray] ={ O };
c) fracoes[3]
d) fracoes[4

e) fracoes[9]1.667;
f) fracoes[6]3.333;
g) cout « setiosflags(ios::fixed 1 ios::showpoint
« setprecision(2) « fracoes[6 1«
«fracoes[9] « endi;
Saída:3.33 1.67.
h) for (int x O; x < tamanhoDoArray; x++
cout« "fracoes [" « x « "j = « fracoes [x
« eridi;
Saída:
fracoes[0] O
fracoes[1]0
fracoes[2] =0
fracoes[3 1 =0
fracoes[4] =0
fracoes[5 1 =0
fracoes[6] =3.333
fracoes[7] =0
fracoes[8) =0
fracoes[9] =1.667
4.4 a) int tabela [tamanhoDoArray] [tamarihoDoArray];
b) Nove elementos.
e) for (x =O; x < tamanhoDoArray; x++
for (y =O; y < tamanhoDoArray; y++
tabela[x][y] x + y;
d)cout « -[O] [1] [2]" « endl;
for (int x =O; x < tamanhoDoArray; x++){
cout « [' « x « "|
for (int y =O; y < tamanhoDoArray; y++
cout « setw(3) « tabela[x] [y] «

```
cout « endl;
```
Saída:
```
[0] [1] [2]
[0] 1 8 0
[1] 2 4 6
[2] 5 0 0
```
4.5 a) Erro: o ponto-e-vírgula no final da diretiva de pré-processador #include.
Correção: elimine o ponto-e-vírgula.
b) Erro: atribuir um valor a uma variável constante usando um comando de atribuição.
Correção: atribuir um valor a uma variável constante em uma declaração const int taxnanhoDoArray.
e) Erro: fazer referência a um elemento além dos limites do array (b [10 **]).**
Correção: mudar o valor final da variável de controle para 9.
d) Erro: uso incorreto de subscritos do array.
Correção: mudar o comando para a [1] [1] = 5;

Exercícios

4.6 Preencha as lacunas em cada uma das sentenças a seguir:
a) C++ armazena listas de valores em _____
b) Os elementos de um array estão relacionados entre si pelo fato de que eles_____
c) Ao se referir a um elemento de array, o número da posição contida entre colchetes é chamado de_____
d) Os nomes dos quatro elementos do array p são _____, _____ e _____
e) Denominar um array, declarar seu tipo e especificar seu número de elementos é chamado de_____ um array.

O o processo de colocar elementos de um array em ordem descendente ou ascendente é chamado de_____
g) Em um aiay bidimensional, o primeiro subscrito (por convenção) identifica a _____de um elemento e o segundo (por convenção) identifica a _____de um elemento.

4.7

falsas.

4.8

h) Um array *m* por *n* contém _____linhas, _____ colunas e_____ elementos.
i) O nome do elemento na linha 3 e coluna 5 do array d é _____

Diga quais das sentenças a seguir são verdadeiras e quais são falsas; explique os motivos pelos quais as sentenças são

a) Para se referir a uma posição ou um elemento em particular dentro de um array. especificamos o nome do array e o

valor daquele elemento.
b) Uma declaração de array reserva espaço para ele.
c) Para indicar que 1 00 posições devem ser reservadas para um array inteiro p. o programador deve escrever a declaração

p[100];

d) Um programa em C++ que inicializa todos os 15 elementos de um array com zeros deve conter um comando for.
e) Um programa em C++ que soma os elementos de um array de dois subscritos deve conter comandos for aninhados.

Escreva comandos em C++ que realizem o seguinte:

a) Exibam na tela o valor do sétimo elemento do array de caracteres f.
b) Leiam um valor para o elemento 4 de um array unidimensional de ponto flutuante b.
c) Inicializem cada um dos 5 elementos de um array unidimensional inteiro g com o valor 8.
d) Somem os elementos de um array de ponto flutuante de 100 elementos.
e) Copiem o array a para a primeira parte do array b. Assuma double a [11] . b [34]

O Determine e imprima o menor e o maior valor contidos em um array de ponto flutuante w com 99 elementos.

Considere um array inteiro, **t,** 2 por 5.
a) Escreva uma declaração para t.
b) Quantas linhas t possui?
c) Quantas colunas t possui?
d) Quantos elementos t possui?
e) Escreva os nomes de todos os elementos da segunda linha de t.
f) Escreva os nomes de todos os elementos da terceira coluna de t.
g) Escreva um único comando que defina como zero o elemento da linha 1 e coluna 2 de t.
h) Escreva uma série de comandos que inicialize como zero cada elemento de t. Não use um laço.
i) Escreva uma estrutura for aninhada que inicialize cada elemento de t como zero.
i) Escreva um comando que entre com os valores dos elementos de t a partir do terminal.
k) Escreva uma série de comandos que determine e imprima o menor valor do array t.
1) Escreva um comando que exiba na tela os elementos da primeira linha de t.
m) Escreva um comando que some os elementos da quarta coluna de t.
n) Escreva uma série de comandos que imprima o array t em um formato organizado de tabela. Liste os subscritos das colunas como cabeçalho ao longo do topo e liste os subscritos das linhas à esquerda de cada linha.

4.10 Use um array unidimensional para resolver o seguinte problema. Uma empresa paga seus vendedores com base em comissões. O vendedor recebe $200 por semana mais 9 por cento de suas vendas brutas daquela semana. Por exemplo, um vendedor que teve vendas brutas de $3000 em uma semana recebe $200 mais 9 por cento de $3000, ou seja. um total de $470. Escreva um programa (usando um array de contadores) que determine quantos vendedores receberam salários nos seguintes intervalos de valores (considere que o salário de cada vendedor é truncado para que seja obtido um valor inteiro):
1) $200-$299
2) $300-$399
3) $400- $499
4) $500-$599
5) $600-$699
6) $700-$799
7) $800-$899
8) $900 -$999
9) $ 1000 em diante

4. 1 1 A classificação com o *huhhle sort* apresentada na Fig. 4. 1 6 é ineficiente para arrays grandes. Faça as modificações simples a seguir para melhorar o desempenho do *bubble sort.*
a) Depois da primeira passagem, garante-se que o maior número esteja no elemento de maior número do array; depois da segunda passagem, os dois maiores números estão em "seus lugares", e assim por diante. Em vez de fazer nove comparações em cada passagem, modifique o *bubble sort* para fazer oito comparações na segunda passagem, sete na terceira e assim por diante.

4.9

r

b) Os dados do array podem estar na ordem adequada ou quase ordenados completamente; assim, por que fazer nove . passagens se um número menor seria suficiente? Modifique a classificação para verificar se alguma permuta foi feita ao final de cada passagem. Se nenhuma permuta foi realizada, os dados já devem estar na ordem adequada e o programa deve ser encerrado. Se foram feitas permutas, é necessária pelo menos mais uma passagem .
4.12 Escreva comandos simples que realizem cada uma das operações seguintes em um array unidimensional:

a) Inicialize com zeros os lO elementos do array inteiro contagem.
b) Adicione 1 a cada um dos 15 elementos do array inteiro bonus.
c) Leia os 12 valores do array de ponto flutuante double temperaturasMensais a partir do teclado.
d) Imprima os 5 valores do array inteiro melhoresResultados em um formato de coluna.
4.13 Encontre o(s) erro(s) em cada um dos seguintes comandos:
a) Assumaque: char str[5] ;

```
cm » str; // Usuário digita alô
```

b) Assuma que: int a [3]

```
cout « a[ 1 ] « ' " « a[ 2 ] « « a[ 3 ] « endi;
c) double f[ 3 ] = { 1.1, 10.01, 100.001, 1000.0001 };
```

d) Assumaque: double d[2][10]

```
d[ 1, 9 ] = 2.345;
```

,/ 4. 14 Modifique o programa da Fig. 4. 1 7 de forma que a função mode seja capaz de manipular um empate no valor da moda. ModiÍique também a função median de forma que seja encontrada a média dos dois valores do meio de um array com número par de elementos.
4.15 Use um array unidimensional para resolver o seguinte problema. Leia 20 números, todos situados entre 10 e 100, inclusive. A medida que cada número for lido, imprima-o somente se não for duplicata de um número já lido. Experimente o "pior caso", no qual todos os 20 números são diferentes. Use o menor array possível para resolver esse problema.
4.16 Classifique os elementos do array vendas bidimensional 3 por 5 para indicar a ordem na qual eles são definidos como zero pelo seguinte segmento de programa:

```
for ( linha =0; linha < 3; linha++
for ( coluna 0; coluna < 5; coluna++
vendas[ linha ][ coluna ] = 0;
```

4.17 Escreva um programa que simule ajogada de dois dados. O programa deve usar rand para lançar o primeiro dado e deve usar rand novamente para lançar o segundo dado. A soma dos dois valores deve então ser calculada. *Nota:* como cada dado pode mostrar um valor inteiro de 1 a 6, a soma dos dois valores variará de 2 a 12, com 7 sendo a soma mais freqüente e 2 e 12 sendo as somas menos freqüentes. A Fig. 4.24 mostra as 36 combinações possíveis dos dois dados. Seu programa deve lançar os dados 34.000 vezes. Use um array unidimensional para registrar o número de vezes que é obtida cada soma possível. Imprima os resultados em formato de tabela. Além disso, determine se as somas são razoáveis, i.e., há seis maneiras de obter 7, portanto, aproximadamente um sexto do total de jogadas deve ser 7.
1 2 3 4 5 6

2
3
4
5
6

Fig. 4.28 As 36 combinações possíveis lançando-se dois dados.

2	3	4	5	6	7
3	4	5	6	**7**	**8**
4	5	6	7	**8**	**9**
5	**6**	**7**	**8**	**9**	**10**
6	7	8	9	10	11
7	8	9	10	11	12

4.IX O que faz o seguinte programa?

```
1 II ex0418.cpp
2 #include <iostream>
3
4 using std: :cout;
5 using std: :endl;
6
7 int whatlsThis( int [],int );
8
```

```
9 int main
lo {
11 const int arraySize =10;
12 inta[arraySize]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
13
14 int result =whatlsThis( a, arraySize );
15
16 cout « "O resultado é « result « endl;
17 return 0;
18 }
19
20 int whatlsThis( int b[J, int size
21 {
22 if ( size=1 )
23 return b[ O ];
24 else
25 return b[ size -1 |+ whatlsThis( b, size -1 );
26 }
```

4.19 Escreva um programa que rode 1000 jogos de *craps* e responda a cada uma das seguintes perguntas:

a) Quantos jogos são vencidos na I jogada, 2 jogada **,..,** 2O jogada e depois da 2O jogada?

b) Quantos jogos são perdidos na i' jogada, 2 jogada 2O jogada e depois da 2O jogada?

e) Quais as probabilidades de se vencer no jogo de *craps? (Nota:* você deve descobrir que *craps é* um dos jogos de cassino mais honestos. O que você acha que isso significa?)

d) Qual a duração média de um jogo de *craps?*

e) As probabilidades de se vencer aumentam com a duração do jogo?

4.20 *(Sistema de reservas de companhias aéreas)* Uma pequena companhia aérea acabou de comprar um computador para o seu novo sistema automático de reservas. Pediram a você que programasse o novo sistema. Você deve escrever um programa para atribuir assentos a cada vôo do único avião da companhia (capacidade: 10 assentos).

Seu programa deve exibir o seguinte menu de alternativas: - Favor digitar 1 para "Primeira Classe" e Favor digitar 2 para "Classe Econômica". Se a pessoa digitar 1, seu programa deve fazer a reserva de um assento na primeira classe (assentos 1 -5). Se a pessoa digitar 2, seu programa deve reservar um assento na classe econômica (assentos 6- **10).** Seu programa deve então imprimir um cartão de embarque indicando o número do assento do passageiro e se ele se encontra na primeira classe ou na classe econômica do avião.

Use um array unidimensional para representaro esquema dos assentos do avião. Inicialize todos os elementos do array com O para indicar que todos os assentos estão livres. A medida que cada assento for reservado, iguale os elementos correspondentes a 1 para indicar que o assento não está mais disponível.

Seu programa nunca deve, obviamente, reservar um assento que **já** tenha sido reservado. Quando a classe econômica estiver lotada, seu programa deve perguntar se a pessoa aceita um lugar na primeira classe (e vice-versa). Em caso positivo, faça a reserva do assento apropriada. Em caso negativo, imprima a mensagem "Próximo vôo sai em 3 horas."

4.21 O que faz o seguinte programa?

```
1 II exO4_21.cpp
2 #include <iostream>
```



```
7
3
4 using std: :cout;
5 using std::endl;
6
7 void someFunction( int [] .int );
8
1 int main O
lo {
1- 11 const int arraySize =10;
12 int a[ arraySize 1 =
13 { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
14
15 cout « "Os valores no array são:" « endl;
16 someFunction( a, arraySize );
17 cout « endl;
18 return 0;
19
20
;i 21 void someFunction( int b[], int size
22 {
```

```
23 if(size>0) {
24 someFunction( &b[ 1 ], size - 1 )
25 cout « b[ 0 ] «
26 }
27 }
```

4.22 Use um array bidimensional para resolver o seguinte problema. Uma empresa tem quatro vendedores (1 a 4) que vendem 5 produtos diferentes (1 a *5).* Uma vez por dia, cada vendedor elabora um memorando de cada tipo diferente de produto vendido. Cada memorando contém o seguinte:
a) O número do vendedor
b) O número do produto
dos jogos de e) O valor total, em dólares, daquele produto vendido naquele dia
Dessa forma, cada vendedor elabora de O a *5* memorandos de vendas por dia. Admita que as informações de todos os memorandos do último mês estão disponíveis. Escreva um programa que leia todas essas informações de vendas do último mês e faça um resumo do total de vendas por vendedor e por produto. Todos os totais devem ser armazenados no array bidimensional vendas. Depois de itador para o processar todas as informações do último mês, imprima os resultados em um formato de tabela, com cada coluna representando um ograma para vendedor diferente e cada linha representando um produto diferente. Faça a soma dos valores de cada linha para obter as vendas
totais de cada produto no último mês; faça a soma dos valores em cada coluna para obter as vendas totais de cada vendedor no último 1asse' e mês. A tabela impressa deve incluir esses totais à direita das linhas somadas e abaixo das colunas somadas.
e um assento
(assentos 6- 4.23 *(Gráficos de tartaruga)* A linguagem Logo, particularmente popular entre os usuários de computadores pessoais, tornou
se encontra famoso o conceito dos *gráficos de tartaruga.* Imagine uma tartaruga mecânica que percorra uma sala sob o controle de um programa em C++. A tartaruga segura uma caneta em uma de duas posições, para cima ou para baixo. Enquanto a caneta está para nos do array baixo, a tartaruga desenha figuras à medida que se move; quando a caneta está para cima, a tartaruga se move livremente, sem s correspon- desenhar nada. Neste problema, você simulará a operação da tartaruga e criará também um bloco de desenho computadorizado.
Use um array piso, 20 por 20, inicializado com zeros. Leia os comandos de um array que os contém. Controle sempre econômica a posição atual da tartaruga e se a caneta está para cima ou para baixo. Admita que a tartaruga sempre começa na posição 0,0 do ositivo, faça com a caneta para cima. O conjunto de comandos da tartaruga que seu programa deve processar são os seguintes:

1 Comando Significado

1 Caneta para cima
2 Caneta para baixo
3 Giro para a direita
4 Giro para a esquerda

Movimento 10 espaços para a frente

(ou outro número diferente de 10) Imprime o array 20 por 20

Fim dos dados (sentinela)

Suponha que a tartaruga esteja em algum lugar próximo do centro do piso. O "programa" a seguir desenharia e imprimiria um quadrado 1 2-por- 12:

2

5,12
3

5,12

3

5,12

3

5,12

1

6

9

À medida que a tartaruga se move com a caneta para baixo, defina os elementos apropriados do array piso como is. Quando o comando 6 (imprimir) for emitido, onde houver um 1 no array exiba um asterisco, ou outro caractere de sua escolha. Sempre que houver um zero, exiba um espaço em branco. Escreva um programa para implementar os recursos do gráfico de tartaruga descritos aqui. Escreva vários programas de gráficos de tartaruga para desenhar formas interessantes. Adicione outros comandos para aumentar a potencialidade de sua linguagem de gráfico de tartaruga.

4.24 *(Passeio do cavalo)* Um dos problemas mais interessantes para os fs do jogo de xadrez é o problema do Passeio do Cavalo, proposto originalmente pelo matemático Euler. A questão é esta: a peça do jogo de xadrez chamada de cavalo pode se mover ao longo de um tabuleiro vazio e passar uma única vez em cada uma das 64 casas? Estudamos aqui esse interessante problema em profundidade.

O cavalo faz movimentos em forma de L (percorre duas casas em uma direção e uma no sentido perpendicular às duas primeiras). Dessa forma, de uma casa no meio do tabuleiro, o cavalo pode fazer oito movimentos diferentes (numerados de O a 7), como mostra a Fig. 4.29.

O

1

2

3

4

5

6

7

O **i 2 3 4 5 6 7**

Comando Significado

5,10
6
9

Fig. 4.29 Os 8 movimentos possíveis do cavalo.

a) Desenhe um tabuleiro de xadrez 8 por 8 em uma folha de papel e tente fazer o Passeio do Cavalo a mão. Coloque um 1 na primeira casa para a qual se mover, um 2 na segunda casa, um 3 na terceira, etc. Antes de começar os movimentos, imagine até onde você chegará, lembrando-se de que um passeio completo consiste em 64 movimentos. Até onde você chegou? Chegou tão perto quanto havia imaginado?
b) Agora, vamos desenvolver um programa que moverá o cavalo pelo tabuleiro de xadrez. O tabuleiro, em si, é representado pelo array bidimensional. 8 por 8. tabuleiro. Cada um dos quadrados (casas é inicializado com o valor zero. Descrevemos cada um dos oito movimentos possíveis em termos de seus componentes horizontais e verticais. Por exemplo, um movimento do tipo O, como mostra a Fig. 4.24, consiste em mover dois quadrados horizontalmente para a direita e um quadrado verticalmente para cima. O movimento 2 consiste em mover um quadrado horizontalmente para a esquerda e dois quadrados verticalmente para cima. Os movimentos horizontais para a esquerda e verticais para cima são indicados por números negativos. Os oito movimentos podem ser descritos por arrays bi-dimensionais, **horizontal e vertical. como segue:**

```
horizontal[ O ] 2
horizontal[ 1 ] 1
horizontal[ 2 J -1
horizontal[ 3 ] = -2
horizontal[ 4 ] = -2
horizontal[ 5 ] -1
horizontal[ 6 ] 1
horizontal[ 7 ] 2
vertical[ O J = -1
vertical[ 1 ] = -2
vertical[ 2 ] = -2
vertical[ 3 ] -1
vertical[ 4 ] = 1
e vertical[ 5 ] 2
vertical[ 6 ] = 2
vertical[ 7 ] =1
```

o As variáveis linhaAtual e colunaAtual indicam a linha e a coluna da posição atual do cavalo. Para fazer um e movimento do tipo numMov. em que numMov está entre O e 7, seu programa usa os comandos

```
linhaAtual + vertical[ numMov ];
colunaAtual + horizontal [ nunMov );
```

Utilize um contador que varie de 1 a 64. Grave a última contagem em cada casa para a qual o cavalo se mover. Lembre-se de testar cada movimento potencial para verificar se o cavalojá esteve naquele quadrado. E, obviamente, teste cada movimento potencial para se certificar de que o cavalo não esteja saindo do tabuleiro. Agora, escreva um programa para mover o cavalo pelo tabuleiro de xadrez. Rode o programa. Quantos movimentos o cavalo fez?
c) Depois de tentar escrever e rodar o programa do Passeio do Cavalo, provavelmente você adquiriu alguns conceitos valiosos. Usaremos esses conceitos para desenvolver uma *heurística* (ou estratégia) para mover o cavalo. A heurística não garante o sucesso, mas uma heurística cuidadosamente desenvolvida aumenta as probabilidades de ele ser atingido. Você pode ter observado que os quadrados extemos são, de alguma forma, mais problemáticos do que os quadrados próximos ao centro do tabuleiro. Na realidade, os quadrados mais problemáticos, ou inacessíveis, são os quatro cantos.
A intuição pode sugerir que você deve tentar mover o cavalo para os quadrados mais problemáticos e deixar livres os quadrados mais fáceis de atingir para que, quando o tabuleiro ficar congestionado próximo ao final do passeio, haja a maior probabilidade de sucesso.
Podemos desenvolver uma "heurística de acessibilidade", classificando cada um dos quadrados de acordo com sua acessibilidade e depois sempre mover o cavalo para o quadrado (dentro dos movimentos em forma de L, obviamente) que seja mais inacessível. Colocamos no array bidimensional acessibilidade os números que indicam a partir de quantos quadrados um determinado quadrado pode ser acessado. Portanto, em um tabuleiro vazio, as casas centrais são classificadas com 8s, os quadrados dos cantos são classificados com 2s e os outros quadrados possuem números de acessibilidade 3, 4 e 6. como é mostrado a seguir

```
2 3 4 4 4 4 3 2
3 4 6 6 6 6 4 3
4 6 8 8 8 8 6 4
4 6 8 8 8 8 6 4
4 6 8 8 8 8 6 4
4 6 8 8 8 8 6 4
3 4 6 6 6 6 4 3


2 3 4 4 4 4 3 2
```

Agora escreva uma versão do programa do Passeio do Cavalo usando a heurística de acessibilidade. Em qualquer tempo, o cavalo deve se mover para o quadrado com menor número de acessibilidade. Em caso de empate, o cavalo pode se mover para qualquer um dos quadrados que empataram. Portanto, o passeio pode começar em qualquer um dos quatro cantos. *(Nota: à* medida que o cavalo se mover no tabuleiro, seu programa deve reduzir os números de acessibilidade quando cada vez mais quadrados forem ocupados. Dessa forma, a qualquer instante durante o passeio o número de acessibilidade de cada quadrado disponível permanecerá igual ao número de quadrados a partir dos quais aquele espaço pode ser alcançado). Rode essa versão de seu programa. Você conseguiu fazer um passeio com- pleto?Agora modifique o programa para executar 64 passeios, um para cada quadrado do tabuleiro. Quantos passeios completos você conseguiu fazer?

d) Escreva uma versão do programa do Passeio do Cavalo na qual, ao encontrar um empate entre dois ou mais quadra- dos, você faça sua escolha verificando os quadrados que podem ser alcançados a partir dos quadrados "empatados". Seu programa deve fazer o movimento do cavalo para o quadrado para o qual o próximo movimento levar ao quadra- do com menor número de acessibilidade.

4.25 *(Passeio do Cavalo: métodos deforça bruta)* No Exercício 4.24, desenvolvemos uma solução para o problema do Passeio do Cavalo. O método usado, chamado de "heurística de acessibilidade", gera muitas soluções e funciona de uma maneira eficiente. Como os computadores continuam a ser cada vez mais poderosos, poderemos resolver muitos problemas com base apenas no poder computacional e com algoritmos relativamente simples. Vamos chamar esse método de resolução de problemas de "força bruta".

a) Use a geração de números aleatórios para permitir que o cavalo percorra o tabuleiro (em seus movimentos permitidos na forma de L, obviamente) de maneira aleatória. Seu programa deve executar um passeio e imprimir o tabuleiro final. Até onde o cavalo chegou?

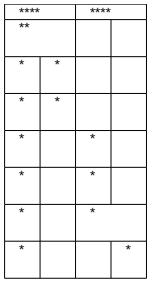
b) Muito provavelmente, o programa anterior levará a um passeio relativamente curto. Agora, modifique seu programa para tentar 1 000 passeios. Use um array unidimensional para controlar o número de passeias por extensão alcançada. Quando seu programa terminar de fazer os 1000 passeios, ele deve imprimir essas informações em um formato de tabela. Qual o melhor resultado?

e) Muito provavelmente, o programa anterior forneceu alguns passeios "respeitáveis", mas nenhum passeio completo. Agora, "retire os limites" e simplesmente deixe seu programa ser executado até que um passeio completo seja produzido. *(Cuidado:* essa versão do programa pode rodar durante horas em um computador poderoso). Mais uma vez, mantenha uma tabela com o número de passeios para cada extensão alcançada e imprima essa tabela quando o primeiro passeio completo for realizado. Quantos passeias seu programa tentou fazer antes de produzir um passeio completo? Quanto tempo levou?

d) Compare a versão de força bruta do Passeio do Cavalo com a versão de heurística de acessibilidade. Qual exigiu um estudo mais cuidadoso do problema? Que algoritmo foi mais difícil de desenvolver? Qual exigiu mais poder computacional? Poderíamos estar certos (antecipadameme) de obter um passeio completo com a heurística de acessibilidade? Poderíamos estar certos (antecipadamente) de obter um passeio completo com o método da força bruta? Analise as vantagens e desvantagens da resolução de problemas em geral por força bruta.

4.26 *(Oito Rainhas)* Outro quebra-cabeça para os fãs do jogo de xadrez é o problema das Oito Rainhas. Resumidamente: é possível colocar oito rainhas em um tabuleiro vazio de xadrez, de forma que nenhuma rainha "ataque" qualquer uma das outras, isto é, de forma que não haja duas rainhas na mesma linha, na mesma coluna ou ao longo da mesma diagonal? Use o método de raciocínio desenvolvido no Exercício 4.24 para formular uma heurística para resolver o problema das Oito Rainhas. Execute seu programa. *(Dica:* é possível atribuir um valor numérico a cada quadrado do tabuleiro indicando quantos quadrados são "eliminados" se uma rainha for colocada. Por exemplo, cada um dos quatro cantos receberia o valor 22, como na Fig. 4.30.) Depois desses "números de eliminações" serem colocados em todas as 64 casas, uma heurística apropriada poderia ser: coloque a próxima rainha no quadrado com menor número de eliminações. Por que essa estratégia é intuitivamente atraente?

Fig. 4.30 As 22 casas eliminadas ao se colocar uma rainha no canto superior esquerdo do tabuleiro.



Bubble Sort - para um array de *n* elementos, devem ser feitas *n* - 1 passagens, e para cada subarray devem ser feitas *n* - 1 comparações para encontrar o menor valor. Quando o subarray a ser processado possuir um elemento, o array estará ordenado. Escreva uma função recursiva selectionSort para implementar esse algoritmo.

4.32 *(Palíndromos)* Um palíndromo é um string que é soletrado da mesma forma da frente para trás ou de trás para a frente. Alguns exemplos de palíndromos são "radar" "orava o avaro" e "socorram marrocos". Escreva uma função recursiva testePalind que retorne 1

se o string armazenado no array for um palíndromo ou O em caso contrário. A função deve ignorar espaços e pontuação no string.
4.33 *(Pesquisa linear)* Modifique o programa da Fig. 4. 1 9 para usar uma função recursiva buscaLinear para realizar a pesquisa linear no array. A função deve receber um array inteiro e o tamanho do array como argumentos. Se a chave de pesquisa for encontrada, retome o subscrito do array; caso contrário, retorne -1.
4.34 *(Pesquisa binária)* Modifique o programa da Fig. 4.20 para usar uma função recursiva buscaBinaria para realizar a pesquisa binária no array. A função deve receber um array inteiro e os subscritos inicial e final como argumentos. Se a chave de pesquisa for encontrada, retome o subscrito do array; caso contrário, retome - 1.
4.35 *(Oito Rainhas)* Modifique o programa Oito Rainhas criado no Exercício 4.26 para que o probkma seja resolvido recursivamente.
4.36 *(Imprimir um array)* Escreva uma função recursiva imprimeArray que tome um array e seu tamanho como argumentos, imprima o array e não retome nada. A função deve parar o processamento e retomar quando receber um array de tamanho zero.
4.37 *(imprimir um* string *de trás para a frente)* Escreva uma função recursiva inverteStririg que tome um array de caracteres como argumento, imprima o array de caracteres na ordem inversa e nada retorne. A função deve parar o processamento e retornar quando for encontrado um caractere nulo de término de *string.*
4.38 *(Encontrar o valor mínimo de um array)* Escreva uma função recursiva minimoArrayRecursiva que tome um array inteiro e o tamanho do array como argumentos e retorne o menor elemento do array. A função deve parar o processamento e retornar quando receber um array de 1 elemento.

5

Ponteiros e *strings*

Objetivos

• Ser capaz de usar ponteiros.
• Ser capaz de usar ponteiros para passar argumentos para funções através de chamadas por referência.
• Entender as relações próximas entre ponteiros, arrays e *strings.*
• Entender o uso de ponteiros para funções.
• Ser capaz de declarar e usar arrays de *strings.*

Os endereços nos são dados para ocultarmos nossos paradeiros.
Saki (H. H. Munro)

Através de indireções, encontre as direções.
William Shakespeare, Hamlet

Muitas coisas, estando em total anuência,
podem funcionar de forma contrária.
William Shakespeare, King Henry V

Você sempre pode encontrar uma prática muito boa para verificar suas referências, senhor!
Dr. Routh

Você não pode confiar num código que você mesmo não criou totalmente. (Especialmente um código feito por empresas que empregam pessoas como eu).
Ken Thompson, 1983 Turing Award Lecture Association for Computing Machinery, mc.

Visão Geral

5.1 Introdução
5.2 Declarações e inicialização de variáveis ponteiro
5.3 Operadores sobre ponteiros
5.4 Chamando funções por referência
5.5 Usando o qualificador const **com ponteiros**
5.6 *Bubbie Sort* usando chamada por referência
57 **Expressões com ponteiros e aritmética de ponteiros**
5.8 A relação entre ponteiros e arrays
5.9 Arrays de ponteiros
5.10 Estudo de caso: uma simulação de embaralhamento e distribuição de cartas
5.11 Ponteiros de função
5.12 Introdução ao processamento de caracteres e *strings*

5.12.1 Fundamentos de caracteres e *strings*
5.12.2 Funções de manipulação de *strings* da biblioteca de tratamento de *strings*
5.13 (Estudo de caso opcional) Pensando em objetos: colaborações entre objetos

Resumo• Terminologia Erros comuns de programação Boas práticas de programação Dicas de desempenho . Dicas de portabilidade. Observações de engenharia de software • Dica de teste e depura ção. Exercícios de auto-revisão . Respostas aos exercícios de auto-revisão • Exercícios Seção especial: construindo seu próprio computador• Mais exercícios sobre ponteiros Exercícios de manipulação de strings• *Seção especial: exercícios avançados de manipulação de* strings. *Um projeto desafiador de manipulação de* strings

5.1 Introdução

Neste capítulo, discutimos uma das características mais poderosas da linguagem de programação C++, o ponteiro. Os ponteiros estão entre os recursos mais difíceis de serem dominados de C++. No Capítulo 3, vimos que teferências podem ser usadas para executar chamadas por referência. Os ponteiros possibilitam aos programas simular chamadas por referência e criar e manipular estruturas de dados dinâmicas, i.e., estruturas de dados que podem crescer e encolher, tais como listas encadeadas, filas, pilhas e árvores. Este capítulo explica conceitos básicos sobre ponteiros. Reforça também a relação íntima entre arrays, ponteiros e *strings,* e inclui uma boa relação de exercícios de processamento de *strings.*

O Capítulo 6 examina o uso de ponteiros com estruturas. Nos Capítulos 9 e 10, veremos que a programação orientada a objetos é executada com ponteiros e referências. O Capítulo 15 introduz técnicas dinâmicas de administração de memória e apresenta exemplos de criação e uso de estruturas de dados dinâmicas.

A visão de arrays e strings como ponteiros é derivada de C. Mais à frente no livro, discutiremos arrays e *strings* como objetos no sentido pleno.

5.2 Declarações e inicialização de variáveis ponteiro

As variáveis ponteiro contêm endereços de memória como seus valores. Normalmente. uma variável contém diretamente um valor específico. Um ponteiro, por outro lado, contém um endereço de uma variável que contém um valor específico. Neste sentido, um nome de variável referencia *diretamente* um valor e um ponteiro referencia *indiretamente* um valor (Fig. *5.* 1). Referenciar um valor através de um ponteiro é chamado de *indireção.*

count count diretamente
referencia uma
variável cujo
valor é 7
countPtr **count countPtr** indiretamente
referencia uma
variável cujo
___ valor é 7

Fig. 5.1 Referenciando direta e indiretamente uma variável.

Ponteiros, como quaisquer outras variáveis, devem ser declarados antes de poderem ser usados. A declaraçao

```
int *countPtr, count;
```

declara a variável **countPtr como** sendo do tipo int * (i.e., **um** ponteiro para **um** valor do tipo inteiro) e é lida como "countptr é um ponteiro para int" ou "countptr aponta para um objeto do tipo inteiro". Além disso, a variável count é declarada como inteira e não como um ponteiro para um inteiro. O * se aplica somente a countPtr na declaração acima. Cada variável que está sendo declarada como um ponteiro deve ser precedida por um asterisco **(*).** Por exemplo, a declaração

double ***xptr, *yptr;**

indica que tanto xPtr como yPtr são ponteiros para valores de tipo double. Quando o * é usado desta maneira em uma declaração, ele indica que a variável que está sendo declarada é um ponteiro. Podemos declarar ponteiros para apontar para objetos de qualquer tipo.

Erro comum de programação 5.1

*Assumir que o * usado para declarar um ponteiro se aplica a todos os nomes de variáveis, em uma lista de variáveis ponteiro separadas por vírgulas, pode fazer com que os ponteiros sejam declarados não como ponteiros. Cada ponteiro deve ser declarado com o * prefixado ao nome.*

***Boa prática** de programação 5.1*

Embora não seja obrigatório fazer isso, incluir as letras **Ptr** *em nomes de variáveis ponteiro torna claro que estas variáveis são ponteiros e precisam ser manipuladas de acordo.*

Os ponteiros deveriam ser inicializados ou quando forem declarados ou em um comando de atribuição. Um ponteiro pode ser inicializado como O, NULL ou um endereço. Um ponteiro com o valor O ou NULL não aponta para nada. NULL é uma constante simbólica definida no arquivo de cabeçalho <iostream> (e em

vários arquivos de cabeçalho da biblioteca padrão). Inicializar um ponteiro com NTJLL é equivalente a inicializar um ponteiro com O. mas, em C++, é preferível usar O. Quando é usado O, ele é convertido em um ponteiro do tipo apropriado. O valor O é o único valor inteiro que pode ser atribuído diretamente a uma variável ponteiro sem fazer primeiro a coerção do inteiro para um tipo de ponteiro. A atribuição do endereço de uma variável para um ponteiro é discutida na Seção 5.3.

Dica de tes(e e depura ção 5.1

Sempre inicialize ponteiros para evitar apontar para áreas de memória desconhecidas ou não-inicializadas.

5.3 Operadores sobre ponteiros

O **&,** ou *operador de endereço,* é um operador unário que retorna o endereço de seu operando. Por exemplo, assumindo as declarações

int y = 5;

int ***yptr;**

o comando

yPtr =

atribui o endereço da variável y à variável ponteiro yPtr. Diz-se, então, que a variável yPtr "aponta para" y. A Fig. 5.2 mostra uma representação esquemática da memória depois que a atribuição precedente foi executada. Na figura, mostramos a "relação de apontar" desenhando uma seta que vai do ponteiro até o objeto por ele apontado.

Fig. 5.2 Representação gráfica de um ponteiro que aponta para uma variável inteira na memória.

A Fig. 5.3 mostra a representação do ponteiro na memória, assumindo que a variável de tipo inteiro y está armazenada na posição 600000 e a variável ponteiro yPtr está armazenada na posição 500000. O operando do operador de endereço deve ser um *ivalue* (i.e., algo ao qual um valor pode ser atribuído, tal como um nome de variável); o operador de endereço não pode ser aplicado para constantes, para expressões que não resultam em referências ou para variáveis declaradas com a classe de armazenamento register.

Fig. 5.3 Representação de y e yptr na memória.

O *operador* ***,** comumente chamado de *operador de indireção* ou *operador de derreferência,* retoma um sinônimo, um nome alternativo ou um apelido para o objeto para qual seu operando (i.e., um ponteiro) aponta. Por exemplo (fazendo referência à Fig. 5.2 novamente), o comando

cout *«* ***yptr** «* eridi;

imprime o valor da variável y, isto é, 5, quase do mesmo modo como o comando

cout *«* y *«* endJ.;

faria. Usar * desta maneira é chamado de *derreferenciar um ponteiro.* Note que um ponteiro derreferenciado também pode ser usado do lado esquerdo de um comando de atribuição, como em

***yptr** = 9;

yptr	y	y
5000001	600000	
1		
6000001		

que atribuiria 9 para y, na Fig. 5.3. O ponteiro derreferenciado pode ser também usado para receber um valor fornecido como entrada, como em

cm » *ypt

O ponteiro derreferenciado é um *Ivalue* ou "valor à esquerda".

Erro comum de programação 5.2

Derreferenciar um ponteiro que não foi corretamente inicializado, ou ao qual não foifeita uma atribuição para apontar para uma posição específica na memória, pode causar um erro fatal durante a execução ou, ainda, pode modificar acidentalmente dados importantes e permitir que o programa execute até o fim, fornecendo resultados incorretos.

Erro comum de programação 5.3

Uma tentativa de derreferenciar um não-ponteiro é um erro de sintaxe.
Erro comum de programação 5.4
Derreferenciar um ponteiro O *normalmente gera um erro fatal durante a execução.*
O programa na Fig. 5.4 demonstra o uso dos operadores sobre ponteiros. As posições de memória são exibidas neste exemplo, pelo operador «, como inteiros hexadecimais (ver o apêndice C, "Sistemas de numeração", para maiores informações sobre inteiros hexadecimais).

```
1 II Fig. 5.4: figO5O4.cpp
2 // Usando os operadores & e *
3 #include <iostreaxn>
4
5 using std::cout;
6 using std: :endl;
7
8 int main()
10 int a; // a é um inteiro
11 int *aPtr; II aPtr é um ponteiro para um inteiro
12
13 a=7;
14 aPtr =&a; // aPtr recebe o endereço de a
15
16 cout « "O endereço de a é -« &a
17 « "\nQ valor de aPtr é -« aPtr;
18
19 cout « "\n\nO valor de a é -« a
20 « "\n0 valor de *aptr é -« *p
21
22 cout « "\n\riMostrando que *e & são os inversos
23 « "um do outro.\n&*aPtr =" « &*aptr
24 « '\*p =" « *&aptr « endl;
25 return 0;
26 }
```

Fig. 5.4 Os operadores de ponteiro & e * (parte 1 de 2).


```
O endereço de a é OxOO6FDF4
O valor de aPtr é OxOO64FDF4
O valor de a é 7
O valor de *aPtr é 7
Mostrando que *e & são os inversos um do outro.
&*aPtr =OxOO64FDF4
*&aptr =OxOO64FDF4
```

Fig. 5.4 Os operadores de ponteiro & e * (parte 2 de 2).

Dica de portabilidade 5.1
___Oformato em que um ponteiro é exibido para o usuário depende da máquina. Alguns valores de ponteiros são exibidos por alguns sistemas como inteiros hexadecimais, enquanto outros sistemas usam inteiros decimais.
Note que o endereço de a e o valor de aPtr são idênticos na saída, confirmando que o endereço de a é realmente atribuído à variável ponteiro aptr. Os operadores **&** e * são inversos um do outro - quando ambos forem aplicados consecutivamente a aPtr, em qualquer ordem, será impresso o mesmo resultado. A Fig. 5.5 mostra a precedência e a associatividade dos operadores apresentados até aqui.

Fig. 5.5 Precedência e associatividade de operadores.

5.4 Chamando funções por referência

Existem três modos de passar argumentos para uma função em C ++ - atraves de *chamada por vaior cnamaaa por refrrência com argumentos de referência* e *chamada por referência com argumentos ponteiros.* No Capítulo 3, comparamos e distinguimos a chamada por valor da chamada por referência com argumentos de referência. Neste capítulo, concentramo-nos na chamada por referência com argumentos ponteiros.
Como vimos no Capítulo 3, return pode ser usado para retornar o valor de uma função para quem a chamou (ou para retornar o controle da função, sem passar de volta um valor). Vimos também que argumentos podem ser passados para uma função usando argumentos de referência, para permitir à função modificar os valores originais dos argumentos (deste modo, mais de um valor pode ser "retornado" por uma função) ou para passar objetos de dados grandes para uma função e evitar o *overhead* de passar os objetos através de uma chamada por valor (a qual, é claro, exige que se faça uma cópia do objeto). Ponteiros, como referências, podem também ser usados para modi

Operadores			Associatividade	Tipo
O	[)		esquerda para a direita	mais alto
++	¨	staticcast<type> ()	esquerda para a direita	pós-fixo
++	¨	+ - **! & ***	direita para a esquerda	unário
*	/	%	esquerda para a direita	multiplicativo
+	˙		esquerda para a direita	aditivo
«	»		esquerda para a direita	inserção/extração
<	<=	> >=	esquerda para a direita	relacional
==	1 =		esquerda para a direita	igualdade
& &			esquerda para a direita	E lógico
1			esquerda para a direita	OU lógico
?:			direita para a esquerda	condicional
=	+=	-= *= */= **%=**	direita para a esquerda	atribuição
˙			esquerda para a direita	vírgula

ficar uma ou mais variáveis de quem chamou ou para passar ponteiros para objetos de dados grandes, evitando o *overhead* de passar os objetos através de uma chamada por valor.

Em C++, os programadores podem usar ponteiros e o operador de indireção para simular uma chamada por referência (exatamente como uma chamada por referência é feita em programas em C). Quando chamarmos uma função com argumentos que devem ser modificados, são passados os endereços dos argumentos. Isto normalmente é feito aplicando-se o operador de endereço (&) ao nome da variável cujo valor será modificado.

Como vimos no Capítulo 4, arrays não são passados usando-se o operador **&** porque o nome do array é a posição inicial do array na memória (o nome de um array é equivalente a &nomeDoArray [O] .i.e.. um nome de um array já é um ponteiro). Quando o endereço de uma variável é passado para uma função, o operador de indireção (*) pode ser usado na função para criar um sinônimo, um nome alternativo ou um apelido para o nome da variável - este, por sua vez, pode ser usado para modificar o valor (se a variável não é declarada como const) que está na posição de memória na função que chamou.

As Figs. 5.6 e 5.7 apresentam duas versões de uma função que eleva um inteiro ao cubo- cubeByValue e cubeByReference. A Fig. *5.6* passa a variável number para a função cubeByValue usando uma chamada por valor. A função cubeByValue eleva ao cubo o seu argumento e passa o novo valor de volta para a função main. usando um comando return. O novo valor é atribuído a number em main. Assim, você tem a oportunidade de examinar o resultado da chamada à função antes de modificar o valor de uma variável. Por exemplo, neste programa, podíamos ter armazenado o resultado de cubeByValue em outra variável, examinado seu valor e atribuído o resultado a number após verificar a consistência desse valor.

```
1 // Fig. 5.6: fig05_06.cpp
2 // Elevar uma variável ao cubo usando uma chamada por valor
3 #include <iostream>
4
5 using std::cout;
6 using std: :endl;
7
8 int cubeByValue( int ); // protótipo
9
10 int main()
11
12 int number = 5;
13
14 cout « "O valor original de nuniber e - « number;
15 nuniber = cubeByValue( number );
16 cout « "\nO novo valor de nuniber é - « number « endl;
17 return 0;
18
19
20 int cubeByValue( int n
21
22 return n * n * n; // eleva ao cubo a variável local n
23
```

Fig. 5.6 Elevando uma variável ao cubo usando uma chamada por valor.

A Fig. 5.7 passa a variável nuniber usando uma chamada por referência - o endereço de nuinber e passado para a função cubeByReference. A função cubeByReference recebe nPtr (um ponteiro para int) como argumento. A função derreferencia o ponteiro e eleva ao cubo o valor apontado por nPtr. Isto muda o valor de nuniber em main. As Figs. 5.8 e 5.9 analisam graficamente os programas nas Figs. 5.6 e 5.7, respectivamente.

```
O valor original de number é 5
O novo valor de number é 125
```



```
1 // Fig. 5.7: figO5_07.cpp
2 // Elevando ao cubo urna variável usando uma chamada
3 II por referência com um ponteiro como argumento
4 #include <iostream>
5
6 using std::cout;
7 using std::endl;
8
9 void cubeByReference( int * ); // protótipo
lo
11 int main()
12
13 int number =5;
14
15 cout « "O valor original de number é - « number;
16 cubeByReference( &nuznber );
17 cout « '\nO novo valor de number é - « number « endl;
18 return 0;
19
20
21 void cubeByReference( int *nptr
22
23 *nptr =*nptr **nPtr **nptr; // eleva ao cubo nuxnber em main
24
```

Fig. 5.7 Elevando uma variável ao cubo usando chamada por referência com um ponteiro como argumento.

Erro comum de programação 5.5

É *um erro não derreferenciar um ponteiro, quando é necessário fazê-lo para obter o valor para o qual o ponteiro aponta.*

Uma função que recebe um endereço como argumento deve definir um argumento ponteiro para receber o endereço. Por exemplo, o cabeçalho da função cubeByReference é

void cubeByReference(int *nptr)

O cabeçalho da função especifica que a função cubeByReference recebe o endereço de uma variável do tipo inteiro *(i.e.,* um ponteiro para inteiro) como argumento, armazena o endereço localmente em nPtr e não retorna um valor. O protótipo de função para cubeByReference contém int * entre parênteses. Como com outros tipos para fins de documentação são ignorados pelo compilador.

No cabeçalho e no protótipo de uma função que espera um array unidimensional como argumento, pode ser usada a notação de ponteiro usada na lista de argumentos de cubeByReference. O compilador não distingue devariáveis, não é necessário incluir nomes de ponteiros em protótipos de funções. Os nomes de argumentos incluídos entre uma função que recebe um ponteiro e uma função que recebe um array unidimensional. Isto, é claro, significa que a função deve saber" quando ela está recebendo um array ou simplesmente uma variável isolada para a qual ela deve executar uma chamada por referência. Quando o compilador encontra um argumento de função para um array unidimensional, da forma int b [ ], o compilador converte o argumento para a notação de ponteiro int * const b (pronunciada como "b é um ponteiro constante para um inteiro" - const é explicado na Seção 5.5). As duas formas de se declarar um argumento de função como um array unidimensional são intercambiáveis.

T

O	valor	original		de nuxnber	é 5
o	novo	valor	de	nuxnber é 125	

Boa prática de programação 5.2
Use uma chamada por valor para passar argumentos para uma função, a menos que o chamador exija explicitamente que a função modifique o valor da variável passada como argumento, no contexto do chamador Este é outro exemplo do princípio de menor privilégio.

int main() nuinber

int nuntber = 1 1 125

number = CcubeByValue (**number** T)

Após **main fazer a** atribuição **a number:**

Antes de **main** chamar cubeByValue:

int main() nuniber

mntnumber=5; I 1

nuxnber cubeByValue (**number);**

Depois de **cubeByValue** receber a chamada:
int main() nuniber

int nuniber = I 1

nuxnber = **cubeByValue** (**nuinber);**

Depois de cubeByValue elevar ao cubo o parâmetro n:

int main() nuinber

int number = I 1

nuniber = **cubeByValue** (**nuniber);**

Depois de cubeByValue retornar a main:

int

int mamnQ number

= 5; 1125 1

nuniber = cubeByValue (nuniber);

Fig. 5.8 Análise de uma chamada por valor típica.

int	**cubeByValue(return n * *fl**	**int n** * *fl;* **n** 1 indefinido
int	**cubeByValue (**	**int n**
	return n * *fl	*
		n
		|

int	**cubeByVa].ue(int n**		
{	returnCn	125 * *	
			n indefinido

int cubeByValue(int n				
return n * *	**n**			
	n indefinido			

cubeByValue(int n		*)*
return fi * *fl* *	fi;	
		n
	indefinido	

328 C++ **COMO PROGRAMAR**

Depois de **cubeByValue** receber a chamada por referência:

de a

int main() nuinber void cubeByReference(int *nptr) refe

daa

int number = 1 1 ***flPtr** = ***nptr** * ***flptr** * ***nptr;** mai

cubeByReference (**&number); nPtr** argi.

indefinido funi

fum

Lepois da chamadapor referência para **cubeByReferencee** antes de ***nptr** ser elevado ao cubo: alte

arra

int main() number void cubeByReference(mnt *nptr tam

{ {

int number = **5;** 1 **5 *nptr** = ***nPtr** * ***nPtr** * ***nptr;** laçc

cubeByReference (**&nuxnber); nPtr**

} 11

Depois de *nptr ser elevado ao cubo: _____ Se

adv

int main() nuniber void cubeByReference(int *nptr

{ { *25*

int nujuber = **5; 125 [*p** = ***nptr** * ***nptr** * ***nptr;J**

cubeByReference (&nuniber); nPtr

i•l

Fig. 5.9 Análise de uma chamada por referência típica com um ponteiro como argumento.

5.5 Usando o qualificador const com ponteiros

O *qualificador* const possibilita ao programador informar ao compilador que o valor de uma variável particular não deve ser modificado.

Observação de engenharia de software ***5.1***

O qualificador const *pode ser usado para forçar o uso do princípio de menor privilégio. O uso do princípio de menor privilégio para projetar software corretamente pode reduzir bastante o tempo de depuração e efeitos colaterais incorretos, tornando o programa mais fácil de se modificar e manter*

Dica de portabilidade 5.2

Embora const *esteja bem-definido em C e C++ ANSI, alguns compiladores não garantem o seu uso correto.*

Ao longo dos anos, um grande volume de código legado foi escrito sem usar const, porque o mesmo não estava disponível nas primeiras versões de C. Por essa razão, existem grandes oportunidades para melhorias na engenharia qua de software do código escrito em versões mais antigas de C. Além disso, muitos programadores, que atualmente des estão usando C e C++ ANSI, não usam const em seus programas porque começaram a programar nas primeiras um versões de C. Estes programadores estão perdendo muitas oportunidades de fazer uso da boa engenharia de softwa- dec re.

Existem seis possibilidades para usar (ou não usar) const com argumentos de funções - duas com passagem de argumentos através de chamadas por valor e quatro com passagem de argumentos através de chamadas por referência. Como escolher uma das seis possibilidades? Guie-se pelo princípio de menor privilégio. Sempre conceda a uma função um acesso suficiente aos dados em seus argumentos para realizar sua tarefa especificada, mas não mais.

No Capitulo 3, explicamos que, quando uma funçào é chamada usando uma chamada por valor, uma cópia do argumento (ou argumentos) é feita na chamada da função e passada para a mesma. Se a cópia é modificada na função, o valor original é mantido, sem mudanças, no chamador. Em muitos casos, um valor passado para uma função é modificado para que a função possa realizar sua tarefa. Porém, em algumas instâncias, o valor não deve ser alterado na função chamada, embora a mesma manipule apenas uma cópia do valor original.

Considere uma função que recebe como argumentos um array unidimensional e seu tamanho e imprime o array. Tal função deveria iterar através do array e dar saída ao valor de cada elemento do mesmo individualmente, O tamanho do array é usado no corpo da função para determinar o índice de valor mais alto do array, de modo que o laço possa terminar quando a impressão estiver completa. O tamanho do array não muda no corpo da função.

Observação de engenharia de software 5.2

***Se** um valor não muda (ou não deve mudar) no corpo de uma função para a qual ele é passado, o argumento deve ser declarado* **const** *para garantir que não seja acidentalmente modificado.*

Se for feita uma tentativa de modificar um valor const, o compilador detectará isto e emitirá ou uma mensagem de advertência ou de erro, dependendo do compilador particular.

Observação de engenharia de software 5.3

Quando é usada uma chamada por valor somente um valor pode ser alterado na função chamada. Esse valor deve ser atribuído do valor de retorno da função. Para modificar múltiplos valores em chamadas de função, vários argumentos são passados através de uma chamada por referência.

***Boa** prática de programação 5.3*

Antes de usar uma função, confira seu protótipo de função para determinar os argumentos que ela pode modificar

Existem quatro modos de passar um ponteiro para uma função: um ponteiro não-constante para dados não-constantes, um ponteiro não-constante para dados constantes, um ponteiro constante para dados não-constantes e um ponteiro constante para dados constantes. Cada combinação fornece um nível diferente de privilégios de acesso.

O nível de acesso mais alto é obtido com um ponteiro não-constante para dados não-constantes - os dados podem ser modificados através do ponteiro derreferenciado e o ponteiro pode ser modificado para apontar para outros dados. A declaração de ponteiros não-constantes para dados não-constantes não inclui const. Tal ponteiro pode ser usado para receber um *string* em uma função que usa aritmética de ponteiros para processar (e, possivelmente, modificar) cada

caractere no *string.* A função convertTouppercase da Fig. *5.10* declara o argumento sPtr **(** char *sPtr **)** como um ponteiro não-constante para dados não-constantes. A função processa o *string* string, um caractere por vez, usando aritmética de ponteiros. A função **islower** recebe um caractere como argumento e devolve true se o caractere for uma letra minúscula e false em caso contrário. Os caracteres no intervalo 'a' a 'z' são convertidos em suas letras maiúsculas correspondentes pela função toupper; outros permanecem inalterados. A função toupper aceita um caractere como argumento. Se o caractere for uma letra minúscula, a letra maiúscula correspondente é retornada; caso contrário, é retornado o caractere original. A função toupper e a função islower fazem parte da biblioteca de manipulação de caracteres <cctype> (veja Capítulo 16, "Bits, caracteres, *strings* e estruturas").
Um ponteiro não-constante para dados constantes é um ponteiro que pode ser modificado para apontar para qualquer item de dado do tipo apropriado, mas os dados para os quais ele aponta nào podem ser modificados através desse ponteiro. Tal ponteiro poderia ser usado para receber como argumento um array para uma função que processa um elemento do array por vez, sem modificar os dados. Por exemplo, a função printCharacters da Fig. 5.11 declara argumentos sPtr como sendo do tipo const char*. A declaração é lida, da direita para a esquerda,

como "sPtr é um ponteiro para uma constante do tipo caractere". O corpo da função usa uma estrutura for para exibir na saída cada caractere no *string,* até ser encontrado o caractere nulo. Depois de cada caractere ser impresso. o ponteiro sPtr é incrementado para apontar para o próximo caractere no *string.*

```
1 // Fig. 5.10: figo5_10.cpp
2 // Convertendo letras minúsculas para letras maiúsculas
3 // usando um ponteiro não-constante para dados não-constantes
4 #include <iostream>
5
6 using std::cout;
7 using std: :endl;
8
9 #include<cctype>
10
11 void convertToUppercase( char *
12
13 int main()
14 {
15 char string[] = "caracteres e $32,98";
16
17 cout « "O string antes da conversão é: -« string;
18 convertTouppercase( string );
19 cout « "\nO string depois da conversão é:
20 « string « endl;
21 return 0;
22
23
24 void convertToUppercase( char *sPtr
25 {
26 while ( != '\O' ){
27
28 if ( islower( *sptr
29 *sPtr toupper( *sPtr ); // converte para inaiúsculas
30
31 ++sPtr; // move sPtr para o próximo caractere
32 )
33 }
```

Fig. 5.11 Imprimindo um *string,* um caractere de cada vez, usando um ponteiro não-constante para F dados constantes (parte 1 de 2).

```
O string antes da conversão é: caracteres e $32,98 O string depois da conversão é:
CARACTERES E $32,98
```

Fig. 5.10 Convertendo um *string* para maiúsculas.

```
1 // Fig. 5.11: figOS 11.cpp
```

2 // Imprimindo um string, um caractere de cada vez,	usando
3 // um ponteiro não-constante para dados constantes	
4 #include <iostream>	
5	
6 using std: :cout;	
7 using std: :endl;	
8	

9 void printCharacters (const char *

10

14

int main

char string[] = “imprime os caracteres de um string”;

15 cout « “O string é:\n”;
16 printCharacters (string);
17 cout « endi;
18 return 0;
19

20

// Em printCharacters, sPtr não pode modificar o caractere // para o qual ele aponta. sPtr é um ponteiro “somente para leitura” void printCharacters(const char *sptr

for (; *sptr != ‘\O’; sPtr++) // nenhuma inicialização cout « *sptr;

Fig. 5.11 Imprimindo um *string,* um caractere de cada vez, usando um ponteiro não-constante para dados constantes (parte 2 de 2).

A Fig. 5. 12 mostra as mensagens de erro de sintaxe produzidas quando se tenta compilar uma função que recebe um ponteiro não-constante para dados constantes e então se tenta usar o ponteiro para modificar os dados.

1 // Fig. 5.12: fig05_12.cpp
2 // Tentando modificar dados através de um
3 // ponteiro não-constante para dados constantes.
4 #include <iostream>
5

6 void f (const int *

7

8 int main O
9
10 int y;

```
11

12 f ( &y ); // f tenta modificação ilegal

13

14 return 0;

15
16

// xPtr não pode modificar o valor // da variável para a qual ele aponta vojd f( const int
*xptr

*xptr = 100;

11
12
13

21
22
23
24
25
26
27

17
18
19
20
21
22

// não se pode modificar um objeto const
```

Fig. 5.12 Tentando modificar dados através de um ponteiro não-constante para dados constantes (parte 1 de 2).

O	string	é:
imprime		os caracteres de um string


```
4 # include <iostream>
5
6 int main O
7
8 int x, y;
9
10 int *const ptr =&x; // ptr é um ponteiro constante para um
11 // inteiro. Um inteiro pode ser modificado
12 // através de ptr, mas ptr sempre aponta
13 // para a mesma posição de memória.
14 *ptr =7;
```

```
15 ptr =
16
17 return O;
18
```

Mensagem de erro do compilador Borland C+ + com linha de comando:

Fig. 5.13 Tentando modificar um ponteiro constante para dados não-constantes (parte 2 de 2).

Erro comum de programação 5.6

É *erro de sintaxe não inicializar um ponteiro declarado como* **const.**

Os menores privilégios de acesso são garantidos por um ponteiro constante para dados constantes. Um ponteiro assim sempre aponta para a mesma posição de memória e os dados nessa posição de memória não podem ser modificados usando o ponteiro. Assim é que um array deve ser passado para uma função que só examina o próprio array, usando a notação de subscrito de arrays, sem modificá-lo. O programa da Fig. 5.14 declara a variável ponteiro ptr como sendo do tipo const int * const. Esta declaração é lida, da direita para esquerda, como "ptr é um ponteiro constante para uma constante do tipo inteiro". A figura mostra as mensagens de erro geradas quando foi feita uma tentativa para se modificar os dados para os quais ptr aponta e quando foi feita uma tentativa de modificar o endereço armazenado na variável ponteiro. Note que nenhum erro é gerado quando tentamos exibir o valor para o qual ptr aponta, porque nada está sendo modificado no comando de saída.

Fig. 5.14 Tentando modificar um ponteiro constante para dados constantes (parte 1 de 2).

Error E2024 figOs 13.cpp 15: Cannot modify a const main()		**object in**	**function**
Mensagem de erro do compilador Microsoft Visual C+ +			
[figos **13.cpp(15) : error C2166: 1-value specifies**		**const object**	

```
1  // Fig. 5.14: figO5l4.cpp  I/ Tentando
2  modificar um ponteiro constante  II para
3  dados constantes.  #include <iostream>
4
5
6  using std::cout; using std: :endl;
7
8
9  int main() { int x =5, y;
10
11
```



```
12
13 const int *const ptr =&x; // ptr é um ponteiro constante para uma
14 // constante inteira. ptr sempre
15 // aponta para a mesma posição e o
16 // inteiro que está armazenado nessa
17 // posição não pode ser modificado.
18 cout « *ptr « endl;
19 *ptr =7;
20 ptr =
21
22 returnO;
23
```

Mensagem de erro do compilador Borland C++ com linha de comando:

Error E2024 fig05 14.cpp 19: Cannot modify a const object in funetion **main()**

Error E2024 fig05 14.cpp 20: Cannot modify a const obect in function main()

Mensagem de erro do compilador Microsofi Visual C+ +

figO5l4.cpp(19) : error C2166: 1-value specifies const object

figO5l4.cpp(20) : error C2166: 1-value specifies const object

Fig. 5.14 Tentando modificar um ponteiro constante para dados constantes (parte 2 de 2).

5.6 *Bubble sort* usando chamada por referência

Modifiquemos o programa *bubble sori* da Fig. 4.16 para usar duas funções - bubbleSort e swap (Fig. 5.15). A função bubbleSort executa a ordenação do array. Ela chama a função swap para permutar os elementos de array array [j 1 e array [j + 1]. Lembre-se de que C÷÷ força obrigatoriamente a ocultação de informação entre funções, de modo que swap não tem acesso a elementos individuais do array em bubbleSort. Como bubbleSort *quer* que swap tenha acesso aos elementos do array que devem ser trocados de posição, bub - bleSort passa cada um destes elementos através de uma chamada por referência para swap - o endereço de cada elemento de array é passado explicitamente. Embora arrays inteiros sejam automaticamente passados por chamadas por referência, elementos individuais de array são escalares e são normalmente passados por chamadas por valor. Portanto, bubbleSort usa o operador de endereço (**&**) para cada elemento de array na chamada swap, como segue, para fazer a chamada por referência:

swap (**&array[** j 1, **&array[** j + 1 1 *);*

```
1 // Fig. 5.15: figO5l5.cpp
2 // Este programa põe valores em um array, classifica os valores em
3 // ordem crescente e imprime o array resultante.
4 #include <iostream>
5
6 using std::cout;
7 using std: :endl;
8
9 #include <iomanip>
```

Fig. 5.15 *Bubble sort com* chamada por referência (parte 1 de 2).

```
for ( i =O; i < arraySize; i++
cout « setw( 4 ) « a[ i ];

cout « endi;

return O;

if ( array[ j ] > array[ + 1 swap( &array[ ], &array[ ÷1) );
```



```
void swap(int *const elementlPtr, int *const element2Ptr

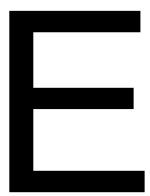
int hold =*elementlptr;
*elementlptr =*element2ptr;

*element2ptr =hold;
```

A função swap recebe &array [j } na variável ponteiro elementlPtr. Por causa do princípio de ocultamento de informação, swap não tem permissão de conhecer o nome array [j J, mas swap pode usar elementlPtr como sinônimo para array [j 1. Deste modo, quando swap referencia elementlPtr, ela está na realidade

```
using std::setw;
void bubbleSort( int -const int );
int main O
const int arraySize =10;
int a[ arraySize ) ={ 2, 6, 4, 8, 10, 12, 89,
int i;
cout « "Itens de dados na ordem original\n";
for ( i =0; i < arraySize; i++
cout « setw( 4 ) « a[ i 1;
bubbleSort( a, arraySize ); // classifica o array cout « "\nltens de dados em ordem
ascendente\n;
```

```
68, 45, 37 };
```



```
void bubbleSort( int *array, const int size void swap( int *const; int *const); for ( int pass =0; pass
< size -1; pass++
for (intj =0; j<size-l; j++)


}
```

Fig. 5.15 *Bubble sort com* chamada por referência (parte 2 de 2).

Itens	de	dados	na	ordem	original					
2		6	4	8	10 12	89	68	45	37	
Itens	de	dados	em	ordem	ascendente					
2		4	6	8	10 12	37	45	68	89	

Erro comum de programação 5.7

Usar o operador **si** zeof *em uma função para achar o tamanho em bytes de um argumento do tipo array resulta no tamanho em bytes de um ponteiro e não no tamanho em bytes do array.*

```
1 II Fig. 5.16: fig0516.cpp
2 1/ O operador sizeof, quando usado com um nome de array,
3 II retorna o número de bytes no array.
4 #include <iostream>
5
6 using std::cout;
7 using std: :endl;
8
9 size_t getsize( double *
10
11 int main()
12 {
13 double array [ 20 1;
14
15 cout « 'O número de bytes no array é
16 « sizeof( array
17 « "\nO número de bytes retornados por getSize é
18 « getSize( array ) « endi;
19
20 return 0;
21 }
22
23 sizet getSize( double *ptr
24
25 return sizeof( ptr );
26
```

Fig. 5.16 O operador **sizeof,** quando aplicado a um nome de array, retorna o número de bytes no array.

O número de elementos em um array pode ser também determinado usando-se os resultados de duas operações sizeof. Por exemplo, considere a seguinte declaração de array:

doublerealArray[22 **];**

Se variáveis do tipo de dados double são armazenadas em 8 bytes de memória, o array realArray contém um total de 176 bytes. Para determinar o número de elementos no array, pode ser usada a expressão seguinte:

sizeofrealArray/ sizeof(double

A expressão determina o número de bytes no array realArray e divide esse valor pelo número de bytes de memória utilizados para armazenar um valor double.

O programa da Fig. 5.17 usa o operador si zeof para calcular o número de bytes usados para armazenar cada um dos tipos de dados normais disponíveis no computador pessoal que estivermos usando.

Dica de portabilidade 5.3

___ O número de bytes usados para armazenar um tipo particular de dados pode variar entre sistemas. Quando estiver escrevendo programas que dependem do tamanho dos tipos de dados e que serão executados em

O	número	de	bytes	no	array é 80
O	número	de	bytes	retornado	por getSize é 4

```
24
25
26
27
28
29
30
31
32


33
34
35
36
37
38
39
40


using std::cout; using std: :endl;


« “\nsizeof $ «
« “\tsizeof(short) =
« ‘\nsizeof i =“«
« “\tsizeof(int) =
« “\nsizeof 1 =“«
« “\tsizeof(long) =
« ‘\nsizeof f =
« “\tsizeof(float) =
« “\nsizeof d =“«


«
«
«
«
«
return 0;


sizeof $
« $izeof( short
sizeof i
« sizeof( int
sizeof 1 « sizeof( lorig sizeof f
« sizeof( float sizeof d


sizeof (char) =1
sizeof (short) =2
sizeof (int) =4
sizeof (long) =4
sizeof (float) =4
sizeof (doubje) =8
sizeof (long double) =8
```

Fig. 5.17 Usando o operador **sizeof** para determinar os tamanhos de tipos de dados padrão.

O operador sizeof pode ser aplicado a qualquer nome de variável, nome de tipo ou valor constante. Quando

aplicado a um nome de variável (o qual não é um nome de array) ou um valor constante, o número de bytes usados

vdrios sistemas de computador use `sizeof` *para determinar o número de kvtes usados para armazenar os tipos de dados.*

```
II Fig. 5.17: figO5l7.cpp
II Demonstrando o operador sizeof #include <iostream>

#include <iomanip>

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23


int main ()
char c;
short s;
int i;
long 1;
float f;
double d;
long double ld;
int array[ 20 J, *ptr =array;
cout « “sizeof c =« sizeof o
« “\tsizeof(char) =‘« sizeof( char


}


“\tsizeof(double) =‘« sizeof( double “\nsizeof ld =« sizeof ld
“\tsizeof(long double) ‘« sizeof( long double “\ns±zeof array « sizeof array
“\nsizeof ptr =“« sizeof ptr
endl;
```

`sizeof` sizeof
sizeof
sizeof sizeof sizeof
sizeof
`sizeof sizeof`

```
c= 1
```

s=2
i=4
1=4
f= 4
d=8
ld = 8
array = 80
ptr = 4

para armazenar o tipo específico de variável ou constante é retornado. Note que os parênteses usados com **sizeof** são exigidos se um nome de tipo é fornecido como operando. Os parênteses usados com sizeof não são exigidos se um nome variável for fornecido como operando. Lembre-se, sizeof é um operador, não uma função.
Erro comum de programação 5.8
Omitir os parênteses em uma operação **sizeof** *quando o operando é um nome de tipo é erro de sintaxe.*

1 *Dica de desempenho 5.2*

f **sizeof** *é um operado unário usado durante a compilação, não uma função executada durante a execução. Deste modo, usar* **sizeof** *não causa impacto negativo sobre o desempenho da execução.*

5.7 Expressões com ponteiros e aritmética de ponteiros

Ponteiros são operandos válidos em expressões aritméticas, expressões de atribuição e expressões de comparação. Porém, nem todos os operadores normalmente usados nestas expressões são válidos com variáveis do tipo ponteiro. Esta seção descreve os operadores que podem ter ponteiros como operandos e como são usados esses operadores.
Um conjunto limitado de operações aritméticas pode ser executado com ponteiros. Um ponteiro pode ser incrementado (++) ou decrementado (--), um inteiro pode ser somado a um ponteiro (+ ou **+), um inteiro pode ser** subtraído de um ponteiro (- ou -=) ou um ponteiro pode ser subtraído de outro.
Suponha que o array int v **[5]** tenha sido declarado e que seu primeiro elemento esteja na posição 3000 da memória. Suponha que o ponteiro vPtr tenha sido inicializado para apontar para v **[O**), i.e., que o valor de vPtr seja 3000. A Fig. 5.18 mostra com um diagrama esta situação para uma máquina com inteiros de 4 bytes. Note que vPtr pode ser inicializado para apontar para o array v com qualquer um dos comandos seguintes
vPtr =
vPtr = &v[O);
posição
3000 3004 3008 **3012 3016**
v[0] v[1] v[2) vE3] v[4)
variável ponteiro vPtr

Fig. 5.18 O array **v** e uma variável ponteiro vPtr que aponta para **v.**

Dica de portabilidade 5.4
___A maioria dos computadores atuais têm inteiros de 2 bytes ou de **4** *bytes. Algumas das máquinas moo recentes usam inteiros de 8 bytes. Como os resultados da aritmética de ponteiros dependem do tamanho dos objetos para os quais um ponteiro aponta, a aritmética de ponteiros é dependente da máquina.*
Na aritmética convencional, a adição **3000 + 2 dá o valor 3002. Este normalmente não é ocaso com a aritmética** de ponteiros. Quando um inteiro é somado a ou subtraído de um ponteiro, o ponteiro não é simplesmente incrementado ou decrementado por esse inteiro, mas pelo tamanho do objeto inteiro ao qual o ponteiro se refere. O número de bytes depende do tipo de dado do objeto. Por exemplo, o comando
vptr **+ 2;**

produziria 3008 (3000 + 2 * 4), supondo que um inteiro seja armazenado em 4 bytes de memória. No array v, vPtr agora apontaria para v **[** 2 **]** (Fig. 5.19). Se um inteiro fosse armazenado em 2 bytes de memória, então o cálculo precedente resultaria na posição de memória 3004 (3000 + 2 * 2). Se o array fosse de um tipo de dado diferente, o comando precedente iria incrementar o ponteiro por duas vezes o número de bytes usados para armazenar um objeto desse tipo de dados. Quando se executa aritmética de ponteiros sobre um array de caracteres, os resultados serão consistentes com a aritmética regular porque cada caractere tem um byte de comprimento.
localização
3000 3004 3008 3012 3016
v[0] **v[1)** v[2] **v[3] v[4)**
variável ponteiro **vPtr**
Fig. 5.19 O ponteiro vptr após operação de aritmética de ponteiros.

Se vPtr foi incrementado para 3016, que aponta para v[4], o comando
vPtr -= 4;
faria vPtr apontar para 3000 - o início do array. Se um ponteiro está sendo incrementado ou decrementado por um, podem ser usados os operadores de incremento (++) e decremento **(---).** Cada um dos comandos
++vPtr;
vPtr++;
incrementam o ponteiro para apontar para a próxima posição no array. Cada um dos comandos
--vPtr;
vPtr--;
decrementa o ponteiro para apontar para o elemento precedente do array.
Variáveis ponteiro que apontam para o mesmo array podem ser subtraídas umas das outras. Por exemplo, se vPtr aponta para a posição 3000 e v2Ptr contém o endereço 3008, o comando
x = v2Ptr - vPtr;
atribuiria a x o número de elementos de array de vPtr até v2Ptr, neste caso, 2. A aritmética de ponteiros não tem sentido, a menos que executada sobre um array. Não podemos assumir que duas variáveis do mesmo tipo sejam armazenadas contiguamente na memória, a menos que sejam elementos adjacentes de um array.

Erro comum de programação 5.9
Usar aritmética de ponteiros sobre um ponteiro que não se refere a um array de valores normalmente é um erro de lógica.

Erro comum de programação 5.10
Subtrair ou comparar dois ponteiros que não se referem a elementos do mesmo array normalmente é um erro de lógica.

Erro comum de programação 5.11
Endereçar além dos extremos de um array, quando usamos aritmética de ponteiros, normalmente é erro de lógica.

Um ponteiro pode ser atribuído a outro ponteiro se ambos os ponteiros são do mesmo tipo. Caso contrário, deve ser usado um operador de conversão de tipo para converter o valor do ponteiro à direita da atribuição para o tipo de ponteiro à esquerda da atribuição. A exceção a esta regra é o ponteiro para void (i.e., void *), o qual é um ponteiro genérico capaz de representar qualquer tipo de ponteiro. Um ponteiro void pode ser atribuído a todos os tipos de pontejro, sem conversão de tipo. Porém, um ponteiro void não pode ser diretamente atribuído a um ponteiro de outro tipo - o ponteiro void deve primeiro ser convertido para o tipo de ponteiro apropriado.
Um ponteiro void não pode ser derreferenciado. Por exemplo, o compilador sabe que um ponteiro para int se refere a quatro bytes de memória em uma máquina com inteiros de 4 bytes, mas um ponteiro void simplesmente contém o endereço de memória de um dado de tipo desconhecido - o número preciso de bytes a que o ponteiro se refere não é conhecido pelo compilador. O compilador deve saber o tipo de dados para determinar o número de bytes a serem dereferenciados para um ponteiro particular. No caso de um ponteiro para void, este número de bytes não pode ser determinado pelo tipo.

Erro comum de programação 5.12
Atribuir um ponteiro de um tipo a um ponteiro de outro (diferente de void *), sem converter o primeiro ponteiro para o tipo do segundo ponteiro, é erro de sintaxe.*

Erro de programação comum 5.13
Derreferenciar um ponteiro void é *um erro de sintaxe.*

Ponteiros podem ser comparados usando-se operadores de igualdade e relacionais, mas tais comparações não têm sentido a menos que os ponteiros apontem para membros do mesmo array. As comparações de ponteiros comparam os endereços armazenados nos ponteiros. Uma comparação de dois ponteiros apontando para o mesmo array podia mostrar, por exemplo, que um ponteiro aponta para um elemento do array de índice mais alto que o outro ponteiro. Um uso comum da comparação de ponteiros é determinar se um ponteiro é O.

5.8 A relação entre ponteiros e arrays

Em C++, arrays e ponteiros estão intimamente relacionados e podem ser usados de maneira *quase* intercambiável, Um nome de array pode ser encarado como um ponteiro constante. Os ponteiros podem ser usados para fazer qualquer operação envolvendo indexação de arrays.

Boa prática de programação 5.4
Use a notação de array em vez da notação de ponteiro quando manipular arrays. Embora o programa possa se tornar ligeiramente mais lento para ser compilado, provavelmente ele será mais claro.

Suponha que o array de inteiros b **[** 5 **]** e a variável ponteiro de inteiros bPtr tenham sido declarados. Como o
nome de array (sem um subscrito) é um ponteiro para o primeiro elemento do array, podemos atribuir a bPtr o
endereço do primeiro elemento no array b com o comando

bPtr =

Isto é equivalente a termos atribuído o endereço do primeiro elemento do array como segue

bPtr = **&b[O**

O elemento b **[3 3** do array pode ser alternativamente referenciado com a expressão de ponteiro

* (bPtr **+** 3

Na expressão acima, o 3 é o *deslocamento* do ponteiro. Quando o ponteiro aponta para o início de um array, o deslocamento indica qual elemento do array devia ser referenciado, e o valor do deslocamento é idêntico ao índice do array. A notação precedente é conhecida como *notação ponteiro/deslocamento.* Os parênteses são necessários

porque a precedência de * é mais alta que a precedência de **+.** Sem os parênteses, a expressão acima somaria 3 ao valor da expressão *bptr (i.e., 3 seria somado a b **[O],** assumindo que bPtr aponta para o início do array). Da mesma maneira que um elemento de array pode ser referenciado com uma expressão de ponteiro, podemos escrever no endereço
&b[3
com a expressão de ponteiro
bPtr **+ 3**
O nome do array pode ser tratado como um ponteiro e usado em aritmética de ponteiros. Por exemplo, a expressão
***(b + 3**
se refere também ao elemento b **[3**] do array. Em geral, todas as expressões com subscritos de array podem ser escritas com um ponteiro e um deslocamento. Neste caso, foi usada a notação ponteiro/deslocamento com o nome do array como um ponteiro. Note que o comando precedente não modifica o nome do array de nenhuma forma; b ainda aponta para o primeiro elemento no array.
Ponteiros podem ser subscritos exatamente como podem os arrays. Por exemplo, a expressão
bPtr[**1**)
se refere ao elemento de array b **[** 1 **3;** esta expressão é chamada de *notação ponteiro/subscrito.*
Lembre-se de que um nome de array é essencialmente um ponteiro constante; sempre aponta para o início do array. Deste modo, a expressão
b + 3
é inválida, porque tenta modificar o valor do nome de array com aritmética de ponteiros.
*Erro comum de programação **5.14***
Embora os nomes de array sejam ponteiros para o início do array e ponteiros possam ser modificados em expressões aritméticas, nomes de array não podem ser modificados em expressões aritméticas porque são ponteiros constantes.

A Fig. 5.20 usa os quatro métodos por nós discutidos para referenciar os elementos de um array: uso de subscritos para o array, ponteiro/deslocamento com o nome do aay como ponteiro, subscritos com ponteiro e ponteiro/deslocamento com um ponteiro - para imprimir os quatro elementos do array **b, de inteiros.**

```
1 // Fig. 5.20: fig052O.cpp
2 // Usando subscritos e notações de ponteiro com arrays
3
4 #include <iostreain>
5
6 using std: :cout;
7 using std::endl;
8
9 int main()
10 {
11 int b[] = { 10, 20, 30, 40 }, i, offset;
12 int *bPtr = b;// configurar bPtr para apontar para array b
13
14 cout « "Array b impresso com:\n"
15 « "Notação de subscrito de array\n";
```

Fig. 5.20 Usando quatro métodos de fazer referência a elementos de array (parte 1 de 2).


```
16
17 for(i0;i<4;i++)
18 cout « "b[" « i « "] = « b[ i 1 « \n';
19
20
21 cout « "\nNotação de ponteiro/deslocamento onde\n"
22 « "o ponteiro é o nome do array\n";
23
24 for ( offset = 0; offset < 4; offset++
25 cout« "*(b + « offset « ") =
26 « '( b + offset ) « \n;
27
28
29 cout « "\nNotação ponteiro/subscrito\n";
30
31 for (i0; i<4; i++)
32 cout « "bPtr[" « i « 9 = " « bptr[ i 1 « '\n';
33
34 cout « "\nNotação ponteiro/deslocamento\n";
35
36 for ( offset = 0; offset < 4; offset++
37 cout« "*(bptr + - « offset « ") =
38 « ( bptr + offset ) « \n';
39
40 return 0;
41 )
Array b impresso com:
Notação de subscrito de array
b[0] = 10
b[1] = 20
b[2] = 30
b[3) = 40
Notação de ponteiro/deslocamento onde
o ponteiro é o nome do array
*(b + 0)= 10
*(b + 1)= 20
*(b + 2)= 30
*(b + 3)= 40
Notação ponteiro/subscrito
bPtr [0] = 10
bPtr [1] = 20
bPtr [2] = 30
bPtr [3] = 40
Notação ponteiro/deslocamento
*(bptr + O ) = 10
```

```
*(bptr + 1 ) = 20
*(bPtr + 2 ) = 30
*(bptr + 3 ) = 40
```

Fig. 5.20 Usando quatro métodos de fazer referência a elementos de array (parte 2 de 2).

Para ilustrarmos mais a intercambiabilidade de arrays e ponteiros, vamos examinar as duas funções de cópia de *strings* - copyl e copy2 - no programa da Fig. 5.21. Ambas as funções copiam um *string* para um array de caracteres. Após compararmos os protótipos de função de copyl e copy2, as funções parecem idênticas (por

causa da intercambiabilidade entre arrays e ponteiros). Estas funções realizam a mesma tarefa, mas ela é executada de forma diferente.

```
1 // Fig. 5.21: figO52l.cpp SI
2 // Copiando um string usando notação
3 // de array e notação de ponteiro.
4 #include <iostream> co
5 err
6 using std: :cout; de
7 using std: :endl; pa
8 nu
9 void copyl( char -, const char *ob
10 void copy2( char , const char *); ar
11
12 int main ()
13
14 char stringl [ 10 ], *string2 = "Olá ! -5.
15 string3,[ 10 ], string4[] = "Tchau !";
16 O
17 copyl( stringl, string2 )
18 cout « "stringl = " « stringl « endi;
19
20 copy2( string3, string4 );
21 cout « "string3 = " « stririg3 « endl; tar
22
23 return 0;
24 )
25
26 // copia s2 para sl usando notação de array
27 void copyl( char *sl, const char *s2 ) A
28 { eh
28 for int i = 0; ( sl[ i ] = s2[ i ] ) != '\O'; i++ )
30 ; // não faz nada no corpo do for re
31 } qu
32 no
33 // copia s2 para sl usando notação de ponteiro ro
34 void copy2( char *sl, const char *s2 ) a
esi
36 for ( ; ( *sl = *s2 ) != '\O'; sl++, s2++
37 ; // não faz nada no corpo do for
38 }
```

Fig. 5.21 Copiando um *string* usando a notação de array e a notação de ponteiro.

A **função copyl** usa a notação de subscrito de array para copiar o *string* **s2 para o** array **de caracteres si.** A função - declara uma variável contadora i, do tipo inteiro, para usar como o índice do array. O cabeçalho da estrutura **for Fi**

executa toda a operação de cópia - seu corpo é o comando vazio, O cabeçalho especifica que i é inicializado com

zero e incrementado de 1 a cada repetição do laço. A condição no for, (si [i 1 = s2 [i]) ! = '\O O
executa a operação de cópia caractere por caractere, de s2 para si. Quando o caractere nulo é encontrado em s 2, ele é atribuído a si e o laço termina, porque o caractere nulo é igual a '\O'. Lembre-se de que o valor de um e comando de atribuição é o valor atribuído ao argumento da esquerda. q A função copy2 usa ponteiros e aritmética de ponteiros para copiar o *string* s2 para o array de caracteres sl. pr
Novamente, o cabeçalho da estrutura for executa a operação de cópia inteira. O cabeçalho não inclui qualquer

string 1 =	**Olá!**
string 3 =	**Tchau!**

inicialização de variável. Como na função copyi, a condição *(* si = * s2) !* = '\O executa a operação de cópia. O ponteiro s2 é derreferenciado e o caractere resultante é atribuído ao ponteiro derreferenciado si. Depois da atribuição na condição, os ponteiros são incremeritados para apontar para o próximo elemento do array si e o próximo caractere do *string* **s2,** respectivamente. Quando o caractere nulo é encontrado em s2. ele é atribuído ao ponteiro derreferenciado si e o laço termina.
Note que o primeiro argumento, tanto para copyi como copy2, deve ser um array grande o bastante para conter o *string* no segundo argumento. Caso contrário, pode ocorrer um erro quando é feita uma tentativa de escrever em uma posição de memória além do limite do array. Note, também, que o segundo argumento de cada função é declarado como const char * (um *string* constante). Em ambas as funções, o segundo argumento é copiado para os primeiros argumentos-os caracteres são copiados do segundo argumento um de cada vez, mas os caracteres nunca são modificados. Assim, sendo o segundo argumento declarado para apontar para um valor constante, é obedecido o princípio de menor privilégio. Nenhuma das funções necessita da capacidade de modificar o segundo argumento, assim nenhuma função recebe essa capacidade.

59 Arrays de ponteiros

Os arrays podem conter ponteiros. Um uso comum de tal estrutura de dados é formar um array de strings, chamado simplesmente de *array de strings.* Cada entrada no array é um *string,* mas em C++ um *string* é essencialmente um ponteiro para seu primeiro caractere. Assim, cada entrada em um array de *strings* é realmente um ponteiro para o primeiro caractere de um *string.* Considere a declaração do array de *string* suit, que poderia ser útil para representar um baralho.

```
const char *suit[ 4 ] = { "Copas", "Ouros",
"Paus", "Espadas" };
```

A parte **suit** *[* **4** *1* da declaração indica um array de 4 elementos. A parte char * da declaração indica que cada elemento do array suit é do tipo "ponteiro para caractere". Os quatro valores colocados no array são "Copas", "Ouros", "Paus" e "Espadas ".Cada um destes valores é armazenado na memória como um *string* de caracteres terminado com um caractere nulo - o qual tem um caractere a mais que o número de caracteres entre aspas. Os quatro *strings* têm 6, 6, 5 e 8 caracteres, respectivamente. Embora pareça que estes *strings* estejam sendo colocados no array suit, só ponteiros estão realmente armazenados no array (Fig. 5.22). Cada ponteiro aponta para o primeiro caractere de seu *string* correspondente. Deste modo, embora o array suit tenha tamanho fixo, ele fornece acesso a *strings* de caracteres de qualquer comprimento. Esta flexibilidade é um exemplo das poderosas capacidades de estruturar dados de C++.

Fig. 5.22 Representação gráfica do array suit.

Os *strings* de suit podem ser colocados em um array bidimensional, no qual cada linha representa um naipe e cada coluna representa uma das cartas de um naipe. Tal estrutura de dados deve ter um número fixo de colunas por linha e esse número deve ser tão grande quanto o maior *string.* Então, é perdida uma quantidade considerável de memória quando um número grande de *strings* é armazenado com a maioria dos strings menores que o *string* mais longo. Na próxima seção, usamos arrays de *slrings* **para** representar um baralho.

suit[O] suit[1) suit[2] suit[31	[! ¡	ø.	**'C'**	**'o']'p '**	**'a'**	**'s'**	**l'\O'**	

'	**'O'**	**]'u']'r ,**	**'o']**	1°1		
'	**'P'**		's'	**'\O'**		
'	**'E'**	iZ]f	IL Z	**'d'**	I'a'	1 's' I'\O' 1

5.10 Estudo de caso: uma simulação de embaralhamento e Pararetira distribuição de cartas correspon

12. A que

Nesta seção, usamos a geração de números aleatórios para desenvolver um programa de simulação do embaralhamento assim, pat
e distribuição de cartas. Este programa pode, então, ser usado para implementar programas que simulam jogos especí-guir o vali
ficos de carta. Para revelar alguns problemas sutis de desempenho, usamos intencionalmente algoritmos de embaralha-de caracte
mento e distribuição subótimos. Nos exercícios, desenvolveremos algoritmos mais eficientes. "Rei **dE**

Usando a abordagem *top-down* de refinamento passo a passo, desenvolvemos um programa que embaralhará um baralho de *52* cartas e, então, fará a distribuição de cada uma das 52 cartas. A abordagem *top-down* é particular-Vamos pn
mente útil para atacar problemas maiores e mais complexos do que os que vimos nos capítulos iniciais.

Usamos um array deck de *4* por 13, bidimensional, para representar o baralho (Fig. 5.23). As linhas corres ponde aos naipes: a linha O corresponde a copas, a linha 1 a ouros, a linha 2 a paus e a linha 3 a espadas. As colunas Nosso pri

correspondem aos valores de face das cartas: O até 9 correspondem a ás até dez, respectivamente, e as colunas 10 até 12 correspondem a valete, dama e rei, respectivamente. Devemos preencher o array suit, do tipo *string,* com *strings* de caracteres representando os quatro naipes e o array **face, do tipo** *string,* com *strings* de caracteres representando os treze valores das cartas.

2 G)
.0 (5. Si O N E _ **"Embaral]**
(i) Q Q) D .E W O W (5 W
1- *O* **O** *(1)* O) O Z O > O
O 1 2 3 4 5 6 7 8 9 101112
Copas O
Ouros 1
Paus 2 "Retire 5
Espadas 3
deck [2] [12] representaoReidePaus A
incorpc
Paus Rei

Fig. 5.23 Representação de um baralho com um array bidimensional.

Este baralho simulado pode ser embaralhado como segue. Primeiro, o array **deck** é limpo com zeros. Então, uma **row** (0-3) e uma column (0-12) são escolhidas ao acaso. O número **1** é inserido no baralho no elemento deck **row**] **[** coluinri **]** do array para indicar que esta carta vai ser a 1 retirada do baralho embaralhado. Este processo continua com os números 2,3 52 sendo aleatoriamente inseridos no array **deck** para indicar quais cartas vão ser colocadas na 2, 3 e 52 posição, no baralho embaralhado. A medida que o array **deck** começa a se encher com números de cartas, é possível que uma carta seja selecionada duas vezes, i.e., deck **[row**] [coluinn **1** será não-zero quando for selecionado. Essa seleção é simplesmente ignorada e outras **rows** e coluniris são repetidamente escolhidas ao acaso, até que seja encontrada uma carta ainda não selecionada. Em algum momento, os números 1 até 52

ocuparão as 52 posições do array **deck. Neste momento,** o baralho está completamente embaralhado. Coloque Este algoritmo de embaralhamento pode ficar sendo executado por um período de tempo indefinidamente expandidi longo se as cartas que já tenham sido embaralhadas são repetidamente selecionadas ao acaso. Este fenômeno é conhecido como *adiamento indefinido.* Nos exercícios, discutimos um algoritmo de embaralharamento melhor, que elimina a possibilidade de ocorrência de um adiamento indefinido.

Dica de desempenho 5.3
___Às vezes, um algoritmo que surge de um modo"natural" pode conter problemas sutis de desempenho, tal como o adiamento indefinido. Procure algoritmos que evitam o adiamento indefinido.

Para retirar a primeira carta, fazemos uma pesquisa no array, procurando um valor de **deck [row] [** colunin correspondente a 1. Isto *é* feito com uma estrutura for aninhada que faz row variar de O até 3 e colunn de O até 12. A que carta do baralho corresponde a posição do array? O array suit foi pré-carregado com os quatro naipes; assim, para obter o naipe, imprimiremos o *string* de caracteres suit [linha]. Semelhantemente, para conseguir o valor de face da carta, imprimimos o *string* de caracteres face [column). Imprimimos também o *string* de caracteres "de" Imprimir estas informações na ordem apropriada nos habilita a imprimir cada carta na forma "Rei de Paus", "Ás de Ouros", e assim por diante.

Vamos prosseguir com o processo *top-down* de refinamento passo a passo. O topo é simplesmente

Embaralha e retira 52 cartas

Nosso primeiro refinamento nos dá:

Inicialize o array naipe (suit)
Inicialize o array face (face)
Inicialize o array baralho (deck)
Embaralhe o baralho
Retire 52 cartas

"Embaralhe o baralho" pode ser expandido como segue:

Para cada uma das 52 cartas
Coloque o número da carta em uma posição desocupada do
array baralho selecionada aleatoriamente

"Retire 52 cartas" pode ser expandido como segue:

Para cada uma das 52 cartas
Ache um número de carta no array baralho e imprima face e naipe da carta

A incorporação destas expansões completa nosso segundo refinamento:

Inicialize o array naipe
Inicialize o array face
Inicia lize o array baralho
Para cada uma das 52 cartas
Coloque o número da carta em uma posição desocupada do
array baralho selecionada aleatoriamente
Para cada uma das 52 cartas
Ache um número de carta no array baralho e imprima face e naipe da carta

"Coloque o número da carta em uma posição desocupada do array baralho selecionada aleatoriarnente" pode ser expandido como segue:

Escolha uma posição do baralho aleatoriamente
Enquanto posição escolhida do baralho já foi escolhida antes
Escolha a posição do baralho aleatoriamente
Coloque o número da carta na posição escolhida do baralho

"Ache um número de carta no array baralho e imprima a face e o naipe da carta" pode ser expandido como segue:

1
Para cada posição do array baralho
Se a posição contém um número de carta
Imprima a face e o naipe da carta

Incorporando estas expansões, obtemos nosso terceiro refinamento:

Inicialize o array naipe
Inicialize o array face
Inicialize o array baralho
Para cada uma das 52 cartas
Escolha uma posição do baralho aleatoriamente
Enquanto posição escolhida do baralho já foi escolhida antes
Escolha a posição do baralho aleatoriamentc
Coloque o número da carta na posição escolhida do baralho
Para cada uma das 52 cartas
Para cada posição do array baralho
Se a posição contém um número de carta

Imprima a face e o naipe da carta
Isto completa o processo de refinamento. Note que esse programa é mais eficiente se as partes do algoritmo referentes ao embaralhamento e retirada da carta do baralho forem combinadas, de modo que cada carta é retirada à medida que é colocada no baralho. Escolhemos programar estas operações separadamente porque, normalmente, as cartas **4** são retiradas depois de terem sido embaralhadas (e não à medida em que são embaralhadas). **4**
O programa de embaralhamento e distribuição de cartas é mostrado na Fig. 5.24 e um exemplo de execução é **4** exibido na Fig. 5.25. Note a formatação da saída usada na função deal: **4**
cout « setw(**5**) **« setiosflags(ios::right**
« **wFace[column**] « **de**
« **setw(8**) « **setiosflags(ios::left**
« **wSuit[row**
« (**card % 2** == **0 ? '\n' :**
O **comando de saída precedente faz com que a face da carta seja mostrada alinhada** *à* **direita em um campo de** 5 caracteres e o naipe em um campo de 8 caracteres. A saída é impressa em um formato de duas colunas. Se a carta que está sendo mostrada está na primeira coluna, uma marca de tabulação é posta na saída, depois da carta, para mover para a segunda coluna; caso contrário, um caractere de nova linha é posto na saída.

Fig. 5.24 Programa de embaralhamento e distribuição de cartas (parte 1 de 2).

```
1  // Fig. 5.24: fig05_24.cpp
2  // Programa de embaralhamento e retirada de cartas
3  #include <iostream>
4
5  using std: :cout;
6  using std::ios;
7
8  #include <iomanip>
9
```



```
10 using std::setw;
11 using std::setiosflags;
12
13 #include <cstdlib>
14 #include <ctime>
15
16 void shuffle(int [][ 13 1 );
17 void deal( const int [][13], const char *[], const char *[] );
18
19 int main()
20 {
21 const char *sujt[ 4 ] =
22 { "Copas", "Ouros", "Paus", "Espadas"
23 const char *face[ 13 ] =
24 { "Ás", "Dois", 'Três", "Quatro",
25 "Cinco", "Seis", "Sete", "Oito",
26 "Nove", "Dez", "Valete", "Dama", "Rei" };
27 intdeck[4][13]={0);
28
29 srand( time( 0 ) );
30
31 shuffle( deck );
32 deal( deck, face, suit );
33
34 return 0;
35 }
36
37 void shuffle(int wDeck[][ 13
```

```
38
39 int row, coluinn;
40
41 for (int card =1; card <=52; card++
42 do{
43 row =rand() % 4;
44 column =rand % 13;
45 )while( wDeck[ row ][ coluinn ]!=0 );
46
47 wDeck[ row ][ column ]=card;
48
49 }
50
51 void deal( const int wDeck[][ 13 ], const char *wFace[J,
52 const char *wSujt[]
53 {
54 for ( int card =1; card <=52; card++
55
56 for ( int row =0; row <=3; row++
57
58 for ( int column = 0; column <= 12; coluxnn++
59
60 if ( wDeck[ row ][ column ] == card
61 cout« setw( 6 ) « setiosflags( ios::right
62 « wFace [ column 1 « ·de
63 « setw( 7 ) « setiosflags( ios::left
64 « wSuit[ row
65 « ( card % 2 ==0 ? '\n' : '\t' )
66
```

Fig. 5.24 Programa de embaralhamento e distribuição de cartas (parte 2 de 2).


```
Seis de Paus Sete
Ás de Espadas As
Ás de Copas Dama
Dama de Paus Sete
Dez de Copas Dois
Dez de Espadas Tres
Dez de Ouros Quatro
Quatro de Ouros Dez
Seis de Ouros Seis
Oito de Copas Tres
Nove de Copas Tres
Dois de Ouros Seis
Cinco de Paus Oito
Dois de Ouros Oito
Cinco de Espadas Rei
Rei de Ouros Valete
Dois de Copas Dama
Ás de Paus Rei
Tres de Paus Rei
Nove de Paus Nove
Quatro de Copas Dama
Oito de Ouros Nove
Valete de Ouros Sete
Cinco de Copas Cinco
Quatro de Paus Valete
Valete de Paus Sete
```

Fig. **5.25** Exemplo de execução do programa de embaralhamento e distribuição de cartas.

Existe uma deficiência no algorilmo desenvolvido acima. Depois que uma é encontrada, ainda que seja encontrada na primeira tentativa, as duas estruturas for internas continuam a procurar nos elementos restantes do **deck uma** carta igual. Nos exercícios, corrigimos esta deficiência.

5.11 Ponteiros de função

Um ponteiro para uma função contém o endereço da função na memória. No Capítulo 4, vimos que um nome de array é, na realidade, o endereço na memória do primeiro elemento do array. De modo semelhante, um nome de função é realmente o endereço na memória do começo do código que executa a tarefa da função. Ponteiros de funções podem ser passados para funções, retornados por funções, armazenados em arrays e atribuídos a outros ponteiros de função.

Para ilustrar o uso de ponteiros de funções, modificamos o programa da Fig. 5.15 criando o programa da Fig. 5.26. Nosso novo programa consiste em main e as funções bubble, swap, ascending e descending. A função bubbleSort recebe um ponteiro para uma função - ou para a função ascending ou para a função descending - como argumento (além de um array de inteiros e do tamanho do array, também passados como argumentos). O programa solicita ao usuário escolher se o array deve ser classificado em ordem ascendente ou em ordem descendente. Se o usuário digitar 1, um ponteiro para a função ascending é passado para a função bubble, fazendo com que o array seja classificado em ordem crescente. Se o usuário digitar 2, um ponteiro para a função **`descending`** é passado para a função bubble. fazendo com que o array seja classificado em ordem decrescente. A saída do programa é mostrada na Fig. 5.27.

`de Ouros` **de Ouros de Ouros de** **`Copas`** **de Paus de Espadas de Espadas de Paus de Espadas** **`de Ouros`** **de Copas de Copas de Paus** **`de Espadas`** **de Paus de Espadas de Espadas de Espadas de Copas de Espadas de Espadas de Ouros** **`de Paus de Ouros`** **de Copas de Espadas**

3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30

31

32
33
34
35

36

37

```
38
39


40
41
42
43
44


45
46
47
48
49
50
51
52
53
54
55
56
57


1 // Fig. 5.26: fig0526.cpp
2 // Programa de classificação de finalidade múltipla, usando ponteiros para função


58
59
60
61
```

Fig. 5.26 Programa de classificação de finalidade múltipla usando ponteiros para função (parte 1 de 3).


```
3 #include <iostream>
4
5 using std::cout;
6 using std::cin;
7 using std::endl;
8
9 #include <iomanip>
10
11 using std::setw;
12
13 void bubble( int [], const int, bool (*) ( int, int ) );
14 bool ascending( int, int );
15 bool descending( int, int );
16
17 int main()
3 18
19 const int arraySize = 10;
20 int order,
21 counter,
22 a[ arraySize ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
23
24 cout « "Digite 1 para classificação em ordem ascendente,\n"
25 « "Digite 2 para classificação em ordem descendente: ";
26 cm » order;
27 cout « "\nDados na ordem original\n";
28
```

```
29 for ( counter =0; counter < arraySize; counter++
30 cout « setw( 4 )« a[ counter J;
31
32 if ( order ==1)
33 bubble( a, arraySize, ascending );
34 cout « '\nDados em ordem ascendente\n";
35 }
36 else {
37 bubble( a, arraySize, descending );
38 cout « "\nDados em ordem descendente\n";
39 }
40
41 for ( counter 0; counter < arraySize; counter++
42 cout « setw( 4 )« a[ counter 1;
43
44 cout « endl;
45 return 0;
46
47
48 void bubble( int work[ ], const int size,
49 bool (*compare) ( int, int)
50 {
51 void swap( int *const, int *const); // protótipo
52
53 for ( int pass =1; pass < size; pass++
54
55 for ( int count =0; count < size -1; count++
56
57 if ( (*compare) ( work[ count ], work[ count + 1 ]
58 swap( &work[ count ], &work[ count + 1 ] );
59 {
60
61 void swap( int *const elementlPtr, int *const element2Ptr
```

Fig. 5.26 Programa de classificação de finalidade múltipla usando ponteiros para função (parte 2 de 3).


```
62 {
63 int temp;
64
65 temp =*elementlptr;
66 *elementlptr =*element2ptr;
67 *element2ptr =temp;
68 }
69
70 bool ascending( int a, int b
71
72 return b < a; // troca se b é menor que a
73 }
74
75 bool descending( int a, int b
76
77 return b > a; // troca se b é maior que a
78 )
```

Fig. 5.26 Programa de classificação de finalidade múltipla usando ponteiros para função (parte 3 de 3).

```
Digite 1 para classificação em ordem ascendente,
Digite 2 para classificação em ordem descendente: 1
Dados na ordem original
2 6 4 8 10 12 89 68 45 37
Dados em ordem ascendente
2 6 4 8 10 12 37 45 68 89
```

```
Digite 1 para classificação em ordem ascendente,
Digite 2 para classificação em ordem descendente: 2
Dados na ordem original
2 6 4 8 10 12 89 68 45 37
Dados em ordem descendente
89 68 45 37 12 10 8 6 4 2
```

Fig. 5.27 Saídas do programa *bubble sortda* Fig. 5.26.

O parâmetro seguinte aparece no cabeçalho de função para bubble:

bool (*compare) (int, int

Isto diz para bubble esperar um argumento que é um ponteiro para uma função que recebe dois argumentos de tipo inteiro, retornando um resultado de tipo booleano. Os parênteses são necessários em volta de * compare porque * tem uma precedência mais baixa que os parênteses em volta dos argumentos da função. Se não incluíssemos os parênteses, a declaração teria sido

```
bool *conpare( int, int
```

que declara uma função que recebe dois inteiros como argumentos e retorna um ponteiro para um booleano.

O argumento correspondente, no protótipo de função de bubble, é

bool (*) (int, int

Note que foram incluídos só tipos, mas para fins de documentação o programador pode incluir nomes que o compilador ignorará.

A função passada para bubble é chamada em um comando if como segue

```
jf ( ( *compare )( work[ courit ),  work[ count + 1]
```

Da mesma maneira que um ponteiro para uma variável é derreferenciado para acessar o valor da variável, um ponteiro para uma função é derreferenciado para executar a função.

A chamada para a função podia ter sido feita sem derreferenciar o ponteiro, como em

```
if ( compare( work[ count 1, work[ count + 1
```

que utiliza o ponteiro diretamente como o nome da função. Preferimos o primeiro método de chamar uma função através de um ponteiro, porque explicitamente mostra que compare é um ponteiro para uma função que é derreferenciado para chamar a mesma. O segundo método de chamar uma função através de um ponteiro faz compare se parecer com uma função real. Isso pode ser confuso para um usuário do programa que gostaria de ver a definição da função compare e acha que ela nunca é definida no arquivo.

Um uso para ponteiros de função é encontrado em sistemas orientados por menus. Um usuário é solicitado a selecionar uma opção de um menu (por exemplo, de 1 até 5). Cada opção é tratada por uma função diferente. Os ponteiros para cada função são armazenados em um array de ponteiros para funções. A escolha do usuário é usada como subscrito do array e o ponteiro que está na posição determinada por esse subscrito no array é usado para chamar a função.

O programa da Fig. 5.28 fornece um exemplo genérico do mecanismo de declarar e usar um array de ponteiros para funções. Três funções são definidas - functionl, function2 e function3 - cada uma aceitando um argumento do tipo inteiro e não retornando nada. Os ponteiros para estas três funções são armazenados no array f, que é declarado como segue:

void (*f[3]) (int) = { functionl, function2, function3 };

A declaração é lida começando no conjunto mais à esquerda de parênteses, como "f é um array de 3 ponteiros para funções que aceitam um int como argumento e retornam void". O array é inicializado com os nomes das três funções (que, repetimos, são ponteiros). Quando o usuário digita um valor entre O e 2, o valor é usado como o subscrito do array de ponteiros para funções. A chamada da função é feita como segue:

```
*f[ choice ])( choice );
```

Na chamada, f [choice] seleciona o ponteiro nacposição choice do array. O ponteiro é derreferenciado para chamar a função e choice é passado como argumento para a função. Cada função imprime o valor do seu argumento e seu nome de função para indicar que a função foi chamada corretamente. Nos exercícios, você desenvolverá um sistema orientado por menus.

Fig. 5.28 Demonstrando um array de ponteiros para funções (parte 1 de 2).

1	// Fig. 5.28: figo5_28.cpp		
2	// Demonstrando um	array	de ponteiros para funções
3	#include <iostream>		
4			
5	using std: :cout;		

6	using std::cin;		
7	using std::endl;		
8			
9	void functionl (int	);	
10	void function2 (int	);	
11	void function3(int	);	
12			
13	int main O		


```
14 { sam
15 void (*f[ 3 ])( int ) = { functionl, function2, function3 }; comi
16 int choice;
17
18 cout« "Digite um número entre O e 2, 3 para terminar: "; 51
19 cm » choice;
20
21 while ( choice >= O && choice < 3 ) { Osc
22 (*f[ choice J)( choice ) deui
23 cout« "Digite um número entre O e 2, 3 para terminar: "; dorc
24 cm » choice; res.1
25 } deu
26 repo
27 cout « "Execução do programa terminada." « endi; den
28 return 0;
29 } car
30 aspa
31 void functionl( int a
32
33 cout « "Você digitou " « a
34 « ", de modo que functionl foi chamada.\n\n";
35
36
37 void function2( int b)
38 { Um
39 cout« "Você digitou " « b deu
40 « ", de modo que function2 foi chamada.\n\n"; cara
41 } poni
42 arra
43 void function3( int c
44 { tipo
45 cout « "Você digitou " « c
46 « ", de modo que function3 foi chamada.\n\n";
47
```

inic

Digite um número entre O e 2, 3 para terminar: O dei

```
Você digitou 0, de modo que functionl foi chamada. co:
Digite um número entre O e 2, 3 para terminar: 1
Você digitou 1, de modo que function2 foi chamada.
Digite um número entre O e 2, 3 para terminar: 2
Você digitou 2, de modo que function3 foi chamada.
Digite um número entre O e 2, 3 para terminar: 3
Execução do programa terminada
```

Fig. 5.28 Demonstrando um array de ponteiros para funções (parte 2 de 2).

Q

na

5.12 Introdução ao processamento de caracteres e *strings* ba

Nesta seção, introduzimos algumas funções comuns da biblioteca padrão que facilitam o processamento de *strings.*
As técnicas discutidas aqui são apropriadas para desenvolver editores de texto, processadores de texto, software para
fazer layout de páginas, sistemas computadorizados para composição de textos e outros tipos de software de proces

sarnento de texto. Aqui, usamos strings baseados em ponteiros. Mais à frente, neste livro, incluímos um capítulo completo sobre strings como objetos no pleno sentido.

5.12.1 Fundamentos de caracteres e *strings*

Os caracteres são os blocos de construção fundamentais dos programas-fonte em C++. Todo programa é composto de uma seqüência de caracteres que - quando agrupados de modo que façam sentido - é interpretada pelo computador como urna série de instruções usadas para realizar uma tarefa. Um programa pode conter ***constantes de caracteres.*** Urna constante de caractere é um valor do tipo inteiro representado corno um caractere entre apóstrofes. O valor de uma constante de caractere é o valor inteiro do caractere no conjunto de caracteres da máquina. Por exemplo, z representa o valor em números inteiros de z (122 no conjunto de caracteres ASCII) e '\n' representa o valor inteiro de nova linha (lO no conjunto de caracteres ASCII).
Um string é urna série de caracteres tratados corno uma unidade. Um string pode incluir letras, dígitos e vários *caracteres especiais,* tais como +, -, , / e $. Os *strings literais* ou *constantes string* são escritos em C++ entre aspas, como segue:

'João **da** Silva" (umnorne)
"Rua A, 1234" (um endereço de rua)
"Porto Alegre, RS" (uma cidade e um estado)
"(51) 654.3210" (um núrnero de telefone)
Um *string* em C++ é um array de caracteres terminado com o *caractere nulo* (' \O'). Um *string* e acessado atraves de um ponteiro para o primeiro caractere no *string.* O valor de um *string* é o endereço (constante) de seu primeiro caractere. Deste modo, em C++, é apropriado dizermos que *um string é um ponteiro constante* - na verdade, um ponteiro para o primeiro caractere do *string.* Neste sentido, *strings* são semelhantes a arrays, porque um nome de array é também um ponteiro constante para o seu primeiro elemento.
Um string pode ser inicializado em uma declaração ou atribuído a um array de caracteres ou a uma variável do tipo char . As declarações

```
char color[ ] = "blue";
const char *colorptr = 'blue";
```

inicializam, cada uma, uma variável para o *string* "blue". A primeira declaração cria um array color de 5 elementos contendo os caracteres 'b'. '1', 'u', 'e' e \0 . A segunda declaração cria a variável ponteiro **colorPtr que** aponta para o `string "blue" em algum lugar da memória.`

Dica de portabilidade ***5.5***

___ *Quando uma variável do tipo* **char** * *é inicializada com um* string *literal, alguns compiladores podem colocar o* string *em uma posição na memória em que o* string *não pode ser modificado. Se você precisa modificar um* string *literal, deve armazená-lo em um array de caracteres para assegurar a possibilidade de mudança em todos os sistemas.*

A declaração char color ***/ /*** = `"blue" poderia também ter sido escrita como`

```
char color[ ] = { 'b', 'l','u'. 'e'. '\O' };
```

`Quando declaramos um array de caracteres para guardar um string, o array deve ser grande o bastante para armazenar` o string e seu caractere terminal nulo. A declaração precedente determina o tamanho do array automaticamente baseando-se no número de inicializadores fornecidos na lista de inicializadores.

Erro comum de programação 5.15

Não alocar espaço suficiente em um array de caracteres para armazenar o caractere nulo que termina um string.

Erro comum de programação 5.16

Criar ou usar um "string" *que não contém um caractere terminal nulo.*

Boa prática de programação 5.5

Quando armazenar um string *de caracteres em um array de caracteres, esteja certo de que o array é grande o bastante para manter o maior* string *que será armazenado. C+ + permite que* strings *de qualquer comprimento sejam armazenados. Se um* string *é mais longo que o array de caracteres em que ele é armazenado, os caracteres além dofim do array serão gravados em cima dos dados que estiverem após o array na memória.* **5.12.2**

`Um` *string* pode ser atribuído a `um` array **usando a extração de** *stream* com cm. Por exemplo, o comando seguinte A biblioti

pode ser usado para atribuir um *string* para o array de caracteres palavra ***/ 20 J :*** procurar
comprim

cm **» palavra; tratameni**
Nc
O *string* fornecido como entrada pelo usuário é armazenado em palavra. O comando precedente lê caracteres até arquivo d
encontrar um espaço, marca de tabulação, nova linha ou indicador de fim de arquivo. Note que o *string* não deveria vos de ca
ter mais que 19 caracteres para deixar um espaço para o caractere terminal nulo. O manipulador de *stream* setw, ned int

introduzido no Capítulo 2, pode ser usado para assegurar que o *string* armazenado em palavra não exceda o tamanho do array. Por exemplo, o comando
`cm » setw( 20 ) » palavra;`
especifica que cm deve ler um máximo de 19 caracteres no array palavra e reservar a 20 `posição no` array para armazenar o caractere nulo terminal do *string.* O manipulador de *stream* **setw** só se aplica ao próximo valor fornecido como entrada.
Em alguns casos, é desejável fornecer uma linha inteira de texto como entrada para um array. Para este fim, C++ oferece a função cm .getline. A função cm .getline aceita três argumentos - um array de caracteres em que a linha de texto será armazenada, um comprimento e um caractere delimitador. Por exemplo, o segmento de programa
char
```
char frase[ 80 J;
cin.getline( frase, 80, \n' );
```
char
declara o array frase de 80 caracteres e então lê uma linha de texto do teclado para o array. A função pára de ler caracteres quando *é* encontrado o caractere delimitador '\n' .quando o indicador de fim de arquivo é inserido pelo usuário ou quando o número de caracteres lido até então é um a menos que o comprimento especificado no segundo argumento (o último caractere no array *é* reservado para o caractere terminador nulo). Se o caractere delimitador *é* char encontrado, ele ***é*** lido e descartado, O terceiro argumento para a ci getline tem '\n' como valor default, de modo que a chamada de função precedente poderia ter sido escrita como segue:
cin.getline (frase, 80);
`int si`
`O Capítulo 11` fornece `uma discussão detalhada de cm .getline e outras funções de entradalsaída.`

Erro comum de programação 5.17

Processar um caractere único como um `string` *pode levar a um erro fatal durante a execução. Um* string *é* **int S** *um ponteiro - provavelmente um inteiro consideravelmente grande. Porém, um caractere é um inteiro pequeno (valores ASCÍI variam de O* a *255). Em muitos sistemas, isso causa um erro porque os endereços de memória baixos são reservados para propósitos especiais, tais como para tratadores de interrupções do sistema operacional - assim, ocorrem "violações de acesso -.*

Fig. 5.

IErro comum de programação 5.18

Passar um caractere como argumento para uma função, quando é esperado um string, *pode levar a um erro fatal durante a execução.*

Erro comum de programação 5.19

Passar um string *como argumento para uma fim ção, quando é esperado um caractere, é um erro de sintaxe.*

5.12.2 Funções de manipulação de *strings* da biblioteca de tratamento de *strings*

A biblioteca de tratamento de *strings* fornece muitas funções úteis para manipular dados de *strings,* comparar *strings,* procurar em *strings* por caracteres e outros *strings,* separação de *strings* em "unidades léxicas" e determinar o comprimento de *strings.* Esta seção apresenta algumas funções comuns de manipulação de *strings* da biblioteca de tratamento de *strings* (parte da biblioteca badrão). As funções são resumidas na Fig. 5.29.
Note que várias funções na Fig. 5.29 contêm argumentos com tipo de dados **size_t.** Este tipo é definido no arquivo de cabeçalho <cstddef> (um arquivo de cabeçalho da biblioteca padrão incluído em muitos outros arquivos de cabeçalho da biblioteca padrão, inclusive <cstring>) como um tipo inteiro sem sinal, tal como unsig- ned `int ou uns igned long.`

Erro comum de programação 5.20

Esquecer de incluir o arquivo de cabeçalho <cstring> *quando usarfunções da biblioteca de tratamento de* strings.

Fig. 5.29 Funções de manipulação de *strings* da biblioteca de tratamento de *strings* (parte 1 de 2).

Prot	ótipo de função Descrição da função
char	*strcpy(char *si, const char *s2
	Copia o *string* s2 no array de caracteres si. Retorna o valor de si.
char	*strncpy(char *si, const char *52, size_t ri
	Copia no máximo n caracteres do *string* s2 para o array de caractere si.
	Retorna o valor de si.
char	*strcat(char *si, const char *s2);
	Anexa a *string* s2 para o *string* si. O primeiro caractere de s2 serão gravados
	em cima do caractere nulo terminal de si. Retorna o valor de si.
char	*strncat(char *si, const char*s2, size_t ri);
	Anexa no máximo n caracteres de *string* s2 para o *string* si. **O** primeiro
	caractere de s2 será gravado por cima do caractere nulo terminal de si.
	Retorna o valor de si.
int	strcmp(const char *si const char*s2);
	Compara o *string* si com o *string* s2. A função retorna um valor O, menor
	que O, ou maior que O, se si for igual a, menor que, ou maior que s2,
	respectivamente.
int	strncmp(const char *si, const char*s2, size_t n);
	Compara até n caracteres do *string* si como o *string* **s2.** A função retorna
	O, menor que O, ou maior que O se si é igual a, menor que, ou maior que **s2,**
	respectivamente.

char *strtok(char *si, const char *s2);

Uma seqüência de chamadas para s trtok quebra o *string* si em "tokens"

-unidades lógicas tais como palavras em uma linha de texto - separados por caracteres contidos no *string* s2. A primeira chamada tem si como o primeiro argumento, e as chamadas subseqüentes para continuar separando em 'tokens" *["tokenizing"j* o mesmo *string* contém NULL como o primeiro argumento. Um ponteiro para o "token" atual **é** retornado por cada chamada. Se não existirem mais "tokens" quando a função é chamada, NULL *é* retornado.

size_t strlen(const char *

Determina o comprimento do *string* s. Retorna o número de caracteres que precedem o caractere nulo terminal.

Fig. 5.29 Funções de manipulação de *strings* da biblioteca de tratamento de *strings* (parte 2 de 2).

A função strcpy copia seu segundo argumento - um *string* - para o seu primeiro argumento - um array de caracteres que deve ser grande o bastante para armazenar o *string* e o seu caractere terminal nulo, que também é copiado. A função strncpy é equivalente a strcpy a não ser pelo fato de que strncpy especifica o número de caracteres a serem copiados do *string* para o array. Note que a função strncpy não copia necessariamente o caractere terminal nulo de seu segundo argumento. O caractere terminal nulo é escrito só se o número de caracteres a ser copiado é pelo menos um a mais do que o comprimento do *string.* Por exemplo, se "teste" é o segundo argumento, um caractere nulo terminal é escrito só se o terceiro argumento de strncpy for pelo menos 6 (5 caracteres em "teste", mais 1 caractere terminal nulo). Se o terceiro argumento for maior do que 6, são acrescentados caracteres nulos ao array até que o número total de caracteres especificados pelo terceiro argumento sejam escritos.

Erro comum de programação 5.21

Não anexar um caractere nulo terminal ao primeiro argumento de uma strncpy. quando o terceiro argumento é menor ou igual ao comprimento do string *no segundo argumento, pode causar erros fatais durante a execução.*

O programa na Fig. 5.30 usa strcpy para copiar o *string* inteiro do array x para o array y e usa strncpy para copiar os primeiros 14 caracteres do array x para o array z. O caractere nulo **('** \0') é acrescentado ao array z porque a chamada a strncpy no programa não escreve um caractere nulo terminal (o terceiro argumento é menor que o comprimento do *string* do segundo argumento).

1 *II* **Fig. 5.30: figO53O.cpp**

```
2 // Usando strcpy e strncpy
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <cstring>
9
10 int main O
ii
i2 char x[] "Programando em C++";
i3 char y[ 25 ], z[ 15 ];
14
iS cout « "O string no array x é: « x
i6 « "\nO string no array y é: « strcpy( y, x
```

Fig. 5.30 Usando strcpy e strncpy (parte 1 de 2). +


```
17 « \n;
18 strncpy( z, x, 14 ); // não copia o caractere nulo
19 z[ 14 :i =
1 20 cout « "O string no array z é: « z « endl;
21
r- 22 return 0;
a. 23 }
```

Fig. 5.30 Usando strcpy e strncpy (parte 2 de 2).

A função strcat concatena seu segundo argumento - um *string* - ao seu primeiro argumento - um array de caracteres contendo um *string.* O primeiro caractere do segundo argumento substitui o caractere nulo (' \0 ') que
le termina o *string* no primeiro argumento. O programador deve assegurar que o array usado para armazenar o primeiro *string* é grande o bastante para armazenar a combinação do primeiro *string* com o segundo *string* e,
[e ainda, o caractere terminal nulo (copiado do segundo *string).* A função strncat concatena um número especificado de caracteres do segundo string ao primeiro *string.* Um caractere terminal nulo é acrescentado ao resultado. O programa da Fig. 5.31 demonstra a função strcat e a função strncat.

```
1 // Fig. 5.31: figO53l.cpp
2 // Usando strcat e strncat
3 #inciude <iostream>
4
5 using std::cout;
6 using std: :endl;
7
8 #include <cstring>
9
10 int main()
11 {
a 12 char si[ 20 ] = "Feliz ";
13 char s2[] = "Ano Novo ";
14 char s3[ 40 ) = "";
15
16 cout « "si = " « si « "\ns2 = " « s2;
17 cout« "\nstrcat(si, s2) = « strcat( si, s2 );
18 cout« "\nstrncat(s3, si, 6) = « strncat( s3, si, 6 );
19 cout« "\nstrcat(s3, si) = « strcat( s3, si ) « endl;
20
21 return 0;
22 }
```

Fig. 5.31 Usando strcat e strncat.

A Fig. 5.32 compara três *strings* usando strcmp e strncmp. A **função strcmp compara seu primeiro argumento** *string* a seu segundo argumento *string,* caractere por caractere. A função retorna zero se os *strings* são iguais, um valor negativo se o primeiro *string* é menor que o segundo *string* e um valor positivo se o primeiro *string* é maior que

```
si = Feliz
s2 = Ano Novo
strcat (si, s2) = Feliz Ano Novo strncat (s3, si, 6) = Feliz
```

```
strcat (s3, si) = Feliz Feliz Ano Novo
```

O	string	no	array x é:	Programando	em	C++
O	string	no	array y é:	Programando	em	C++
O	string	no	array z é:	Programando	em	

o segundo *string.* A função strncmp é equivalente strcmp, a não ser pelo fato de que strncmp faz a comparação até um número especificado de caracteres. A função strncmp não compara os caracteres seguintes a um caractere nulo em um *string.* O programa imprime o valor de tipo inteiro retornado por cada chamada da função.

```
1 // Fig. 5.32: figo5_32.cpp
2 // Usando strcmp and strncmp
3 #include <iostream>
4
5 using std::cout;
6 using std: :endl;
7
8 #include <iomanip>
9
10 using std::setw;
ii
12 #include <cstring>
13
14 int main O
15 {
16 char *si = "Feliz Ano Novo";
17 char *s2 = "Feliz Ano Novo";
18 char *3 = "Feliz Natal";
19
20 cout « "si =" « si « "\ns2 =" « s2
21 « "\ns3 =" « s3 « "\n\nstrcinp(si, s2) =
22 « setw ( 2 ) « strcmp( si, s2
23 « "\nstrcmp(si, s3) =" « setw(2)
24 « strcmp( si, s3 ) « "\nstrcmp(s3, si) =
25 « setw( 2 ) « strcnip( s3, si );
26
27 cout« "\n\nstrncntp(si, s3, 6) =" « setw( 2
28 « strncmp( si, s3, 6 ) « "\nstrncmp(si, s3, 7) =
29 « setw( 2 ) « strncmp( si, s3, 7
30 « "\nstrncmp(s3, si, 7) =
31 « setw( 2 ) « strncmp( s3, si, 7 ) « endl;
32 return 0;
33 1
si =Feliz Ano Novo
s2 =Feliz Ano Novo
s3 =Feliz Natal 2K
strcmp(si, s2) O
strcmp(si, s3) =-i
strcmp(s3, si) =i
strncmp(si, s3, 6) =O
strncmp(si, s3, 7) =-i
strncmp(s3, si, 7) =1
```

Fig. 5.32 Usando strcmp e **strncmp.**

Erro comum de programação 5.22

Assumir que strcmp *e* strncmp *retornam um quando seus argumentos forem iguais* é *um erro de lógica. Ambas as funções retornam zero (o valor falso em C++) para igualdade. Então, quando* strings *são testados quanto* à *igualdade, o resultado das funções* strcmp *ou* strncmp *devem ser comparados com* O *para determinar se os* strings *são iguais.*

Só para entender o que quer dizer um *string* ser "maior do que" ou "menor do que" outro *string,* considere o processo de colocar uma série de sobrenomes em ordem alfabética. O leitor iria, sem dúvida alguma, colocar "Maria" antes de "Pedro", porque a primeira letra de "Maria" vem antes da primeira letra de "Pedro" no alfabeto. Mas o alfabeto é mais do que só uma lista de 26 letras - é uma lista ordenada de caracteres. Cada letra ocupa uma posição específica dentro da lista. **"Z"** é mais que do só uma letra do alfabeto; "Z" é, especificamente, a 26 letra do alfabeto.

Como o computador sabe que uma letra vem antes de outra? Todos os caracteres são representados dentro do computador como códigos numéricos; quando o computador compara dois *strings,* ele compara na verdade os códigos numéricos dos caracteres nos *strings.*

Dica de portabilidade 5.6

___ Os códigos numéricos internos usados para representar caracteres podem ser diferentes em computado- res dfe rentes.

Dica de portabilidade 5.7

___Nãoftça testes explícitos do valor do código de caracteres em ASCII, como por exemplo em: if (ch == 65) ; *em vez disso, use sempre a constante de caractere correspondente, como em:* if (ch == 'A').

Em um esforço para padronizar as representações de caracteres, a maioria dos fabricantes de computador projetou suas máquinas para utilizar um dos dois esquemas populares de codificação - *ASCI!* ou *EBCDJC.* ASCII significa "American Standard Code for Information Interchange" e EBCDIC significa "Extended Binary Coded Decimal Interchange Code". Existem outros esquemas de codificação, mas estes dois são os mais populares.

ASCII e EBCDIC são chamados de *códigos de caracteres* ou *conjuntos de caracteres.* As manipulações de strings e caracteres envolvem, na realidade, a manipulação dos códigos numéricos apropriados e não dos caracteres. Isto explica a intercambialidade de caracteres e valores inteiros pequenos em C++. Uma vez que é significativo se dizer que um código numérico é maior do que, menor do que ou igual a outro código numérico, torna-se possível relacionar vários caracteres ou strings um ao outro, referindo-se aos códigos dos caracteres. O Apêndice B contém os códigos de caracteres em ASCII.

A função strtok é usada para quebrar um *string* em uma série de "unidades léxicas". Uma unidade léxica é uma seqüência de caracteres separados por *caracteres delimitadores* (normalmente por espaços ou sinais de pontuação). Por exemplo, em uma linha de texto, cada palavra pode ser considerada uma unidade léxica e os espaços que separam as palavras podem ser considerados delimitadores.

São necessárias múltiplas chamadas a strtok para quebrar um *string* em unidades léxicas (assumindo-se que o string contenha mais do que uma unidade léxica). A primeira chamada a strtok contém dois argumentos, um string para ser separado em unidades léxicas e outro *string* contendo caracteres que separam as unidades léxicas (i.e., delimitadores). No programa da Fig. 5.33, o comando

tokenPtr = strtok(string, -

atribui a tokenPtr um ponteiro para a primeira unidade léxica em string. O segundo argumento de strtok, indica que as unidades léxicas em string são separadas por espaços. A função strtok procura pelo primeiro caractere em string que não é um caractere delimitador (espaço). Esse começa a primeira unidade léxica. A função então acha o próximo caractere delimitador no *string* e substitui este por um caractere nulo (' \O'). Este termina a unidade léxica atual. A função strtok salva um ponteiro para o próximo caractere que vem em seguida à unidade léxica em string, retornando um ponteiro para a unidade léxica atual.

As chamadas subseqüentes a strtok para continuar a separar string em unidades léxicas contêm NULL como o primeiro argumento. O argumento NULL indica que a chamada a strtok deve continuar o processo de separação em unidades léxicas a partir da posição em string salva pela última chamada a strtok. Se não restam mais unidades léxicas quando strtok é chamada, strtok retorna NULL. O programa da Fig. 5.33 usa strtok para separar em unidades léxicas o *string* **"Esta** é urna frase com 8 unidades léxicas". Cada unidade léxica é impressa separadamente. Note que strtok modifica o *string* fornecido como entrada; por isso, se o *string* precisar ser usado novamente no programa, após a chamada a strtok, deve ser feita antes uma cópia do *string.*


```
1 // Fig. 5.33: fig0533.cpp
2 // Usando strtok 6 u
3 #include <iostrearn> 7
5 using std::cout; 9
6 using std::endl; 10 i
8 #include <cstring> 11 {
10 int main O 14
11 { 15
12 char string[] = Esta é urna frase com 8 unidades léxicas"; 16
13 char *tokenptr; 17
14 18
15 cout« "O string a ser separado em unidades léxicas é:\n" « string 19
```

```
16 « "\n\nAs unidades léxicas são:\n"; 20
17 21
18 tokenPtr = strtok( string, -22
20 while ( tokenPtr != NULL ){
21 cout « tokenPtr « '\n';
22 tokenPtr = strtok( NULL, --
25 return 0;
26
```

Fig.5

```
O string a ser separado em unidades J.éxicas é:
Esta é urna frase com 8 unidades léxicas
As unidades léxicas são: 5 13
Esta cIa
uma
frase Esta é
com orient
8 tecnic
unidades Parac
léxicas Incluí
```

biblio,

Fig. 5.33 Usando strtok.

cam u
erece

Erro comum de programa ção 5.23

Não perceber que strtok *modifica o* string *que será separado em unidades léxicas e, então, tentar usar esse* string *como se ele fosse o* string *original inalterado.*

A função strlen aceita um *string* como argumento e retorna o número de caracteres no *string* - o caractere nulo terminal não é incluído no comprimento. O programa da Fig. 5.34 demonstra a função strlen. A mci no fim nos m

```
1 // Fig. 5.34: fig0534.cpp Remo
2 // Usando strlen emno
3 #include <iostrearn> classe hora"
```

Fig. 5.34 Usando **strlen** (parte 1 de 2).


```
4
5 using std::cout;
6 using std: :endl;
7
8 #include <cstring>
9
10 int main O
11
12 char *stringl = "abcdefghijklmnopqrstuvwxyz";
13 char *string2 "cinco;
14 char *string3 = 'Porto Alegre";
15
16 cout « "O comprimento de \" « stringl
17 « "V' é -« strlen( stringl
18 « "\nO comprimento de \" « string2
g 19 « \" é -« strlen( string2
20 « "\nO comprimento de \" « string3
21 « \" é -« strlen( string3 ) « endi;
22
23 return 0;
24
```

Fig. 5.34 Usando **strlen** (parte 2 de 2).

5.13 (Estudo de caso opcional) Pensando em objetos: colaborações entre objetos

Esta é a última tarefa do nosso projeto orientado a objetos antes de começarmos nosso estudo da

programação orientada a objetos em C++ no Capítulo **6.** Depois de discutirmos as colaborações entre objetos nesta seção e técnicas de OOP no Capítulo 6, você estará preparado para começar a codificar o simulador de elevador em C÷+. Para completar o simulador de elevador, você também precisará das técnicas de C++ discutidas nos Capítulos 7 e **9.** Incluímos no fim dessa seção uma lista de recursos para UML existentes na lnternet e na World Wide Web e uma bibliografia de referência para UML.
Nesta seção, concentramo-nos nas colaborações (interações) entre objetos. Quando dois objetos se comunicam um com o outro para executar determinada tarefa, diz-se que eles *colaboram* - os objetos fazem isso enviando e recebendo mensagens. Uma *colaboração* consiste em:
usar 1. um objeto de uma classe
2. envia uma mensagem particular
3. para um objeto de uma outra classe
nulo
A mensagem enviada pela primeira classe invoca uma operação da segunda classe. Na seção "Pensando em objetos" no fim do Capítulo *4,* determinamos muitas das operações das classes em nosso sistema. Nesta seção, concentramonos nas mensagens que invocam estas operações. A Fig. 5.35 é a tabela de classes e frases com verbos da Seção 4.10. Removemos todas as frases com verbos que não correspondem a operações. As frases restantes são as colaborações **em** nosso sistema. Associamos as frases "fornece a hora para o *scheduler"* e "fornece a hora para o elevador" com a classe Building porque decidimos que o edifício irá controlar a simulação. Associamos as frases "incrementa a hora" e "obtém a hora" com a classe Building pela mesma razão.

O	**comprimento**	**de**	**"abcdefghijklmnopqrstuvwxyz" é 26**		
O	**comprimento**	**de**	**"cinco"**	**é 5**	
O	**comprimento**	**de**	**"Porto**	**Alegre" é 12**	

Fig. 5.35 Lista modificada de frases com verbos para as classes no sistema.

Examinamos a lista de verbos para determinar as colaborações em nosso sistema. Por exemplo, a classe Elevator lista a frase "desliga o botão do elevador". Para cumprir essa tarefa, um objeto da classe Elevator precisa enviar a mensagem resetButton para um objeto da classe ElevatorButton, invocando a operação resetButton daquela classe. A Fig. 5.36 lista todas as colaborações que podem ser percebidas a partir da nossa tabela de frases.

Um objeto `Floor`
`FloorBul`
`E1evato`

Door
Beil

`Light`

`Bui1din`

Fig. 5.36
Diagrama
Consideren do elevadoi ção. Tanto objetos inte as interaçõt
A Fi que objetos andar. Con retângulo q

Fig. 5.36 Colaborações no sistema do elevador (parte 1 de 2).

Fig. 5.37

Classe	Frases com verbos
Elevator	desliga o botão do elevador, soa a campainha do elevador, sinaliza sua chegada a um andar, abre a porta, fecha a porta
Clock	**bate** a cada segundo
Scheduler	diz a uma pessoa para entrar em um andar, verifica se um andar está desocupado
Person	aperta o botão do andar, aperta o botão do elevador, entra no elevador, sai do elevador
F].oor	desliga o botão do andar, apaga a luz, acende a luz
FloorButton	chama o elevador
ElevatorButton	avisa o elevador para se preparar para sair
Door	(abertura da porta) avisa a pessoa para sair do elevador, (abertura da porta) avisa a pessoa para entrar no elevador
Beil	
Light	
Building	incrementa a hora, obtém a hora, fornece a hora para o *scheduler,* fornece a hora para o elevador

Um objeto da classe	Envia a mensagem	Para um objeto da classe
`Elevator`	`resetButton ringBell elevatorArrived openDoor closeDoor`	`ElevatorButton Beli Floor Door Door`
`Clock`		
`Scheduler`	`stepOntoFloor isOccupied`	`Person Floor`
`Person`	`pressButton pressButton passengerEnters passengerExits personArrives`	`FloorButton ElevatorButton Elevator Elevator FJ.oor`

Fig. **5.36** Colaborações no sistema do elevador (parte 2 de 2).

Diagramas de colabora ções

Consideremos, agora, os objetos que precisam interagir para que as pessoas em nossa simulação possam entrar e sair do elevador quando ele chega em um andar. A UML oferece o *diagrama de colabo rações* para modelar esta interação. Tanto os diagramas de colaborações quanto os diagramas de seqüência fornecem informação sobre como os objetos interagem, mas cada diagrama tem uma ênfase diferente. Diagramas de seqüência se concentram em *quando* as interações ocorrem. Diagramas de colaborações se concentram em *quais objetos participam* da interação.

A Fig. 5.37 mostra um diagrama de colaborações que modela as interações entre objetos no sistema à medida que objetos da classe person entram e saem do elevador. A colaboração começa quando o elevador chega em um andar. Como em um diagrama de seqüência, um objeto em um diagrama de colaborações é representado por um `retângulo que contém o nome do objeto.`

3 : elevatorArrived()
4.1.1: passengerExits()

Fig. 5.37 Diagrama de colaborações para entrada e saída de passageiros.

Um objeto da classe	Envia a mensagem	Para um objeto da classe
Floor	resetButton turnOff turnOn	FloorButton Light Light
FloorButton	summonElevator	Elevator
ElevatorButton	prepareToLeave	Elevator
Door	exitElevator enterElevator	Person Person
Bel].		
Light		
Building	tick getTime processTime processTime	Clock Clock Scheduler Elevator

Objetos que colaboram entre si são conectados por linhas cheias e mensagens são passadas entre os objetos, ao longo destas linhas, na direção mostrada pelas setas. O nome da mensagem aparece próximo à seta.

A *seqüência de mensagens* em um diagrama de colaborações progride, em ordem numérica. do menor para o maior. Neste diagrama, a numeração começa com a mensagem 1 .O elevador envia esta mensagem (resetButton) para o botão do elevador para desligar o botão. O elevador, então, envia a mensagem ringBell (mensagem 2) para a campainha. Então, o elevador avisa o andar de sua chegada (mensagem 3), de modo que o andar possa desligar o botão e acender a luz (mensagens 3.1 e 3.2, respectivamente).

Depois que o andar desligou o botão e acendeu a luz, o elevador abre a porta (mensagem 4). Neste ponto, a ***R*** porta envia a mensagem exitElevator (mensagem 4.1) para o objeto passenger.' O objeto passenger A avisa o elevador de sua intenção de sair através da mensagem passengerExits (mensagem 4.1.1). es

Depois que a pessoa que estava no elevador saiu, a pessoa que está esperando no andar (o objeto pu waitingPassenger) pode entrar no elevador. Observe que a porta envia a mensagem enterElevator (mensagem 4.2) para o objeto waitingPassenger depois que o objeto passenger envia a mensagem passengerExits para o elevador (mensagem 4. 1 .1 **).** Esta seqüência assegura que uma pessoa que está no andar irá esperar até que a pessoa que está no elevador saia antes de entrar no elevador. O objeto waitingPassenger entra no elevador através da mensagem pas sengerEnters (mensagem 4.2.1).

Resumo

Agora, temos uma listagem razoavelmente completa das classes de que precisaremos para implementar nosso simulador de elevador, bem como das interações entre os objetos destas classes. No próximo capítulo, começamos nosso estudo de programação orientada a objetos em C++. Depois de ler o Capítulo 6, estaremos prontos para escrever uma porção substancial do simulador do elevador em C++. Depois de completar o Capítulo 7, implementaremos um simulador de elevador completo que funciona. No Capítulo 9, discutimos como usar herança para explorar o que há de comum entre as classes, a fim de minimizar a quantidade de software necessária para implementar um sistema.

Vamos resumir o processo de projeto orientado a objetos que usamos nos Capítulos 2 a 5.

O. Na fase de análise, reúna-se com os clientes (as pessoas que querem que você construa seu sistema) e obtenha tantas informações sobre o sistema quanto possível. Com essas informações, crie os casos de uso que descrevem as maneiras pelas quais os usuários interagem com o sistema (em nosso estudo de caso, não nos concentramos na fase de análise; os resultados desta fase estão representados na definição do problema e o caso de uso derivado desta definição). Lembramos novamente que sistemas do mundo real freqüentemente têm muitos casos de uso.

1. Comece a localizar as classes no sistema listando os substantivos na definição do problema. Filtre essa lista eliminando substantivos que claramente representam atributos de classes e outros substantivos que claramente não fazem parte do sistema de software que está sendo modelado. Crie um diagrama de classes que modele as classes no sistema e seus relacionamentos (associações).

2. Extraia os atributos de cada classe da definição do problema, listando palavras e frases que descrevem cada classe no sistema.

3. Aprenda mais sobre a natureza dinâmica do sistema. Crie um diagrama de mapa de estados para descobrir como as classes no sistema mudam ao longo do tempo.

4. Examine verbos e frases com verbos associados a cada classe. Use estas frases para extrair as operações das classes em nosso sistema. Diagramas de atividades podem ajudar a modelar detalhes destas operações.

w1

No mundo real, uma pessoa andando no elevador irá esperar até que a porta abra para sair do elevador. Precisamos modelar este comportamen- w1 to; portanto, fazemos a porta enviar uma mensagem para o objeto passenger no elevador. Esta mensagem representa uma indicação visual
para a pessoa no elevador. Quando a pessoa recebe esta indicação, ela pode sair do elevador.

5. Examine as colaborações entre vários objetos. Use diagramas de seqüência e de colaborações para modelar essas interações. Adicione atributos e operações às classes à medida que o processo de projeto revelar a necessidade delas.
6. Nesse ponto, nosso projeto provavelmente ainda tem poucas peças faltando. Estas se tornarão visíveis à medida que implementarmos nosso simulador de elevador em C++, a partir do Capítulo 6.

Recursos de UML na Internet e na World Wide Web

A seguir há uma relação de recursos para a UML na Internet e na World Wide Web. Tais recursos incluem as especificações da UML 1.3 e outros materiais de referência, recursos genéricos, tutoriais, FAQs, artigos, trabalhos publicados e software.

Referências

www. omg org
Omg.org é o site principal do Object Management Group (OMG). O OMG é o grupo responsável por supervisionar a manutenção e as revisões futuras da UML. O site deles na Web contém informações sobre a UML e outras tecnologias de orientação a objetos.
www. rational com
A Rational Software Corporation foi quem desenvolveu inicialmente a UML. Tal site na Web contém informação sobre a UML e seus criadores . Grady Booch, James Rumbaugh e Ivar Jacobson.
www . omg org/cgi-bin/doc?ad/99-06-09
Esse endereço contém versões das especificações oficiais da UML 1.3 nos formatos PDF e ZTP.
www.omg.org/techprocess/meetings/schedule/UMLl.4RTF.htm1
A OMG mantém nesse site informações a respeito das especificações da UML 1.4.
www. rational . com/uml/resources/quick/index. j tmpl
O manual de referência rápida para UML da Rational Software Corporation.
www . holub. com/class/oo design/uml html
Esse sue oferece uma ficha de referência rápida para UML detalhada, com comentarios adicionais.
softdocwiz com/UML . htm
Kendall Scott. autor de diversos recursos para UML, mantém um dicionário de UML nesse site.

Recursos

www. omg. org/uml/
A página de recursos para UML da OMG.
www. rational . com/uml/index. jtmp].
A página de recursos para UML da Rational Software Corporation.
www.p1atinum.com/corp/um1/um1.htxn
A Platinum Technology, sócia da UML Partners, mantém uma página de recursos para UML nesse endereço.
www. cetus-].ink . org/oo um].. htm].
Este site contém centenas de links para sites sobre UML, incluindo informações, tutoriais e software.
www. um].-zone . com
Este site contém uma enorme quantidade de informações sobre UML, incluindo artigos e links para grupos de notícias e para outros sites.

home .pacbeil net/ckobryn/umi htini
Este site é mantido por Cris Kobryn, arquiteto de software com experiência em UML. Contém informações gerais e links para sites importantes na Web.
www . methods-toois com/cgi-bin/DiscussionUML. cgi
Esse site contém a página inicial para um grupo de discussão sobre UML.
www . pois co. uk/usecasezone/index htm

Esse site oferece recursos e artigos sobre aplicações de casos de uso.
www. ics uci edu/pub/arch/uini/umi books and toois htmi
Esse site contém links para informações sobre outros livros sobre a UML, bem como uma lista de ferramentas que suportam a notação da UML.
home earthiink net/-saihir/
Sinan Si Alhir, autor de ***UML*** *in a Nutshell,* mantém um site nesse endereço que inclui links para muitos recursos para UML.

Software

www. rationai com/products/rose/index. tmpl
Este site é a homepage para a ferramenta de modelagem visual para UML, Rational RoseTM, da Rational Software Corporation. Você pode baixar uma versão de teste deste endereço e usá-la sem custo por um período de tempo limitado.
www. rosearchitect. comi
Rosearchitect.com é uma revista on-line, publicada pela Rational Software Corporation, que trata da modelagem em UML usando Rational Rose.
www .advancedsw. comi
A Advanced Software Technologies é autora do GDPro, uma ferramenta de modelagem visual para UML. Você pode baixar uma versão de teste do site da Web deles e usá-la sem custo por um período de tempo limitado.
www visualobject. comi
A Visual Object Modelers criou uma ferramenta de modelagem visual para UML. Você pode baixar uma versão limitada de demonstração do site da Web deles e usá-la sem custo por um período de tempo limitado.
www .microgoid. comibversion2istage/product htmi
A Microgold Software, lnc. criou WithClass, um aplicativo para projeto de software que suporta a notação da UML.
www. iysator. iiu. sei-ai].aidiaidia htini
Dia é uma ferramenta de diagramação gtk+ que pode desenhar diagramas de classes UML. Dia roda sob UNIX, mas o site da Web também inclui um link para uma versão para Windows.
dir. iycos comiComputersiSoftwareiObjectOrientediMethodoiogiesiUiTooisi Este site lista dezenas de ferramentas de modelagem para UML e suas homepages.
www methods-toois comitoois,/modeiing htmi
Este site contém uma lista de muitas ferramentas de modelagem de objetos, incluindo aquelas que suportam a UML.

Artigos e publica ções

www. omg. orginewsipr99iUML 2001 CACMOct99y29-Kobryn pdf
Este artigo, escrito por Cris Kobryn, explora o passado, o presente e o futuro da UML.

www. sdmagazine .com/umi/focus .rosenberg. htm
se Aqui você encontrará um artigo com dicas sobre como incorporar a UML a seus projetos.
www. db. informatik .uni-bremen. de/urnlbib/
A Bibliografia de UML oferece nomes e autores de muitos artigos relacionados à UML. Você pode pesquisar artigos por autor ou título.
usecasehelp. com/wp/white_papers .htm
Este site mantém uma lista de publicações sobre aplicações da modelagem de casos de uso à análise e projeto de sistemas.
ue www.ratio.co.uk/white. html
Você pode ler uma publicação que descreve um processo para OOAD usando a UML neste site. O artigo também inclui alguma implementação em C++.
www. tucs. fiipublicationsitechreportsiTR234 .pdf
Este arquivo contém um estudo de caso de OOAD para um gravador de som digital usando a UML.
www. conailen. com/whitepapers/webapps/ModelingWebApplications .htm
Este site contém um estudo de caso que modela aplicações para a Web usando a UML.
are www. sdmagazine .com/
p0 O site da Software Development Magazine Online tem um repositório de muitos artigos sobre a UML. Você pode procurar por assunto ou navegar pelos títulos dos artigos.

Tutoriais

www. qoses .com/education/
Este site contém um banco de tutoriais criados pelo autor da UML Kendall Scott e mantidos por Qoses.
www. qoses .com/education/tests/testO2 .html
Você pode fazer um teste on-line sobre UML neste endereço. Os resultados lhe são enviados por e-mail.
www. rational .comiproductsiroseitryititutorialiindex. j trnpl

ão A Rational Software Corporation oferece um arquivo com um tutorial para o Rational Rose neste site.

FAQs (Perguntas mais freqüentes)

www. rational .com/uml/gstart/'faq. jtmpl
Este é o endereço de FAQs sobre UML da Rational Software Corporation.
usecasehelp. com/faqifaq.htm
as Este site contém um pequeno arquivo de FAQs mantido por *usecasehelp.com.*
www. jguru. comi jguruifaqi
Digite UML na caixa de procura para acessar uma lista de FAQs sobre UML deste site.
www.uml-zone .comiumlfaq. asp
Este site contém uma pequena lista de FAQs sobre UML mantida por *uml-zone.com.*

Bibliografia

(A198) Alhir, *S. UML in a Nutshell.* Cambridge: O'Reily & Associates, mc. 1998.
(Bo99) Booch, G., Rumbaugh, J. e Jacobson, 1. *The Unified Modeling Language User Guide.* Reading, MA:

Addison-Wesley, 1999.

(Fi98) Firesmith, D. G. e B. Henderson-Sellers. "Clarifying Specialized Forms ofAssociation in UML and OML'. • o quali
Journal ofObject-oriented Programming, May 1998: 47-50. ser mod
(Fo97) FowlerM. e Scott, K. *UML Distilled. Applying the Standard Object Modeling Language.* Reading, MA: • Se for f
Addison-Wesley, 1997. mensag
(JoOO) Johnson, L.J. "Model Behavior". *Enterprise Development* May 2000: 20-28. • Existem
ponteirc
(Mc98) McLaughlin, M. e A. Moore. "Real-Time Extensions to the UML". *Dr Dobb's Journal* December 1998: dados a
82-93.
• Arrays
(Me98) Melewski, D. "UML Gains Ground". *Application Development Trends* October 1998: 34-44. primem
(Me97) Melewski, D. "UML: Ready for Prime *Time?"Application Development Trends* November 1997: 30-44. Para pa
ço do ei
(Me99) Melewski, D. "Wherefore and what 110w, *UML?" Application Development Trends* December 1999: 61-
68. • C++ foi
dados),
(Mu97) Muiler, *P. Instant UML.* Birmingham, UK: Wrox Press Ltd., 1997.
• Quandc
(Pe99) Perry, P. "UML Steps to the Plate". *Application Development Trends* May 1999: 33-36. • o opere
(Ru99) Rumbaugh, J., Jacobson, 1. e Booch, *G. The Unified Modeling Language Reference Manual.* Reading, • As opei
MA: Addison-Wesley, 1999. pontein
outro.
(Sc99) Schmuller, J. *Sam's Teach Yourself UML in 24 Ilours.* Indianapolis: Macmillan Computer Pubhshing,
1999. • Quand
.....objeto
(UML99) *The UnifiedModeling Language Specification: Version 1.3.* Framingham, MA: Object Management Group
(OMG), 1999. • Operaç
Todos
• Ao exe

Resumo cada ca

• Ponteiros são variáveis que contêm como seus valores os endereços de outras variáveis. • Ponteir
operaç
• A declaração ponteir
int *ptr; atribuíi
declara ptr como um ponteiro para um objeto de tipo int e é lida, "ptr é um ponteiro para int". O *, como usado aqui em • Um po:
uma declaração, indica que a variável *é* um ponteiro. • Ponteir
• Existem três valores que podem ser usados para inicializar um ponteiro: O. NULL ou um endereço de um objeto do mesmo mente
tipo. Inicializar um ponteiro com O e inicializar esse mesmo ponteiro com NULL são idênticos. • Ponteir
• O único inteiro que pode ser atribuído a um ponteiro sem coerção é zero. • Um no
• O operador & (endereço) retoma o endereço de seu operando. Na not:
• O operando do operador de endereço deve ser um nome de variável (ou outro ivalue); o operador de endereço não pode ser • Todas
aplicado a constantes, a expressões que não retomam um ivalue ou a variáveis declaradas com a classe de armazenamento do arra
register.
• Umno
• O operador *, chamado de indireção, referenciamento indireto ou operador de derreferencia, retorna um sinonimo, nome
alternativo ou apelido para o nome do objeto a que seu operando aponta na memória. Isto é chamado de "derreferenciar o • Arrays

ponteiro".
• Umpo
• Ao chamar uma função com um argumento que a função que chamou quer modificar, o endereço do argumento pode ser • passado. A função chamada então usa o operador de indireção (*) para modificar o valor do argumento na função que chamou.
• Uma função que recebe um endereço como argumento deve incluir um ponteiro como seu argumento correspondente.
• Não é necessário incluir os nomes de ponteiros em protótipos de função; só é necessário incluir os tipos dos ponteiros. Os • A fun nomes de ponteiro podem ser incluídos por razões de documentação, mas o compilador os ignorará. prograi termin:

• O qualificador const possibilita ao programador informar ao compilador que o valor de uma variável particular não deve ser modificado.
• Se for feita uma tentativa para modificar um valor const, o compilador a detecta e emite ou uma advertência ou uma mensagem de erro, dependendo do compilador particular usado.
• Existem quatro meios para passar um ponteiro para uma função: um ponteiro não-constante para dados não-constantes, um ponteiro não constante para dados constantes, um ponteiro constante para dados não-constantes e um ponteiro constante para dados constantes.
• Arrays são automaticamente passados por referência usando ponteiros porque o valor do nome do array é o endereço do primeiro elemento do array.
• Para passar um único elemento de array através de uma chamada por referência usando ponteiros, deve ser passado o endereço do elemento específico do array.
• C++ fornece o operador unário especial sizeof para determinar o tamanho de um array (ou de qualquer outro tipo de dados), em bytes, durante a compilação do programa.
• Quando aplicado ao nome de um array, o operador sizeof retoma o número total de bytes no array como um valor inteiro.
• O operador sizeof pode ser aplicado a qualquer nome de variável, tipo ou constante.
• As operações aritméticas que podem ser executadas com ponteiros são incrementar (++) um ponteiro, decrementar **(- -)** um ponteiro, somar (+ ou ***+=)*** um ponteiro e um inteiro, subtrair **(-** ou -=) um ponteiro e um inteiro e subtrair um ponteiro de outro.
• Quando um inteiro é somado ou subtraído de um ponteiro, o ponteiro é incrementado ou decrementado pelo tamanho que um objeto do tipo apontado por esse ponteiro ocupa, multiplicado pelo inteiro.
• Operações de aritmética de ponteiros só devem ser executadas sobre porções contíguas de memória, tais como um array. Todos os elementos de um array são armazenados contiguamente na memória.
• Ao executar aritmética de ponteiros sobre um array de caracteres, os resultados são iguais aos da aritmética regular porque cada caractere é armazenado em um (1) byte de memória.
• Ponteiros podem ser atribuídos um ao outro se ambos os ponteiros são do mesmo tipo. Caso contrário, deve ser feita uma operação de coerção. A exceção a esta regra é um ponteiro para void. o qual é um tipo de ponteiro genérico que pode manter ponteiros de qualquer tipo. Ponteiros de outros tipos podem ser atribuídos a ponteiros para void. Um ponteiro nulo pode ser atribuído a um ponteiro de outro tipo só com uma conversão de tipo explícita.
• Um ponteiro void não pode ser derreferenciado.
• Ponteiros podem ser comparados usando-se os operadores relacionais e de igualdade. As comparações de ponteiros normalmente só têm significado se os ponteiros apontam para membros do mesmo array.
• Ponteiros podem ser indexados exatamente como os nomes de array.
• Um nome de array é equivalente a um ponteiro constante para o primeiro elemento do array.
• Na notação ponteiro/deslocamento, o deslocamento é o mesmo que um índice de array.
• Todas as expressões de array com subscritos podem ser escritas com um ponteiro e um deslocamento, usando-se ou o nome do array como um ponteiro ou um ponteiro separado que aponta para o array.
• Um nome de array é um ponteiro constante que sempre aponta para a mesma posição na memória.
• Arrays podem conter ponteiros.
• Um ponteiro de uma função contém o endereço da posição de memória onde inicia o código da função na memória.
• Ponteiros de funções podem ser passados para funções, retornados por funções, armazenados em arrays e atribuídos a outros ponteiros.
• Um uso comum de ponteiros de função se encontra em sistemas orientados por menos. O ponteiro de função é usado para chamar a função correspondente a um item particular do menu.
• A função strcpy copia seu segundo argumento - um *string* - para o seu primeiro argumento - um array de caracteres. O programador deve assegurar que o array de destino seja suficientemente grande para armazenar o *string* e seu caractere terminal nulo.

A função strncpy é equivalente a strcpy, a não ser pelo fato de que uma chamada a strncpy especifica o número de caracteres a serem copiados do *string* para o array. O caractere terminal nulo só será copiado se o número de caracteres a serem copiados for um a mais que o comprimento do *string.*
• A função strcat concatena seu segundo argumento *srring* - inclusive com o caractere terminal nulo - a seu primeiro argumento *string.* O primeiro caractere do segundo *string* substitui o caractere nulo (\O) do primeiro *string.* O programador deve assegurar que o array usado para armazenar o primeiro *string* é grande o bastante para armazenar tanto o primeiro *string* como o segundo *string.*
• A função strncat concatena um número especificado de caracteres do segundo *string* ao primeiro *string.* Um caractere terminal nulo é acrescentado ao resultado.
• A função strcmp compara seu primeiro argumento *string* com seu segundo argumento *string,* caractere por caractere. A função retoma O se os *strings* são iguais, um valor negativo se o primeiro *string* é menor que o segundo *string* e um valor positivo se o primeiro *string* for maior que o segundo *string.*

• A função strncmp é equivalente a strcmp, a não ser pelo fato de que strncmp compara um número de caracteres especificado. Se o número de caracteres em um dos *strings* é menor que o número de caracteres especificado, strncmp compara os caracteres até encontrar o caractere nulo no *string* menor.
• Uma seqüência de chamadas a strtok separa um *string* em unidades léxicas que são separadas por caracteres contidos em um segundo argumento *string.* A primeira chamada contém o *string* para ser separado em unidades léxicas como primeiro argumento e chamadas subseqüentes para continuar a separar em unidades léxicas o mesmo *string* podem conter NULL como primeiro argumento. Cada chamada retoma um ponteiro para a unidade léxica atual. Se não existirem mais unidades léxicas quando strtok é chamado, o valor NULL é retomado.
• A função strlen aceita um *string* como um argumento e retorna o número de caracteres no *string* - o caractere terminal nulo não é incluído no comprimento do *string.*

Terminologia

adiamento indefinido
anexar strings a outros *strings*
aritmética de ponteiros
array de ponteiros
array de strings
ASCII
atribuição de ponteiro
chamada por referência
chamada por referência simulada
chamada por valor
classes, responsabilidades e colaborações (CRC) código de caractere
código numérico de um caractere
comparação de ponteiros
comparação de *strings*
comprimento de um *string*
concatenação de *string*
conjunto de caracteres

corist

constante de caractere constante de *string* copiar *strings*

<cstririg>
decrementar um ponteiro delimitador
derreferenciar um ponteiro deslocamento
EBCDIC
expressão de ponteiro

incrementar um ponteiro
indexação de ponteiro
indireção
inicializar um ponteiro
islower
literal
notação ponteiro/deslocamento operador de derreferenciamento (*) operador de endereço (&)
operador de indireção (*)
ponteiro
ponteiro constante
ponteiro constante para dados constantes ponteiro constante para dados não-constantes ponteiro de caractere
ponteiro de função
ponteiro não-constante para dados constantes ponteiro não-constante para dados não-constantes ponteiro NULL
ponteiro para uma função
ponteiro para void (void *) princípio do menor privilégio processamento de *string*
processamento de texto
referenciar diretamente uma variael referenciar indiretamente uma variável separar *strings* em unidades léxicas

strca
strcnT
strcp
string
string Ii
strle strnc

strnc
strnc

Tem

colabor diagran interaç linha cl da U

linha cl

Erro

5.1
5.2
5.3
5.4
5.5
5.6
5.7
5.8
5.9
5.10
5.11
5.12
5.13
5.14
5.15
5.16
5.17
5.18
5.19
5.20
5.21
5.22

sizeof
somando um ponteiro e um inteiro

`strcat strtok`
strcmp subscritar um ponteiro
strcpy subtraindo dois ponteiros
string subtraindo um inteiro de um ponteiro
string literal tipos de ponteiros
`strleri toupper`
strncat unidade léxica
strncmp void * (ponteiro para void)
`strncpy`

Terminologia de "Pensando em objetos"

colaboração mensagem
diagrama de colaborações números em diagramas de colaborações da UML
interação entre objetos objetos que participam de interações
linha cheia com seta na ponta em diagramas de colaborações quando ocorrem as interações
da UML retângulo em diagrama de colaborações da UML
linha cheia em diagramas de colaborações da UML seqüência de mensagens

Erros comuns de programação

5.1 Assumir que o usado para declarar um ponteiro se aplica a todos os nomes de variáveis, em uma lista de variáveis ponteiro separadas por vírgulas, pode fazer com que os ponteiros sejam declarados não como ponteiros. Cada ponteiro deve ser declarado com o prefixado ao nome.

5.2 Derreferenciar um ponteiro que não foi corretamente inicializado, ou ao qual não foi feita uma atribuição para apontar para uma posição específica na memória, pode causar um erro fatal durante a execução ou, ainda, pode modificar acidentalmente dados importantes e permitir que o programa execute até o fim, fornecendo resultados incorretos.
5.3 Uma tentativa de derreferenciar um não-ponteiro é um erro de sintaxe.
5.4 Derreferenciar um ponteiro O normalmente gera um erro fatal durante a execução.
5.5 E um erro não derreferenciar um ponteiro, quando é necessário fazê-lo para obter o valor para o qual o ponteiro aponta.
5.6 E erro de sintaxe não inicializar um ponteiro declarado como const.
5.7 Usar o operador **sizeof** em uma função para achar o tamanho em bytes de um argumento do tipo array resulta no tamanho em bytes de um ponteiro e não no tamanho em bytes do array.
5.8 Omitir os parênteses em uma operação sizeof quando o operando é um nome de tipo é erro de sintaxe.
5.9 Usar aritmética de ponteiros sobre um ponteiro que não se refere a um array de valores normalmente é um erro de lógica.
5.10 Subtrair ou comparar dois ponteiros que não se referem a elementos do mesmo array normalmente é um erro de lógica.
5.1 **1** Endereçar além dos extremos de um array, quando usamos aritmética de ponteiros, normalmente é erro de lógica.
5.12 Atribuir um ponteiro de um tipo a um ponteiro de outro (diferente de void *), sem converter o primeiro ponteiro para o tipo do segundo ponteiro, é erro de sintaxe.
5.13 Derreferenciar um ponteiro void é um erro de sintaxe.
5.14 Embora os nomes de array sejam ponteiros para o início do array e ponteiros possam ser modificados em expressões aritméticas, nomes de array não podem ser modificados em expressões aritméticas porque são ponteiros constantes.
5.15 Não alocar espaço suficiente em um array de caracteres para armazenar o caractere nulo que termina um *string.*
5.16 Criar ou usar um *"string"* que não contém um caractere terminal nulo.
5.17 Processar um caractere único como um *string* pode levar a um erro fatal durante a execução. Um *string* é um ponteiro - provavelmente um inteiro consideravelmente grande. Porém, um caractere é um inteiro pequeno (valores ASCII variam de O a 255). Em muitos sistemas, isso causa um erro porque os endereços de memória baixos são reservados para propósitos especiais, tais como para tratadores de interrupções do sistema operacional - assim, ocorrem "violações de acesso".
5.18 Passar um caractere como argumento para uma função, quando é esperado um *string,* pode levar a um erro fatal durante a execução.
5.19 Passar um *string* como argumento para uma função, quando é esperado um caractere, é um erro de sintaxe.
5.20 Esquecer de incluir o arquivo de cabeçalho <cstring> quando usar funções da biblioteca de tratamento de *strings.*
5.21 Não anexar um caractere nulo terminal ao primeiro argumento de uma strncpy, quando o terceiro argumento é menor ou igual ao comprimento do *string* no segundo argumento, pode causar erros fatais durante a execução.
5.22 Assumir que strcmp e strncmp retomam um quando seus argumentos forem iguais é um erro de lógica. Ambas as funções retornam zero (o valor falso em C++) para igualdade. Então, quando *strings* são testados quanto à igualdade, o resultado das funções strcmp ou strncmp devem ser comparados comO para determinar se os *strings* são iguais.

5.23 Não perceber que strtok modifica o *string* que será separado em unidades léxicas e, então, tentar usar esse *string* como se ele fosse o *string* original inalterado.

Boas práticas de programação

5.1 Embora não seja obrigatório fazer isso, incluir as letras Ptr em nomes de variáveis ponteiro torna claro que estas variáveis são ponteiros e precisam ser manipuladas de acordo.
5.2 Use uma chamada por valor para passar argumentos para uma função, a menos que o chamador exija explicitamente que a função modifique o valor da variável passada como argumento, no contexto do chamador. Este é outro exemplo do princípio de menor privilégio.
5.3 Antes de usar uma função, confira seu protótipo de função para determinar os argumentos que ela pode modificar.
5.4 Use a notação de array em vez da notação de ponteiro quando manipular arrays. Embora o programa possa se tornar ligeiramente mais lento para ser compilado, provavelmente ele será mais claro.
5.5 Quando armazenar um *string* de caracteres em um array de caracteres, esteja certo de que o array é grande o bastante para manter o maior *string* que será armazenado. C++ permite que strings de qualquer comprimento sejam armazenados. Se um *string* é mais longo que o array de caracteres em que ele é armazenado, os caracteres além do fim do array serão gravados "em cima" dos dados que estiverem após o array na memória.

Dicas de desempenho

5.1 Passe objetos grandes, tais como estruturas, usando ponteiros para dados constantes ou referências para dados constantes, para obter os benefícios de desempenho de uma chamada por referência e a segurança de uma chamada por valor.
5.2 sizeof é um operado unário usado durante a compilação, não uma função executada durante a execução. Deste modo, usar sizeof não causa impacto negativo sobre o desempenho da execução.
5.3 As vezes, um algoritmo que surge de um modo "natural" pode conter problemas sutis de desempenho, tal como o adiamento indefinido. Procure algoritmos que evitam o adiamento indefinido.

Dicas de portabilidade

5.1 O formato em que um ponteiro é exibido para o usuário depende da máquina. Alguns valores de ponteiros são exibidos por alguns sistemas como inteiros hexadecimais, enquanto outros sistemas usam inteiros decimais.
5.2 Embora const esteja bem-definido em C e C++ ANSI, alguns compiladores não garantem o seu uso correto.
5.3 O número de bytes usados para armazenar um tipo particular de dados pode variar entre sistemas. Quando estiver escrevendo programas que dependem do tamanho dos tipos de dados e que serão executados em vários sistemas de

computador, use sizeof para determinar o número de bytes usados para armazenar os tipos de dados.
5.4 A maioria dos computadores atuais têm inteiros de 2 bytes ou de 4 bytes. Algumas das máquinas mais recentes usam inteiros de 8 bytes. Como os resultados da aritmética de ponteiros dependem do tamanho dos objetos para os quais um ponteiro aponta, a aritmética de ponteiros é dependente da máquina.
5.5 Quando uma variável do tipo char * é inicializada com um *string* literal, alguns compiladores podem colocar o *string* em uma posição na memória em que o *string* não pode ser modificado. Se você precisa modificar um *string* literal, deve armazená-lo em um array de caracteres para assegurar a possibilidade de mudança em todos os sistemas.
5.6 Os códigos numéricos internos usados para representar caracteres podem ser diferentes em computadores diferentes.
5.7 Não faça testes explícitos do valor do código de caracteres em ASCII, como por exemplo em: if (ch == 65) em vez disso, use sempre a constante de caractere correspondente, como em: if (ch == A').

Observações de engenharia de software

5.1 O qualificador corist pode ser usado para forçar o uso do princípio de menor privilégio. O uso do princípio de menor privilégio para projetar software corretamente pode reduzir bastante o tempo de depuração e efeitos colaterais incorretos, tornando o programa mais fácil de se modificar e manter.
5.2 Se um valor não muda (ou não deve mudar) no corpo de uma função para a qual ele é passado, o argumento deve ser declarado const para garantir que não seja acidentalmente modificado.
5.3 Quando é usada uma chamada por valor, somente um valor pode ser alterado na função chamada. Esse valor deve ser atribuído do valor de retorno da função. Para modificar múltiplos valores em chamadas de função, vários argumentos são
passados através de uma chamada por referência.

g como 5.4 Colocar protótipos de função nas definições de outras funções força o princípio de menor privilégio, restringindo chama da para as funções às funções em que os protótipos aparecerem.
5.5 Quando passar um array para uma função, passe também o tamanho do array (em vez de incluir na função o conhecimen t do tamanho do array). Isto ajuda a tornar a função mais genérica. Funções genéricas são freqüentemente reutilizáveis
em muitos programas.

ue estas *Dica de teste e depura ção*

mte que ...
iplo do 5.1 Sempre inicialize ponteiros para evitar apontar para áreas de memória desconhecidas ou não-inicializadas.

omar *Exercícios de auto-revisão*

nte para 5.1 Responda a cada uma das seguintes frases:
idos. Se a) Um ponteiro é uma variável que contém como seu valor o _____ de outra variável.
iy serão b) Os três valores que podem ser usados para inicializar um ponteiro são _____. ____ ou
c) O único inteiro que pode ser atribuído a um ponteiro é _____
5.2 Afirme se a frase seguinte é *verdadeira ou falsa.* Se a resposta for *falsa,* explique por quê.
a) O operador de endereço *&* só pode ser aplicado a constantes, expressões e variáveis declaradas com a classe de armazenamento regis ter.
onstan- b) Um ponteiro que é declarado como void pode ser derreferenciado.
c) Ponteiros de tipos diferentes não podem ser atribuídos um ao outro sem uma operação de conversão de tipo.
orno 5.3 Responda a cada um dos seguintes ítens. Assuma que números de ponto flutuante com precisão simples são armazenados
o em 4 bytes e que o endereço de início do array é na posição 1002500 da memória. Cada parte do exercício deve usar os resultados de partes anteriores, quando necessário.
a) Declare um array nuinbers. de tipo double, com 10 elementos, e inicialize os elementos com os valores O . 0, 1 . 1, 2.2 9. 9. Suponha que a constante simbólica **SIZE** tenha sido definida como 10.
b) Declare um ponteiro nPtr que aponta para um objeto de tipo double.
c) Imprima os elementos do array nuinbers usando a notação de subscrito para arrays. Use uma estrutura for e xibidos assuma que a variável de controle do for, do tipo inteiro, tenha sido declarada. Imprima cada número com precisão
de 1 posição à direita da casa decimal.
d) Dê dois comandos separados que atribuem o endereço do início do array numbers à variável ponteiro nPtr.
rescre- e) Imprima os elementos do array nunibers usando a notação ponteiro/deslocamento com o ponteiro nPtr.
mputa- f) Imprima os elementos do array nuinbers usando a notação ponteiro/deslocamento com o nome do array como ponteiro.
usam g) Imprima os elementos do array numbers indexando o ponteiro nPtr.
ais um h) Refira-se ao elemento 4 do array numbers. usando as notações de índice de array, ponteiro/deslocamento e a nota çã com o nome do array como ponteiro, a notação de índice de ponteiro com nPtr e a notação ponteiro/desloca strin mento com nPtr.
1, deve i) Assumindo-se que nPtr aponta para o início do array numbers. que endereço é referenciado por nPtr + 8? Que valor está armazenado nessa posição?

ates. j) Assumindo-se que nPtr aponta para riumbers [5 *1* . qual endereço é referenciado por nPtr depois de riPtr - elo vez = 4 ser executado? Qual valor está armazenado nessa posição?
5.4 Para cada um dos seguintes itens, escreva um único comando para executar a tarefa indicada. Assuma que as variáveis de ponto flutuante numberi e number2 foram declaradas e que numberi foi inicializado com 7 . 3. Assuma, também, que a variável ptr é do tipo char * e os arrays si [100] e s2 [**100** *1* são do tipo char.

a) Declare a variável fPtr como um ponteiro para um objeto do tipo double.
menor b) Atribua o endereço da variável numberi à variável ponteiro fPtr.
rretos. c) Imprima o valor do objeto apontado por fptr.
d) Atribua o valor do objeto apontado por fPtr à variável number2.
ve ser e) Imprima o valor de nuxnber2.
t) Imprima o endereço de nuinberl.
‘ve ser g) Imprima o endereço armazenado em fPtr. O valor impresso é o mesmo que o endereço de nuinberi?
os são h) Copie o *string* armazenado no array s2 para o array si.
1) Compare o *string* em si ao *string* em s2. Imprima o resultado.

i) Anexe lO caracteres do *string* em s2 ao *string* em si.
k) Determine o comprimento do *string* em si. Imprima o resultado.
1) Atribua a ptr a posição da primeira unidade léxica em s2. Unidades léxicas em s2 são separadas por vírgulas.
5.5 Faça cada um dos seguintes itens:
a) Escreva o cabeçalho da função para uma função exchange que aceita como argumentos dois ponteiros para os números de ponto flutuante com precisão dupla x e y e não retoma um valor.
b) Escreva o protótipo da função para a função do item (a).
c) Escreva o cabeçalho da função para uma função evaluate que retoma um inteiro e que aceita como argumentos um inteiro x e um ponteiro para a função poiy. A função poiy aceita um argumento de tipo inteiro, retornando um inteiro.

```
d) Escreva o protótipo da função para a função do item (c).
```

e) Mostre dois métodos diferentes de inicializar o array de caracteres vogal com o *string* de vogais “AEIOU”.
5.6 Ache o erro em cada um dos segmentos do programa seguinte. Suponha que existam as seguintes declarações:

```
int *zptr; // zPtr referenciará o array z
int *aptr = O;
void *sptr = 0;
int nuinber, i;
int z [5] = (l, 2, 3, 4, 5 };
sPTr = z;
a) ++zPtr;
b) //usa o ponteiro para obter o primeiro valor do array
number = zPtr;
c) // atribui o elemento 2 do array (o valor 3) a nuniber
number *zptr[ 2 ];
d) //imprime todo o array z
for ( i = 0; i <= 5; i++
cout « zPtr[ i ] « endl;
e) //atribui o valor apontado por sPtr para number
nuriber = *sptr;
f) ++z;
g) char s[ 10 ];
cout « strncpy(s, “oi!”, 3) « endi;
h) char s[ 12 ];
strcpy( s, “Bem-vindo ao lar!” );
i) if ( strcmp( stringi, string2
cout «“Os strings são iguais” « endi;
5.7 O que é impresso, se for o caso, quando cada um dos comandos seguintes é executado? Se o comando contem um erro, descreva o erro e indique como corrigi-lo. Assuma as seguintes declarações de variáveis:
char si[ 50] = “jack”, s2[ 50 ] = “jill”, s3[ 50 ], *sptr;
a)cout « strcpy( s3, s2 ) « endi;
b)cout « strcat( strcat( strcpy( s3, si),” e -),s2 )« endi;
c)cout « strlen( si ) + strlen( s2 ) « endl;
d)cout « strlen( s3 ) « endi;
```

Respostas dos exercícios de auto-revisão

5.1 a) endereço. b) 0, NULL, um endereço. c) 0.
5.2 a) Falsa. O operador de endereço só pode ser aplicado a variáveis e não pode ser aplicado a constantes, expressões ou variáveis declaradas com a classe de armazenamento regis ter.
b) Falsa. Um ponteiro void não pode ser derreferenciado porque não existe nenhum modo de saber exatamente quantos bytes de memória deveriam ser derreferenciados.

c) Falsa. Aos ponteiros de tipo void podem ser atribuídos ponteiros de outros tipos. Os ponteiros de tipo void podem

ser atribuídos a ponteiros de outros tipos somente com uma conversão explícita de tipo.
5.3 a) double nuiubers[SIZE] ={ 0.0, 1.1, **2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9 };**
b) double *nptr;
c) cout« setiosflags(ios::I fixed ios::showpoint
« setprecision(1);
for (i = **0; i < SIZE; i++**
cout « numbers[i] «
d) nPtr = numbers;
nPtr = &numbers [0];
e) cout« setiosflags(ios::fixed 1 ios::showpoint
« setprecisiori(1);
for (i = 0; i < SIZE; i++
cout« *(nptr + i) «

O cout « setiosflags (ios : : 1 fixed ios : showpoint
j « setprecision (1);
for (i = 0; i < SIZE; i++
cout « * (riumbers + i) « ·
g) cout « setiosflags (ios : : 1 fixo ios : : showpoirit
«setprecision(1);
for (i = 0; i < SIZE; i ++
cout« nPtr[i] «
h) nuxnbers[3
*(nunibers + 3)
nPtr[3
*(nPtr + 3
i) Oendereçoéloo2500 +8 * 4 =1002532. Ovaloré8.8.
j) Oendereçodenumbers[5] é1002500 + 5 *4 = 1002520.
OendereçodenPtr-=4 é1002520 - **4** * **4** = **1002504.**
O valor nessa posição é 1 . **1.**
5.4 a) doubie *fPtr;
b) fPtr = &numberl;
c) cout « "O valor de *fptr é - « *f « endi;
d) nuinber2 = *fptr;
e) cout « "O valor de number2 é - « number2 « endi;
f) cout « "O endereço de numberi é - « &numberl « endi;
g) cout « **"O endereço armazenado em** fPtr é - « fPtr « **endi;**
Sim, o valor é o mesmo.
h) strcpy(si, s2
i) cout « "strcmp(sl, s2) = - «strcmp(si, s2) « endi;
j) strncat(si, s2, 10
k) cout « "strlen(sl) = - «strlen(si) «endi;
1) ptr = **strtok(s2,** -;);
5.5 a) void exchange(doubie *, doubie *y
b) void **exchange** (double *, doubie *
c) int evaluate(int x, int (*poly) (int
d) int evaluate(int, int (*) (**int**));
e) char vogai[] = "AEIOU";
char vogai[] = { **'A', 'E', '1', 'O', 'U', '\O');**
5.6 a) Erro: zPtr não tinha sido inicializado.
Correção: inicialize zPtr com zPtr =
b) Erro: o ponteiro não é derreferenciado.
Correção: mude o comando para number = ***zptr;**
c) Erro: **zPtr [2]** não é um ponteiro e não devia ser derreferenciado.
Correção: mude *zptr[2] para **zPtr[2].**
d) Erro: referindo-se a um elemento do array fora dos limites do mesmo com indexação de ponteiro. Correção: mude o operador relacional na estrutura for para < para evitar ultrapassar o fim do array.
e) Erro: derreferenciar um ponteiro void.
Correção: a fim de desreferenciar o ponteiro, ele deve primeiro ser convertido para um ponteiro para dados de tipo inteiro.

Mude o comando acima para riumber = * (int *) sPtr;
f) Erro: tentar modificar um nome de array com aritmética de ponteiros.
Correção: use uma variável ponteiro, em vez do nome do array, para poder efetuar aritmética de ponteiros ou use um

subscrito junto com o nome do array para se referir a um elemento específico.
g) Erro: a função strncpy não escreve um caractere nulo terminal no array s porque seu terceiro argumento é igual ao *Nota:* comprimento do *string* Oi ! - *capaz*

Correção: mude o valor do terceiro argumento de s trncpy para 4 ou atribua \ O · a s [3] para assegurar que o caractere nulo terminal é anexado ao *string.* 5.12
h) Erro: o array de caracteres s não é grande o bastante para armazenar o caractere nulo terminal. pôquer
Correção: declare o array com mais elementos.
i) Erro: a função strcmp retornará O se os *strings* forem iguais; então, a condição na estrutura if será falsa e a exibição/impressão do comando não será executada.
Correção: compare, explicitamente, o resultado de strcmp com O na condição da estrutura *if.*
5.7 a) jili
b) jack e jili
c)8 -
d) 13 :l3
poquei

Exercícios 5.14

cartea
5.8 Afirme se o item seguinte é *verdadeiro oufalso. Sefalso,* explique por quê. avaliai
a) Dois ponteiros que apontam para dois arrays diferentes não podem ser comparados de modo que a comparação faça substit sentido. dor. ((
b) Como o nome de um array é um ponteiro para o primeiro elemento do array, nomes de array podem ser manipulados exatamente da mesma maneira que ponteiros. 5.15
sendo,
5.9 Responda a cada um dos seguintes ítens. Assuma que inteiros sem sinal são armazenados em 2 bytes e que o endereço maos inicial do array está na posição 1002500 na memória. voce
a) Declare um array do tipo unsigned int chamado value. com 5 elementos, e inicialize os elementos com os result inteiros pares de 2 até 10. Assuma que a constante simbólica SIZE tenha sido definida como 5. dificil
b) Declare um ponteiro vPtr que aponta para um objeto de tipo unsigned int.
e) Imprima os elementos do array values usando a notação de subscrito para array. Use uma estrutura for e assuma que a variável inteira de controle i tenha sido declarada. inefic
d) Dê dois comandos separados que atribuem o endereço do início do array values à variável ponteiro vPtr. um aI
e) Imprima os elementos do array values usando a notação ponteiro/deslocamento.

O Imprima os elementos do array values usando a notação ponteiro/deslocamento, com o nome do array como ponteiro. para
g) Imprima os elementos do array values indexando o ponteiro para o array. deves
h) Retira-se ao quinto elemento de values usando a notação de subscrito de array, a notação ponteiro/deslocamento esta com o nome do array como o ponteiro, a notação de índice de ponteiro e a notação ponteiro/deslocamento. ShU±
i) Qual endereço é referenciado por vPtr + 3? Qual valor está armazenado nesta posição? -

j) Assumindo-se que vPtr aponta para values [*4*], qual endereço é referenciado por vptr? Qual valor está çao ai armazenado nessa posição? depoi
progr
5.10 Para cada um dos seguintes itens, escreva um comando único que executa a tarefa indicada. Assuma que as variáveis parar inteiras longas valuel e value2 tenham sido declaradas e que valuel tenha sido inicializado.
a) Declare a variável lPtr como um ponteiro para um objeto do tipo long.
b) Atribua o endereço da variável valuel à variável ponteiro lPtr.
e) Imprima o valor do objeto apontado por lPtr. A
d) Atribua o valor do objeto apontado por lPtr à variável value2.
e) Imprima o valor de value2.
f) Imprima o endereço de valuel.
g) Imprima o endereço armazenado em lPtr. O valor impresso é o mesmo que o endereço de valuel?
5.11 Faça cada um dos seguintes itens. 2
a) Escreva o cabeçalho da função para a função zero que aceita como argumento um array biglntegers de intei ro longos e não retoma um valor.
b) Escreva o protótipo da função para a função do item (a). -

Fig.

Fig. 5.39 Exemplo de embaralhamento do array deck.

5.1 7 *(Simulação: a tartaruga e a lebre)* Neste problema, você recriará a clássica corrida da tartaruga e a lebre. Você usará a geração de números aleatórios para desenvolver uma simulação deste evento memorável.
Nossos contendores começam a corrida no "quadrado 1" de 70 quadrados. Cada quadrado representa uma posição possível ao longo do percurso da corrida. A linha de chegada o quadrado 70. O primeiro contendor que alcançar, ou passar, pelo quadrado 70 é recompensado com um balde de cenouras e alface frescas. O percurso é em uma montanha escorregadia, ladeira acima. Assim, ocasionalmente os contendores perdem terreno.

Existe um relógio que marca cada segundo. A cada “tique” do relógio, seu programa deve ajustar a posição dos animais de
acordo com as seguintes regras:

Escorregada 20% 6 quadrados à esquerda
Caminhada lenta 30% 1 quadrado à direita
Lebre Dorme 20% Nenhum movimento
Grande pulo 20% 9 quadrados à direita
Grande escorregada 1 0% 1 2 quadrados à esquerda
Pulo pequeno 30% 1 quadrado à direita
Pequena escorregada 20% 2 quadrados à esquerda
São usadas variáveis para manter o acompanhamento das posições dos animais (posições são os números 1 a 70). Cada animal começa na posição 1 (i.e., a “faixa de largada”). Se um animal, ao partir do quadrado 1 , escorregar, volte o animal para o quadrado 1.
Gere as porcentagens na tabela precedente por um meio da geração aleatória de um inteiro no intervalo 1 < *i* < 10. Para a tartaruga, execute uma “caminhada rápida” quando 1 < *i* < *5,* uma “escorregada” quando 6 < *i* < *7* ou uma “caminhada lenta” quando 8 < *i* < lO. Use uma técnica semelhante para mover a lebre.
Comece a corrida imprimindo
BANG ! I ‘ !‘
E ELES PARTIRAM ‘ *! .* !‘
Para cada “tique” do relógio (i.e., cada repetição do laço), imprima uma linha de 70 posições mostrando a letra **T** na posição da tartaruga e a letra **L** na posição da lebre. Ocasionalmente, os contendores caem sobre o mesmo quadrado. Neste caso, a tartaruga morde a lebre e seu programa deve imprimirAl ‘ *! !* começando naquela posição. Todas as outras posições de impressão diferentes de T, L ou o AI ! ! (no caso de um empate) devem estar em branco.
Depois de imprimir cada linha, teste se um ou outro animal alcançou ou passou pelo quadrado 70. Nesse caso, imprima o vencedor e termine a simulação. Se a tartaruga ganhar, imprima TARTARUGA VENCE 1 ‘ VALEU *.! !* Se a lebre ganhar, imprima Lebre ganha . Ohhh …Se ambos os animais chegarem à posição 70 no mesmo “tique” do relógio, você pode optar por favorecer a tartaruga (“o menos favorecido dos dois”) ou imprimir É **EMPATE.** Se nenhum animal ganhar, execute o laço para simular novamente o próximo “tique” do relógio. Quando você estiver pronto para executar seu programa, junte um grupo de torcedores para assistir à corrida. Você ficará admirado ao ver como o público fica envolvido!

Exe	**mplo do**	**array d**	**eck embaralhado**										
	O	**1**	**2**	**3**	**4**	*5*	*6*	*7*	**8**	**9**	**10**	**11**	**12**
O	**19**	**40**	**27**	**25**	**36**	**46**	**10**	**34**	**35**	**41**	**18**	**2**	**44**
1	**13**	**28**	**14**	**16**	**21**	**30**	**8**	**11**	**31**	**17**	**24**	**7**	**1**
2	**12**	**33**	**15**	**42**	**43**	**23**	**45**	**3**	**29**	**32**	**4**	**47**	**26**
3	**50**	**38**	**52**	**39**	**48**	**51**	**9**	**5**	**37**	**49**	**22**	**6**	**20**

Animal	**Tipo de movimento**	**Porcentagem do tempo**	**Movim**	**ento efetivo**
Tartaruga	Caminhada rápida	50%	3 quad	rados à direita

Seção especial: construindo seu próprio computador

Nos próximos problemas, faremos um desvio temporário do mundo da programação em linguagens de alto nível. “Descascaremos” um computador e examinaremos sua estrutura interna. Introduziremos a programação em linguagem de máquina e escreve- remos vários programas em linguagem de máquina. Para fazer desta experiência uma coisa realmente valiosa, construiremos então um computador (através da técnica de simulação por software do mesmo) com o qual você poderá executar seus programas em linguagem de máquina!
5.1k *(Programação em linguagem de máquina)* Vamos criar um computador ao qual chamaremos de Simpletron. Como seu nome indica, é uma máquina simples, mas, como logo veremos, também poderosa. O Simpletron executa programas escritos na única

linguagem que ele entende diretamente; isto é, a Simpletron Machine Language ou, abreviadamente, SML.

O Simpletron contém um registrador especial *acumulador* em que as informações são postas antes do Simpletron usar essas informações em cálculos ou examiná-la de vários modos. Todas as informações no Simpletron são manipuladas em termos de palavras. Uma palavra é um número decimal de quatro dígitos, com sinal, tal como +3364. -1293, +0007. -0001, etc. O Simpletron é equipado com uma memória de 100 palavras e estas palavras são referenciadas por seus números de posição na memória: 00, 01 99. Antes de executar um programa em SML, devemos *carregar* ou colocar o programa na memória. A primeira instrução (ou comando) de todo programa em SML é sempre colocada na posição 00 O simulador começará a execução nesta posição.
Cada instrução escrita em SML ocupa uma palavra da memória do Simpletron (portanto, instruções são números decimais de quatro dígitos com sinal). Devemos assumir que o sinal de uma instrução de SML é sempre mais, porém o sinal de uma palavra de dados pode ser ou mais ou menos. Cada posição na memória do Simpletron pode conter uma instrução, o valor de um dados usado por um programa ou uma área de memória não-usada (e, conseqüentemente, indefinida). Os primeiros dois dígitos de cada instrução em SML são o *código de operação,* que especifica a operação a ser executada. Os códigos de operação da SML são mostrados na Fig. 5.40.

e

Fig. 5.40 Códigos de operação da Simpletron Machine Language (SML) (parte 1 de 2).

Código de operação	Significado
Operações de entrada/saída: corist int READ = 10	Lêumapalavradotecladoparaumaposiçãoespecíficana memória.
const int WRITE = 1 1 ;	Escreve uma palavra de uma posição específica da memória na tela.
Operações de cargaJarmazenamento: const int LOAD = 20 ;	Carrega uma palavra de uma posição específica na me- mória para o acumulador.
const int STORE = 21 ;	Armazena uma palavra do acumulador em uma posição específica da memória.
Operações aritméticas: const int ADD = 30 ;	Soma uma palavra de uma posição específica na memória à palavra no acumulador (deixa o resultado no acumula- dor).
const int SUBTRACT = 31 ;	Subtrai uma palavra de uma posição específica na memória da palavra no acumulador (deixa o resultado no acumulador).
const int DIVIDE = 32 ;	Divide a palavra que está no acumulador pela palavra de uma posição específica na memória (deixa o resultado no acumulador).
const int MULTIPLY = 33 ;	Multiplica uma palavra de uma posição específica na me- mória pela palavra no acumulador (deixa o resultado no acumulador).

Fig. 5.40 Códigos de operação da Simpletron Machine Language (SML) (parte 2 de 2).
Os últimos dois dígitos de uma instrução de SML são o *operando* - o endereço da posição de memória que contém a palavra à qual a operação se aplica.

Agora, vamos examinar vários programas simples em SML. O primeiro programa em SML (Exemplo 1) lê dois números do teclado e calcula e imprime sua soma. A instrução +1007 lê o primeiro número do teclado e o coloca na posição 07 (que foi inicializada com zero). Então, a instrução +1008 lê o próximo número para a posição 08 . A instruçãoloadcarrega (copia) o primeiro número no acumulador e a instrução *add,* +3008. soma o segundo número ao número no acumulador. *Todas as instruções aritméticas de SML deixam seu resultado no acumulador.* A instrução *store,* +2109. armazena (copia) o resultado de volta na posição de memória 09 de onde a instrução *write,* +1109. pega o número e imprime o mesmo (como um número decimal de quatro dígitos com sinal). A instrução *halt,* +4300, termina a execução.

O programa em SML do Exemplo 2 lê dois números do teclado e determina e imprime o de valor maior. Note o uso da instrução +4107 como uma transferência de controle condicional, quase a mesma coisa que faz o comando if em C++.

```
00 +1009 (Read A)
01 +1010 (Read B)
02 +2009 (Load A)
03 +3110 (Subtract B)
04 +4107 (Branch negative to 07)
05 +1109 (Write A)
06 +4300 (Halt)
07 +1110 (Write B)
08 +4300 (Halt)
09 +0000 (Variável A)
```

10 +0000 (Variável B)

Código de operação	Significado
Operações de transferência de controle: const int BRANCH = 40;	Desvia para uma posição específica na memória.
const int BRANCHNEG = 41;	Desvia para uma posição específica na memória se o acumulador for negativo.
const int BRANCHZERO = 42;	Desvia para uma posição específica na memória se o acumulador for zero.
const int HALT = 43;	Parada-o programa completou sua tarefa.

Exemplo 1 Posição	Número	Instrução	
00	**+1007**	**(Read A)**	
01	**+1008**	**(Read B)**	
02	**+2007**	**(Load A)**	
03	**+3008**	**(Add B)**	
04	**+2109**	**(Store C)**	
05	**+1109**	**(Write C)**	
06	**+4300**	**(Halt)**	
07	+0000	**(Variável**	**A)**
08	**+0000**	**(Variável**	**B)**
09	**+0000**	**(C - Resultado)**	

Exemplo 2
Posição Número Instrução

Agora, escreva programas em SML para realizar cada uma das seguintes tarefas.
a) Use um laço controlado por uma sentinela para ler números positivos e calcular e imprimir sua soma.
b) Use um laço controlado por um contador para ler sete números, alguns positivos e alguns negativos, e calcular e imprimir sua média.
c) Leia uma série de números e determine e imprima o maior número. O primeiro número lido indica quantos números devem ser processados.
o acu 5.1
(Um simulador de computador) Poderá, a princípio, parecer estranho, mas nesse problema você vai construir seu próprio o acu- 1 computador. Não, você não estará soldando componentes de hardware. Em vez disso, você usará a poderosa técnica da *simulação baseada em software* para criar um *modelo em software* do Simpletron. Você não ficará desapontado. Seu simulador do Simpletron tornará o computador que você está usando um Simpletron e você realmente será capaz de executar. testar e depurar os programas em SML escritos no Exercício 5.18.
Quando você executar seu simulador do Simpletron, deve começar imprimindo:
*** **Bem-vindo ao Simpletron'**
lavra à
*** **Por favor, digite seu programa,** uma **instrução**
imeros *** **(ou palavra de dados) de cada vez. Eu digitarei o**
uefoi *** número da posição e um ponto de interrogação (?) .
pia)o *** **Então você digita a palavra para aquela posição.**
instru- *** **Digite a sentinela - 99999 para terminar a entrada** ***
evolta *** **do seu programa.** ***
roalde
Simule a memória do Simpletron com um array memory com 100 elementos. Agora, assuma que o simulador está sendo executado, e examinemos o diálogo na medida em que digitamos o programa do Exemplo 2 do Exercício 5.18:
00 ? +1009
01 ? +1010
02 ? +2009
03 ? +3110
04 ? +4107
05 ? +1109
06 ? +4300

07 ? +1110
08 ? +4300
09 ? +0000
10 ? +0000
11 ? -99999
*** **Carga do programa completa**
alção ***Começa a execução do programa

Agora, o programa em SML foi armazenado no array memory. A seguir, o Simpletron executa seu programa em SML. A execução começa com a instrução na posição 00 e, como em C++, continua na seqüência, a menos que seja desviada para alguma outra parte do programa por uma transferência de controle.

Use a variável accumulator para representar o registrador acumulador. Use a variável counter para manter o controle da posição na memória que contém a instrução que está sendo executada. Use a variável operationCode para indicar a operação que está sendo atualmente executada, i.e., os dois dígitos da esquerda da palavra que contém a instrução. Use a variável operand para indicar a posição da memória sobre a qual a instrução atual opera. Deste modo, operand é composto pelos dois dígitos mais à direita da instrução que está sendo atualmente executada. Não execute instruções diretamente da memória. Em vez disso, transfira apróxima instrução que vai serexecutadadamemóriaparauma variável instructionRegister. Então, "retire" os doisdígitos da esquerda e coloque-os em operationCode: "retire" os dois dígitos da direita e coloque os mesmos em operand. Quando o Simpletron começa a execução, os registradores especiais são todos inicializados com zero.

Agora, vamos fazer um "passeio" pela execução da primeira instrução de SML. +1009, na posição de memória 00. Isto é chamado de um *ciclo de execução de instrução.*

O counter nos diz a posição da próxima instrução a ser executada. Pegamos o conteúdo dessa posição de memory usando o comando de **C++**

instructionRegister = memory[counter];

J

O código de operação e o operando são extraídos do registrador de instrução pelos comandos

```
operationCode = instructionRegister / 100;
operand = instructionRegister % 100;
```

Agora, o Simpletron deve determinar que o código da operação é realmente um *read (versus* um *write,* um *load,* etc.). Um comando **switch** diferencia as doze operações de SML.

Na estrutura switch, o comportamento de várias instruções de SML é simulado como segue (deixamos as outras para o leitor):

read: **cm » memory[operand];**

load: accumulator = **memory [operand];**

add: `accumulator += memory[ operand ];`

branch: Discutiremos as instruções de desvio em breve.

halt: Esta instrução imprime a mensagem

`***Execução do Simpletron terminada ***`

Ela então imprime o nome e conteúdo de cada registrador, como também o conteúdo completo de memória. Tal relatório é freqüentemente chamado de *dump de computador* (não, um *dump* de computador não é um lugar para onde os computadores velhos vão). Para ajudá-lo a programar sua função *dump,* um exemplo do formato do *dump é* mostrado na Fig.5.41 .Note que um *dump,* depois de ser executado um programa de Simpletron, mostraria os valores reais de instruções e dados no momento do término da execução.

REGISTRADORES:
acumulator +0000
counter 00
iristructionRegister +0000
operationCode 00
operand 00
MEMÓRIA:
O 1 2 3 4 5 6 7 8 9
O +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
10 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
20 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
30 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
40 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000

50 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
60 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
70 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
80 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
90 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000

Fig. 5.41 Exemplo de *dump.*

Vamos continuar com a execução da primeira instrução do nosso programa - **+100 9 na posição de memória 00. Como** indicamos, o comando switch simula esta execução com o comando de C++

`cm » memory[ operarid ];`

Um ponto de interrogação `(?)deve ser exibido na tela, antes do cm ser executado, para solicitar que o usuário forneça um dado de` entrada. O Simpletron espera o usuário digitar um valor e apertar a tecla *Return.* **O valor é então** armazenado na posição `09.`

Nesse momento, a simulação da primeira instrução está completa. Tudo que falta é preparar o Simpletron para executar a próxima instrução. Uma vez que a instrução que acabou de ser executada não era uma transferência de controle, necessitamos somente incrementar o registrador contador de instruções, como segue:

```
++counter;
```

isso completa a execução simulada da primeira instrução. O processo inteiro (i.e., o ciclo de execução da instrução) começa novamente, com a busca da próxima instrução a ser executada.

Consideremos, agora, como instruções de desvio - transferências de controle - são simuladas. Tudo o que necessitamos fazer é ajustar o valor no contador de instruções apropriadamente. Então, a instrução de branch incondicional (4 O) é simulada dentro da estrutura switch como

```
counter = operand;
```

O desvio condicional branchzero (desvia se o acumulador for zero) é simulado como

```
if ( accumulator == O)
couriter = operand;
```

Nesse ponto, você deveria implementar seu simulador do Simpletron e executar cada um dos programas em SML que você escreveu no Exercício 5.18. Você pode enfeitar a SML com características adicionais e incorporar essas ao seu simulador.

Seu simulador deve verificar vários tipos de erros. Durante a fase de carga do programa, por exemplo, cada número que o usuário digita para memory no Simpletron deve estar no intervalo -9999 a +999. Seu simulador deve usar um laço while para testar se cada número fornecido está nesse intervalo e, se não estiver, continuar solicitando ao usuário paira redigitar o número até que forneça um número correto.

Durante a fase de execução, seu simulador deveria testar a ocorrência de vários erros sérios, tais como tentativas de divisão por zero, tentativas de execução de códigos de operações inválidas, estouros *(overflow)* do acumulador (i.e., operações aritméticas que resultam em valores maiores do que + 9 9 9 9 ou menores do que - 9 9 9 9) e coisas semelhantes. Tais erros sérios são chamados de *errosfatais.* Quando um erro fatal é descoberto, seu simulador deveria imprimir uma mensagem de erro tal como:

```
*** Tentativa de dividir por zero
*** Execução do Simpletron terminada anormalmente ***
```

e imprimir um *dump* completo de computador no formato que já discutimos anteriormente. Isto ajudará o usuário a localizar o erro no programa.

Mais exercícios sobre ponteiros

5.20 Modifique o programa de embaralhar e distribuir cartas da Fig. 5.24, de maneira que o embaralharamento e a retirada sejam executados pela mesma função (shuffleAndDeai). A função deveria conter uma estrutura de laços aninhados semelhante à função shuffle na Fig. 5.24.

5.21 O que faz esse programa?

```
1 // ex0521.cpp
2 #include <iostream>
3
4 using std::cout;
5 using std::cin;
6 using std::endl;
7
8 void mysteryl( char *, const char *
9
10 int main O
11
12 char stringl [ 80 ) , string2 [ 80 ]
13
```



```
14 cout « "Forneça dois strings: ";
15 cio » stringl » string2;
```

```
16 mysteryl( stringl, string2 );
17 cout « stringl « endi;
18 return O;
19}
20
21 void mysteryl (char *sl, const char *s2
22
23 while (*sl ! '\O'
24 ++sl;
25
26 for (; si *s2; si++, s2++
27 ; II comando vazio
28 }
```

5.22 O que faz este programa ?

```
1 II exOS_22.cpp
2 #inciude <iostream>
3
4 using std::cout;
5 using std::cin;
6 using std: :endi;
7
8 int miystery2 (const char *);
9
io int main O
ii{
12 char string[ 80 1;
13
14 cout « Forneça um string: ;
15 cm » string;
16 cout « mystery2( string ) « endi;
17 return 0;
18 J
19
20 int mystery2( const char *s
21 {
22 int x;
23
24 for (x = 0; '\O'; s++
25 ++x;
26
27 return x;
28 }
```

5.23 Ache o erro em cada um dos seguintes segmentos de programa. Se o erro puder ser corrigido, explique como.

a) int

cout « nuinber « endi;

b) double *realptr;

```
long *mntegerPtr;
integerPtr = realPtr;
```

c) int *x, y;

```
x = y;
```


d) char s[ ] "este é um array de caracteres;

```
for (; *s != '\O'; s++)
cout « *s «
```

e) short *fljjflp, result;

```
void *genericptr = numPtr;
```

result=*genericPtr + 7;

```
f) double x 19.34;
double xPtr =
cout « xPtr « endi;
g) char *S;
cout « s« endi;
```

5.24 *(Quicksort)* Nos exemplos e exercícios do Capítulo 4, discutimos as técnicas de classificação *bubble sort, bucket sort* e *selection*

sort. Agora, apresentamos a técnica recursiva de classificação *Quicksort.* O algoritmo básico para um array unidimensional de valores é como segue:
a) *Etapa de particionamento:* pegue o primeiro elemento do array não-classificado e determine sua posição final no array classificado, i.e., todos os valores à esquerda de um elemento no array são menores que o elemento e todos os valores à direita de um elemento no array são maiores que o elemento. Agora, temos um elemento em sua posição correta e dois subarrays não-classificados.
b) *Etapa recursiva:* execute a etapa 1 para cada subarray não-classificado.
Cada vez que a etapa 1 é executada sobre um subarray, outro elemento é colocado em sua posição final do array classificado e dois subarrays não-classificados são criados. Quando um subarray consiste em apenas um elemento, ele deve estar classificado, então esse elemento está em sua posição final.

O algoritmo básico parece bastante simples, mas como é que nós determinamos a posição final do primeiro elemento de cada subarray? Por exemplo, considere o seguinte conjunto de valores (o elemento em negrito é o elemento particionador, que será colocado em sua posição final no array classificado):
37 2 6 4 89 8 10 12 68 45
a) A partir do elemento mais à direita do array, compare cada elemento até que um elemento menor que 37 seja achado, e então troque 37 e esse elemento de posição. O primeiro elemento menor que 37 é 12, assim 37 e 12 são trocados. O novo array fica:
12 2 6 4 89 8 10 37 68 45

O elemento 12 está em itálico para indicar que acabou de ser trocado com 37.
b) A partir da esquerda do array, mas começando com o elemento depois de 12, compare cada elemento até que um elemento maior que 37 seja encontrado, então troque 37 com esse elemento. O primeiro elemento maior que 37 é 89, assim 37 e 89 são trocados. O novo array fica:
12 2 6 4 37 8 10 *89* 68 45
e) A partir da direita, mas começando com o elemento antes de 89, compare cada elemento até que um elemento menor que 37 seja achado, então troque 37 com esse elemento. O primeiro elemento menor que 37 é 10, assim 37 e 10 são trocados. O novo array fica:
12 2 6 4 *10* 8 37 89 68 45
d) A partir da esquerda, mas começando com o elemento depois de 10, compare cada elemento até que um elemento maior que 37 seja achado, então troque 37 com esse elemento. Não existem mais elementos maiores que 37, de modo que, quando comparamos 37 com ele mesmo, sabemos que 37 foi colocado em sua posição final no array ordenado.
' Uma vez que a partição tenha sido aplicada sobre o array acima, existem dois subarrays não-classificados. O subarray com valores menores que 37 contém 12, 2, 6, 4, lO e 8. O subarray com valores maiores que 37 contém 89, 68 e 45. A classificação continua com ambos os subarrays sendo particionados da mesma maneira que o array original.
Baseado na discussão precedente, escreva a função recursiva quickSort para classificar um array de inteiros unidimensional. A função deveria receber como argumentos um array de inteiros, um subscrito de início e um subscrito de fim. A função partition deveria ser chamada por quickSort para executar o passo de particionamento.

5.25 *(Travessia de um labirinto)* A seguinte grade de sustenidos (#) e pontos (**.**) *é* uma representação de um labirinto como um array bidimensional

```
##########
#_ . _# .....-
###.#. . .
#. . . _###.#. -
#### . # . # . # . #
# . . # . # . # . # .
# # . # . # . # . # - #
# . #.#
###### . ### . #
# . #_ .
##########
```

No array bidimensional precedente, os sustenidos (#) representam as paredes do labirinto e os pontos representam quadrados nos caminhos possíveis através do labirinto. Os movimentos só podem ser feitos para uma posição no array que contém um ponto.
Existe um algoritmo simples para atravessar um labirinto que garante que se encontre a saída (assumindo-se que exista uma saída). Se não existir uma saída, você chegará novamente à posição de início no labirinto. Coloque sua mão direita sobre a parede à sua direita e comece a caminhar para frente. Nunca remova a mão da parede. Se o labirinto vira para a direita, você segue a parede à direita. Desde que não remova a mão da parede, eventualmente você chegará à saída do labirinto. Pode haver um caminho menor que aquele que você tomou, mas você estará seguro de que sairá do labirinto se seguir o algoritmo proposto.
Escreva a função recursiva mazeTraverse para caminhar através do labirinto. A função deveria receber como argumentos um array de caracteres 1 2 por 1 2 representando o labirinto e a posição inicial no mesmo. A medida que mazeTraverse tenta localizar a saída do labirinto. deve colocar o caractere x em cada quadrado no caminho. A função deveria exibir o labirinto depois de cada movimento; assim; o usuário pode assistir como a travessia do labirinto é resolvida.
5.26 *(Gerando labirintos aleatoriamente)* Escreva uma função mazeGenerator que aceita como um argumento um array de caracteres bidimensional, de 12 por 12, e produz aleatoriamente um labirinto. A função deveria também fornecer as posições de início e de fim do labirinto. Teste sua função mazeTraverse do Exercício *5.25* usando vários labirintos gerados aleatoriamente.
5.27 *(Labirintos de qualquer tamanho)* Generalize as funções mazeTraverse e mazeGenerator dos Exercícios 5.25 e **5.26** para trabalhar com labirintos de qualquer largura e comprimento.
5.28 *(Arrays de ponteiros de funções)* Reescreva o programa da Fig. 4.23 para usar uma interface orientada por menus. O programa deveria oferecer ao usuário 5 opções, como segue (estas deveriam ser exibidas na tela):

Digite uma opção:

```
O Imprimir o array de notas
1 Achar a nota mínima
2 Achar a nota máxima
3 Imprimir a média de todas as provas para cada estudante
4 Encerrar o programa
```

Uma restrição sobre o uso de arrays de ponteiros para funções é que todos os ponteiros devem ser do mesmo tipo. Os ponteiros devem apontar para funções com o mesmo tipo de retorno e que recebem argumentos do mesmo tipo. Por essa razão, as funções na Fig. 4.23 devem ser modificadas, de modo que elas retomem o mesmo tipo e aceitem os mesmos argumentos. Modifique as funções minimum e maximum para imprimir o valor do mínimo ou do máximo e não retomar nada. Para a opção 3, modifique a função average da Fig. 4.23 para mostrar na saída a média para cada estudante (não um estudante específico). A função average não deveria retomar valores e aceitar os mesmos argumentos que printArray, minimum e **maximum.** Armaze

ne os ponteiros para as quatro funções no array processGrades e use a opção digitada pelo usuário como índice para o array para chamar cada função.

5.29 *(Modificações no simulator do Simpletron)* No Exercício 5. 1 9, você escreveu uma simulação, em software, de um computador que executa programas escritos na Simpletron Machine Language (SML). Neste exercício, propomos-lhe várias modifica- ções e melhnria para o Simulator Simpletron. Nos Exercícios 15.26 e 15.27. propomos construir um compilador que converte programas escritos em uma linguagem de programação de alto nível (uma variação de BASIC) para a SML. Algumas das modificações e melhorias seguintes podem ser necessárias para executar os programas produzidos pelo compilador. *(Nota:* algumas modificações podem conflitar com outras e, portanto, precisam ser feitas separadamente.)

a) Estenda a memória do Simulador do Simpletron para conter 1000 posições de memória para possibilitar ao Simpletron processar programas maiores.

b) Permita que o simulador execute cálculos com o operador módulo. Isto exige uma instrução adicional na Simpletron Machine Language.

e) Permita que o simulador execute cálculos de exponenciação. Isto exige uma instrução adicional na Simpletron Machine Language.

d) Modifique o simulador para usar valores hexadecimais em lugar de valores inteiros para representar instruções da Simpletron Machine Language.

e) Modifique o simulador para permitir a exibição/impressão de um caractere nova linha. Isto exige uma instrução adicional na Simpletron Machine Language.

f) Modifique o simulador para processar valores de ponto flutuante, além de valores inteiros.

g) Modifique o simulador para tratar entrada por *strings.* Sugestão: cada palavra do Simpletron pode ser dividida em dois grupos, cada um mantendo um inteiro de dois dígitos. Cada inteiro de dois dígitos representa o equivalente, em decimal, ao código ASCII de um caractere. Adicione uma instrução de linguagem de máquina que aceitará como entrada um *string* e armazenará o início do mesmo em uma posição de memória específica do Simpletron. A primeira metade da palavra nessa posição será o número de caracteres no *string* (i.e., o comprimento do *string).* Cada meia palavra sucessiva conterá um caractere ASCII expresso como dois dígitos decimais. A instrução de linguagem de máquina converte cada caractere para o seu equivalente em ASCII e atribui o mesmo a uma meia palavra.

h) Modifique o simulador para tratar a saída de *strings* armazenados no formato do item (g). Sugestão: acrescente uma instrução de linguagem de máquina que imprimirá um *string* que começa em uma determinada posição de memória do Simpletron. A primeira metade da palavra nessa posição é o número de caracteres no *string* (i.e., o comprimento do *string).* Cada meia palavra subseqüente contém um caractere ASCII expresso como dois dígitos decimais. A instrução de linguagem de máquina confere o comprimento e imprime o *string,* traduzindo cada número de dois dígitos para o seu caractere equivalente.

i) Modifique o simulador para incluir a instrução SMLDEBUG. que imprime um *dump* de memória depois que cada instrução é executada. Atribua o código de operação 44 a S4LDEBUG. A palavra +4401 liga o modo de depuração e +4400 desliga o modo de depuração.

5.3() O que faz esse programa?

```
1 II ex0530.cpp
2 #include <iostream>
3
4 using std: :cout;
5 using std::cin;
6 using std::endl;
7
8 bool mystery3( const char corist char *
9
lo int main O
11 {
12 char stringl[ 80 ], string2[ 80 ];
13
14 cout « 'Forneça dois strings: ";
15 cm » stringl » string2;
16 cout « "O resultado é
17 « miystery3( stringl, string2 ) « endi;
18
```

1

```
19 return 0;
20
```



```
21
22 bool mystery3( const char *s1, const char *s2
23 {
24 for (; *s1 != '\0' && *s2 = '\0'; sl++, s2++
25
26 if ( *s1 != *s2
27 return false;
28
29 return true;
30
```

Exercícios de manipulação de strings

5.31 Escreva um programa que usa a função strcmp para comparar dois *strings* fornecidos como entrada pelo usuário. O programa deveria informar se o primeiro *string* é menor do que, igual a ou maior do que o segundo *sfring.*

5.32 Escreva um programa que usa a função s trncmp para comparar dois *strings* fornecidos pelo usuário. O programa deveria receber como entrada o número de caracteres a serem comparados. O programa também deveria informar se o primeiro string é menor do que, igual a ou maior do que o segundo *string.*

5.33 Escreva um programa que usa a geração de números aleatórios para criar frases. O programa deve usar quatro arrays de ponteiros do tipo char, chamados artigo, substantivo, verbo e preposicao. O programa deve criar uma sentença selecionando uma palavra ao acaso de cada array, na seguinte ordem: artigo. substantivo, verbo. preposicao. ar- tigo e substantivo. A medida que cada palavra é escolhida, ela deve ser concatenada com as palavras precedentes em um array grande o bastante para conter toda a frase. As palavras devem ser separadas por espaços. Quando a frase final for mostrada na saída, ela deve começar com uma letra maiúscula e terminar com um ponto final. O programa deve gerar 20 dessas frases.

Os arrays devem ser preenchidos como segue: o array artigo deve conter os artigos "o", "a". "um", "uma, "ai- gum" e "qualquer'; o array substantivo deve conteros substantivos menino. "menina", "cachorro". "cidade e carro ; o array verbo deve conter os verbos "dirigiu", 'pulou" . correu . caminhou e saltou ; o array preposição deve conter as preposições "para", 'de", "acima de", "debaixo de" e "sobre".

Depois do programa precedente estar escrito e funcionando, modifique o programa para produzir uma pequena história consistindo em várias destas frases (que tal a possibilidade de um escritor aleatório de trabalhos de conclusão **?!).**

5.34 *(Limericks)* Um *limerick* é um poema humorístico de cinco linhas, no qual a primeira e a segunda linhas rimam com a quinta e a terceira linha rima com a quarta. Usando técnicas semelhantes àquelas desenvolvidas no Exercício *5.33,* escreva um programa em C+ + que produza *1imerick* aleatoriamente. Sofisticar este programa para produzir bons *limericks* é um problema desafiador, mas o resultado compensa o esforço!

5.35 Escreva um programa que codifica frases da língua inglesa em pig *latin.* O "pig *latin"* é uma forma de linguagem codificada, freqüentemente usada para diversão. Existem muitas variações nos métodos usados para formar frases em "pig *latin".* Para simplificar, use o seguinte algoritmo:

Para formar uma frase em "pig *latin"* partindo de uma frase inglesa, separe a frase em palavras, com a função strtok. Para traduzir cada palavra inglesa para uma palavra de "pig latim", coloque a primeira letra da palavra inglesa no fim da mesma e acrescente as letras "ay". Deste modo, a palavra "jump" se toma "uxnpj ay". a palavra "the" se toma hetay" e a palavra 'computer" se torna omputercay'. Espaços entre palavras permanecem como espaços. Assuma o seguinte: a frase inglesa consiste em palavras separadas por espaços, não existe nenhum sinal de pontuação e todas as palavras têm duas ou mais letras. A função printLatinWord deve exibir cada palavra. Sugestão: toda vez que uma unidade léxica é encontrada em uma chama- da a strtok, passe a unidade léxica por ponteiro para a função printLatinWord e imprima a palavra em "pig *latin".*

5.36 Escreva um programa que recebe como entrada um número de telefone como um *string* na forma (555) 555-5555. O programa deve usar a função strtok para extrair o código de área como uma unidade léxica, os primeiros três dígitos do número do telefone como uma unidade léxica e os últimos quatro dígitos do número do telefone também como uma unidade léxica. Os sete dígitos do número do telefone devem ser concatenados em um *string.* O programa deve converter o *string* do código de área para int e converter o *string* do número do telefone para iong. Tanto o código de área como o número do telefone devem ser impressos.

5.37 Escreva um programa que recebe como entrada uma linha de texto, separa a linha em unidades léxicas usando a função strtok e mostra na saída as unidades léxicas na ordem inversa.

5.3S Use as funções de comparação de *string* discutidas na Seção 5. 1 2.2 e as técnicas para classificação de arrays desenvolvi- das no

Capítulo 4 para escrever um programa que coloca em ordem alfabética uma lista de *strings.* Use os nomes de 10 ou 15 cidades de sua região como dados para seu programa.

5.39 Escreva duas versões de cada uma das funções de cópia e concatenação de *strings* da Fig. 5.29. A primeira versão deve usar subscritos de arrays e a segunda versão deve usar ponteiros e aritmética de ponteiros.

5.40 Escreva duas versões de cada função de comparação de *strings* da Fig. 5.29. A primeira versão deve usar subscritos de arrays e a segunda versão deve usar ponteiros e aritmética de ponteiros.

5.41 Escreva duas versões da função strlen na Fig. 5.29. A primeira versão deve usar subscritos de arrays e a segunda versão deve usar ponteiros e aritmética de ponteiros.

Seção especial: exercícios avançados de manipulação de strings

Os exercícios precedentes estão ligados ao texto e foram concebidos para testar a compreensão pelo leitor dos conceitos fundamentais de manipulação de *strings.* Esta seção inclui uma relação de exercícios de manipulação de *strings* intermediária e avançada. O leitor vai achar estes problemas difíceis e desafiadores, mas, ainda assim, agradáveis. Os problemas variam consideravelmente em dificuldade. Alguns exigem uma hora ou duas de codificação e implementação de programas. Outros são úteis para trabalhos de laboratório que podem exigir duas ou três semanas de estudo e implementação. Alguns são projetos de conclusão desafiadores.

542 *(Análise de texto)* A disponibilidade de computadores com capacidades de manipulação de *strings* resultou em algumas abordagens bastante interessantes para analisar a obra escrita de grandes autores. Muita atenção foi concentrada sobre se William Shakespeare até mesmo existiu. Alguns estudiosos acreditam que existe evidência significativa indicando que Christopher Marlowe ou outros autores escreveram realmente as obras primas atribuídas a Shakespeare. Os investigadores usaram computadores para achar semelhanças na escrita destes dois autores. Este exercício examina três métodos para analisar textos por computador.

a) Escreva um programa que lê várias linhas de texto digitadas no teclado e imprime uma tabela indicando o número de ocorrências de cada letra do alfabeto no texto. Por exemplo, a frase

`To be, or not be: that is the question:`

contém a letra "a" três vezes, a letra "b" duas vezes, nenhum "e", etc.

b) Escreva um programa que lê várias linhas de texto e imprime uma tabela indicando o número de palavras de uma letra, palavras de duas letras, palavras de três letras, etc. que aparecem no texto. Por exemplo, a frase

`Whether 'tis nobler in the mmd to suffer`

a contém

a

Comprimento da palavra Ocorrências 1

`1 0`

2 2

3 1

4 2 (incluindo tis)

5 O

6 2

7 **`1`**

c) Escreva um programa que lê várias linhas de texto e imprime uma tabela indicando o número de ocorrências de cada palavra diferente no texto. A primeira versão de seu programa deveria incluir as palavras na tabela na mesma ordem em que aparecem no texto. Por exemplo, as linhas

`To be or not be: that is the question:`
`Whether tis nobler` `iii` **`the mmd to suffer`**

contêm três vezes a palavra "tu", duas vezes a palavra "be", uma vez a palavra "or", etc. Um relatório mais interessante (e útil) seria um no qual as palavras estão classificadas em ordem alfabética.

5.43 *(Processamento de texto)* Uma função importante em sistemas de processamento de texto *éjustiflcar* - fazer o alinhamento das palavras tanto pela margem esquerda como pela margem direita de uma página. Isto gera um documento com aparência profissional, que dá a impressão de ter sido preparado em um equipamento de composição de textos profissional e não em uma máquina de escrever. Ajustificação do texto pode ser feita, em sistemas de computador, inserindo-se caracteres em branco entre cada uma das palavras em uma linha, de forma que a palavra mais à direita fica alinhada com a margem direita.

Escreva um programa que lê várias linhas de texto e imprime esse texto em formato justificado. Assuma que o texto será impresso em papel de 8 1/2 polegadas de largura e que são deixadas margens de uma polegada, tanto no lado esquerdo como no lado direito da página impressa. Assuma que o computador imprime 10 caracteres por polegada horizontal. Portanto, seu programa deve imprimir 6 1/2 polegadas de texto ou 65 caracteres por linha.

5.44 *(Imprimindo datas em váriosformatos)* Datas são comumente impressas em vários formatos diferentes na correspondência comercial. Dois dos formatos mais comuns são:

`21/07/55 e 21 de julho de 1955`

Escreva um programa que lê uma data no primeiro formato e imprime essa data no segundo formato.

5.45 *(Proteção de cheques)* Computadores são freqüentemente empregados em sistemas de impressão de cheques, tais como folha de pagamento e aplicativos de contas a pagar. Circulam muitas histórias estranhas relativas a cheques de pagamento semanais sendo impressos (por engano) com quantias de mais de **$** 1 milhão. As quantias misteriosas são impressas por sistemas de impressão de

cheques computadorizados por causa de erros humanos e/ou falhas de máquina. Os projetistas de sistemas introduzem controles em seus sistemas para prevenir a emissão de tais cheques errados.
Outro problema sério é a alteração intencional do valor de um cheque por alguém que pretende obter dinheiro em espécie usando um desses cheques de forma fraudulenta. Para impedir que uma quantia de dinheiro seja alterada, a maioria dos sistemas de impressão de cheques computadorizados empregam uma técnica chamada de *proteção de cheque.*
Os cheques destinados a serem impressos por computador contêm um número fixo de espaços em que o computador pode imprimir uma quantia. Suponha que um cheque de pagamento contém oito espaços em branco nos quais o computador deve imprimir a quantia de um cheque de pagamento semanal. Se a quantia for grande, então todos aqueles oito espaços serão preenchidos; por exemplo:
1 .230 .60 (valor do cheque)
12345678 (números de posições)

Por outro lado, se a quantidade fosse menor que **$** 1 .000,00, vários dos espaços seriam normalmente deixados em branco. Por exemplo,

```
99,87
12345678
```

contém três espaços em branco. Se um cheque for impresso com os espaços em branco, é mais fácil para alguém alterar a quantia do cheque. Para impedir que um cheque seja alterado, muitos sistemas de impressão de cheques inserem *asteriscos à esquerda* para proteger a quantia impressa, como a seguir:

```
***99,87
12345678
```


Escreva um programa que recebe como entrada uma quantia de dinheiro para ser impressa em um cheque e então imprime a quantia no formato de cheque protegido, com asteriscos à esquerda, se necessário. Assuma que nove espaços estão disponíveis para imprimir uma quantia.
5.46 *(Escrevendo o equivalente à quantia do cheque por extenso)* Continuando a discussão do exemplo anterior, que- remos reiterar a importância de projetar sistemas de impressão de cheques para impedir a alteração dos valores dos mesmos. Um método de segurança comum exige que o valor do cheque seja impresso tanto em números como também por extenso. Ainda que alguém possa alterar o valor numérico do cheque, é extremamente difícil mudar as palavras que espressam a quantia.
Muitos sistemas de impressão de cheques computadorizados não imprimem o valor do cheque por extenso. Talvez a razão principal para esta omissão seja o fato de que a maioria das linguagens de alto nível usadas em aplicativos comerciais não contém recursos de manipulação de strings adequados. Outra razão é que a lógica para escrever o equivalente em palavras do valor numérico de um cheque é um pouco elaborada.
Escreva um programa que recebe como entrada o valor numérico de um cheque e escreve o equivalente por extenso desse valor. Por exemplo, a quantia R$ l 12,43 deveria ser impressa como
Cento e doze reais e quarenta e três centavos

5.47 (Código Morse).Talvez o mais famoso de todos os esquemas de codificação seja o código Morse, desenvolvido **por Samuel Morse em** 1832 para ser usado com o sistema de telégrafo. O código Morse atribui uma série de pontos e traços para cada letra do alfabeto, cada dígito e alguns caracteres especiais (tais como ponto, vírgula, dois-pontos e ponto-e-vírgula). Em sistemas que usam sons, o ponto é representado por um som curto e o traço representado por um som longo. Outras representações dos pontos e traços são usadas em sistemas que utilizam luzes e sistemas de sinaliza- ção por meio de bandeiras.
Um espaço ou simplesmente a ausência de um ponto ou um traço indica uma separação entre palavras. Em um sistema que usa sons, um espaço é indicado por um pequeno período de tempo durante o qual nenhum som é transmitido. A versão internacional do código Morse é mostrada na Fig. 5.42.

Fig. 5.42 As letras do alfabeto expressas em código Morse internacional.

Escreva um programa que lê uma frase da língua portugesa e converte a frase em código Morse (obviamente, não use acentos nem caracteres especiais, tais como cedilha). Escreva também um programa que lê uma frase codificada em

1 Caractet	:9O		Caracter e	Códig o
1	··		*Dígitos*	
J	·—	·	1	·—
K	—·—		2	··—
L	·—··		3	···—
M	—		4	····—

N	-.		5	
O	---		6	-....
P	.--.		7	--...
Q	--.-		8	---..
R	.-.		9	----.
S	...		O	

Morse e converte a mesma para o seu equivalente na língua portuguesa. Use um branco entre cada letra codificada em Morse e três brancos entre cada palavra codificada em Morse.

5.48 (*Um programa de conversão de medidas)* Escreva um programa que ajudará o usuário a fazer conversões de medidas. Seu programa deve permitir ao usuário especificar os nomes das unidades como *strings* (i.e., centímetros, litros, gramas, etc. para o sistema métrico decimal e polegadas, quartos, libras, etc. para o sistema inglês) e deve responder a perguntas simples, tais como

"Quantas polegadas correspondem a 2 metros?"
"Quantos litros correspondem a 10 quartos?"

Seu programa deve reconhecer conversões inválidas. Por exemplo, a pergunta

"Quantos pés correspondem a 5 quilogramas?"

não tem significado porque pé" é uma unidade de comprimento, enquanto que quilograma" é uma unidade de peso.

Um projeto desafiador de manipulação de strings

5.49 *(Um gerador de palavras cruzadas)* A maioria das pessoas já resolveu problemas de palavras cruzadas, mas poucas tentaram criar um. Criar um problema de palavras cruzadas é uma tarefa difícil. Aqui é sugerido, como um projeto de manipula- ção de *strings,* um gerador de palavras cruzadas, *o* que exige sofisticação e esforço significativos. O programador deve solucionar muitos problemas para fazer funcionar até o mais simples programa gerador de palavras cruzadas. Por exemplo, como representar a grade de um problema de palavras cruzadas no computador? Deveríamos usar uma série de *strings,* ou usar arrays bidimensionais? O programador necessita de uma fonte de palavras (por exemplo, um dicionário computadorizado) e o programa necessita fazer referência a essa fonte diretamente. De que forma deveriam ser armazenadas estas palavras para facilitar as manipula- ções complexas exigidas pelo programa? O leitor ambicioso desejará gerar a "parte de dicas" do quebra-cabeça, na qual breves sugestões para cada palavra na horizontal e cada palavra na vertical estão impressas para os solucionadores do quebra-cabeça. Somente imprimir uma versão do próprio quebra-cabeça em branco já não é um problema simples.

Classes e abstração de dados

Objetivos

. Compreender os conceitos de engenharia de software de encapsulamento e ocultação de dados.
. Compreender as noções de abstração de dados e tipos de dados abstratos (ADTs).
. Ser capaz de criar ADTs em C++, ou seja, classes.
. Compreender como criar, usar e destruir objetos de classes.
• Ser capaz de controlar o acesso a membros de dados e fun . ções membro de objetos.
• Começar a apreciar o valor da orientação a objetos.

My object ali subii,ne
1 shali achieve in time.

W. S. Gilbert

É este um mundo no qual devemos esconder nossas virtudes? William Shakespeare, Twelfth Night

Yourpublic servants serve you right.
Adlai Stevenson

Rostos privados em lugares públicos
São mais sábios e simpáticos
Do que rostos públicos em lugares privados.

W. H. Auden

6

Visão geral

6.1 Introdução
6.2 Definições de estrutura
6.3 Acessando membros de estruturas
64 Implementando um tipo Time definido pelo usuário com uma Struct
6.5 Implementando um tipo de dado abstrato Time com uma Class
6.6 Escopo de classe e acesso a membros de classes
6.7 Separando a interface da implementação

6.8 Controlando o acesso a membros
6.9 Funções de acesso e funções utilitárias
6.10 Inicializando objetos de classes: construtores
6.1 1 Usando argumentos default com construtores
6.12 Usando destruidores
6.13 Quando construtores e destruidores são chamados
6.14 Usando membros de dados e funções membro
6.15 Uma armadilha sutil: retornando uma referência para um membro de dados private
6.16 Atribuição usando cópia membro a membro default
6.17 Reutilização de software
6.18 (Estudo de caso opcional) Pensando em objetos: começando a programar as classes para o simulador de elevador
Resumo . Terminologia • Erros comuns de programação • Boas práticas de programação • Dicas de desempenho • Observações de engenharia de software • Dicas de teste e depuração • Exercícios de auto-revisão• Respostas aos exercícios de auto-revisão • Exercícios

6.1 Introdução

Agora começamos nossa introdução à orientação a objetos em C++. Por que adiamos a programação orientada a objetos em C++ até o Capítulo 6? A resposta é que os objetos que construiremos serão compostos, em parte, por segmentos de programas estruturados; assim, precisávamos primeiro estabelecer uma base em programação estruturada.

Através de nossas seções "Pensando em objetos", nos finais dos Capítulos 1 a 5, introduzimos os conceitos básicos (i.e., "pensar em objetos") e a terminologia (i.e., "falar em objetos") de programação orientada a objetos em C++. Nestas seções especiais, também discutimos as técnicas do *projeto orientado a objetos (OOD, object-oriented design):* analisamos uma definição típica de problema, que pedia que se construísse um sistema (um simulador de elevador), determinamos quais classes eram necessárias para implementar o sistema, determinamos de quais atributos os objetos dessas classes necessitavam possuir, determinamos quais os comportamentos que os objetos dessas classes necessitavam exibir e especificamos como os objetos necessitavam interagir uns com os outros para atingir os objetivos globais do sistema.

Revisaremos brevemente alguns conceitos-chave e a terminologia da orientação a objetos. A programação orientada a objetos (OOP, *object-oriented programming) encapsula* dados (atributos) e funções (comportamento) em pacotes chamados de *classes;* os dados e funções de uma classe estão intimamente ligados. Uma classe é como uma planta de uma casa. A partir da planta de uma casa, um construtor pode construir uma casa. A partir de uma

r

classe, um programador pode criar um objeto. Uma planta de uma casa pode ser reutilizada muitas vezes para fazer muitas casas. Uma classe pode ser reutilizada muitas vezes para criar muitos objetos da mesma classe. As classes têm a propriedade de *ocultação de informações.* Isso significa que, embora os objetos da classe possam saber como comunicar-se uns com os outros através de *inteifaces* bem-definidas, normalmente não é permitido às classes saber como outras classes são implementadas . os detalhes de implementação estão escondidos dentro das próprias dasses. Certamente, é possível dirigir um carro eficazmente sem conhecer os detalhes de como motores, transmissões e sistemas de escapamento funcionam internamente. Veremos por que a ocultação de informações é tão crucial para a boa engenharia de software.

Em C e outras *linguagens de programação procedurais,* a programação tende a ser *orientada a ações,* enquanto que em C++, idealmente, a programação é *orientada a objetos.* Em C, a unidade de programação é *afunção.* Em C++. a unidade de programação é a *classe,* a partir da qual os objetos são eventualmente *instanciatados* (i.e., criados).
Os programadores de C concentram-se em escrever funções. Os grupos de ações que executam alguma tarefa são reunidos em funções e funções são agrupadas para formar programas. Os dados são certamente importantes em C, mas a visão é que os dados existem principalmente por causa das ações que as funções executam. Os *verbos* em uma especificação de sistema ajudam o programador de C a determinar o conjunto de funções que trabalharão juntas para implementar o sistema.
Os programadores de C++ concentram-se em criar seus próprios *tipos definidos pelo usuário,* chamados de *classes.* As classes também são chamadas de *tipos definidos pelo programador* Cada classe contém dados, bem como o conjunto de funções que manipulam os dados. Os componentes de dados de uma classe são chamados de *membros de dados.* As funções componentes de uma classe são chamadas de *funções membro* (ou *métodos,* em outras linguagens orientadas a objeto). Da mesma maneira que uma instância de um tipo primitivo da linguagem, tal como int, é chamada de *variável,* uma instância de um tipo definido pelo usuário (i.e., uma classe) é chamada de *objeto.* [Na comunidade C++, os termos variável e objeto são freqüentemente usados de modo intercambiável]. O foco da atenção em C++ é sobre as classes e não as funções. Os *substantivos* em uma especificação de sistema ajudam o programador de C++ a determinar o conjunto de classes que serão usadas para criar os objetos que trabalharão juntos para implementar o sistema.
As classes em C++ são uma evolução natural da noção de struct de C. Antes de prosseguir com os aspectos específicos de classes no desenvolvimento em C++, discutiremos estruturas e construiremos um tipo definido pelo usuário baseado em uma estrutura. As fraquezas que exporemos nesta abordagem ajudarão a motivar a noção de `classe.`

6.2 Definições de estrutura

As estruturas são tipos de dados agregados construídos usando-se elementos de outros tipos, incluindo outras structs. Considere a seguinte definição de estrutura:

```
struct Time
int hour; // 0-23
int minute; 1/ 0-59
int second; // 0-59
```

A palavra-chave struct inicia a definição de uma estrutura. O identificador Time é a *etiqueta da estrutura,* que dá nome à definição da estrutura e é usado para declarar variáveis do *tipo da estrutura.* Neste exemplo, o novo nome de tipo é Time. Os nomes declarados entre as chaves da definição de estrutura são os *membros* da estrutura. Os mcm- bros de uma mesma estrutura devem ter nomes diferentes, mas duas estruturas diferentes podem conter membros com os mesmos nomes sem que haja conflito. Cada definição de estrutura deve terminar com um ponto-e-vírgula. A explicação precedente é válida também para classes, como logo veremos; estruturas e classes são bastante semelhantes em C++.
A definição de Time contém três membros do tipo int - hour, minute e second. Os membros da estrutura podem ser de qualquer tipo e uma estrutura pode conter membros de muitos tipos diferentes. Uma estrutura

no pode, porém, conter uma instância de si própria. Por exemplo, um membro do tipo Time não pode ser declarado na definição de estrutura de Time. No entanto, um ponteiro para uma outra estrutura do tipo Time pode ser incluído. Uma estrutura contendo um membro que é um ponteiro para o mesmo tipo de estrutura é chamada de *estrutura auto-referente.* As estruturas auto-referentes são úteis para formar ligações entre estruturas de dados, tais como as listas encadeadas, filas, pilhas e árvores que veremos no Capítulo *15.*
A definição de estrutura precedente não reserva qualquer espaço na memória; porém, essa definição cria um novo tipo de dados que é usado para declarar variáveis. As variáveis do tipo estrutura são declaradas como as variáveis de outros tipos. A declaração

```
Time timeObject, timeArray[ 10 J, *tjmeptr,
&timeRef = timeObject;
```

declara timeObject como uma variável do tipo Time, timeArray como um array com 10 elementos do tipo Time. timePtr como um ponteiro para um objeto do tipo Time e timeRef como uma referência para um objeto do tipo Time que é inicializada com timeObect.

6.3 Acessando membros de estruturas

Membros de uma estrutura (ou de uma classe) são acessados usando-se os *operadores de acesso a membros* - o *operadorponto* **(.)** e o *operador seta (->).* O operador ponto acessa um membro de estrutura ou classe através do nome de variável do objeto ou através de uma referência para o objeto. Por exemplo, para se imprimir o membro hour da estrutura timeObect. usamos o comando

```
cout « timeObject.hour;
```

Para imprimir o membro hour da estrutura referenciada por timeRef. usa-se o comando

```
cout « timeRef.hour;
```

O operador seta - que consiste em um sinal de menos **(-)** e um sinal maior que (>) sem os espaços intermediários -

acessa um membro de estrutura ou membro de classe através de um ponteiro para o objeto. Suponha que o ponteiro timePtr tenha sido declarado para apontar para um objeto do tipo Time e que o endereço da estrutura timeObect tenha sido atribuído a timePtr. Para imprimir o membro hour da estrutura timeObect com o ponteiro timeptr, use os comandos

```
timePtr = &timeObject;
cout « timeptr -> hour;
```

A expressão timeptr -> hour é equivalente a (*timeptr) . hour, que derreferencia o ponteiro e acessa o membro hour usando o operador ponto. Os parênteses são necessários aqui porque o operador ponto *(.)* tem uma precedência mais alta que o operador de derreferenciamento de ponteiro *(*).* O operador seta e o operador ponto, juntamente com parênteses e colchetes *(/* **J),** têm a segunda precedência mais alta de operador (depois do operador de resolução de escopo introduzido no Capítulo 3) e se associa da esquerda para a direita.

Erro comum de programação 6.1

A expressão (*timeptr) .hour *se refere ao membro* hour *da* struct *apontadapor* timePtr. *Omitir os parênteses, como* cio timePtr . hour, *seria um erro de sintaxe porque* . *tem uma precedência mais alta que* · *assim, a expressão seria executada como se estivesse incluída entre parênteses, como em (* timePtr . hour *) . Isso seria um erro de sintaxe porque com um ponteiro você deve usar o opera- dor seta para se referir a um membro.*

6.4 Implementando um tipo Time definido pelo usuário com uma struct

A Fig. 6. 1 cria o tipo de estrutura definido pelo usuário Time com três membros inteiros: hour. minute e second. O programa define uma única estrutura Time chamada dinnerTime e usa o operador ponto para inicializar os membros da estrutura com os valores 1 8 para hour, 30 para minute e O para second. O programa então imprime a hora em formato militar (também chamado de "formato universal") e formato padrão. Note que as funções de impressão recebe referências para estruturas Time constantes. Isto faz com que estruturas do tipo Time sejam passadas para as funções de impressão por referência, deste modo eliminando o *overhead* de cópia associado com a passagem de estruturas por valor para funções - e o uso de const evita que a estrutura Time seja modificada pelas funções de impressão. No Capítulo *7,* discutimos objetos const e funções membro const.

```
1 II Fig. 6.1: figO6Ol.cpp
2 // Cria uma estrutura, inicializa seus membros e imprime a estrutura.
3 #include <iostream>
4
5 using std: :cout;
6 using std::endl;
7
8 struct Time { 1/ definição da estrutura
9 int hour; // 0-23
10 int minute; // 0-59
11 int second; II 0-59
12
13
14 void printMilitary ( const Time & ) ; II protótipo
15 void printStandard ( const Time & ) ; II protótipo
16
17 int main(
18 {
19 Time dinnerTime; II variável do novo tipo Time
20
21 II inicializa os membros com valores válidos
22 dinnerTime.hour = 18;
23 dinnerTime.minute = 30;
24 dinnerTime.second 0;
25
26 cout « "O jantar será servido às
27 printNilitary( dinnerTime );
28 cout « - hora militar,\ncjue é ";
29 printStandard( dinnerTime );
30 cout « - na hora padrão.\n';
31
32 II inicializa os membros com valores inválidos
33 dinnerTime.hour = 29;
```

```
34 dinnerTime.minute 73;
35
36 cout « "\nHora com valores inválidos:
37 printNilitary ( dinnerTime );
38 cout « endl;
39 return 0;
40 }
41
42 /1 Imprime a hora no formato militar
43 void printMilitary( const Time &t
```

Fig. 6.1 Criando a estrutura, inícializando seus membros e imprimindo a estrutura (parte 1 de 2).


```
44 {
45 cout « ( t.hour < 10 ? O" : ) « t.hour «
46 « ( t.minute < 10 ? "O": ) « t.minute;
47 }
48
49 II Imprime a hora no formato padrão
50 void printStandard( const Time &t
51 {
52 cout « ( ( t.hour == O fl t.hour 12 ) ?
53 12 : t.hour % 12
54 « « ( t.minute < 10 ? "O" : ) « t.minute
55 « « ( t.second < 10 ? "O" : ) « t.second
56 « ( t.hour < 12 ? da manhã" : da tarde" );
57 }
O jantar será servido às 18:30, hora militar,
qie é 6:30:00 da tarde na hora padrão.
Hora com valores invãlidos: 29:73
```

Fig. 6.1 Criando a estrutura, inicializando seus membros e imprimindo a estrutura (parte 2 de 2).

1 *Dica de desempenho 6.1*

-F *As estruturas são geralmente passadas através de chamadas por valor Para evitar o* overhead *de copiar uma estrutura, passe a estrutura através de uma chamada por referência.*

Observação de engenharia de software 6.1

___ *Para evitar o* overhead *de u,na chamada por valor e ainda obter o beneficio de que o dados originais da f unção que chamou estejam protegidos contra modificações, passe parâmetros de tamanho grande como referências* **const.**

Existem desvantagens em se criar novos tipos de dados com estruturas desta maneira. Uma vez que a inicialização não é especificamente exigida, *é* possível se ter dados não-inicializados e os problemas daí decorrentes. Ainda que os dados sejam inicializados, eles podem não ser corretamente inicializados. Valores inválidos podem ser atribuídos aos membros de uma estrutura (como fizemos na Fig. 6. 1) porque o programa tem acesso direto aos dados. Nas linhas 30 e 31, o programa foi capaz de atribuir facilmente valores inválidos aos membros hour e minute do objeto dinnerTime do tipo Time. Se a implementação da struct for mudada (por exemplo, a hora poderia ser representada como o número de segundos decorridos desde a meia-noite), todos os programas que usam a struct deverão ser mudados. Isso acontece porque o programador manipula diretamente a representação dos dados. Não existe nenhuma "interface" para ela para assegurar que o programador use os serviços daquele tipo de dados corre- tamente e para assegurar que os dados permaneçam em um estado consistente.

Observação de engenharia de software 6.2

___ É *importante escreverprogramas que sejam compreensíveis efáceis de se manter A mudança é a regia e não a exceção. Os programadores deveriam prever que seu código será modificado. Como veremos, o uso de classes podefacilitar a possibilidade de mudanças em programas.*

Existem outros problemas associados com estruturas ao estilo de C. Em C, as estruturas não podem ser impressas como uma unidade; em vez disso, seus membros devem ser impressos e formatados um de cada vez. Poderia ser escrita uma

função para imprimir os membros de uma estrutura em algum formato apropriado. O Capítulo 8, “Sobrecarga de operadores”, ilustra como sobrecarregar o operador **«** para possibilitar que objetos de um tipo de estrutura ou tipo de classe sejam impressos facilmente. Em C, as estruturas não podem ser comparadas em sua totalidade; elas devem ser comparadas membro a membro. O Capítulo 8 também ilustra como sobrecarregar opera- dores de igualdade e operadores relacionais para comparar objetos dos tipos estrutura e classe de C++.

A seção seguinte reimplementa nossa estrutura Time como uma classe C++ e demonstra algumas das vantagens de se criar *tipos de dados abstratos* ou classes. Veremos que classes e estruturas podem ser usadas quase que de forma idêntica em C++. A diferença entre as duas é a acessibilidade default associada com os membros de cada uma. Isso será explicado em breve.

65 Implementando um tipo de dado abstrato Time com uma class

As classes habilitam o programador a modelar objetos que têm *atributos* (representados como *membros de dados)* e *comportamentos* ou *operações* (representados *comofunções membro).* Tipos contendo membros de dados e funções membro são definidos em C++ usando a palavra-chave class.

As funções membro são às vezes chamadas de *métodos* em outras linguagens de programação orientadas a objetos, e são invocadas em resposta a *mensagens* enviadas para um objeto. Uma mensagem corresponde a uma chamada da função membro enviada de um objeto para outro ou enviada de uma função para um objeto.

Uma vez que uma classe tenha sido definida, o nome da classe pode ser usado para declarar objetos daquela classe. A Figura 6.2 contém uma definição simples para a classe Time.

Fig. 6.2 Uma definição simples da classe Time.

- Nossa definição da classe Time começa com a palavra-chave class. O *corpo* da definição da classe é delineado açao com chaves à esquerda e à direita ({ e }). A definição da classe termina com um ponto-e-vírgula. Nossa definição aque da classe Time e nossa definição da estrutura Time contêm três membros inteiros cada uma: hour, minute e) UldOs
second.
s. Nas
te do 7lrzl ***Erro comum de programação 6.2***
na ser
ruct *Esquecer o ponto-e-vírgula, nofim de uma definição de classe (ou estrutura), é um erro de sintaxe.*
. Não
ceife- As partes restantes da definição da classe são novas. Os rótulos public : e private : são chamados de *especificadores de acesso a membros.* Quaisquer membros de dados ou funções membro declarados depois do especificador de acesso a membros public (e antes do próximo especificador de acesso a membro) são acessíveis - onde quer que o programa tenha que acessar um objeto da classe Time. Qualquer membro de dados ou funções co *e* membro declaradas depois do especificador de acesso a membro private (e até o próximo especificador de *g* acesso a membro) são acessíveis somente a funções membro da classe. Os especiticadores de acesso a membros são *o uso* sempre seguidos por um dois-pontos (:) e podem aparecer várias vezes, e em qualquer ordem, em uma definição de
classe. No restante do texto, referir-nos-emos aos especificadores de acesso a membros como publie e private ressas (sem o dois-pontos). No Capítulo ***9,*** introduziremos um terceiro especificador de acesso a membro, protected, ia ser quando estudarmos herança e o papel que ela desempenha na programação orientada a objetos.
poSdO ***Boa prática de programação 6.1***
1 sua *Para maior clareza e legibilidade, use cada especificador de acesso a membro só uma vez em uma defini-*)pera- *ção de classe. Coloque membros* public *em primeiro lugar, onde eles são maisfáceis de se localizar*

1	`class Time`		
2	publie:		
3	`Time( );`		
4	`void setTime( int,`	`int,`	`int );`
5	`void printMilitary(`	`);`	
6	`void printStandard(`	`);`	
7	`private:`		
8	`inthour; II O`	- `23`	

9	intminute; /1 o	-59	
lo	intsecond II o	-59	
11	);		


```
e 12 void setTime( int, int, int ) ; /1 inicializa hora, minuto, segundo
s 13 void printMilitary() ; II imprime no formato de hora militar
D 14 void printStandard() ; II imprima hora no formato padrão
15 private:
s 16 int hour; /1 o -23
17 int minute; /1 o -59
18 int second; 1/ o -59
19
20
21 1/ construtor de Time inicializa cada membro de dados com zeros para
e 22 /1 garantir que todos os objetos do tipo Time começam em um estado consistente.
23 Time::Time() { hour = minute = second 0;
24
25 /1 Inicializa um novo valor de Time usando hora militar. Executa testes
26 1/ de validade sobre os valores de dados. Zera valores inválidos.
27 void Time::setTime( int h, int m, int s
28 {
s 29 hour (h>0 &&h<24 ) ?h :0;
s 30 minute = ( m >= O && m < 60 ) ? m : 0;
s 31 second=(s>=0&&s<60)?s:0;
e 32 }
33
s 34 II Imprime Time no formato militar
) 35 void Time: :printMilitary()
36 {
37 cout « ( hora < 10 ? "0" : - ) « hour « - :
38 « ( minuto < 10 ? "0" : - ) « minute;
39
40
41 1/ Imprime Time no formato padrão
42 void Time: :printStandard()
43 {
44 cout« ( ( hour == 0 | hour == 12 ) ? 12 : hour % 12
45 « - : - « ( minute < 10 ? 0' : - ) « minute
46 « „ : - « ( second < 10 ? "0" : ) « second
47 « ( hour < 12 ? - da manhã" : - da tarde' );
48
49
50 II Programa para testar a classe Time simples
51 int main()
52 {
53 Time t; II instancia (cria) objeto t da classe Time
54
55 cout « 'A hora militar inicial é
56 t.printNilitary();
57 cout « "\nA hora padrão inicial é
58 t.printStandard();
59
60 t.setTime( 13, 27, 6 );
61 cout « \n\nA hora militar depois de setTime é ";
62 t.printMilitary();
63 cout « "\nA hora padrão depois de setTime é ";
64 t.printStandardo);
65
```

```
66 t.setTime( 99, 99, 99 ); // tenta inicialização inválida
67 cout« '\n\nApós tentar inicialização inválida:"
68 « "\n Hora militar: ";
69 t.printMilitary();
```

Fig. 6.3 Implementação do tipo de dados abstrato Time como uma classe (parte 2 de 3).


```
70 cout « "\nHora padrão: "; Uma
71 t.printStandard() ; dessa
72 cout « endi; destri
73 return O; pe1o
74 } Discu
```

Fig. 6.3 Implementação do tipo de dados abstrato **Time** como uma classe (parte 3 de 3).

Repetindo, note que os membros de dados hour, minute e second são precedidos pelo especificador de acesso a membros private. Os membros de dados privados de uma classe normalmente não estão acessíveis fora da classe. (Repetindo, veremos no Capítulo 7 *quefriends* de uma classe podem acessar os membros privados da mesma). A filosofia aqui é que a representação real dos dados usados dentro da classe não interessa aos clientes da mesma. Por exemplo, seria perfeitamente razoável para a classe representar a hora internamente como o número de segundos decorridos desde a meia-noite. Os clientes poderiam usar as mesmas funções membro public e obter os mesmos resultados, sem estarem cientes disso. Neste sentido, diz-se que a implementação de uma classe está *escondida* de seus clientes. Tal *ocultação de informações* promove a possibilidade de mudanças dos programas e simplifica a percepção do cliente de uma classe.

Observação de engenharia de software 6.3

___Os clientes de uma classe usam-na sem conhecer os detalhes internos de como a classe é implementada. Se a implementação de classe for mudada (por exemplo, para melhorar o desempenho), desde que a interface da classe permaneça constante, o códigofonte do cliente da classe não necessita mudar (embora o cliente possa necessitar ser recompilado). Isso torna muito maisfácil modificar sistemas.

Neste programa, o construtor de Time inicializa os membros de dados com O (i.e., o equivalente em hora militar a 12 da manhã). Isto assegura que o objeto estú em um estado consistente quando é criado. Valores inválidos não podem ser armazenados nos membros de dados de um objeto do tipo Time porque o construtor é automaticamente chamado quando o objeto de tipo Time é criado e todas as tentativas subseqüentes de um cliente de modificar os dados membros são verificadas pela função setTime.

Observação de engenharia de software 6.4

___As funções membro são normalmente menores que as funções em programas não-orientados a objetos porque os dados armazenados nos membros de dados, idealmente, jáforam validados por um construtor e/ou porfunções membro que armazenam novos dados. Como os "dados já estão no objeto", a função membrofreqüentemente chamada não tem nenhum argumento ou, pelo menos, menos argumentos que as chamadas típicas de função em linguagens não-orientadas a objetos. Deste modo, as chamadas são pequenas, as definições defunção idem e os protótipos defunção também são pequenos.

Note que os dados membros de uma classe não podem ser inicializados onde são declarados no corpo da classe. Estes membros de dados deveriam ser inicializados pelo construtor da classe ou poderiam ter valores atribuídos a eles por funções de inicialização.

Erro comum de programação 6.4

Tentar inicializar um membro de dados de uma classe explicitamente, na definição da classe, é um erro de sintaxe.

memi client
A dei decla meml defini
Note defin funçí class
dach
é pre **ter os** class
Embi
a fun
a mei
para
mi:

expli coloc

A hora militar inicial é 00:00		
A hora padrão inicial é 12:00:00 da manhã		
A hora militar depois de setTime é 13:27		
A hora padrão depois de setTime é 1:27:06	**da**	**tarde**
Após tentar inicialização inválida:		
Hora militar: 00:00		
Hora padrão: 12:00:00 da manhã		

Uma função com o mesmo nome que a classe, mas precedida por um *caractere til* ***(-), é*** chamada de *destruidor* dessa classe (este exemplo não inclui um destruidor explicitamente, assim o sistema inclui um para você). O destruidor faz a "faxina de término" em cada objeto da classe antes de a memória para o objeto ser recuperada pelo sistema. Destruidores não podem aceitar argumentos e não podem, conseqüentemente, ser sobrecarregados. Discutiremos construtores e destruidores em maiores detalhes mais à frente, neste capítulo e no Capítulo 7.

Note que as funções que a classe oferece ao mundo exterior são precedidas pelo especificador de acesso a membro public. As funções públicas implementam os comportamentos ou serviços que a classe oferece a seus clientes - comumente chamados de *inteiface* da classe ou *interface* public.

Observação de engenharia de software 6.5

___ Clientes têm acesso à interftwe de uma classe, mas não devem ter acesso à implementação de uma classe.

A definição de classe contém declarações dos membros de dados da classe e das funções membro da classe. As declarações das funções membro são os protótipos de função que discutimos nos capítulos anteriores. As funções membro podem ser definidas dentro de uma classe. mas é uma boa prática de programação definir as funções fora da definição da classe.

Observação de engenharia de software 6.6

Declararfunções membro dentro da definição de uma classe (através de seus protótipos) e definir essas ***f*** *unçõesfora da definição da classe separa a inteiface da implementação da classe. Isso promove a boa engenharia de software. Os clientes de uma classe não podem ver a implementação dasfunções membro da classse e não precisam ser recompilados se a implementação muda.*

Note o uso do *operador binário de resolução de escopo* ***(*** : : ***)*** em cada definição de função membro em seguida à definição da classe na Fig. 6.3. Uma vez que é definida uma classe e são declaradas suas funções membros, as funções membros devem ser definidas. Cada função membro da classe pode ser diretamente definida no corpo da classe (em lugar de incluir o protótipo de função da classe) ou a função membro pode ser definida depois do corpo da classe. Quando uma função membro é definida depois da definição da classe correspondente, o nome da função é precedido pelo nome da classe e o operador binário de resolução de escopo ***(*** : : ***).*** Como classes diferentes podem ter os membros com os mesmos nomes, o operador de resolução de escopo "amarra" o nome do membro ao nome da classe para identificar de maneira ánica as funções membro de uma classe particular.

Erro comum de programação 6.5

Q *uando estiver definindo funções membro de uma classe fora dessa classe, omitir o nome da classe e operador de resolução de escopo aplicado ao nome dafunção é um erro de sintaxe.*

Embora uma função membro declarada em uma definição de classe possa ser definida fora da definição dessa classe, a função membro ainda está dentro do *escopo da classe,* i.e., seu nome é conhecido só dos outros membros da classe, a menos que seja referenciado através de um objeto da classe, uma referência a um objeto da classe ou um ponteiro para um objeto da classe. Falaremos mais sobre escopo de classes mais à frente.

Se uma função membro é definida em uma definição de classe, a função membro é automaticamente colocada mime. Funções membro definidas fora de uma definição de classe podem ser transformadas em funções mime explicitamente, usando-se a palavra-chave mnimne. Lembre-se de que o compilador reserva-se ao direito de não colocar mnimne qualquer função.

1 ***Dica de desempenho 6.2***

.1:. *Definir uma pequenafunção membro dentro da definição de uma classe automaticamente a coloca* inimne *(se o compilador decidir assim). Isto pode melhorar o desempenho, mas não promove a melhor engenharia de software porque os clientes da classe serão capazes de ver a implementação dafunção, e seu código deve ser recompilado se a definição dafunção* mnimne *muda.*

6.6 Escopo de classe e acesso a membros de classes

Os membros de dados de uma classe (variáveis dec'aradas na definição da classe) e funções membro (funções declaradas na definição da classe) pertencem ao *escopo da classe.* As funções não-membro são definidas como *escopo de arquivo.*

Dentro do escopo de uma classe, os membros da classe são imediatamente acessíveis por todas as funções membro daquela classe e podem ser referenciados por nome. Fora do escopo de uma classe, os membros da classe são referenciados através de um dos *handles** de um objeto - um nome de objeto, uma referência a um objeto ou um ponteiro para um objeto. [Veremos no Capítulo 7 que um *handie* implícito é inserido pelo compilador em toda referência a um membro de dados ou a uma função membro em um objetoj.

As funções membro de uma classe podem ser sobrecarregadas, mas só por outras funções membro da classe.

Para sobrecarregar uma função membro. simplesmente forneça na definição de classe um protótipo para cada versão da função sobrecarregada e forneça uma definição de função separada para cada versão da função.

Variáveis definidas em uma função membro têm *escopo defunção* - são conhecidas somente naquela função. Se uma função membro define uma variável com o mesmo nome que uma variável com escopo de classe. a variável com escopo de classe *é* escondida pela variável com escopo de função no escopo da função. Uma variável escondida como essa pode ser acessada precedendo-se o seu nome com o nome da classe seguido pelo operador de resolução de escopo **(: :).** Variáveis globais escondidas podem ser acessadas com o operador de resolução de escopo unário (ver Capítulo 3).

Os operadores utilizados para acessar membros de classes são idênticos aos operadores usados para acessar membros de estruturas. O *operadorde seleção de membroponto* **(.)** *é* combinado com o nome de um objeto ou com uma referência a um objeto para acessar os membros do objeto. O *operador de seleção de membro seta* **(->) é** combinado com um ponteiro para um objeto para acessar os membros daquele objeto.

A Fig. 6.4 usa uma classe simples, chamada Count, com o membro de dados public x, do tipo int. e a função membro public print, para ilustrar o acesso aos membros de uma classe com os operadores de seleção de membro. O programa cria (define) três variáveis relacionadas ao tipo Count **-** counter. **counterRef (uma** referência a um objeto do tipo Count) e counterPtr (um ponteiro para um objeto do tipo Count). A variável **counterRef** *é* definida para fazer referência a counter e a variável counterPtr é definida para apontar para **counter.** *E importante notar que o membro de dados xfoi criado aqui como* public *simplesmente para de- inonstrar como membros públicos são acessados usando* handles *(i.e., um nome, uma referência ou um ponteiro). Como já falamos, dados são tipicamente declarados como* **private.** *como faremos na maioria dos exemplos subseqüentes.* No Capítulo **9,** "Herança", às vezes declararemos dados como protected.

```
1 II Fig. 6.4: figO6O4.cpp
2 // Demonstrando os operadores de acesso a membros de classes . e ->
3 II
4 II ATENÇÃO - NOS EXEMPLOS FUTUROS EVITAREMOS DADOS PÚBLICOS
5 #include <iostream>
6
7 using std: :cout;
8 using std: :endl;
9
lo II Classe Count simples
11 class Count {
12 public:
13 int x;
14 void print() {cout « x « endi; }
15 };
16
17 int main()
18
```

Fig. 6.4 Acessando os membros de dados e as funções membro de um objeto através de cada tipo de *handie* de objeto - através do nome do objeto, através de uma referência e através de um ponteiro para o objeto (parte 1 de 2).

* N. de R.T.: Uma tradução seria *manípulo,* ou seja, algo que se pode usar para acessar e manipular outro objeto; no entanto, o termo original em inglês é largamente usado e, por isso, foi mantido no texto.


```
19 Count counter, 1/ cria objeto contador
20 *counterptr =&counter, 1/ ponteiro para contador
21 &counterRef =counter; /1 referência para contador
```

```
22
23 cout « "Atribui 7 a x e imprime usando o nome do objeto:
24 counter.x = 7; /1 atribui 7 para o membro de dados x
25 counter.print() ; II chama a função membro print
26
27 cout « "Atribui 8 para x e imprime usando urna referência: ";
28 counterRef.x 8; 1/ atribui 8 para o membro de dados x
29 counterRef.print() ; II chama a função membro print
30
31 cout « "Atribui 10 para x e imprime usando um ponteiro: ";
32 counterPtr->x = 10; II atribui 10 para membro de dados x
33 counterPtr->print() ; II chama a função membro print
34 return 0;
35
Atribui 7 para x e imprime usando o nome do objeto: 7
Atribui 8 para x e imprime usando uma referência: 8
Atribui 10 para x e imprime usando um ponteiro: 10
```

Fig. 6.4 Acessando os membros de dados e as funções membro de um objeto atraves ae caoa upo ae *handie* de objeto - através do nome do objeto, através de uma referência e através de um ponteiro para o objeto (parte 2 de 2).

6.7 Separando a interface da implementação

Um dos princípios fundamentais da boa engenharia de software é separar a interface da implementação. Isso torna mais fácil modificar os programas. Até onde interessa aos clientes de uma classe, mudanças na implementação da classe não afetam o cliente, desde que a interface da classe originalmente fornecida ao cliente se mantenha inalterada (a funcionalidade da classe pode ser expandida para além da interface original).

Observação de engenharia de software 6.10

____Coloque a declaração da classe em um arquivo de cabeçalho para ser incluído por qualquer cliente que queira usar a classe. Isso constitui a interface pública da classe (e fornece ao cliente os protótipos de

f unção de que ele necessita para poder chamar asfunções membro da classe). Coloque as definições das

f unções membro da classe em um arquivo-fonte. Isso constitui a implementação da classe.

Observação de engenharia de software 6.11

____Os clientes de uma classe não necessitam de acesso ao código-fonte da classe para utilizá-la. Porém, os clientes necessitam poder ligar seu código ao código objeto da classe. Isto encoraja vendedores de sofiware independente (JSVs, independent software vendors) *a oferecer bibliotecas de classes para venda ou licença de uso. Os ISVs oferecem em seus produtos somente os arquivos de cabeçalho e os módulos objeto. Nenhuma informação proprietária é revelada - como seria o caso sefosse fornecido código-fonte. A comunidade de usuários de C+ + se beneficia tendo mais bibliotecas de classe disponíveis, produzidas por ISVs.*

Na realidade, as coisas não são tão cor-de-rosa. Os arquivos de cabeçalho contêm partes da implementação e sugestões sobre outras partes da implementação. Funções membro mime, por exemplo, necessitam estar em um arquivo de cabeçalho, de forma que quando o compilador compilar um código-fonte cliente, o cliente possa incluir (com a diretiva #include do pré-processador) a definição da função mime no seu lugar. Os membros privados são listados na definição da classe no arquivo de cabeçalho, de modo que estes membros sejam visíveis para os clientes,

Porém, os software ou licença l'o. Nenhuomunidar ISVs.
o e sugesum arquiIuir (com vados são s clientes,

embora não possam acessar os membros private. No Capítulo 7, mostramos como usar uma assim denominada *classe proxy* para esconder até os dados private de uma classe dos clientes da mesma.

Observação de engenharia de software 6.12

As infrrniações importantes sobre a interface de uma classe deveriam ser incluídas no arquivo de cabeça- lho. As informações que serão usadas só internamente na classe e não serão necessitadas por clientes da classe deveriam ser incluídas no arquivo-fonte não-divulgado. Isso é outro exemplo do princípio do mínimo privilégio.

A Fig. 6.5 divide o programa da Fig. 6.3 em múltiplos arquivos. Quando se desenvolve um programa em C++, cada definição de classe normalmente é colocada em um *arquivo de cabeçalho* e as definições das funções membro daquela classe são colocadas em *arquivos de código-fonte* com o mesmo nome básico. Os arquivos de cabeçalho são incluídos

(através de #include) em cada arquivo em que a classe é usada e o arquivo de código-fonte *é* compilado e ligado com o arquivo contendo o programa principal. Veja a documentação do seu compilador para determinar como compilar e ligar programas que consistem de múltiplos arquivos-fonte.

A Fig. 6.5 consiste no arquivo de cabeçalho timel .h em que a classe Time *é* declarada. o arquivo timel .cpp em que as funções membro da classe Time são definidas e o arquivo figO6O5 .cpp em que a função main *é* definida. A saída deste programa é idêntica à do programa da Fig. 6.3.

```
1 II Fig. 6.5: timel.h
2 l/Declaração da classe Time
3 II As funções membro são definidas em timel.cpp
4
5 II evita inclusões múltiplas de arquivo de cabeçalho
6 #ifndef TIME1H
7 #define TIME1H
```

```
23
24
25
26
27 using std::cout;
28
```

aa tipo cie teiro para

Isso torna entação da inalterada
1iente que tótipos de nições das

```
8
9
lo
11
12
13
14
15
16
17
18
19
20
21
```

```
II Definição do tipo de dados abstrato Time class Time
```

public:

```
Time();
void setTime( int, int, int ); void printMilitary();
void printStandard();
private:
int hour;
int minute;
int second;
```

};

```
II construtor
II inicializa hora, minuto, segundo
II imprime no formato de hora militar
II imprime no formato de hora padrao
```

II O .23
II O .59

```
II O - 59

22 #endif
```

Fig. 6.5 Separando a interface da classe **Time** da implementação - **timel** . **h.**

```
II Fig. 6.5: timel.cpp
II Definições das funções membro para a classe Time #include <iostream>

29 #include "timel.h
30
31 I/ O construtor de Time inicializa cada membro de dados com zeros.
```

Fig. 6.5 Separando a intertace da classe Time da implementação - **timel** . cpp (parte 1 de 2).


```
32 II Para que todos os objetos do tipo Time comecem em um estado consistente.
33 Time::Time() { hour = minute = second = O;}
34
35 II Inicializa um novo valor do tipo Time usando hora militar. Testa a
36 /1 validade dos dados. Inicializa valores inválidos com zeros.
37 void Time::setTime( int h, int m, int s
38 {
39 hour = (h>=O &&h<24 ) ?h :0;
40 minute = ( m >= O && m < 60 ) ? m : 0;
41 second = ( s >= O && s < 60 ) ? s : 0;
42 )
43
44 II Imprime Time no formato militar
45 void Time: :printNilitary()
46 {
47 cout « ( hour < 10 ? "0" : - ) « hour « :"
48 « ( minute < 10 ? '0" : " ) « minute;
49 }
50
51 II Imprime hora no formato padrão
52 void Time: :printStandard()
53 {
54 cout« ( ( hour == O hour == 12 ) ? 12 : hour % 12
55 « ":" « ( minute < 10 ? "0" : " ) « minute
56 « ":" « ( second < 10 ? "0" : " ) « second
57 « ( hour < 12 ? da manhã" : " da tarde" );
58 }
```

Fig. 6.5 Separando a interíace da classe Time da implementação - timel . cpp (parte 2 de 2).

```
59 II Fig. 6.5: figO6O5.cpp
60 II Programa para testar a classe Timel
61 /1 NOTA: Compilar com timel.cpp
62 #include <iostream>
63
64 using std::cout;
65 using std::endl;
66
67 #include "timel.h"
68
69 II Programa para testar a classe Time simples
```

```
70 int maia O
71 {
72 Time t; II instancia o objeto t da classe Time
73
74 cout « A hora militar inicial é ;
75 t.printNilitary();
76 cout « "\nA hora padrão inicial é
77 t.printStandard();
78
79 t.setTime( 13, 27, 6 )
80 cout « \n\nHora militar depois de setTime é ;
81 t.printNilitary();
82 cout « "\nHora padrão depois de setTime é
83 t.printStandard O;
84
```

Fig. 6.5 Separando a interface da classe Time da implementação - figo6_05 . cpp (parte 1 de 2).


```
risistente. 85 t.setTime( 99, 99, 99 ) ; II tenta inicialização inválida
86 cout « "\n\nApós tentar inicialização inválida:\n"
87 « "Hora militar: ";
'esta a 88 t.printMilitary();
89 cout « \nHora padrão: ;
90 t.printStandard();
91 cout « endi;
92 return O;
93
A hora militar inicial é 00:00
A hora padrão inicial é 12:00:00 da manhã
Hora militar depois de setTime é 13:27
Hora padrão depois de setTime é 1:27:06 PM
Depois de tentar inicialização inválida:
Hora militar: 00:00

Hora padrão: 12:00:00 da manhã 1
```

Fig. 6.5 Separando a interface da classe Time da implementação - figO6O5 . cpp (parte 2 de 2).

Note que a declaração da classe está inclusa no seguinte código para o pré-processador:

```
II evita inclusões múltiplas de arquivo de cabeçalho
#ifndef TIME1H
#define TIME1H
```

Je 2).

```
#endif
```

Quando construirmos programas maiores, outras definições e declarações também serão colocadas em arquivos de cabeçalho. As diretivas para o pré-processador precedentes evitam que o código entre #ifndef e #endif seja incluído novamente no programa se o nome TIME1Hjá foi definido. Se o cabeçalho ainda não foi incluído em um arquivo, o nome TI1H é definido pela diretiva #define e os comandos do arquivo de cabeçalho são incluídos. Se o cabeçalhojá foi incluído anteriormente, TIlHjá está definido e o arquivo de cabeçalho não é incluído novamente. As tentativas de incluir (inadvertidamente) várias vezes um arquivo de cabeçalho acontecem tipicamente em programas grandes, com muitos arquivos de cabeçalho, os quais, eles mesmos, podem incluir outros arquivos de cabeçalho. Nota: a convenção que usamos para o nome constante simbólico nas diretivas para o pré-processador é simplesmente o nome do arquivo de cabeçalho com o caractere sublinhado (_) substituindo o ponto.



Dica de teste e depuração 6.2

Use as diretivas para o pré-processador #ifndef, #define *e* #endif *para evitar que arquivos de cabeçalho sejam incluídos mais de uma vez em um programa.*

Boa prática de programação 6.2

Use o nome do arquivo de cabeçalho com o ponto substituído por um sublinhado nas diretivas para o pré-processador #ifndef *e* #define *de um arquivo de cabeçalho.*

6.8 Controlando o acesso a membros

Os especiticadores de acesso a membro public e private (e protected. como veremos no Capítulo 9, "Herança") são

usados para controlar o acesso a membros de dados e funções membro de uma classe. O modo de acesso default para classes é private: assim, todos os membros depois do cabeçalho da classe e antes do primeiro especificador de acesso a membro são private. Após cada especificador de acesso a membro, o modo que foi
1 de 2). invocado por esse especificador de acesso a membro se aplica até o próximo especificador de acesso a membro ou

até a chave à direita de término **(})** da definição da classe. Os especificadores de acesso a membro public, private e protected podem ser repetidos, porém tal uso é raro e pode ser confuso.

Os membros privados de uma classe podem ser acessados somente por funções membro (e friends, como veremos no Capítulo 7) dessa classe. Os membros public de uma classe podem ser acessados por qualquer função no programa.

O propósito primário dos membros public é apresentar aos clientes da classe uma visão dos *serviços* (comportamentos) que a classe oferece. Este conjunto de serviços forma a *interface pública* (public) da classe. Os clientes da classe não precisam se preocupar com a forma como a classe realiza suas tarefas. Os *membros privados* (private) de uma classe, assim como as definições de suas funções membro públicas, não são acessíveis aos clientes de uma classe. Esses componentes formam a *implementação* da classe.

Observação de engenharia de software 6.13

C++ encoraja a construção de programas que sejam independentes da implementação. Quando muda a implementação de uma classe usada por código independente de implementação, aquele código não necessita ser modificado. Se qualquer parte da interface da classe muda, o código independente de implementação precisa ser recompilado.

Erro comum de programação 6.6

Uma tentativa por uma função que não é um membro de uma classe particular (ou um friend *dessa classe) de acessar um membro privado dessa classe é um erro de sintaxe.*

A Fig. 6.6 demonstra que membros de classe private só são acessíveis através da interface pública da classe usando funções membro public. Quando este programa é compilado, o compilador gera dois erros. declarando que o membro private especificado em cada comando não está acessível. A Fig. 6.6 faz a inclusão de timel .h e é compilada com timel .cpp da Fig. 6.5.

Boa prática de programação 6.3

Se você optarpor listar os membros private *primeiro, em uma definição de classe, use explicitamente o espec(ficador de acesso a membro* private, *apesar dofato de que* private é *assumido por default. Isso melhora a clareza do programa. Nossa preferência é listar primeiro os membros* public *de uma classe para enfatizar a interface da classe.*

```
1 II Fig. 6.6: figO6O6.cpp
2 /1 Demonstra erros resultantes de tentativas
3 II para acessar membros private de classes.
4 #include <iostreain>
5
6 using std: :cout;
7
8 #include "timel.h
9
lo int main O
11 {
12 Time t;
13
14 II Erro: 'Time: :hour' não está acessível
15 t.hour = 7;
16
17 II Erro: 'Time::minute' não está acessível
18 cout « "minuto =" « t.minute;
19
20 return O;
21 }
```

Fig. 6.6 Tentativa errônea de acessar membros private de uma classe (parte 1 de 2).

Fig. 6.6 Tentatíva errônea de acessar membros private de uma classe (parte 2 de 2).

Boa prática de programação 6.4

Apesar dofato de que os especificadores de acesso a membro public *e* private *podem ser repetidos e misturados, liste todos os membros* public *de uma classe em um grupoprimeiro e então liste todos os membros* private *em outro grupo. Issofocaliza a atenção do cliente sobre a interfacepública da classe, em vez de sobre a implementação da classe.*

Observação de engenharia de software 6.14

___Mantenha todos os membros de dados de uma classe private. *Forneçafunções membro* public *para inicializar os valores dos membros de dados* private *e obter os valores dos membros de dados* private.

Esta arquitetura ajuda a esconder a implementação de uma classe de seus clientes, o que reduz bugs e aumenta afacilidade do programa ser modificado.

Um cliente de uma classe pode ser uma função membro de outra classe ou pode ser uma função global (i.e., uma função ao estilo de C "solta" ou "livre" no arquivo e que não é uma função membro de qualquer classe).

O acesso default a membros de uma classe é private. O acesso a membros de uma classe pode ser explicitamente inicializado com public, protected (como veremos no Capítulo 9) ou private. O acesso default a membros de struct é pub].ic. O acesso a membros de uma struct também pode ser explicitamente inicializado com public, protected ou private.

Observação de engenharia de software 6.15

___Os projetistas de classes usam membros private, protected *e* publie *para garantir a noção de ocultação de informações e seguir o princípio do mínimo privilégio.*

Só porque dados de classe são private não significa necessariamente que os clientes não possam fazer mudanças nesses dados. Os dados podem ser mudados através de funções membro ou, ainda, por friends daquela classe. Como veremos, estas funções devem ser projetadas visando-se a assegurar a integridade dos dados.

O acesso a dados private de uma classe deveria ser cuidadosamente controlado pelo uso de funções membro chamadas de *funções de acesso* (também chamadas de *accessor methods).* Por exemplo, para permitir aos clientes ler o valor de dados private, a classe pode oferecer uma função *get.* Para permitir aos clientes modificar

riO
ao

Mensagens de erro do compilador Borland C+ + com linha de comando

ii-
os
os

```
Timel .cpp:
Fig0606 .cpp:
Error E2247 figO6O6.cpp 15:
'Time: :hour' is not accessible in function main() Error E2247 figO6_06.cpp 18:
'Time::minute' is not accessible in function main())
*** 2 errors in Compile
```

Mensagens de erro do compilador Microsofi Visual C+ +

a
e-
le

Compiling ..

```
FigO6_06 .cpp:
D:\figO6O6.cpp(15) : error C2248: 'hour' : cannot access private member declared in class 'Time'
D:\FigO6_06\timel.h(18) : see declaration of 'hour'
D:\FigO6_06.cpp(18) : error C2248: 'minute' : cannot access private member declared in class 'Time'
D:\timel.h(19) : see declaration of 'minute'
Error executing cl.exe.
test.exe - 2 error(s) .O warning(s)
```

se lo

te Ir.

dados private. a classe pode oferecer uma função *set.* Tal modificação pode parecer que viola a noção de dados private. Mas uma função membro *set* pode oferecer recursos para a validação de dados (tal como uma verificação de intervalo de validade) para assegurar que o valor seja corretamente inicializado. Uma função *set* pode também fazer a tradução entre a forma dos dados usados na interface e a forma usada na implementação. Uma função *get* não necessita expor os dados em seu formato "bruto"; em vez disso, a função *get* pode editar os dados e limitar a visão deles que o cliente terá.

Observação de engenharia de software 6.16

_____ O projetista de classes não necessitafornecerfunções set *e/ou* get *para cada item de dados* private; *esses recursos devem serfornecidos somente quando for apropriado. Se o serviço é útil para o código cliente, esse serviço deve ser incluído na interface* public *da classe.*



Dica de teste e depuração 6.3

Tornar os membros de dados de uma classe private *e asfunções membro da classe pub].icfacilita a depuração porque os problemas com manipulações de dados ficam restritos ou a funções membro da classe ou a* friends *da classe.*

6.9 Funções de acesso e funções utilitárias

Nem todas as funções membro necessitam ser tornadas public para atuar como parte da interface de uma classe. Algumas funções membro permanecem private e servem *comofunções utililitórias* para as outras funções da classe.

Observação de engenharia de software 6.17

Funções membro tendem a se enquadrar em várias categorias diferentes: funções que lêem e retornam o valor de membros de dados private: *funções que inicializam o valor de membros de dados* private:

funções que implementam os serviços da classe; efunções que executam várias tarefas mecânicas para a classe, tais como inicializar objetos da classe, atribuir a objetos da classe, fazer a conversão entre classes e tipos primitivos ou entre classes e outras classes e administrar a memória para objetos da classe.

As funções de acesso podem ler ou exibir dados. Outro uso comum das funções de acesso é testar a verdade ou falsidade de condições - tais funções são freqüentemente denominadasfunções *predicado.* Um exemplo de uma função predicado seria uma função isEmpty para uma classe contêiner qualquer - uma classe capaz de manter muitos objetos - tal como uma lista encadeada, uma pilha ou uma fila. Um programa testaria isEmpty antes de tentar ler outro item do objeto contêiner. Uma função predicado isFuil poderia testar um objeto de uma classe contêiner para determinar se ele não tem mais espaço para novos items. Um conjunto de funções predicado úteis para nossa classe Time poderia ser isAN e isPM.

A Fig. 6.7 demonstra a noção de *umafunção utilitária* (também chamada *defunção auxiliar).* Uma função utilitária não faz parte da interface de uma classe; porém, é uma função membro private que suporta a operação das funções membro public da classe. As funções utililitárias não são projetadas para serem usadas pelos clientes de uma classe.

Fig. 6.7 Usando uma função utilitária - **salesp . h** (parte 1 de 2).

```
1   // Fig. 6.7: salesp.h
2   // Definição da classe SalesPerson
3   // Funções membro definidas em salesp.cpp
4   #ifndef SALESPH
5   #define SALESPH
6
7   class SalesPerson
8   public:
9   SalesPersonO;                    // construtor
lo  void getSalesFromUserO;          // lê dados de vendas do teclado
```

```
11 void setSales( int, double ); II Usuário fornece dados
12 II de vendas de um mês
13 void printAnnua1Sa1es
15 private:
16 double totaiAnnua1Sa1es // função utilitária
17 double sales[ 12 ]; // 12 dados de vendas mensais
18 );
19
20 #endif
```

Fig. 6.7 Usando a função utilitária salesp.h (parte 2 de 2).

```
21 // Fig. 6.7: salesp.cpp
22 II Funções membro para a classe SalesPerson
23 #include <iostream>
24
25 using std: :cout;
26 using std::cin;
27 using std::endl;
28
29 4include <iomanip>
30
31 using std::setprecision;
32 using std: :setiosflags;
33 using std::ios;
34
35 #include "salesp.h"
36
37 II Função construtor inicializa array
38 SalesPerson: :SalesPerson()
39
40 for ( int i = 0; i < 12; i++
41 sales[ i J 0.0;
42
43
44 II Função para obter 12 dados de vendas
45 II fornecidos pelo usuário pelo teclado
46 void SalesPerson: :getSalesFromUser(.)
47
48 double salesFigure;
49
50 for(inti1;i<=12;i+4-){
51 cout « 'Digite o volume de vendas para o mês -« i « -: ";
52
53 cm » salesFigure;
54 setSales ( i ,salesFigure );
55 )
56 }
57
58 1/ Função para inicializar um dos 12 dados de vendas mensais.
59 II Note que o valor de mês deve ser de O até 11.
60 void SalesPerson: :setSales ( int month, double amount
61 {
62 if ( month >= 1 && month <= 12 && amount > O
63 sales[ month -1 ] amount; /1 ajusta para subscritos de O a 11
64 else
65 cout « "Mês ou valor de vendas inválido" « endi;
66 }
67
```

Fig. 6.7 Usando uma função utlitária salesp.cpp (parte 1 de 2).


```
68 1/ Imprime o total anual de vendas
69 void SalesPerson: :printAnnualSales()
```

```
70 {
71 cout « setprecision( 2
72 « setiosflags( ios::fixed 1 ios::showpoint
73 « "\no total anual de vendas é: $"
74 « totalAnnualSales() « endl;
75 }
76
77 1/ Função utilitária private para totalizar vendas do ano
78 double Salesperson: :totalAnnualSales()
79 {
80 double total 0.0;
81
82 for (int i =0; i < 12; i++
83 total += sales[ i ];
84
85 return total;
86 }
```

Fig. 6.7 Usando uma função utilitária - salesp . cpp (parte 2 de 2).

A classe SalesPerson tem um array com 12 valores de vendas mensais inicializados pelo constmtorcom zeros e preenchidos com valores fornecidos pelo usuário através da função setSales. A função membro pública printAnnualSales imprime o total das vendas para os últimos 12 meses. A função utilitária totalAnnualSales totaliza os 12 dados de vendas mensais para auxiliar printAnnualSales. A função membro printAnnualSales edita os dados de vendas para o formato de quantia em dólares. Note que main só contém uma seqüência simples de chamadas de funções membro não existe nenhuma estrutura de controle.

```
87 1/ Fig. 6.7: fígO6O7.cpp
88 1/ Demonstrando uma função utilitária
89 II Compila com salesp.cpp
90 #include "salesp.h"
91
92 int main O
93 {
94 SalesPerson s; // cria objeto s da classe SalesPerson
95
96 s.getSalesFromUserO; II note a sequência simples do código
97 s.printAnnualSalesO; /1 não há nenhuma estrutura de controle em main
98 return 0;
99 }
```

```
Digite o voluiie de vendas para o mês 1: 5314.76
Digite o volume de vendas para o mês 2: 4292.38
Digite o volume de vendas para o mês 3: 4589.83
Digite o volume de vendas para o mês 4: 5534 .03
Digite o volume de vendas para o mês 5: 4376.34
Digite o volume de vendas para o mês 6: 5698.45
Digite o volume de vendas para o mês 7: 4439.22
Digite o volume de vendas para o mês 8: 5893.57
Digite o volume de vendas para o mês 9: 4909.67
Digite o volume de vendas para o mês 10: 5123.45
Digite o volume de vendas para o mês 11: 4024.97
Digite o volume de vendas para o mês 12: 5923.92
O total anual de vendas é: $60120.59
```

Fig. 6.7 Usando uma função utilitária - figO6O7 . cpp.

Observação de engenharia de software 6.18

_____ Um fenômeno observado na programação orientada a objetos é que, uma vez que esteja definida uma classe, criar e manipular objetos daquela classe geralmente envolve apenas efetuar uma seqüência simples de chamadas a funções membro - poucas, senão nenhuma, estruturas de controle são necessárias. Em contraste com isso, é comum ter-se estruturas de controle na implementação das funções membro de uma classe.

6.10 Inicializando objetos de classes: construtores

Quando um objeto de uma classe é criado, seus membros podem ser inicializados pela função *construtor* daquela classe. Um construtor é uma função membro da classe com o mesmo nome da classe. O programador fornece o construtor, que é então invocado automaticamente toda vez que é criado (instanciado) um objeto daquela classe. Construtores podem ser sobrecarregados para oferecer várias maneiras de inicializar objetos de uma classe. Os membros de dados devem ser inicializados em um construtor da classe, ou então seus valores podem ser *estabelecidos* mais à frente, depois de o objeto ser criado. Entretanto, é considerada uma boa prática de programação e engenharia de software assegurar que um objeto seja completamente inicializado antes que o código cliente invoque as funções membro do objeto. Em geral, você não deve confiar no código cliente para assegurar que um objeto seja inicializado adequadamente.

Erro comum de programação 6.7
Os membros de dados de uma classe não podem ser inicializados na definição da classe.

Erro comum de programação 6.8
Tentar declarar um tipo de retorno para um construtor e/ou tentar retornar um valor de um construtor são erros de sintaxe.

Boa prática de programação 6.5
Quando for apropriado (quase sempre), forneça um construtor para assegurar que todo objeto seja corretamente inicializado com valores que tenham significado. Em particular membros de dados do tipo ponteiro deveriam ser inicializados com algum valor de ponteiro válido ou com O.



Dica de teste e depura ção 6.4

Toda função membro (e friend*) que modifica membros de dados* private *de um objeto deve assegurar que os dados permaneçam em um estado consistente.*

Quando um objeto de uma classe é declarado, podem ser fornecidos *inicializadores* entre parênteses, à direita do nome do objeto e antes do ponto-e-vírgula. Esses inicializadores são passados como argumentos para o construtor da classe. Logo veremos vários exemplos dessas *chamadas* para um *construtor.* [Nota: embora os programadores não chamem explicitamente construtores, os programadores ainda podem fornecer dados que são passados para construtores como argumentosi.

6.11 Usando argumentos default com construtores

O construtor de timel .cpp (Fig. 6.5) inicializou hour. minute esecond com O (i.e., meia-noite em hora militar). Construtores podem conter argumentos default. A Fig. 6.8 redefine a função construtor de Time para incluir parâmetros default com valor zero para cada variável. Fornecer parâmetros default ao construtor faz com que. ainda que nenhum valor seja fornecido em uma chamada para o construtor, o objeto ainda tenha assegurada uma inicialização em um estado consistente, devido aos parâmetros default. Um construtor fornecido pelo programador .que torna default todos os seus parâmetros (ou que não exija explicitamente nenhum argumento) é também um *construtor default,* i.e., um construtor que pode ser invocado sem parâmetros. Pode haver somente um construtor default por classe.


```
1 // Fig. 6.8: time2.h
2 // Declaração da classe Time
3 // Funções membros são definidas em time2.cpp
4
```

```
1 5 // diretivas para o pré-processador que evitam
6 // inclusões múltiplas de arquivos de cabeçalho
7 #ifndef TIME2H
8 #define TIME2H
-9
10 // Definição do tipo de dados abstrato Time
11 class Time
12 public:
13 Time( int =0, int =0, int =0 ); // construtor default
14 void setTime( int, int, int ); // inicializa hora, minuto, segur
15 void printMilitaryQ; // imprime hora no formato nilit
16 void printStandardQ; // imprime hora no formato padrã
17 private:
18 inthour; //0-23
19 int minute; // O -59
20 int second; // O -59
21 };
22
23 #endif
```

Fig. 6.8 Usando um construtor com argumentos default - time2 - h.

```
24 // Fig. 6.8: time2.cpp
25 // Definições de funções membro para a classe Time
26 #include <iostream>
27
28 using std::cout;
29
30 #include “time2.h”
31
32 // Construtor de Time inicializa cada membro de dados com zeros
33 // Garante que todos os objetos do tipo Tine comecem em um estado consisteM
34 Time: :Time( int hr, int mi int sec
35 { setTime( hr, mi sec );
36
37 // Inicializa um novo valor de Time usando hora militar. Faz testes d
38 // validade com os valores de dados. Inicializa valores inválidos com zej
39 void Time::setTime( int h, int m, int s
40 {
41 hour =(h>=0 &&h<24 ) ?h :0;
42 minute (m>0&&m<60) ?m: 0;
43 second = ( s >= O && s < 60 ) ? s : 0;
44 }
45
46 // Imprime Time no formato militar
47 void Time::printNilitary()
48 {
4 49 cout« ( hour <10 ? “0” : “ ) « hour « “:“
50 « ( minute < 10 ? “0” : “ ) « minute;
51 }
52
53 // Imprime Time no formato padrão
54 void Time::printStandard()
55 {
56 cout « ( (hour == O hour == 12) ? 12 : hour % 12
```

Fig. 6.8 Usando um construtor com argumentos default - time2 - cpp (parte 1 de 2).

```
58
59
60
```



```
« ":" « ( minute < 10 ? "0" : "" ) « minute
« ":" « ( second < 10 ? "0" : "" ) « second
« ( hour < 12 ? " da manhã" : " da tarde" );
```

Fig. 6.8 Usando um construtor com argumentos default - time2 - cpp (parte 2 de 2).

```
61 // Fig. 6.8: figO6O8.cpp
62 // Demonstrando uma função
63 // construtor default para a classe Time
64 #include <iostream>
65
66 using std: :cout;
67 using std: :endl;
68
69 #include "time2.h"
70
```

Fig. 6.8 Usando um construtor com argumentos default - figO 6_08 - cpp (parte 1 de 2).

71	int	main O
72		
73		Time tl, // usados defaults para todos os argumentos
74		t2(2), // minute e second definidos por default
75		t3(21, 34), // second definido por default
76		t4(12, 25, 42), // todos os valores especificados
77		t5(27, 74, 99); // todos os valores especificados inválidos
78		
79		cout « "Construído com:\n"
80		« "todos os argumentos definidos por default:\n
81		tl.printNilitaryQ;
82		cout« "\n ";
83		tl.printStandardO;
84		
85		cout« "\nhora especificada; minuto e segundo definidos por default:"
86		« "\n ";
87		t2.printNilitaryO;
88		cout « "\n
89		t2.printStandard
90		
91		cout« "\nhora e minuto especificados; segundo definido por default:"
92		« "\n ";
93		t3.printNilitaryO;
94		cout« "\n ";

```
95          t3.printStandard
96
97          cout« "\nhora, minuto e segundo especificados:"
98          « "\n ";
99          t4.printMilitaryO;
100         cout« "\n
101         t4.printStandardO;
102
103         cout« "\ntodos os valores especificados inválidos:"
104         « "\n ";
105         t5.printNilitaryQ;
106         cout« "\n ";
107         t5.printStandardO;
108         cout « endl;
109
110         return 0;
111 }
```



```
Construído com:
todos os argumentos definidos por default:
00:00
12:00:00 da manhã
hora especificada; minuto e segundo definidos por default:
02:00
2:00:00 da manhã
hora e minuto especificados; segundo definido por default:
21:34
9:34:00 da tarde
hora, minuto e segundo especificados:
12:25
12:25:42 da tarde
todos os valores especificados inválidos:
00:00
12:00:00 da manhã
```

Fig. 6.8 Usando um construtor com argumentos default . figO 6_O 8. **cpp** (parte 2 de 2).

Nesse programa, o construtor chama a função membro setTime com os valores passados para o construtor (ou os valores default) para garantir que o valor fornecido para hour esteja no intervalo O a 23 e que os valores para minute e second estejam no intervalo O a 59. Se um valor estiver fora do intervalo, ele é inicializado com zero por setTime (isto é um exemplo de como garantir que um membro de dados permaneça em um estado consistente).

Note que o construtor de Time poderia ser escrito de forma a incluir os mesmos comandos que a função membro setTime. Isto pode ser ligeiramente mais eficiente porque a chamada extra a setTime é eliminada. Porém, codificar o construtor de Time e a função membro setTime de forma idêntica torna a manutenção deste programa mais difícil. Se a implementação da função membro setTime mudar, a implementação do construtor de Time deve mudar em conseqüência. Fazer o construtor de Time chamar setTime diretamente exige que quaisquer mudanças na implementação de setTime sejam feitas só uma vez. Isto reduz a probabilidade de um erro de programação quando a implementação for alterada. Além disso, o desempenho do construtor de Time pode ser melhorado declarando-se o construtor explicitamente inline ou definindo-se o construtor na definição de classe (o que, implicitamente, torna inline a definição da função).

Observação de engenharia de software 6.19

___Se uma função membro de uma classe já oferece toda ou parte da funcionalidade exigida por um construtor (ou outra função membro) da classe, chame aquela função membro de dentro do construtor (ou de outra função membro). isso simpflfica a manutenção do código e reduz a probabilidade de um erro se for modificada a implementação do código. Como regra geral: evite repetir código.

Boa** prática de programação **6.6

Declare valores default para os argumentos de uma função somente no protótipo da função dentro da definição da classe, no arquivo de cabeçalho.

Erro** comum de programação **6.9

Especificar inicializadores default para a mesma função membro, tanto em um arquivo de cabeçalho como na definição da função membro.

Nota: qualquer mudança nos argumentos default de um método requer que o código cliente seja recompilado. Se é provável que os valores dos argumentos default irão mudar, use funções sobrecarregadas em vez de argumentos default. Assim, se muda a implementação de uma função membro, o código cliente não precisa ser recompilado.

A Fig. 6.8 inicializa cinco objetos Time - um com todos os três argumentos inicializados por default na chamada do construtor, um com um argumento especificado, um com dois argumentos especificados, um com três

argumentos especificados e um com três argumentos inválidos especificados. O conteúdo de cada um dos membros de dados do objeto, depois da instanciação e inicialização, é exibido.

Se nenhum construtor está definido para uma classe, o compilador cria um construtor default. Tal construtor não executa qualquer inicialização; assim, quando o objeto é criado, não é garantido que esteja em um estado consistente.

Observação de engenharia de software 6.20

É possível que uma classe não tenha um construtor default, se forem construídos quaisquer construtores e nenhum deles for explicitamente um construtor default.

6.12 Usando destruidores

Um *destruidor é* uma função membro especial de uma classe. O nome do destruidor para uma classe é formado pelo caractere *til* **(-)** seguido pelo nome da classe. Essa convenção de nomenclatura é intuitivamente atraente porque, como veremos em um capítulo mais à frente, o operador til é o operador de complemento sobre bits e, de certo modo, o destruidor é o complemento do construtor.

O destruidor de uma classe é chamado quando um objeto é destruído - por exemplo, para objetos automáticos, quando a execução de programa deixar o escopo em que um objeto daquela classe foi instanciado. Na verdade, o destruidor propriamente dito não destrói o objeto - ele executa a *faxina de término* antes de o sistema recuperar de volta a memória liberada pelo objeto, de forma que essa memória possa ser reutilizada para manter novos objetos.

Um destruidor não recebe nenhum argumento e não retorna nenhum valor. Uma classe pode ter só um destruidor - não é permitido sobrecarregar um destruidor.

Erro comum de programação 6.10

É um erro de sintaxe tentar passar argumentos para um destruidon especificar um tipo de retorno para um destruidor (nem **void** *pode ser especificado), retornar valores de um destruidor ou sobrecarregar um destruidor*

Note que não foram fornecidos destruidores para as classes apresentadas até agora. Logo veremos vários exemplos de classes com destruidores úteis. No Capítulo 8, veremos que destruidores são apropriados para classes cujos objetos contêm memória alocada dinamicamente (para arrays e *strings,* por exemplo). No Capítulo 7, discutimos como alocar e desalocar memória dinamicamente.

Observação de engenharia de software 6.21

Como veremos (ao longo do restante do livro), construtores e destruidores têm uma importância muito maior em C++ e na programação orientada a objetos do que foi possível transmitir em nossa breve introdução aqui.

6.13 Quando construtores e destruidores são chamados

Construtores e destruidores são chamados automaticamente. A ordem em que essas chamadas de função são feitas depende da ordem em que a execução entra e sai do escopo em que os objetos são instanciados. Geralmente, chamadas para um destruidor são feitas na ordem contrária das chamadas para um construtor. Porém, como veremos na Fig. 6.9, a classe de armazenamento dos objetos pode alterar a ordem em que o destruidores são chamados.

Construtores são chamados para objetos definidos no escopo global antes de qualquer outra função (inclusive main) em que o arquivo começa a execução (embora a ordem de execução de construtores de objetos globais entre arquivos não seja garantida). Os destruidores correspondentes são chamados quando main termina ou a função **exit é chamada** (veja o Capítulo 18 para obter mais informações sobre essa função). Não são chamados destruidores para objetos globais se o programa é terminado com uma chamada à função abort (veja o Capítulo 18 para obter mais informações sobre esta função).

Construtores são chamados para objetos locais automáticos quando a execução alcança o ponto onde os objetos são definidos. Os destruidores correspondentes são chamados quando os objetos deixam o escopo (i.e., quando a execução deixa o bloco em que são definidos). Construtores e destruidores para objetos automáticos são chamados toda vez que os objetos entram e saem de seus escopos.

Construtores são chamados para objetos locais static somente uma vez, quando a execução atinge pela primeira vez o ponto onde os objetos são definidos. Os destruidores correspondentes são chamados quando main termina ou a função exit é chamada. Não são chamados destruidores para objetos static se o programa é terminado com uma chamada à função abort.

O programa da Fig. 6.9 demonstra a ordem em que construtores e destruidores são chamados para objetos de tipo CreateAndDestroy em vários escopos. O programa define first no escopo global. Seu construtor é chamado quando o programa começa a execução e seu destruidor é chamado ao término do programa após todos os outros objetos serem destruídos.

```
1 // Fig. 6.9: create.h
2 // Definição da classe CreateAndDestroy
3 // Funções membro definidas em create.cpp
4 #ifndef CREATEH
5 #define CREATE H
6
7 class CreateAndDestroy {
8 public:
9 CreateAndoestroy( int ); // construtor
10 CreateAndDestroy // destruidor
11 private:
12 int data;
13 };
14
15 #endif
```

Fig. 6.9 Demonstrando a ordem em que construtores e destruidores são chamados - **create .h.**

```
16 1/ Fig. 6.9: create.cpp
17 // Definições de funções membro para a classe CreateAndDestroy
18 #include <iostream>
19
20 using std: :cout;
21 using std::endl;
22
23 #include “create.h”
24
25 CreateAndDestroy: :CreateAndDestroy( int value
26 {
27 data = value;
28 cout « “Construtor do objeto - « data;
29 }
30
31 CreateAndDestroy: : -CreateAndDestroy O
32 { cout « “Destruidor do objeto - « data « endl;
```

Fig. 6.9 Demonstrando a ordem em que construtores e destruidores são chamados - **create .cpp.**

```
33 // Fig. 6.9: figO6O9.cpp
34 // Demonstrando a ordem em que construtores e
35 // destruidores são chamados.
```

Fig. 6.9 Demonstrando a ordem em que construtores e destruidores são chamados - figo6 09. **cpp** (parte 1 de 2).

```
#include <iostream>

using std: :cout; using std::endl;

#include “create .h”
```

```
CreateAndDestroy first( 1 ); // objeto global


int main O
cout « (global criado antes de main)” « endl;


CreateAndDestroy second( 2 ); // objeto local cout « -(local automático em main)” « endl;


static CreateAndDestroy third( 3 ); // objeto local cout « -(local estático em main)” «
endl;


create() // chama a função para criar objetos


CreateAndDestroy fourth( 4 ); // objeto local cout « -(local automático em main)” « endl; return 0;


/1 Função para criar objetos void create( void


CreateAndDestroy fifth( 5 );
cout « -(local automático em create) « endl;


static CreateAndoestroy sixth( 6 );
cout « (local estático em create)” « endl;


Construtor
Construtor
Construtor
Construtor
Construtor
Construtor
Destruidor
Destruidor
Construtor
Destruidor
Destruidor
Destruidor
Destruidor
Destruidor


do objeto 1
do objeto 2
do objeto 3
do objeto 5
do objeto 6
do objeto 7
do objeto 7
do objeto 5
do objeto 4
do objeto 4
do objeto 2
do objeto 6
do objeto 3
do objeto 1


(local
(local
(local
(local
(local
```

```
void create ( void );

// protótipo

36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75

CreateAridDestroy seventh( 7 );
cout « "(local automático em create)" « endl;

(global criado antes de main)

automático em main)
estático em main)
automático em create)
estático em create)
automático em create)

(local automático em main)
```

Fig. 6.9 Demonstrando a ordem em que construtores e destruidores são chamados . fig06 09. cpp (parte 2 de 2).

A função main declara três objetos. Os objetos **second e fourth** são objetos automáticos locais, e o objeto classe

third é um objeto local estático. Os construtores para cada um destes objetos são chamados quando a execução lando alcança o ponto onde cada objeto é declarado. Os destruidores para os objetos fourth e second são chamados, nessa ordem, quando o fim de main é atingido. Como o objeto third é estático, ele existe até o término da hour execução do programa. O destruidor para o objeto third é chamado antes do destruidor para first, mas após de dad todos os outros objetos serem destruídos. de dad
A função create declara três objetos - fifth e seventh são objetos automáticos locais e sixth é um Cada objeto local estático. Os destruidores para os objetos seventh e fifth são chamados, nessa ordem, quando o fim funçõe de create é alcançado. Como sixth é estático, ele existe até o término do programa. O destruidor para **sixth** valore *é* chamado antes dos destruidores para thirde first, mas após todos os outros objetos serem destruídos. 1 seguid membr
exibiçã

6.14 Usando membros de dados e funções membro comze

uma ch
Os membros de dados private de uma classe só podem ser manipulados por funções membro (e friends) da membr classe. Uma manipulação típica poderia ser o ajuste do saldo bancário de um cliente (por exemplo, um membro de funcion
dados private de uma classe BankAccount) por uma função membro computelnterest. nho. N
As classes freqüentemente fornecem funções membro public *set* e *get* para permitir que os clientes da sobre o classe *inicializem* (i.e., escrevam) ou *obtenham* (i.e., leiam) os valores de membros de dados private. Estas funções não necessitam ser chamadas especificamente de *sei'* e *get,* mas elas freqüentemente são chamadas assim. Mais especificamente, uma função membro que *inicializa* o membro de dados interestRate seria tipicamente chamado de setlnterestRate, e uma função membro que *obtém a* **interestRate seria tipicamente chama d** de getlnterestRate. Funções *get* são também comumente chamadas de funções de "consulta".
Pode parecer que oferecer tanto o recurso *inicializar* como o recurso *obter* seja essencialmente o mesmo que tornar public os membros de dados. Esta é outra sutileza de **C++** que torna a linguagem tão desejável para a engenharia de software. Se um membro de dados é public, então o membro de dados pode ser lido ou escrito *à* vontade por qualquer função no programa. Se um membro de dados é private, uma função public *get* certa- 1 mente pareceria permitir a outras funções no programa lerem os dados à vontade, mas a função *get* poderia controlar 2 a formatação e exibição dos dados. Uma função public *sei* poderia - e provavelmente o faria - monitorar cuida- **3 „** dosamente qualquer tentativa de modificar o valor do membro de dados. Isto asseguraria que o novo valor fosse **4** apropriado para aquele item de dados. Por exemplo, uma tentativa para *inicializar* odiado mês como 37 poderia ser 5 // rejeitada, uma tentativa para *inicializar* o peso de uma pessoa com um valor negativo também poderia ser rejeitada, 6 // uma tentativa para *inicializar* uma quantidade numérica com um valor alfabético e uma tentativa para *inicializar 7* **#i** uma nota em um exame com 185 (quando o intervalo apropriado é zero a 100) poderiam ser rejeitadas, etc. 8 #cL
9
Observação de engenharia de software 6.22 **10 c1**
11 pu]
_____ *Tornar os membros de dados* **private** *e controlar o acesso, especialmente acesso para escrever, àqueles* **12** *membros de dados através de funções membro* public *ajuda a assegurar a integridade dos dados.* **13**
14



Dica de teste e depuração 6.5 **15**

16
Os benefícios da integridade de dados não são obtidos automaticamente apenas porque os membros de **17** *dados são declarados como* **priva te** - *o programador deve providenciar a verificação de validade.* 18
C+ **+,** *porém, fornece uma estrutura na qual os programadores podem projetar programas melhores de* **19** *maneira conveniente.* **20**
Boa prática de programação 6.7
Funções membro que inicializam *os valores de dados privados devem verificar se os novos valores preten-* **24** *didos são apropriados; se não forem, afunção* set *deve colocar os membros de dados* **private** *em um estado consistente apropriado.* **27**
O cliente de uma classe deve ser notificado quando for feita uma tentativa de atribuir um valor inválido a um pr membro de dados. As funções de *inicialização* de uma classe são freqüentemente escritas para retornar valores indicando que foi feita uma tentativa de atribuir dados inválidos a um objeto da classe. Isso habilita os clientes da **Fig. 6.10**

classe a testar os valores de retorno de funções de *inicialização* para determinar se o objeto que eles estão

manipulando é um objeto válido e tomar a ação apropriada se o objeto não for válido.
A Fig. 6.10 estende nossa classe Time para incluir funções *get* e *set* para os membros de dados privados hour. minute e second. As funções de *inicialização* controlam estritamente a inicialização dos membros de dados. As tentativas de deixar qualquer membro de dados com um valor incorreto fazem com que o membro de dados seja inicializado com zero (deixando deste modo os membros de dados em um estado consistente). Cada função *get* simplesmente retorna o valor apropriado do membro de dados. O programa primeiro usa as funções de *inicialização* para *inicializar* os membros de dados private do objeto t do tipo Time com valores válidos e, então, usa as funções *get* para recuperar os valores e enviá-los para a saída de dados. Em
seguida, as funções de *inicialização* tentam *inicializar* os membros hour e second com valores inválidos e o membro minute com um valor válido, e então funções *get* recuperam os valores para a saída de dados. A exibição confirma que valores inválidos fornecidos fazem com que os membros de dados sejam inicializados com zero. Finalmente, o programa *inicializa* a hora como 11:58:00 e incrementa o valor de minute por 3 com uma chamada para a função incrementMinutes. A função incrementMinutes é uma função não- membro, que usa as funções membro *get* e *set* para incrementar corretamente o membro minute. Embora isso funcione, incorremos no *overhead* de múltiplas chamadas de função, o que impacta negativamente o desempenho. No próximo capítulo, discutiremos a noção de funções friend como um meio de eliminar este impacto sobre o desempenho.

Erro comum de programação ***6.11***

Um construtor pode chamar outras funções membro da classe, tais como funções set *ou* get, *mas, como o construtor está inicializando o objeto, os membros de dados podem ainda não estar em um estado consistente. Usar membros de dados antes de eles serem corretamente inicializados pode causar erros de lógica.*

```
1 // Fig. 6.10: time3.h
2 II Declaração da classe Time
3 // Funções membro definidas em time3.cpp
4
5 II diretivas para o pré-processador que previnem
6 II inclusões múltiplas de arquivos de cabeçalho
7 #ifndef TIME3H
8 #define TIME3H
9
class Time
public:
Time( int =0, int =0, int O ); //construtor


10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27


1/ funções de inicialização void setTime( int, int, int void setHour( int );
void setMinute ( int ); void setSecond( int );


) ; //
/1
/1
//
```

```
1/ funções get int getHourO; int getNinuteO; int getSecond

inicializa hora, minuto, segundo inicializa hora
inicializa minuto
inicializa segundo

// retorna
// retorna
// retorna

hora minuto
segundo

void printMilitaryO; /1 saída da hora no formato militar void printStandardQ; /1 saída da hora no formato padrão
28 private:
29 int hour; // O - 23
```

Fig. 6.10 Usando funções *sete get*- **time3.h** (parte 1 de 2).


```
30 int minute; // O - 59
31 int second; // O - 59
32 };
33
34 #endif
```

Fig. 6.10 Usando funções *sete get*- time3 .h (parte 2 de 2).

```
35 // Fig. 6.10: time3.cpp
36 // Definições de funções membro para a classe Time
37 #include <iostream>
38
39 using std: :cout;
40
41 #include "time3.h
42
43 /1 Função construtor para inicializar dados privados
44 // Chama a função membro setTime para inicializar variáveis
45 // Valores default são O (ver definição da classe)
46 Time: :Time( int hr, int mm, int sec
47 { setTime( hr, mm, sec ); }
48
49 // Inicializa os valores de hora, minuto e segundo
50 void Time::setTime( int h, int m, int s
51 {
52 setflour( h );
53 setMinute( m );
54 setSecond( s );
55
56
57 // Inicializa o valor de hora
58 void Time: :setHour( int h
59 (hour= (h>=0&&h<24) ?h :0; }
60
61 // Inicializa o valor de minuto
62 void Time::setMinute( int m
63 {minute=(m>=0&&m<60)?m:0;}
64
65 // Inicializa o valor de segundo
66 void Time::setsecond( int s
```

```
67 {second=(s>=0&&s<60)?s:0;)
68
69 // Obtém o valor da hora
70 int Time::getHour() { return hour; )
71
72 // Obtém o valor do minuto
73 int Time::getNinute() { return minute;
74
75 // Obtém o valor do segundo
76 int Time: :getSecond() { return second;
77
78 II Imprime Time no formato militar
79 void Time::printNilitary()
80
81 cout « ( hour < 10 ? 0" : ) « hour «
82 « ( minute < 10 ? "0" : ) « minute;
```

Fig. 6.10 Usando funções *sete get-* time3 .cpp (parte 1 de 2).


```
83 }
84
85 // Imprime Time no formato padrão
86 void Time: :printStandard()
87
88 cout« ( ( hour == O hour = 12 ) ? 12 : hour % 12
89 « ":" « ( minute < 10 ? "0" : "" ) « minute
90 « ":" « ( second < 10 ? "O" : " ) « second
91 « ( hour < 12 ? " da manhã : da tarde' );
92
```

Fig. 6.10 Usando funções *sete get-* time3 .cpp (parte 2 de 2).

```
93 // Fig. 6.10: figo6lO.cpp
94 // Demonstrando as funções get e set da classe Time
95 #include <iostream>
96
97 using std: :cout;
98 using std::endl;
99
100 4finclude "time3.h"
101
102 void incrementMinutes( Time &, const int );
103
104 int main O
105
106 Time t;
107
108 t.setHour( 17 );
109 t.setNinute( 34 );
110 t.setSecond( 25 );
111
112 cout « Resultado da inicialização com todos os valores válidos:\n"
113 « " Hora: " « t.getHour()
114 « " Minuto: " « t.getMinute()
115 « Segundo: " « t.getSecondO;
116
117 t.setHour( 234 ); // hora inválida inicializada com O
118 t.setNinute( 43 );
119 t.setSecond( 6373 ); // segundo inválido inicializado com O
120
121 cout « "\n\nResultado de tentar inicializar hora e segundo
122 « "inválidos:\n Hora: " « t.getHour()
123 « " Minuto: " « t.getMinute()
124 « " Segundo: " « t.getSecond() « "\n\n";
```

```
125
126 t.setTime( 11, 58, O );
127 incrementNinutes( t, 3 );
128
129 return 0;
130 }
131
132 void incrementNinutes( Time &tt, const int count
133 {
134 cout « "Incrementando minuto " « count
135 « vezes:\nHora de início:
136 tt.printStandardO;
137
```

Fig. 6.10 Usando funções *sete get-* figo6 lO.cpp (parte 1 de 2).


```
138 for ( int i = 0; i < count; i++
139 tt.setMinute( ( tt.getNinute() + 1 ) % 60 );
140
141 if ( tt.getMinute() == O
142 tt.setflour( ( tt.getHour() + 1 ) % 24 );
143
144 cout « "\nxninuto + 1: ";
145 tt.printStandard
146
147
148 cout « endl;
149
Resultado da inicialização com todos os valores válidos:
Hora: 17 Minuto: 34 Segundo: 25
Resultado de tentar inicializar hora e segundo inválidos:
Hora: O Minuto: 43 Segundo: O
Incrementando minuto 3 vezes:
Hora de início: 11:58:00 da manhã
minuto + 1: 11:59:00 da manhã
minuto + 1: 12:00:00 da tarde
minuto + 1: 12:01:00 da tarde
```

Fig. 6.10 Usando funções *sete get-* **figo6 lO.cpp** (parte 2 de 2).

Usar funções *set* certamente é importante do ponto de vista de engenharia de software porque elas podem executar verificações de validade. Tanto as funções *set* como *get* têm outra vantagem importante para a engenharia de software. Fi

Observação de engenharia de software 6.23

Acessar dados private *através de funções membro* set *e* get *não só protege os membros de dados de receber valores inválidos, como também isola os clientes da classe da representação dos membros de dados. Deste modo, se a representação dos dados muda,; por alguma razão (tipicamente para reduzir o volume de armazenamento exigido ou para melhorar o desempenho), somente as funções membro necessitam ser mudadas - os clientes não necessitam ser mudados desde que a inteiface fornecida pelas funções membro permaneça a mesma. Os clientes podem, porém, necessitar ser recompilados.*

6.15 Uma armadilha sutil: retornando uma referência a um membro de dados private

Uma referência para um objeto é um nome alternativo *(alias)* para o *nome* do objeto e pode ser, conseqüentemente, usado no lado esquerdo de um comando de atribuição. Neste contexto, a referência toma-se um *Ivalue* perfeitamente aceitável, que pode receber um valor. Uma maneira de usar esse recurso (infelizmente!) é ter uma função membro public de uma classe que retorna uma referência não-const para um membro de dados private daquela classe. A Fig. 6.11 usa uma classe Time simplificada para demonstrar o retorno de uma referência para um membro de dados private. Tal retorno, na realidade, torna uma chamada para a função badSetHour um *alias* para o membro de dados private hour! A chamada da função pode ser usada de qualquer forma que o membro de dados private possa ser usado, inclusive como um *lvalue* em um comando de atribuição!

Boa prática de programação 6.8
Nunca faça uma função membro public *retornar uma referência* não-const *(ou um ponteiro) para um membro de dados* private. *Retornar uma referência como essa viola o encapsulamento da classe. Na* **Fig**

verdade, retornar qualquer referência ou ponteiro para dados private já torna o código cliente dependente da representação dos dados da classe. Assim, deve ser evitado retornar ponteiros ou referências para dados private.

```
1 // Fig. 6.11: time4.h
2 // Declaração da classe Time
3 // Funções membro definidas em time4.cpp
4
5 // diretivas para o pré-processador que previnem
6 // inclusões múltiplas de arquivos de cabeçalho
7 #ifndef TIME4H
8 #define TIME4H
9
10 class Time
11 public:
12 Time( int O, int =0, int =O );
13 void setTime( int, int, int );
14 int getHour();
15 int &badSetHour( int ); // retorno de referência PERIGOSA
16 private:
17 int hour;
18 int minute;
19 int second;
20
21
22 #endif
```

Fig. 6.11 Retornando uma referência a um membro de dados private - time4 - h.

```
23 // Fig. 6.11: time4.cpp
24 // Definições das funções membro para a classe Time
25 #include "time4.h"
26
27 // Função construtor para inicializar dados privados
28 // Chama a função membro setTime para inicializar variáveis
29 // Valores default são O (veja definição da classe)
30 Time::Time( int hr, int mi int s
31 { setTime( hr, mi sec ); }
32
33 // Inicializa os valores de hora, minuto e segundo.
34 void Time::setTime( int h, int m, int s
35 {
36 hour (h>0&&h<24)?h:0;
37 minute =( m >O && m < 60 ) ? m : 0;
38 second =( s >=0 && s < 60 ) ? s : O;
39 }
40
41 // Obtém o valor da hora
42 int Time::getHour() { return hour; }
43
44 // PRÁTICA RUIM DE PROGRAMAÇÃO
45 // Retornar urna referência para um membro de dados privado
46 int &Time: :badSetHour( int hh
47 {
```

Fig. 6.11 Retornando uma referência a um membro de dados private - time4 - cpp (parte 1 de 2).


```
48 hour= (hh>=0 &&hh<24 ) ?hh :0;
49
```

```
50 return hour; // retorna referência PERIGOSA
51
```

Fig. 6.11 Retornando uma referência a um membro de dados private - time4 - cpp (parte 2 de 2).

```
52 // Fig. 6.11: figO6ll.cpp
53 /1 Demonstrando uma função membro pública que retorna
54 // uma referência para um membro de dados privado.
55 // A classe Time foi simplificada para este exemplo.
56 #include <iostream>
57
58 using std::cout;
59 using std: :endl;
60
61 #include "time4.h"
62
63 int main()
64
65 Time t;
66 int &hourRef =t.badSetHour( 20 );
67
68 cout « "Hora antes da modificação: -« hourRef;
69 hourRef =30; // modificação com valor inválido
70 cout « "\nHora depois da modificação: -« t.getHour
71
72 // Perigoso: uma chamada de função que retorna
73 // uma referência pode ser usada como um ivalue!
74 t.badSetHour(12) =74;
75 cout « "\n\n**********************************\n
76 « "PRÁTICA RUIM DE PROGRAMAÇÃO!!! ! ! ! ! \n"
77 « "badSetHour como um lvalue, Hora: -
78 « t.getHour()
79 « "\n********************************** « endi;
80
81 return 0;
82
Hora antes da modificação: 20
Hora depois da modificação: 30
PRÁTICA RUIM DE PROGRAMAÇÃO! ! !!!!!
badSetHour como um lvalue, Hora: 74
```

Fig. 6.11 Retornando uma referência a um membro de dados private - figO 6_li. cpp

O programa começa declarando o objeto t, do tipo Time, e a referência hourRef, à qual é atribuída a referência retornada pela chamada t . badSetHour (20) . O programa!÷xibe o valor do *alias* `hourRef`. Em seguida, o *alias* é usado para inicializar o valor de hour com 30 (um valor inválido) e o valor é novamente exibido. Finalmente, a própria chamada da função é usada como um *Ivalue,* ao qual é atribuído o valor 74 (outro valor inválido), e o valor é exibido.

6.16 Atribuição usando cópia membro a membro default

O operador de atribuição (=) pode ser usado para atribuir um objeto a um outro objeto do mesmo tipo. Tal atribuição é, por default, executada por *cópia membro a membro* - cada membro de um objeto é copiado (atribuído) individualmente para o mesmo membro em um outro objeto (ver Fig. 6.12). (Nota: a cópia membro a membro pode causar problemas sérios quando usada com uma classe cujos membros de dados contêm memória alocada dinamicamente; no Capítulo 8, "Sobrecarga de operadores", discutiremos esses problemas e mostraremos como tratá-los).

Os objetos podem ser passados como argumentos de função e podem ser retornados por funções. Tal passagem e retorno são executados, por default, através de chamada por valor - uma cópia do objeto é passada ou retornada (apresentamos vários exemplos no Capítulo 8).

Dica de desempenho 6.4

___Passar um objeto através de chamada por valor é bom do ponto de vista de segurança porque a função chamada

não tem nenhum acesso ao objeto original, mas as chamadas por valor podem degradar o desempenho quando um programa fizer uma cópia de um objeto grande. Um objeto pode ser passado através de uma chamada por referência, passando um ponteiro ou uma referência para o objeto. A chamada por referência oferece um bom desempenho, mas é inferior do ponto de vista da segurança porque a função chamada recebe acesso ao objeto original. A chamada por referência const *é uma alternativa segura e de bom desempenho.*

```
1 // Fig. 6.12: figO6l2.cpp
2 // Demonstrando que objetos de classes podem ser atribuidos
3 II uns aos outros usando a cópia membro a membro por default
4 #include <iostream>
5
6 using std: :cout;
7 using std: :endl;
8
9 II Classe Date simples
10 class Date
11 public:
12 Data( int =1, int =1, int =1990 ); // construtor default
13 void printO;
14 private:
15 int month;
16 int day;
17 int year;
18 };
19
20 I/ Construtor simples de Date, sem consistência de intervalos
21 Date::Date( int m, int d, int y
22 {
23 month =m;
24 day=d;
25 year =
26 }
27
28 // Imprime a Date no formato dd/mm/aaaa
29 void Date: :print()
30 { cout « day « '1 « month « '1' « year;
31
32 int main O
33 {
34 Date datel( 7, 4, 1993 ), date2; // date2 recebe default de 1/1/90
```

Fig. 6.12 Atribuindo um objeto a um outro por cópia membro a membro default (parte 1 de 2).


```
35
36 cout « 'datei =
37 datei.printO;
38 cout « "\ndate2 =
39 date2.printO;
40 NoCapft
41 date2 datei; II atribuição por cópia, membro a membro, por default membro
42 cout « "\n\nApós cópia default membro a membro, date2 =" tes daqu
43 date2 print () ; colocand
44 cout « endi;
45 dados);u
46 return 0; docoma
47 } Floord
```

para

```
datei =4/7/1993 eitensde
date2 =1/1/1990
Após cópia default membro a membro, date2 =4/7/1993 E
```

Fig. 6.12 Atribuindo um objeto a um outro por cópia membro a membro default (parte 2 de 2).

6.17 Reutilização de software

As pessoas que escrevem programas orientados a objetos se concentram em implementar classes úteis. Existe uma tremenda oportunidade de capturar e catalogar classes de forma que elas possam ser acessadas por grandes segmentos da comunidade de programação. Existem muitas *bibliotecas de classes* e outras estão sendo desenvolvidas em todo o mundo. Esforços estão sendo feitos no sentido de tomar essas bibliotecas amplamente acessíveis. Softwares estão sendo cada vez mais construídos a partir de componentes existentes, bem-definidos, cuidadosamente testados, bem-documentados, portáveis e amplamente disponíveis. Este tipo de reutilização de software acelera o desenvolvimento
de softwares poderosos e de alta qualidade. O *desenvolvimento rápido de aplicativos (RAD, rapid aplications development)* através dos mecanismos de reutilização de componentes se tornou um campo importante. Problemas significativos, porém, devem ser resolvidos antes que todo o potencial da reutilização de software possa ser percebido. Necessitamos de esquemas de catalogação, esquemas de licenciamento, mecanismos de proteção, para assegurar que as cópias mestres de classes não sejam corrompidas, esquemas de descrição, de maneira que os projetistas de novos sistemas possam determinar se objetos existentes satisfazem suas necessidades, mecanismos de navegação e busca, para determinar quais classes estão disponíveis e quão adequadas elas são para os requisitos do desenvolvedor de software, etc. Muitos problemas de pesquisa e desenvolvimento interessantes necessitam ser resolvidos. Existe uma grande motivação para resolver esses problemas porque o valor potencial de suas soluções é enorme.

6.18 (Estudo de caso opcional) Pensando em objetos: começando a programar as classes para o simulador de elevador Fig. 6.13

Nas seções "Pensando em objetos", nos Capítulos 1 a 5, introduzimos os fundamentos da orientação a objetos e

Implemei

desenvolvemos um projeto orientado a objetos para um simulador de elevador. No corpo do Capítulo 6, introduzi- Para que ur
mos os detalhes da programação com classes de C++. Agora, começamos a implementar nosso projeto orientado a para o obje
objetos em C++. Nesta seção, usaremos nossos diagramas de classes UML para esboçar os arquivos de cabeçalho de o objeto da
C++ que definem nossas classes. A Fig. 5.3

Implementação: visibilidade ____

No corpo do Capítulo 6, introduzimos os especificadores de acesso **`public e private. Antes de criarmos os`** *N. de R.T.:
arquivos de cabeçalho das classes, precisamos primeiro considerar quais elementos de nosso diagrama de classes inglês é lar
devem ser public e quais elementos devem ser private. Em situaçõ
ponteiros (o

Observação de engenharia de software 6.24

___ *Cada elemento de uma classe deve ter visibilidade* private *até que possa ser provado que o elemento necessita de visibilidade* public.

No Capítulo 6, discutimos porque membros de dados deveriam, em geral, ser private. mas e quanto às funções membro? As operações de uma classe são suas funções membro. Estas operações precisam ser invocadas por clientes daquela classe; portanto, as funções membro devem ser public. Na UML, visibilidade public é indicada colocando-se um sinal de mais (+) antes de um elemento particular (i.e., uma função membro ou um membro de dados); um sinal de menos (-) indica visibilidade private. A Fig. 6.13 mostra nosso diagrama de classes atualizado com as notações de visibilidade incluídas. (Observe que adicionamos a operação personArrives à classe Floor do nosso diagrama de seqüência na Fig. 4.27). Na medida em que escrevemos os arquivos de cabeçalho de C++ para as classes em nosso sistema, automaticamente colocamos os itens designados com + nas seções public e itens designados com - nas seções private das declarações de classes.

processTime(time : int) : void

Fig. 6.13 Diagrama completo de classes com notações de visibilidade.

<nenhuma ainda> runSimulation() : void

Implementação: handies

Para que um objeto da classe A possa se comunicar com um objeto da classe B, o objeto da classe A deve ter um *handle**

para o objeto da classe B. Isto significa que ou o objeto da classe A precisa conhecer o nome do objeto da classe B ou
o objeto da classe A precisa manter uma referência (Seção 3. 17) ou um ponteiro (Capítulo *5)* para o objeto da classe B.
A Fig. 5.36 continha uma lista de colaborações entre os objetos de nosso sistema. As classes na coluna à esquerda da

· N. de R.T.: uma tradução seria ***manípulo,*** ou seja, algo que se pode usar para acessar e manipular outro objeto; no entanto, o termo original em
inglês é largamente usado e, por isso, foi mantido no texto.

Em situações em que o nome do objeto da classe B não está disponível para o objeto da classe A, vamos preferir usar referências em vez de
ponteiros (onde apropriado), porque referências são inerentemente mais seguras do que ponteiros.

Elevator

crrentFloor: int = 1 direction enum = up capacity: int = 1 arrivalTime : int moving : bool = false

Clock

time : int = O getTime () int tick() void

Doar

open : bool = false openDoor() : void closeDoor() : void

processTime (time : int) : void personEnters() void
personExits() : void
summonElevator() void prepareToLeave() : void

Floor

occupied : bool = false elevatorArrived() : void isOccupied(): bool

Scheduler

BeII
<nenhuma ainda>

floorlArrivalTime int floor2ArrivalTime : int

ringBell() : void

FloorButton

pressed bool = false pressButton() : void resetButton() : void

Person

Light

ID: int
stepOntoFloor() : void exitElevator() : void enterElevator() : void

ElevatorButton

pressed : bool = false resetButton() : void pressButton() : void

on : bool = false
turnOff() : void
turnOn() void
Building

tabela precisam de um *handie* para cada uma das classes na coluna à direita da tabela, para enviar mensagens àquelas classes. A Fig. 6.14 lista os ***handies*** para cada classe, com base na informação mostrada na tabela da Fig. 5.36.
No corpo do Capítulo 6, discutimos como implementar ***handies*** em C++ como referências e ponteiros para classes (e, novamente, vamos preferir referências em vez de ponteiros, onde apropriado). Essas referências então se tornam atributos (dados) da classe. Enquanto não discutirmos composição, no Capítulo 7, não podemos representar cada item da Fig. 6.14cm nossos arquivos de cabeçalho de classes. Discutiremos tais casos especiais em breve.

Fig. 6.14 Lista de *handies* para cada classe.

Implementação: arquivos de cabeçalho de classes

Agora que discutimos a programação de classes em C++, estamos prontos para começar a escrever o código para nosso simulador de elevador. Nesta seção, examinamos os arquivos de cabeçalho para cada classe em nosso sistema. Na seção "Pensando em objetos" no fim do Capítulo 7, apresentamos o código C++ completo, que funciona, para o simulador. No Capítulo 9, modificamos aquele código para incorporar herança.
Para demonstrar a ordem em que construtores e destruidores são executados, codificaremos um construtor e um destruidor para cada uma de nossas classes que simplesmente exibem mensagens indicando que tais funções estão sendo executadas. Incluímos os protótipos do construtor e do destruidor em nossos arquivos de cabeçalho; incluímos suas implementações nos arquivos .cpp apresentados no Capítulo 7.
A Fig. 6.15 lista o arquivo de cabeçalho para a classe Bell. Trabalhando a partir de nosso diagrama de classes (Fig. 6.13), declaramos um construtor, um destruidor (linhas 8 e 9) e a função membro ringBell (linha 10); cada uma destas funções membro tem visibilidade public. Não identificamos nenhum outro elemento public ou private para esta classe, de modo que nosso arquivo de cabeçalho está completo.

```
1  // bell.h
2  // Definição para a classe Bell
3  #ifndef BELLH
4  #define BELL_H
5
6
7
8
9
10
11 );
12
13 #endif
```

```
1 // c
2 /1 D
3 #ifn
4 #def
5
```

```
6 das
7  publi
8 Clc
9 -Cl
10 voi
11 int
12 priv
13 int
```

A Fig. 6.16
e as funções
em nosso ai
Uma vez po
objeto da cl

```
14 };
15
16 #end:
```

Fig. 6.16 /
AFig.6.171
a 14) as opa
6.13). Tamb
da pessoa. L
Objet
dinâmica, à'
diferentemei
objetos dina
classe Pers

```
1 //
2 // DE
3 #ifnc
4 #defi
5
6 class
7  publi
8 Per,
9 -Pei
10 int
11
12 voic
13 voic
14 voic
15 priva
16 int
17 };
18
19 #endi
```

Fig. 6.17 Ar

```
class Beil {
public:
BellO;
-Bell()
void ringBellQ;
```

// **construtor** // **destruidor** // **toca a** campainha

// **BELLH**

Fig. 6.15 Arquivo de cabeçalho da classe Bell.

Classe	Handles		
Elevator	ElevatorButton, Beli, Floor,	Door	
Clock			
Scheduler	Person, Floor		
Person	FloorButton, ElevatorButton,	Elevator,	Floor
Floor	FloorButton, Light		
FloorButton	Elevator		
ElevatorButton	Elevator		
Door	Person		
Beil			
Light			
Building	Clock, Scheduler, Elevator		

A Fig. 6.16 lista o arquivo de cabeçalho para a classe Clock. Incluímos um construtor e um destruidor (linhas 8 e 9) e as funções membro public tick *()* e getTime *O* (linhas 10 e 11) da Fig. 6.13. Implementamos o atributo time em nosso arquivo de cabeçalho de classe declarando um membro de dados private time, do tipo int (linha 13). Uma vez por segundo em nossa simulação, um objeto da classe Building invoca a função membro getTime de um objeto da classe Clock para obter o valor atual de time e invoca a função membro tick para incrementar time.

```
// clock.h
// Definição para a classe Clock
#ifndef CLOCKH
#define CLOCK H
6 class Clock
7 public:
8 ClockO;
9 -Clock();
10 void tickO;
11 int getTime
12 private:
13 int time;
14 };
15
16 #endif // CLOCKH
```

Fig. 6.16 Arquivo de cabeçalho da classe Clock.

A Fig. 6.17 lista o arquivo de cabeçalho para a classe Person. Declaramos o atributo ID (linha 16) e (nas linhas 12 a 14) as operações stepOntoFloor, enterElevator e exitElevator de nosso diagrama de classes (Fig. 6.13). Também declaramos uma função membro public getlD (linha 10), que retorna o número de identificação da pessoa. Usaremos esta operação para monitorar as pessoas em nossa simulação.

Objetos da classe Person não são criados no início da simulação - eles são criados aleatoriamente, de forma dinâmica, à medida que a simulação é executada. Por essa razão, precisamos implementar objetos da classe Person diferentemente dos objetos de nossas outras classes em nosso sistema. Depois de discutirmos como criar novos objetos dinamicamente, no Capítulo 7, acrescentaremos elementos significativos ao arquivo de cabeçalho para a classe Person.

```
// person.h
// Definição da classe Person
#ifndef PERSONH
#define PERSONH
6 class Person
7 public:
8 Person( int );
9 PersonO;
10 int getIDO;
11
void stepOntoFloorQ;
void enterElevator
void exitElevator
```

```
private:
int ID; // número de identificação único da pessoa
19 #endif // PERSONH


1
2
3
4
5


// construtor
// destruidor
// incrementa o relógio em um segundo
// retorna a hora atual do relógio


1
2
3
4
5


// construtor
// destruidor
// retorna a identificação da pessoa


12
13
14
15
16
17
18
```

Fig. 6.17 Arquivo de cabeçalho da classe **Person.**

A Fig. 6.18 lista o arquivo de cabeçalho para a classe Door. Declaramos **um** construtor e um destruidor (linhas 8 e 9) e as funções membro public openDoor e closeDoor nas linhas 11 e 12. Declaramos o membro de dados private open da classe na linha 14. A tabela na Fig. 6.14 mostra que a classe Door precisa de um *handie* para a classe Person. Entretanto, como os objetos da classe Person são criados dinamicamente em nosso sistema, ainda estamos inseguros, neste ponto, sobre como implementar *handies* para objetos da classe Person. Depois de discutirmos

objetos criados dinamicamente, no Capítulo 7, teremos uma idéia melhor de como implementar *handies* para a classe Person.

II door.h

```
// Definição para a classe Door
#ifndef DOORH
#define DOORH
class Door {
public:
Door();
-Door();
```

Fig. 6.18 **Arquivo** de cabeçalho da classe Door.

Listamos o arquivo de cabeçalho para a classe Light na Fig. 6.19. A informação do diagrama de classes na Fig. 6.13 nos leva a declarar as funções membro **public turnOn e turnOff e o membro** de dados private, do tipo bool, on. Nesse arquivo de cabeçalho, também introduzimos algo novo em nossa implementação a necessidade de distinguir entre objetos diferentes da mesma classe em nosso sistema. Sabemos que a simulação contém dois objetos da classe Light: um objeto pertence ao primeiro andar e o outro objeto pertence ao segundo andar. Quere-

mos ser capazes de distinguir entre estes dois objetos para fins de saída de dados, de modo que precisamos dar um 10
nome a cada um deles. Portanto, acrescentamos a linha 14 **11**
char ***ne;** *II* **em e andar a luz está ligada**
à seção private da declaração da classe. Também adicionamos um parâmetro char * ao construtor (linha 8), de
modo que ele possa inicializar o nome de cada objeto da classe Light.

```
1
2
3
4
5
6
7
8
9
10


13
14
15
16
17
```

Fig

AF

um
mcii
a sii
dccl

// **construtor**
// **destruidor**
// aberta ou fechada

```
11 void openDoor
12 void closeDoorO;
13 private:
14 bool open;
15 };
16
17 #endif // DOORH
```

Tam vai invo
naFi
estes obje

```
15
16
17
18
19
20
21
```

Fig.

A Fig funçõ dor. A

incluí como

Fig. 6.19 Arquivo de cabeçalho da classe **`Light`** (parte 1 de 2).

1	// **light.h**			
2	// **Definição para a**		**classe Light**	
3	**#ifndef LIGHTH**			
4	**#define LIGHTH**			
5				
6	**class Light** {			
7	public:			
8	**Light(char** *); //		**construtor**	
9	**-Light();**	//	**destruidor**	
10	**void turnOn**	//	**liga a luz**	
11	**void turnOff** O;	//	**desliga a**	**luz**
12	**private:**			

13 bool on; // ligada ou desligada
14 char *name; // em que **andar a luz está ligada**
15 };
16
17 #endif // LIGHTH

Fig. 6.19 Arquivo de cabeçalho da classe Light (parte 2 de 2).

A Fig. 6.20 lista o arquivo de cabeçalho para a classe **`Building.`** A seção public da declaração da classe inclui um construtor, um destruidor e a função membro runSimulation da Fig. 6.13. Quando identificamos pela primeira vez a operação runSimulation, no Capítulo 4, não sabíamos que objeto iria invocar a função para iniciar a simulação. Agora que já discutimos classes em C++, sabemos que um objeto da classe Building precisa ser declarado em main e que main irá então invocar runSimulation. O código no programa principal é:

Buildirig building; // cria o objeto Buildirig
building.runSimulationQ; // invoca runSimulation

Também optamos por incluir um parâmetro do tipo int na declaração de runSimulation. O objeto building vai executar a simulação do elevador pelo número de segundos passado para o objeto através desse parâmetro. A invocação anterior de runSimulation incluiria, então, um número indicando a duração da simulação. A tabela na Fig. 6.14 indica que a classe Building precisa de *handies* para seus objetos compostos. Não podemos implementar estes *handies* neste ponto porque ainda não discutimos composição. Portanto, retardamos a implementação dos objetos componentes da classe Building até o Capítulo 7 (ver os comentários nas linhas 14 a 18 na Fig. 6.20).

1 // building.h
2 // **Definição** para a classe Building
3 #ifndef BUILDINGH
4 #define BUILDINGH
5
6 class Building
7 public:
8 Building // construtor
9 -BuildingO; // **destruidor**
10
11 // executa a simulação por um período de tempo especificado
12 void runSimulationO;
13 private:
14 // No Capítulo 7, mostramos como incluir:
15 // um objeto da classe Clock

```
16 // um objeto da classe Scheduler
17 // um objeto da classe Elevator
18 // dois objetos da classe Floor
19
20
21 #endif // BUILDINGH
```

Fig. 6.20 Arquivo de cabeçalho da classe Building.

A Fig. 6.21 lista o arquivo de cabeçalho para a classe ElevatorButton. Declaramos o atributo pressed, as funções membro pressButton e resetButton (do diagrama de classes da Fig. 6.13) e o construtor e o destruidor. A Fig. 6.14 afirma que a classe ElevatorButton precisa de um *handie* para o elevador. Na linha 19

Elevator &elevatorRef;

incluímos este *handie* (observe que usamos uma referência para implementar o *handie).* No Capítulo 7, discutimos como enviar mensagens para o elevador usando esta referência.

Uma referência precisa ser inicializada quando é declarada, mas não estamos autorizados a atribuir um valor para um membro de dados de uma classe no arquivo de cabeçalho. Portanto, uma referência deve ser inicializada no construtor; passamos uma referência Elevator ao construtor, como um parâmetro, na linha 10.

A linha 6

```
class Elevator;


// declaração antecipada (forward)
```

é uma *declaração antecipada (forward)* da classe Elevator. A declaração antecipada nos permite declarar uma referência para um objeto da classe Elevator no arquivo de cabeçalho da classe ElevatorButton.2

```
1 // elevatorButton.h
2 // Definição para a classe ElevatorButton
3 #ifndef ELEVATORBUTTQNH
4 #define ELEVATORBUTTONH
5


6 class Elevator;


7


class ElevatorButton
public:
ElevatorButton( Elevator & );
-ElevatorButton O;


void pressButton void resetButtonQ;
private:
bool pressed;


// referência ao elevador do botão Elevator &elevatorRef;
#endif // ELEVATORBUTTONH


// construtor
// destruidor


// pressiona o botão
```

II **desliga o botão**
II **estado do botão**

Fig. 6.21 Arquivo de cabeçalho da classe **ElevatorButton.**

A Fig. 6.22 lista o arquivo de cabeçalho para a classe FloorButton. Esse arquivo de cabeçalho é idêntico ao arquivo de cabeçalho para a classe ElevatorButton, exceto pelo fato de declararmos um membro de dados private floorNumber, do tipo int. Objetos da classe FloorButton não precisam saber a que andar eles pertencem, para fins de saída de dados do simulador. O número do andar é passado como um argumento para o construtor, para fins de inicialização (linha 10).

Fig. 6.22 Arquivo de cabeçalho da classe FloorButton (parte 1 de 2).

// **declaração antecipada (forward)**

8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

1:
1:

14

1

lE
1
2C

21
F

2 Usar a declaração antecipada (onde possível), em vez de incluir o cabeçalho completo, ajuda a evitar um problema do pré-processador chamado de inclusão circular. Discutimos o problema da inclusão circular em mais detalhes no Capítulo 7.

1	*II* **floorButton.h**						
2	*II* **Definição para a classe FloorButton**						
3	**#ifndef FLOORBUTTONH**						
4	**#define FLOORBUTTONH**						
5							
6	**class Elevator;**			*II*	**declaração**	**antecipada**	**(forward)**
7							
8	**class FloorButton**						

9	public:						
10	FloorButton(int, Elevator &	);	//	construtor			


```
11 -FloorButtonQ; // destruidor
12
13 void pressButtonO; II pressiona o botão
14 void resetButton(); II desliga o botão
15 private:
16 int floorNuinber; // número do andar do botão
17 bool pressed; II estado do botão
18
19 // referência ao elevador do botão
20 Elevator &elevatorRef;
21 );
22
23 #endif // FLOORBUTTONH
```

Fig. 6.22 Arquivo de cabeçalho da classe FloorButton (parte 2 de 2).

A Fig. 6.23 lista o arquivo de cabeçalho para a classe Scheduler. Nas linhas 23 e 24

```
int floorlArrivalTime;
int floor2ArrivalTime;
```

declaramos os **membros de dados private da classe Scheduler. que** correspondem aos **atributos que identificamos para esta classe (Fig.** 6.13). Na linha 12, declaramos a função membro public processTime, que corresponde à operação que identificamos na seção "Pensando em objetos", no fim do Capítulo 4. Nas linhas 15 a 19, declaramos as funções que identificamos no diagrama de seqüência da Fig. 4.27. Cada uma dessas funções recebe como parâmetro uma referência para um objeto da classe Floor. Note que não listamos essas funções como operações (i.e., funções membro public) porque tais métodos não são invocados por objetos clientes. Em vez disso, estes métodos são usados somente pela classe Scheduler para executar suas próprias ações internas. Portanto, colocamos estes métodos na seção private da declaração da classe.

Nas linhas 21 e 22, declaramos os *handies* identificados na Fig. 6.14. Novamente, implementamos cada *handie* como uma referência para um objeto da classe Floor. A classe Scheduler precisa destes *handies* para que ela possa enviar a mensagem isOccupied para os dois andares na simulação (ver diagrama na Fig. 4.27) Também precisamos fazer uma declaração antecipada da classe Floor na linha 6 para que possamos declarar as referências.

```
1 I/ scheduler.h
2 // Definição para a classe Scheduler
3 #ifndef SCHEDULERH
4 #define SCHEDULERH
5
6 class Floor; II declaração antecipada (forward)
7
8 class Scheduler {
9 public:
10 Scheduler( Floor &, Floor & ); // construtor
11 -Schedu1erO; // destruidor
12 void processTime( int ); II inicializa a hora do scheduler
13 private:
14
15 // método que escalona as chegadas para um andar específico
16 void scheduleTime( Floor & );
17
18 // método que retarda as chegadas para um andar específico
19 void delayTime( Floor & );
20
21 Floor &floorlRef;
```

Fig. 6.23 Arquivo de cabeçalho da classe Scheduler (parte 1 de 2).


```
22 Floor &floor2Ref;
23 int floorlArrivalTime;
24 int floor2ArrivalTime;
25 };
```

```
27 #endif // SCHEDULERH
```

Fig. 6.23 Arquivo de cabeçalho da classe Scheduler (parte 2 de 2).

A Fig. 6.24 lista o arquivo de cabeçalho para a classe Floor. Declaramos as funções membro public elevatorArrived, isOccupied e personArrives da Fig. 6.13. Também declaramos a função membro public elevatorLeaving na linha 26. Adicionamos essa função membro para que o elevador possa avisar ao andar quando estiver se preparando para sair. O elevador invoca a operação **elevatorLeaving e o andar** responde desligando a luz.

Na linha 31, adicionamos um membro de dados private à classe - adicionamos esse valor para fins de saída de dados, da mesma maneira que fizemos com o membro de dados floorNuitber da classe FloorButton. Também adicionamos um parâmetro do tipo int ao construtor, de modo que o construtor possa inicializar o membro de dados. Também declaramos o *handie* para a classe Elevator identificado na Fig. 6.14. Deixamos para o Capítulo 7 a declaração dos membros componentes da classe Floor (ver linhas 28 e 33).

```
1 // floor.h
2 // Definição para a classe Floor
3 #ifndef FLOORH
4 #define FLOOR H
5 .
6 class Elevator; // declaração antecipada (forward)
1
8 class Floor {
9 public:
10 Floor( int, Elevator & ); // construtor
11 -Floor(); /1 destruidor
12
13 // retorna true se o andar está ocupado
14 bool isOccupiedO;
15
16 // retorna o número do andar
17 int getwuniber();
18
19 /1 passa um handle para uma nova pessoa chegando no andar
20 void personArrivesO;
21
22 // avisa o andar que o elevador chegou
23 void elevatorArrivedQ;
24
25 // avisa o andar que o elevador está saindo
26 void elevatorLeavingO;
27
28 // declaração do componente FloorButton (ver Capítulo 7)
29
30 private:
31 int floorNumber; // o número do andar
32 Elevator &elevatorRef; // ponteiro para o elevador
33 // declaração do componente Light (ver Capítulo 7)
34);
35
36 #endif // FLOORH
```

Fig. 6.24 Arquivo de cabeçalho da classe Floor.

Listamos o arquivo de cabeçalho para a classe Elevator na Fig. 6.25. Na seção public do arquivo de cabeçalho, declaramos as operações summonElevator, prepareToLeave e processTime listadas na Fig. 6.13. Para diferenciar entre pessoas no andar e pessoas no elevador, renomeamos as duas últimas operações listadas sob a classe Elevator. Chamamos estas operações de passengerEnters e passengerExits e as declaramos na seção public do arquivo de cabeçalho. Também declaramos uma referência para cada um dos dois andares (linhas 38 e 39); o construtor (linha 10) inicializa essas referências.

Na seção private do arquivo de cabeçalho, declaramos os atributos moving, direction, currentFloor e arrivalTime da Fig. 6.13. Não precisamos declarar o atributo capacity; em vez disso, escreveremos nosso código de forma a assegurar que somente uma pessoa possa estar no elevador a cada momento.

```
1 // elevator.h
2 // Definição para a classe Elevator
3 #ifndef ELEVATORH
```

```
4 #define ELEVATORH
5
6 class Floor; // declaração antecipada (forward)
7
8 class Elevator
9 public:
10 Elevator( Floor &, Floor & ); // construtor
11 ElevatorO; // destruidor
12
13 // solicita que o elevador atenda a um determinado andar
14 void summonElevator( int );
15
16 // prepara o elevador para sair
17 void prepareToLeave
18
19 // fornece a hora para o elevador
20 void processTime ( int );
21
22 // avisa o elevador que o passageiro está entrando
23 void passengerEntersO;
24
25 // avisa o elevador que o passageiro está saindo
26 void passengerExitsO;
27
28 // declaração do componente ElevatorButton (ver Capítulo 7)
29 private:
30 bool moving; // estado do elevador
31 int direction; // direção atual
32 int currentFloor; // posição atual
33
34 // hora para chegar em um andar
35 int arrivalTime;
36
37 // referências aos andares atendidos pelo elevador
38 Floor &floorlRef;
39 Floor &floor2Ref;
40
41 // declaração do componente Door (ver Capítulo 7)
42 // declaração do componente Bell (ver Capítulo 7)
43 };
44
45 #endif // ELEVATORH
```

Fig. 6.25 Arquivo de cabeçalho da classe Elevator.

Conclusão E

Na próxima seção "Pensando em objetos", apresentamos o código completo para nossa simulação de elevador. U Usaremos os conceitos apresentados no próximo capítulo para implementar relacionamentos compostos, criação • E dinâmica de objetos da classe Person e membros de dados e funções static e const. Na seção "Pensando em E objetos", no fim do Capítulo 9, usamos herança para aprimorar ainda mais nosso projeto e implementação de simu lado de elevador orientada a objetos.

Resumo

• Estruturas são tipos de dados agregados construídos a partir de dados de outros tipos.
• O
• A palavra-chave struct introduz uma definição de estrutura. O corpo de uma estrutura é delimitado por chaves ({ e }). 1 Toda definição de estrutura deve terminar com um ponto-e-vírgula.

•O

Um nome de etiqueta de uma estrutura pode ser usado para declarar variáveis de um tipo de estrutura.

•O

• As definições de estrutura não reservam espaço na memória; elas criam novos tipos de dados que são usados para declarar **dl** variáveis.

• Os membros de uma estrutura ou uma classe são acessados usando-se os operadores de acesso a membros . o operador ponto • **O** (.)e o operador seta **(->).** O operador ponto acessa um membro de uma estrutura através do nome da variável do objeto ou rn de uma referência para o objeto. O operador seta acessa um membro de uma estrutura através de um ponteiro para o objeto. 01
• As desvantagens de se criar novos tipos de dados com structs são a possibilidade de ter dados não-inicializados; inicialização • imprópria; todos os programas usando uma struct devem ser mudados se a implementação da struct muda; e nenhuma ur proteção é fornecida para assegurar que os dados sejam mantidos em um estado consistente, com valores de dados apropria dos • C
• As classes habilitam o programador a modelar objetos com atributos e comportamentos. Os tipos de classe podem ser defini- • dos em C++ usando as palavras-chave class e struct, mas a palavra-chave class é normalmente usada para este propósito. • *Q*
• O nome da classe pode ser usado para declarar objetos daquela classe. CO
• As definições de classes começam com a palavra-chave das s. O corpo da definição da classe é delimitado por chaves ({ e • }**).** As definições de classe terminam com um ponto-e-vírgula. • C
• Qualquer membro de dados ou função membro declarada depois de public: em uma classe é visível para qualquer função • Se com acesso a um objeto da classe.
pe
• Qualquer membro de dados ou função membro declarada depois de private: só é visível para friends e outros membros daclasse. **(I**
• Especificadores de acesso a membros sempre terminam com dois-pontos (:)e podem aparecer múltiplas vezes e em qualquer ordem em uma definição de classe.
L)e
• Os dados privados não são acessíveis de fora da classe, ser
• A implementação de uma classe deveria ser ocultada de seus clientes. • O
exe
• Um construtor e uma funçao membro especial, com o mesmo nome que a classe, que é usada para inicializar os membros de um objeto da classe. Um construtor de uma classe é chamado quando um objeto daquela classe é instanciado.
• A função com o mesmo nome que a classe, mas precedido com um caractere til é chamada de destruidor. ***Tem***
• O conjunto de funções membro public de uma classe é chamado de interface ou interface pública da classe. arquiv
• Quando uma função membro é definida fora da definição da classe, o nome da função é precedido pelo nome da classe e o arqulv operador de resolução de escopo binário (: :). atnbut
d1as
• As funções membro definidas usando o operador de resolução de escopo fora de uma definição de classe estão dentro do classei escopo daquela classe, cliente
• As funções membro definidas em uma definição de classe são automaticamente colocadas mime. Ao compilador é reser- codigo vado o direito de não colocar qualquer função mune. compol
constru

• Chamar funções membro é mais conciso do que chamar funções em programação procedural porque a maioria dos dados usados pela função membro está diretamente acessível no objeto.
• Dentro do escopo de uma classe, podem ser feitas referências aos membros daquela classe simplesmente por seus nomes. Fora do escopo de uma classe, os membros da classe são referenciados ou através de um nome de objeto, uma referência para um objeto ou um ponteiro para um objeto.
• Operadores de seleção de membro e -> são usados para acessar membros de classes.
• Um princípio fundamental da boa engenharia de software é separar a interface da implementação.
• As definições de classes são normalmente colocadas em arquivos de cabeçalho e as definições de funções membro são normalmente colocadas em arquivos de código-fonte com o mesmo nome básico.
• O modo de acesso default para classes é private, de modo que todos os membros depois do cabeçalho da classe e antes do primeiro especificador de acesso a membro são considerados private.
• Os membros públicos de uma classe apresentam uma visão dos serviços que a classe oferece aos clientes da classe.
• O acesso a dados privados de uma classe pode ser cuidadosamente controlado através do uso de funções membro chamadas de funções de acesso. Se uma classe quiser permitir que os clientes leiam dados private, ela poder oferecer uma função *get.* Para habilitar os clientes a modificar dados private. a classe pode oferecer uma função *set.*
• Os membros de dados de uma classe são normalmente declarados private e as funções membro de uma classe são normalmente declaradas public. Algumas funções membro podem ser private e servem como funções utililitárias para as outras funções da classe.
• Os membros de dados de uma classe não podem ser inicializados em uma definição de classe. Eles devem ser inicializados em um construtor ou seus valores podem ser *inicializados* depois de seu objeto ser criado.
• Construtores podem ser sobrecarregados.
• Uma vez que um objeto de classe seja corretamente inicializado, todas as funções membro que manipulam o objeto devem assegurar que o objeto permaneça em um estado consistente.
• Quando um objeto de uma classe é declarado, podem ser fornecidos inicializadores. Estes inicializadores são passados para o construtor da classe.
• Construtores podem especificar argumentos default.
• Construtores não podem especificar tipos de retomo, nem podem tentar retomar valores.
• Se nenhum construtor for definido para uma classe, o compilador cria um construtor default. Um construtor default fomecido pelo

compilador não executa qualquer inicialização; assim, quando o objeto é criado, não é garantido que esteja em um estado consistente.
• O destruidor de um objeto automático é chamado quando o objeto sai do escopo. Na verdade, o destruidor propriamente dito não destrói o objeto, mas executa a faxina" de término antes de o sistema recuperar a memória usada pelo objeto.
• Destruidores não recebem argumentos e não retomam valores. Uma classe pode ter só um destruidor (destruidores não podem ser sobrecarregados).
• O operador de atribuição é usado para atribuir um objeto a outro objeto do mesmo tipo. Tal atribuição é normalmente executada por cópia membro a membro default. A cópia membro a membro não é ideal para todas as classes.

Terminologia

arquivo de cabeçalho construtor default
arquivo de código-fonte controle de acesso a um membro
atributo cópia membro a membro
class definição de classe
classe *proxy* desenvolvimento rápido de aplicativos (RAD)
cliente de uma classe destruidor
código reutilizável encapsulamento
comportamento escopo de arquivo
construtor escopo de classe

especificadores de acesso a membros estado consistente de um membro de dados

estrutura
estrutura auto-referente extensibilidade
função auxiliar
função de acesso
função de consulta
função *get*
função membro
função membro mime função não-membro função predicado
função *set*
função utilitária
implementação de uma classe

inicializador de um membro inicializar um objeto de classe instância de uma classe
instanciar um objeto de uma classe interface de uma classe

interface publie de uma classe membro de dados
mensagem
objeto

Terminologia de "Pensando em objetos"

declaração antecipada *(forward) handie*
problema da inclusão circular referências vs. ponteiros
símbolo -- para visibilidade private

Erros comuns de programação

objeto global
objeto local estático objeto local não-estático ocultação de informações operador de referência &
operador de resolução de escopo (: :)

operador de resolução de escopo binário (: :)
operador de seleção de membro seta **(.>)**
operador de seleção membro (.e **->)**
operador seleção de membro ponto **(.)**
operador seletor de membro de classe **(.)**

princípio do mínimo privilégio private
programação orientada a objetos (OOP) programação procedural projeto orientado a objetos (OOD) protected
pubiic
reutilização de software serviços de uma classe til **(-)** em nome de destruidor

tipo de dados
tipo de dados abstratos (ADT) tipo definido pelo programador tipo definido pelo usuário

símbolo -+- para visibilidade pubiic visibilidade
visibilidade private visibilidade pubiic

A expressão (*timePtr) hour se refere ao membro hour da struct apontada por timePtr. Omitir os parênteses, como em *timeptr.hour. seria um erro de sintaxe porque. tem uma precedência mais alta que *; assim, a expressão seria executada como se estivesse incluída entre parênteses, como em * (timeptr. hour) . Isso seria um erro de sintaxe porque com um ponteiro você deve usar o operador seta para se referir a um membro.
Esquecer o ponto-e-vírgula, no fim de uma definição de classe (ou estrutura), é um erro de sintaxe. Especificar um tipo de retorno e/ou um valor de retorno para um construtor é um erro de sintaxe. Tentar inicializar um membro de dados de uma classe explicitamente, na definição da classe, é um erro de sintaxe.
Quando estiver definindo funções membro de uma classe fora dessa classe, omitir o nome da classe e operador de resolução de escopo aplicado ao nome da função é um erro de sintaxe.
Uma tentativa por uma função que não é um membro de uma classe particular (ou um friend dessa classe) de acessar um membro privado dessa classe é um erro de sintaxe.
Os membros de dados de uma classe não podem ser inicializados na definição da classe.
Tentar declarar um tipo de retomo para um construtor e/ou tentar retornar um valor de um construtor são erros de sintaxe. Especificar inicializadores default para a mesma função membro, tanto em um arquivo de cabeçalho como na definição da função membro.
E um erro de sintaxe tentar passar argumentos para um destruidor, especificar um tipo de retorno para um destruidor (nem void pode ser especificado), retornar valores de um destruidor ou sobrecarregar um destruidor.
Um construtor pode chamar outras funções membro da classe, tais como funções *set* ou *get,* mas, como o construtor está inicializando o objeto, os membros de dados podem ainda não estar em um estado consistente. Usar membros de dados antes de eles serem corretamente inicializados pode causar erros de lógica.

6.1

6.2
6.3
6.4
6.5

6.6

6.7
6.8
6.9

6.10

6.11

Boas práticas de programação

6.1 Para maior clareza e legibilidade, use cada especificador de acesso a membro só uma vez em uma definição de classe. Coloque membros public em primeiro lugar, onde eles são mais fáceis de se localizar.
6.2 Use o nome do arquivo de cabeçalho com o ponto substituído por um sublinhado nas diretivas para o pré-processador #ifndef e #define de um arquivo de cabeçalho.
6.3 Se você optar por listar os membros private primeiro, em uma definição de classe, use explicitamente o especificador de acesso a membro private. apesar do fato de que private é assumido por default. Isso melhora a clareza do
programa. Nossa preferência é listar primeiro os membros public de uma classe para enfatizar a interface da classe.

6.4 Apesar do fato de que os especificadores de acesso a membro public e private podem ser repetidos e misturados, liste todos os membros public de uma classe em um grupo primeiro e então liste todos os membros private em outro grupo. Isso focaliza a atenção do cliente sobre a interface pública da classe, em vez de sobre a implementação da classe.

6.5 Quando for apropriado (quase sempre), forneça um construtor para assegurar que todo objeto seja corretamente inicializado com valores que tenham significado. Em particular, membros de dados do tipo ponteiro deveriam ser inicializados com
algum valor de ponteiro válido ou com O.
6.6 Declare valores default para os argumentos de uma função somente no protótipo da função dentro da definição da classe, no arquivo de cabeçalho.
6.7 Funções membro que *inicializam* os valores de dados privados devem verificar se os novos valores pretendidos são apropriados; se não forem, a função *set* deve colocar os membros de dados private em um estado consistente apropriado.
6.8 Nunca faça uma função membro publie retornar uma referência não-const (ou um ponteiro) para um membro de dados private. Retornar uma referência como essa viola o encapsulamento da classe. Na verdade, retornar qualquer referência ou ponteiro para dados private já torna o código cliente dependente da representação dos dados da classe. Assim, deve ser evitado retornar ponteiros ou referências para dados private.

Dicas de desempenho

6.1 As estruturas são geralmente passadas através de chamadas por valor. Para evitar o *overhead* de copiar uma estrutura, passe a estrutura através de uma chamada por referência.
6.2 Definir uma pequena função membro dentro da definição de uma classe automaticamente a coloca iriline (se o compilador decidir assim). Isto pode melhorar o desempenho, mas não promove a melhor engenharia de software porque os clientes da classe serão capazes de ver a implementação da função, e seu código deve ser recompilado se a definição da função mime muda.
6.3 Na realidade, objetos contêm só dados, de modo que objetos são muito menores do que se também contivessem funções. A aplicação do operador sizeof a um nome de classe ou a um objeto daquela classe retornará somente o tamanho dos dados da classe. O compilador cria (somente) uma cópia das funções membro, separada de todos os objetos da classe. Todos os objetos da classe compartilham essa cópia das funções membro. Cada objeto, naturalmente, necessita de sua própria cópia dos dados da classe porque tais dados podem variar entre os objetos. O código da função não é modificável (também chamado de *código reentrante* ou *procedimento puro)* e pode, conseqüentemente, ser compartilhado por todos os objetos de uma classe.
6.4 Passar um objeto através de chamada por valor é bom do ponto de vista de segurança porque a função chamada não tem nenhum acesso ao objeto original, mas as chamadas por valor podem degradar o desempenho quando um programa fizer uma cópia de um objeto grande. Um objeto pode ser passado através de uma chamada por referência, passando um ponteiro ou uma referência para o objeto. A chamada por referência oferece um bom desempenho, mas é inferior do ponto de vista da segurança porque a função chamada recebe acesso ao objeto original. A chamada por referência const é uma alternativa segura e de bom desempenho.

Observações de engenharia de software

6.1 Para evitar o *overhead* de uma chamada por valor e ainda obter o benefício de que os dados originais da função que chamou estejam protegidos contra modificações, passe parâmetros de tamanho grande como referências const.
6.2 E importante escrever programas que sejam compreensíveis e fáceis de se manter. A mudança é a regra e não a exceção. Os programadores deveriam prever que seu código será modificado. Como veremos, o uso de classes pode facilitar a
possibilidade de mudanças em programas.

6.3 Os clientes de uma classe usam-na sem conhecer os detalhes internos de como a classe é implementada. Se a implementação de classe for mudada (por exemplo, para melhorar o desempenho), desde que a interface da classe permaneça constante, o código-fonte do cliente da classe não necessita mudar (embora o cliente possa necessitar ser recompilado). Isso toma muito mais fácil modificar sistemas.
6.4 As funções membro são normalmente menores que as funções em programas não-orientados a objetos porque os dados armazenados nos membros de dados, idealmente, já foram validados por um construtor e/ou por funções membro que armazenam novos dados. Como os "dados já estão no objeto", a função membro freqüentemente chamada não tem nenhum argumento ou, pelo menos, menos argumentos que as chamadas típicas de função em linguagens não-orientadas a objetos. Deste modo, as chamadas são pequenas, as definições de função idem e os protótipos de função também são pequenos.
6.5 Clientes têm acesso à interface de uma classe, mas não devem ter acesso à implementação de uma classe.
6.6 Declarar funções membro dentro da definição de uma classe (através de seus protótipos) e definir essas funções fora da definição da classe separa a interface da implementação da classe. Isso promove a boa engenharia de software. Os clientes de uma classe não podem ver a implementação das funções membro da classse e não precisam ser recompilados se a implementação muda.
6.7 Somente as funções membro mais simples e as funções membro mais estáveis (i.e., sua implementação provavelmente não muda) devem ser definidas no cabeçalho da classe.
6.8 Usar uma abordagem de programação orientada a objetos freqüentemente pode simplificar a chamada de funções, reduzindo o

número de argumentos a serem passados. Esse benefício da programação orientada a objetos deriva-se do fato de que o encapsulamento de membros de dados e funções membro dentro de um objeto dá às funções membro o direito de acessar os membros de dados.

6.9 Um tema central deste livro é "reutilize, reutilize, reutilize". Discutiremos cuidadosamente várias técnicas para "polir" as classes, visaido a facilitar a sua reutilização. Concentrar-nos-emos em "lapidar classes valiosas" e criar "ativos de software" valiosos.

6.10 Coloque a declaração da classe em um arquivo de cabeçalho para ser incluído por qualquer cliente que queira usar a classe. Isso constitui a interface pública da classe (e fornece ao cliente os protótipos de função de que ele necessita para poder chamar as funções membro da classe). Coloque as definições das funções membro da classe em um arquivo-fonte. Isso constitui a implementação da classe.

6.11 Os clientes de uma classe não necessitam de acesso ao código-fonte da classe para utilizá-la. Porém, os clientes necessitam poder ligar seu código ao código objeto da classe. Isto encoraja vendedores de software independente (ISVs, *independent software vendors)* a oferecer bibliotecas de classes para venda ou licença de uso. Os ISVs oferecem em seus produtos somente os arquivos de cabeçalho e os módulos objeto. Nenhuma informação proprietária é revelada - como seria o caso se fosse fornecido código-fonte. A comunidade de usuários de C++ se beneficia tendo mais bibliotecas de classe disponíveis, produzidas por ISVs.

6.12 As informações importantes sobre a interface de uma classe deveriam ser incluídas no arquivo de cabeçalho. As informações que serão usadas só internamente na classe e não serão necessitadas por clientes da classe deveriam ser incluídas no arquivo-fonte não-divulgado. Isso é outro exemplo do princípio do mínimo privilégio.

6.13 C++ encoraja a construção de programas que sejam independentes da implementação. Quando muda a implementação de uma classe usada por código independente de implementação, aquele código não necessita ser modificado. Se qualquer parte da interface da classe muda, o código independente de implementação precisa ser recompilado.

6.14 Mantenha todos os membros de dados de uma classe private. Forneça funções membro public para inicializar os valores dos membros de dados private e obter os valores dos membros de dados private. Esta arquitetura ajuda a esconder a implementação de uma classe de seus clientes, o que reduz bugs e aumenta a facilidade do programa ser modificado.

6.15 Os projetistas de classes usam membros private, protected e public para garantir a noção de ocultação de informações e seguir o princípio do mínimo privilégio.

6.16 O projetista de classes não necessita fornecer funções *set* e/ou *get* para cada item de dados private: esses recursos devem ser fornecidos somente quando for apropriado. Se o serviço é útil para o código cliente, esse serviço deve ser incluído na interface public da classe.

6.17 Funções membro tendem a se enquadrar em várias categorias diferentes: funções que lêem e retornam o valor de mcmbros de dados private: funções que inicializam o valor de membros de dados private: funções que implementam os serviços da classe; e funções que executam várias tarefas mecânicas para a classe, tais como inicializar objetos da classe, atribuir a objetos da classe, fazer a conversão entre classes e tipos primitivos ou entre classes e outras classes e administrar a memória para objetos da classe.

6.18 Um fenômeno observado na programação orientada a objetos é que, uma vez que esteja definida uma classe, criar e manipular objetos daquela classe geralmente envolve apenas efetuar uma seqüência simples de chamadas a funções membro - poucas, senão nenhuma, estruturas de controle são necessárias. Em contraste com isso, é comum ter-se estruturas de controle na implementação das funções membro de uma classe.

6.19 Se uma função membro de uma classe já oferece toda ou parte da funcionalidade exigida por um construtor (ou outra função membro) da classe, chame aquela função membro de dentro do construtor (ou de outra função membro). Isso simplifica a manutenção do código e reduz a probabilidade de um erro se for modificada a implementação do código. Como regra geral: evite repetir código.

6.20 E possível que uma classe não tenha um construtor default, se forem construídos quaisquer construtores e nenhum deles for explicitamente um construtor default.

6.21 Como veremos (ao longo do restante do livro), construtores e destruidores têm uma importância muito maior em C++ e na programação orientada a objetos do que foi possível transmitir em nossa breve introdução aqui.

6.22 Tornar os membros de dados private e controlar o acesso, especialmente acesso para escrever, àqueles membros de dados através de funções membro publie ajuda a assegurar a integridade dos dados.

6.23 Acessar dados private através de funções membro *set* e *get* não só protege os membros de dados de receber valores inválidos, como também isola os clientes da classe da representação dos membros de dados. Deste modo, se a representação dos dados mudar, por alguma razão (tipicamente para reduzir o volume de armazenamento exigido ou para melhorar o desempenho), somente as funções membro necessitam ser mudadas - os clientes não necessitam ser mudados desde que a interface fornecida pelas funções membro permaneça a mesma. Os clientes podem, porém, necessitar ser recompilados.

6.24 Cada elemento de uma classe deve ter visibilidade private até que possa ser provado que o elemento necessita de visibilidade public.

Dicas de teste e depura ção

6.1 O fato de que chamadas de funções membro geralmente não passam nenhum, ou substancialmente menos, argumentos do que chamadas de função convencionais em linguagens não-orientadas a objetos, reduz a probabilidade de passar os argumentos errados, os tipos errados de argumentos e/ou a quantidade errada de argumentos.

6.2 Use as diretivas para o pré-processador #ifndef. #define e #endif para evitar que arquivos de cabeçalho sejam incluídos mais de uma vez em um programa.

6.3 Tornar os membros de dados de uma classe private e as funções membro da classe public facilita a depuração porque os problemas com manipulações de dados ficam restritos ou a funções membro da classe ou a friends da classe.

6.4 Toda função membro (e friend) que modifica membros de dados private de um objeto deve assegurar que os dados permaneçam em um estado consistente.

6.5 Os benefícios da integridade de dados não são obtidos automaticamente apenas porque os membros de dados são declarados como private - o programador deve providenciar a verificação de validade. C++, porém, fornece uma estrutura

na qual os programadores podem projetar programas melhores de maneira conveniente.

Exercícios de auto-revisão

6.1 Preencha os espaços em branco em cada um dos seguintes itens:
a) A palavra-chave _____ introduz uma definição de estrutura.
b) Membros de classe são acessados através do operador junto com o nome de um objeto da classe ou através do operador junto com um ponteiro para um objeto da classe.
c) Membros de uma classe especificados como _____ são acessíveis somente a funções membros e friends da classe.
d) Um _____ é uma função membro especial usada para inicializar os membros de dados de uma classe.
e) O acesso default para membros de uma classe é _____
f) Uma função _____ é usada para atribuir valores a membros de dados private de uma classe.
g) pode ser usada para atribuir um objeto de uma classe a outro objeto da mesma classe.
h) Funções membro de uma classe são normalmente declaradas _____ e membros de dados de uma classe são normalmente declarados _____
i) Uma função _____ é usada para recuperar valores de dados de private de uma classe.
j) O conjunto de funções membro public de uma classe é chamado de _____ da classe.
k) Uma implementação de classe é dita ocultada de seus clientes ou _____
1) As palavras-chave _____ e _____ podem ser usadas para introduzir uma definição de classe.
m) Membros de uma classe especificados como _____ são acessíveis em qualquer lugar em que um objeto da classe está no escopo.

6.2 Encontre o(s) erro(s) em cada um dos seguintes itens e explique como corrigi-lo(s).
a) Assuma que o seguinte protótipo seja declarado na classe Time.

```
void ~Time (int );
```

b) O seguinte código é uma definição parcial da classe Time.

```
class Time
public:
// protótipos de funções
private:
int hour = 0;
int minute = 0;
int second = 0;
```

c) Suponha que o seguinte protótipo seja declarado na classe Employee.

```
int Employee( const char *, const char *
```

Respostas aos exercícios de auto-revisão

6.1 a) struct. b) ponto **(.),** seta **(->).** c) private. d) construtor, e) private. f) *set.* g) cópia membro a membro default (executada pelo operador de atribuição). h) public, private. i) *gel.* j) interface. k) encapsulada. 1) class, struct. m) public.

6.2 a) Erro: destruidores não podem retomar valores ou receber argumentos.
Correção: remova o tipo de retorno void e o parâmetro int da declaração.
b) Erro: membros não podem ser explicitamente inicializados na definição de classe.
Correção: remova a inicialização explícita da definição da classe e inicialize os membros de dados em um construtor.
c) Erro: construtores não podem retornar valores.
Correção: remova o tipo de retorno int da declaração.

Exercícios

6.3 Qual é o propósito do operador de resolução de escopo?

6.4 Compare e contraste as noções de struct e de class em C++.

6.5 Forneça um construtor que é capaz de usar a hora atual fornecida pela função time ( ) - declarada no cabeçalho time .h da biblioteca padrão C - para inicializar um objeto da classe Time.

6.6 Crie uma classe chamada Complexos para fazer aritmética com números complexos. Escreva um programa para testar sua classe.
Os números complexos têm a forma
parteReal + partelmaginaria *j
onde i é
Use variáveis double para representar os dados private da classe. Forneça uma função construtor que possibilita que um objeto desta classe seja inicializado quando é declarado. O construtor deve conter valores default no caso de nenhum inicializador ser fornecido. Forneça funções membro public para cada um dos seguintes itens:
a) Adição de dois números Complexos: as partes reais são somadas juntas e as partes imaginárias são somadas juntas.
b) Subtração de dois números Complexos: a parte real do operando direito é subtraída da parte real do operando esquerdo e a parte imaginária do operando direito é subtraída da parte imaginária do operando esquerdo.
c) Imprimir números do tipo Complexos na forma (a, b) onde a é a parte real e b é a parte imaginária.

6.7 Crie uma classe chamada Racional para fazer aritmética com frações. Escreva um programa para testar sua classe. Use variáveis inteiras para representar os dados private da classe - o numerador e o denominador. Forneça uma função construtor que permita que um objeto desta classe seja inicializado quando é declarado, O construtor deve conter valores default no caso de nenhum inicializador ser fornecido e deve armazenar a fração em formato reduzido (i.e., a fração
2
4
seria armazenada no objeto como 1 no numerador e 2 no denominador). Forneça funções membro public para cada um dos seguintes itens:
a) Adição de dois números do tipo Racional. O resultado deve ser armazenado em forma reduzida.
b) Subtração de dois números do tipo Racional. O resultado deve ser armazenado em forma reduzida.
c) Multiplicação de dois números do tipo Racional. O resultado deve ser armazenado em forma reduzida.
d) Divisão de dois números do tipo Racional. O resultado deve ser armazenado em forma reduzida.
e) Imprimir números do tipo Racional no formato a/b, onde a é o numerador e b é o denominador.
f) Imprimir números do tipo Racional em formato de ponto flutuante.

6.8 Modifique a classe Time da Fig. 6. 10 para incluir uma função membro tick que incrementa a hora armazenada em um objeto do tipo Time em um segundo. O objeto do tipo Time deve sempre permanecer em um estado consistente. Escreva um programa que testa a função membro tick em um laço que imprime a hora no formato padrão durante cada repetição do laço, para ilustrar que a função membro tick funciona corretamente. Não deixe de testar os seguintes casos:
a) Incrementar para o próximo minuto.
b) Incrementar para a próxima hora.
c) Incrementar para o dia seguinte (i.e., 11:59:59 da noite para 12:00:00 da manhã).

6.9 Modifique a classe Date da Fig. 6.12 para executar a verificação de erros sobre os valores de inicializadores fornecidos para os membros de dados month, day e year. Forneça também uma função membro nextDay para incrementar o dia por um. O objeto Date deve sempre permanecer em um estado consistente. Escreva um programa que testa a função nextDay em um laço que imprime a data durante cada repetição do laço para ilustrar que a função nextDay funciona corretamente. Não deixe de testar os seguintes casos:
a) Incrementar para o próximo mês.
b) Incrementar para o próximo ano.

6.10 Combine a classe Time modificada do Exercício 6.8 e a classe Date modificada do Exercício 6.9cm uma classe chamada DateAndTime (no Capítulo 9 discutiremos herança, que nos habilitará a realizar esta tarefa rapidamente sem modificar as definições de classes existentes). Modifique a função tick para chamar a função nextDay se a hora for incrementada para o dia seguinte. Modifique as funções printStandard e printNilitary para enviar para a saída a data, além da hora. Escreva um programa para testar a nova classe DateAndTime. Faça um teste específico, incrementando a hora para o dia seguinte.

6.11 Modifique as funções *set* no programa da Fig. 6. 10 para retomar valores de erro apropriados se for feita uma tentativa de *inicializar* um membro de dados de um objeto da classe Time com um valor inválido.

6.12 Crie uma classe Retangulo. A classe tem atributos comprimento e largura, cada um com valor default igual a 1. Ela tem funções membro que calculam o perimetro e a area do retângulo. Forneça funções ***Sete*** *get,* tanto para o compri - mento como para a largura. As funções *set* devem verificar se o comprimento e a largura são números de ponto flutuante maiores que 0.0 e menores que 20.0.

6.13 Crie uma classe Retangulo mais sofisticada que a que você criou no Exercício 6.12. Esta classe armazena só as coordenadas cartesianas dos quatro cantos do retângulo. O construtor chama uma função ***Set*** que recebe quatro conjuntos de coordenadas e verifica se cada um deles está no primeiro quadrante sem que nenhum valor de coordenada x ou y seja maior que 20.0. A função ***Set*** também verifica se as coordenadas fornecidas de fato especificam um retângulo. As funções membro calculam o comprimento, largura, perimetro e area. O comprimento é a maior das duas dimensões. Inclua uma função predicado quadrado, que determina se o retângulo é um quadrado.

6.14 Modifique a classe Retangulo do Exercício 6.13 para incluir uma função desenhar que exibe o retângulo dentro de uma "caixa" de 25 por 25, abrangendo a parte do primeiro quadrante em que o retângulo se situa. Inclua uma função inicializarCaractereDePreenchimento para especificar o caractere com o qual o corpo do retângulo será desenhado. Inclua uma função inicializarCaractereDoPerimetro para especificar o caractere que será usado para desenhar

a borda do retângulo. Se for ambicioso, você pode incluir funções para mudar a escala do retângulo, girá-lo e movê-lo dentro da parte designada do primeiro quadrante.

6.15 Crie uma classe InteiroGigante que usa um array de dígitos de 40 elementos para armazenar inteiros tão grandes quanto 40 dígitos. Forneça as funções membro leialnteiroGigante. imprimalnteiroGigante. adicionelnteirosGigantes e subtraialnteirosGigantes. Para comparar objetos do tipo InteiroGigante. providencie funções elgualA. naoElgualA, eMaiorQue. eMenorQue. eMaiorQueOulgualAc eMenorQueOulgualA
- cada uma dessas funções é uma "função predicado', que simplesmente retorna true se a relação entre os dois inteiros gigantes for verdadeira e retoma false se a relação não for verdadeira. Forneça uma função predicado eZero. Se você for ambicioso, forneça também funções membro multipliquelnteirosGigantes. dividalnteirosGigantes e `modulolnteirosGigantes.`

6.16 `Crie uma classe JogoDaVelha que lhe permitirá escrever um programa completo para jogar o jogo da velha. A classe` contém como dados private um array de inteiros bidimensional, 3 `por` 3. **O** construtor `deve inicializar` o tabuleiro vazio todo com zeros. Permita que duas pessoas possam jogar. Onde quer que o primeiro jogador faça sua jogada, coloque um dígito 1 no quadrado especificado; coloque um dígito 2 onde quer que o segundo jogador jogue. Cada jogada deve ser em um quadrado vazio. Após cada jogada, determine se o jogo foi ganho por um dos dois jogadores ou se o jogo terminou sem vencedor. Se você for ambicioso, modifique seu programa de forma que o computador faça as jogadas para um dos jogadores automaticamente. Além disso, permita ao jogador especificar se ele quer começar primeiro ou depois. Se você for excepcionalmente ambicioso, desenvolva um programa que jogará o jogo da velha em três dimensões, com um cubo de 4 por 4 por 4 células (Cuidado:

este é um projeto extremamente desafiante, que pode consumir muitas semanas de esforço!).

7

Classes: parte II

Objetivos

• Ser capaz de criar e destruir objetos dinamicamente.
• Ser capaz de especificar objetos const (constantes) e fun- ções membro const.
• Entender a finalidade das funções e classes friend.
• Entender como usar membros de dados e funções membro static.
• Entender o conceito de uma classe contêiner.
• Entender a noção de classes de iteradores que percorrem os elementos de classes contêineres.
• Entender o uso do ponteiro this.

But what, to serve ourpri vate ends,
Forbids the cheating ofourfriends?
Charles Churchili

Em vez dessa divisão absurda de sexos, deveriam
classificar as pessoas como estáticas e dinâmicas.
Evelyn Waugh

E sobretudo isso: sê fiel a ti mesmo.
William Shakespeare, Hamlet

Não tenha amigos iguais a você.
Confúcio

Visão geral

7.1 **Introdução**
7.2 Objetos const **(constantes) e funções membro const**
7.3 Composição: objetos como membros de classes
7.4 Funções friend e classes friend
7.5 Usando o ponteiro this
7.6 Alocação dinâmica de memória com os operadores new e delete
7.7 Membros de classe static
7.8 Abstração de dados e ocultação de informações
7.8.1 Exemplo: tipo de dado abstrato array
7.8.2 Exemplo: tipo de dado abstrato *string*
7.8.3 Exemplo: tipo de dado abstrato fila
7.9 Classes contêiner e iteradores
7.10 Classes proxy
7.11 (Estudo de caso opcional) Pensando em objetos: programando as classes para o simulador de elevador
Resumo . Terminologia . Erros comuns de programação • Boas práticas de programação Dicas de desempenho• Observações de engenharia de software Dicas de teste e depura ção Exercícios de auto-revisão• Respostas aos

exercícios de auto-revisão • Exercícios

7.1 Introdução

Neste capítulo, continuamos o nosso estudo de abstração de dados e classes. Discutimos muitos tópicos mais avançados e assentamos a base para a discussão de classes e sobrecarga de operadores no Capítulo 8. A discussão nos Capítulos 6 a 8 encoraja que os programadores usem objetos, o que chamamos *de programação baseada em objetos (OBP object-based programming).* A seguir, os Capítulos 9 e 10 introduzem herança e polimorfismo - as técnicas da verdadeira *programação orientada a objetos (OOP).* Nestes e em vários capítulos subseqüentes, usamos *strings* ao estilo de programação em C que introduzimos no Capítulo 5. Isto ajudará o leitor a dominar o complexo tópico de ponteiros em C e prepará-lo para o mundo profissional, no qual o leitor verá muito código legado em C, desenvolvido durante as últimas duas décadas. No Capítulo 19, discutimos o novo estilo de *strings,* isto é, *strings* como objetos de classes no sentido pleno da palavra. Deste modo, o leitor familiarizar-se-á com os dois métodos mais comuns de criar e manipular *strings* em C++.

7.2 Objetos const (constantes) e funções membro const

Ternos enfatizado o *princípio do mínimo privilégio* como um dos princípios mais fundamentais da boa engenharia de software. Iremos ver como esse princípio se aplica a objetos.

Alguns objetos necessitam ser modificáveis e outros não. O programador pode usar a palavra-chave const para especificar que um objeto não é modificável e que qualquer tentativa de modificar o objeto deve ser considerada um erro de sintaxe. Por exemplo,

```
const Time noon ( 12, 0, 0 );
```

declara um objeto const noon da classe Time e o inicializa às 12 horas.

```
// Fig. 7.1: time5.h
// Declaração da classe Time.
// Funções membro definidas em time5.cpp
#ifndef TIME5H
#define TIME5H
7 class Time { 60
public:
Time( int =0, int =0, int =0); // construtor default
// funções de inicialização
void setTime( int, int, int
void setHour( int );
void setMinute ( int );
void setSecond( int );
// funções obter (normalmente declaradas const)
int getHour() const; // retorna hour
int getNinute() const; // retorna minute
int getSecond() const; // retorna second
// funções de impressão (normalmente declaradas const)
void printMilitary() const; // imprime hora militar
void printStandard // imprime hora padrão
private:
int hour;
int minute;
int second;
83
84
85
```

Fig. 7.1 Usando uma classe Time com objetos const e funções membro const - time5 .h. 86

(na linha 108). O programa também ilustra as três outras combinações de chamadas de funções membro a objetos - uma função membro não-const a um objeto não-const (linha 100), uma função membro const a um objeto não const (linha 104) e uma função membro const a um objeto const (linhas 106 e 107). As mensagens geradas por dois compiladores populares para funções membros não-const que chamam um objeto const são mostradas na janela de saída.

Boa prática de programação 7.1

Declare como cons **t** *todas as funções membro que não necessitam modificar o objeto corrente, deforma que você possa usá-las sobre um objeto* const *se necessitar*

```
1
2
3
4
5
6


41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59


8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31


// inicializa time // inicializa hour
// inicializa minute
// inicializa second


61
62
63
64
65
66
67
68
69
70
```

71
72
73
74
75
76
77
78
79
80
81
82

// o -23
// O -59
// O -59

#endif

87
88

Fig.
2 de

89
90
91

92
Fig.
(part

Fig. 7.1 Usando uma classe Time com objetos const e funções membro const - time5 . h (parte 1 de 2).

32 33 34	// Fig. 7.1: time5.cpp // Definições das funções #include <iostream>	membro	para a classe Time.
35			
36	using std::cout;		
37			
38	#include "time5.h"		
39			
40	// Função construtor para	inicializar dados	privados.

os- 41 /1 Valores default são O (ver definição da classe).
jeto 42 Time::Time(int hr, int mm, int sec)
ens 43 { setTime(hr, mm, sec); }
sao 45 /1 Inicializa os valores de hour, minute e second.
46 void Time::setTime(int h, int m, int s
47
48 setHour(h);
49 setMinute(m);
50 setSecond(s);
51

```
52
53 // Inicializa o valor de hour
54 void Time::setHour( int h
55 {hour=(h>=0&&h<24)?h:0;}
56
57 /1 Inicializa o valor de minute
58 void Time: :setMinute( int m
59 {minute= (m>=0 &&m< 60) ?m :0;)
60
61 // Inicializa o valor de second
62 void Time::setSecond( int s
63 ( second =( s >=o && s < 60 )? s : 0; }
64
65 /1 Obtém o valor de hour
66 int Time: :getHour() const { return hour;
67
68 // Obtém o valor de minute
69 irit Time: :getMinute() const { return minute;
70
71 // Obtém o valor de second
72 int Time: :getSecond() const { return second;
73
74 // Exibe o tempo no formato militar: HH:MM
75 void Time: :printMilitary() const
76
77 cout « ( hour < 10 ? '0" : -) « hour «
78 « ( minute < 10 ? "0' : -) « minute;
79 }
80
81 /1 Exibe o tempo no formato padrão: HH:MM:SS da manhã (ou tarde)
82 void Time::printStandard() // deveria ser const
83
84 cout « ( ( hour == 12 ) ? 12 : hour % 12 ) «
85 « ( minute < 10 ? "0" : - ) « minute «
86 « ( second < 10 ? "0' : - ) « second
87 « ( hour < 12 ? - da manhã" : - da tarde");
88 )
```

Fig. 7.1 Usando uma classe Time com objetos const e funções membro const - time5 . cpp (parte 2 de 2).

```
89 1/ Fig. 7.1: figo7_01.cpp
90 // Tentando acessar um objeto const
91 // com funções membro não-const.
92 #include "time5.h"
```

Fig. 7.1 Usando uma classe Time com objetos const e funções membro const - figo7_01 . cpp (parte 1 de 2).


```
93
94 int main()
95 O
96 Time wakeUp( 6, 45, O ); // objeto não-constante
97 const Time noon( 12, 0, O ); // objeto constante
98
99 // FUNÇÃO MEMBRO OBJETO
100 wakeUp.setHour( 18 ); // não-const não-const
101
102 noon.setHour( 12 ); // não-const const
103
```

```
104 wakeUp.getHourf); // const não-const
105
106 noon.getMinuteO; // const const
107 noon.printMi1itary // const const
108 noon.printStandardO); // não-const const
109 return 0;
110 }
```

Mensagens de erro do compilador Borland C++ com linha de comando

```
FigO7Ol .cpp:
Warning W8037 FigO7 01.cpp 14: Non-const function Time::setHour(int) called for const
object in function main()
Warning W8037 FigO7 01.cpp 20: Non-const function Time: :printStandard(int) called for
const object in function main()
Turbo Incremental Link 5.00 Copyright (c) 1997, 2000 Borland
```

Mensagens de erro do compilador Microsoft Visual C++

```
Compiling...
FigO 7_Ol .cpp
d:figo7_01.cpp(14): error C2662: 'setHour' : cannot convert 'this' pointer from'const
class Time' to 'class Time &
Conversion loses qualifiers
d:\fig07_01.cpp(20): error C2662: 'printStandard : cannot convert 'this' pointer from
'const class Time' to 'class Time &'
Conversion loses qualifiers
time 5 .cpp
Error executing cl.exe.
test.exe -2 error(s), O warning(s)
```

Fig. 7.1 Usando uma classe Time com objetos const e funções membro const -fig07 01 .cpp (parte 2 de 2).

Note que, muito embora um construtor deva ser uma função membro não-const, ainda assim ele pode ser chamado para um objeto const. A definição do construtor de Time nas linhas 42 e 43

```
Time::Time( int hr, int mm, int sec
{ setTime( hr, mm, sec); }
```

mostra que o construtor de Time chama outra função membro não-const -setTime -para executar a inicialização de um objeto de tipo Time. Invocar uma função membro não-const a partir de uma chamada para o construtor de um objeto const é permitido. A característica de ser const de um objeto é assegurada a partir do momento em que o construtor completa a inicialização do objeto e até que o destruidor daquele objeto seja chamado.

Observação de engenharia de software 7.4

_____ *Um objeto* cons **t** *não pode ser modificado por atribuição; por isso ele deve ser inicializado. Quando um membro de dados de uma classe é declarado* **const,** *um* inicializador de membro *deve ser usado para fornecer ao construtor o valor inicial do membro de dados de um objeto* da *classe.*

Note também que a linha 108 (linha 20 no arquivo-fonte)

noon.printStandard O; *II* non-const const

gera um erro de compilação, embora a função membro printStandard da classe Time não modifique o objeto sobre o qual é invocada. Não modificar um objeto não é suficiente para indicar um método const. O método também precisa ser explicitamente declarado const.

A Fig. 7.2 demonstra o uso de um inicializador de membro para inicializar o membro de dados const increment da classe Increment. O construtor para Increment é modificado, como segue:

Increment::Increment(`int a, int i`

`increment( i`

{ count = **a;** }

A notação : increment (i) inicializa increment com o valor i. Se são necessários vários inicializadores de membros, simplesmente inclua-os em uma lista separada por vírgulas depois dos dois-pontos. Todos os membros de dados *podem* ser inicializados usando sintaxe de inicializador de membro, mas consts e referências *devem* ser inicializados desta maneira. Mais tarde, neste capítulo, veremos que objetos membro devem ser inicializados deste modo. No Capítulo 9, quando estudarmos herança, veremos que as partes da classe base, em classes derivadas, também devem ser inicializadas deste modo.



Dica de teste e depura ção 7.1

Sempre declare funções membro como const *se elas não modificarem o objeto. Isso pode ajudar a eliminar muitos bugs.*

```
1 // Fig. 7.2: fig07_02.cpp
2 // Usando um inicializador de membro para inicializar
3 // urna constante de um tipo de dado primitivo.
4 #include <iostream>
5
6 using std: :cout;
7 using std: :endl;
8
9 class Increment
10 public:
11 Increment( int c = 0, int i = 1 );
12 void addlncrement() { count += increment;
13 void print() const;
14
15 private:
16 int count;
17 const int increment; // membro de dados const
18 };
19
20 // Construtor para a classe Increment
21 Increment: :Increment( int c, int i
```

Fig. 7.2 Usando um inicializador de membro para inicializar uma constante de um tipo de dado primitivo (parte 1 de 2).


```
22 : increment( i ) // inicializador para membro const
23 { count = e;
24
25 // Imprime os dados
26 void Increment: :print() const
27
28 cout « "count = " « count
29 « ", increment = " « increment « endi;
30 }
31
32 int main O
33
34 Increment value( 10, 5 );
35
36 cout « 'Antes de incrementar: ";
37 value.printO;
38
39 for(intj=0;j<3;j++){
40 value.addlncrementO;
41 cout « "Após incrementar " « + 1 « ":
42 value.printQ;
43
44
45 return 0;
46
Antes de incrementar: count = 10, increment = 5
Após incrementar 1: count = 15, increment = 5
Após incrementar 2: count = 20, increment = 5
Após incrementar 3: count = 25, increment =
```

Fig. **7.2** Usando um inicializador de membro para inicializar uma constante de um tipo de dado primitivo (parte 2 de 2).

A Fig. 7.3 ilustra as mensagens de erros de compilação emitidas por dois compiladores populares de C++ para um programa que tenta inicializar increment com um comando de atribuição em vez de com um inicializador de membro.

```
1 // Fig. 7.3: figO7O3.cpp
2 // Tentando inicializar uma constante de uni
3 // tipo de dado primitivo com uma atribuição.
4 #include <iostreani>
5
6 using std: :cout;
7 using std: :endl;
8
9 class Increment {
10 public:
11 Increment( int c =0, int i =1 );
12 void addlncrement() { count +increment; }
13 void print() const;
14 private:
15 int count;
```

Fig. 7.3 Tentativa errônea de inicializar uma constante de um tipo de dado primitivo por atribuição (parte 1 de 2).


```
16 const int increment;
17
18
19 // Construtor para a classe Increment
20 Increment: :Increment( int c, int i
21 { // Membro constante 'increment' não está inicializado
22 count =
23 increment =i; // ERRO: não se pode modificar um objeto const
24 )
25
26 // Imprime os dados
27 void Increment: :print() const
28 {
29 cout« "count =" « count
30 « ", increment =" « increment « endl;
31
32
33 int main()
34 {
35 Increment value( 10, 5 );
36
37 cout « 'Antes de incrementar:
38 value.printO;
39
40 for(mntj=0;j<3;j++){
41 value.addlncrementQ;
42 cout « "Após incrementar " « j «
43 value.printO);
44 }
45
46 return 0;
47 )
```

Mensagens de erro do compilador Borland C++ com linha de comando

```
Fig0703.cpp:
de Warning W8038 FigO7 03.cpp 21: Constant member 'Increment: :increment' is not
initialized in function Increment: :Increment(int,int)
Error E2024 FigO7 03.cpp 23: Cannot modify a const object in function
Increment: : Increment (int, int)
Warning W8057 FigO7 03.cpp 24: Parameter 'i' is never used in function
Increment: : Increment(int,int)
*** 1 errors in Compile ***
```

Mensagens de erro do compilador Microsoft Visual C++

```
Compiling...
Fig0703 cpp
D:\Fig0703.cpp(21) : error C2758: 'increment' : must be initialized in
constructor base/member initializer list
D:\Fig0703.cpp(16) : see declaration of 'increment'
D:\Fig0703.cpp(23) : error C2166: 1-value specifies const object
Error executing cl.exe.
test.exe 2 error(s), O warning(s)
rte
```

Fig. 7.3 Tentativa erronea de inicializar uma constante de um tipo de dado primitivo por atribuiçao (parte 2 de 2).

Erro comum de programação 7.5

Não definir um inicializador de membro para um membro de dados **const** *é um erro de sintaxe.*

Observação de engenharia de software 7.5

____ Membros de classe constantes (objetos const *e "variáveis"* const) *devem ser inicializados com a sintaxe de inicializador de membro; não são permitidas atribuições.*

Note que a função print na linha 27 é declarada const. É razoável, embora estranho, rotular essa função como const, porque, provavelmente, nunca teremos um objeto Increment const.

Observação de engenharia de software 7.6

É *uma boa prática declarar como* const *todas as funções membro de uma classe que não modficam o objeto sobre o qual elas operam. Ocasionalmente, isto será uma anomalia, porque você não tem nenhuma intenção de criar objetos* **const** *daquela classe. Entretanto, declarar como* **const** *tais funções membro oferece um benefi'cio: se você, inadvertidamente, modificar o objeto naquela função membro, o compilador emitirá uma mensagem de erro de sintaxe.*

® *Dica de teste e depura ção 7.2*

As linguagens como C+ **+** *são como "alvos móveis pois evoluem.* É *provável que mais palavras-chave sejam incorporadas à linguagem. Evite usar palavras muito importantes, tais como "objeci como identificadores. Embora "object" não seja atualmente uma palavra-chave em C++, no futuro ela o poderá ser Assim, futuros compiladores poderiam tornar inoperante o código existente.*

C++ fornece uma palavra-chave chamada mutable que afeta o tratamento de objetos const em um programa. Discutiremos a palavra-chave mutable no Capítulo 21.

7.3 Composição: objetos como membros de classes

Um objeto da classe RelogioDespertador necessita saber quando ele deve soar seu alarme; assim, por que não incluir um objeto do tipo Time como membro do objeto RelogioDespertador? Tal recurso é chamado de *composição.* Uma classe pode ter objetos de outras classes como membros.

Observação de engenharia de software 7.7

____ A forma mais comum de reutilização de software é a composição, na qual uma classe tem objetos de outras classes como membros.

Quando um objeto é criado, seu construtor é chamado automaticamente, de modo que necessitamos especificar como os argumentos são passados para construtores de objetos membro. Os objetos membro são construídos na ordem em que são declarados (não na ordem em que são listados na lista de inicializadores de membros do construtor) e antes de os objetos de classe que os incluem (às vezes denominados *objetos hospedeiros)* serem construídos.

A Fig. 7.4 usa a classe Employee e a classe Date para demonstrar objetos como membros de outros objetos. A classe Eniployee contém membros de dados privados firstName. lastName. birthDate e hireDate. Os membros birthDate e hireDate são objetos const da classe Date, que contém os membros de dados privados month, day e year. O programa instancia um objeto Employee e inicializa e exibe seus membros de dados. Note a sintaxe do cabeçalho da função, na definição do construtor de Employee:

Employee: **`:Employee( char *fne, char *1ne,`**
`int bmonth, int bday, int byear,`
`int hmonth, int hday, int hyear`
`birthDate( bmonth, bday, byear`),
`hireDate( hmonth, hday, hyear`

O construtor recebe oito argumentos (fnaine. mame, bmonth. bday. byear. hmonth, hday e hyear). Os dois-pontos **(:)** no cabeçalho separa os inicializadores de membros da lista de parâmetros. Os inicializadores de membros especificam os argumentos de Employee que estão sendo passados para os construtores dos objetos membro Date. Os argumentos

bmonth, bday e byear são passados para o construtor do objeto birthDate e os argumentos hmonth. hday e hyear são passados para o construtor do objeto hireDate. Inicializadores de membros múltiplos são separados por vírgulas.

```
1 II Fig. 7.4: datel.h
2 1/ Declaração da classe Date.
3 II Funções membro definidas em datel.cpp
4 #ifndef DATE1_H
5 #define DATE1_H
6
7 class Date
na 8 public:
9 Date( int =l, int =1, int =1900 ); // construtor default
la- 10 void print() const; II imprime a data no formato mês/dia/ano
11 'DateO; // fornecido para confirmar a ordem de destruição
12 private:
13 int month; // 1-12
.14 int day; II 1-31 dependendo do mês
ive 15 int year; // qualquer ano
'nO 16
'rá 17 // função utilitária para testar validade do dia para mês e ano
18 int checkDay(int);
19
20
21 #endif
```

Fig. 7.4 Usando inicializadores para objetos membro .datel .h.

```
22 // Fig. 7.4: datel.cpp
não 23 /1 Definições de funções membro para a classe Date.
de 24 #include <iostream>
25
26 using std::cout;
27 using std: :endl;
.28
de 29 #include "datel.h"
30
31 1/ Construtor: confirma valor adequado para month
32 // chama a função utilitária checkDay para
33 // confirmar valor adequado para day.
,na .
34 Date: :Date( int mn, int dy, int yr
35
dOs. 36 if ( mn > O && mn < 12 ) // valida o mês
bje- 37 month =mn;
te. 38 else {
idos 39 month =1;
sde 40 cout « "Mês « mn « inválido. Inicializa mês com 1.\n";
41 }
42
43 year yr; 1/ deveria validar yr
44 day =checkDay( dy ); /1 valida o dia
45
```

Fig. 7.4 Usando inicializadores para objetos membro .datei .cpp (parte 1 de 2).


```
46 cout « Construtor do objeto Date para a data ";
47 print(); // interessante: uma função print sem argumentos
48 cout « endi;
49 }
50
51 // Imprime o objeto Date no formato mês/dia/ano
52 void Date::print() const
53 { cout « month « '1' « day « '1' « year;
```

```
54
55 // Destruidor: fornecido para confirmar a ordem de destruição
56 Date::-Date()
57
58 cout « "Destruidor do objeto Date para a data ";
59 print();
60 cout « endi;
61
62
63 // Função utilitária para confirmar o valor apropriado
64 // do dia, com base no mês e no ano.
65 // O ano 2000 é um ano bissexto?
66 int Date::checkDay( int testDay
67 {
68 static const int daysPerMonth[ 13 ] =
69 {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
70
71 if ( testDay > O && testDay <= daysPerMonth[ month
72 return testDay;
73
74 if ( month == 2 && // Fevereiro: testa se é ano bissexto
75 testDay == 29 &&
76 (year % 400 == 011 1/ano 2000?
77 ( year % 4 == O && year %100 ! O ) ) ) 1/ ano 2000 ?
78 return testDay;
79
80 cout « "Dia « testDay « inválido. Inicializa dia com
81
82 return 1; // deixa o objeto em estado consistente se o valor for inadequa&
83 }
```

Fig. 7.4 Usando inicializadores para objetos membro datei **cpp.**

Lembre-se de que membros e referências const também são inicializados na lista de inicializadores de membro (no Capítulo 9, veremos que as partes da classe base de classes derivadas também são inicializadas deste modo). classe Date e a classe Empioyee incluem uma função destruidor que imprime uma mensagem quando um objet do tipo Date ou um objeto do tipo Empioyee é destruído, respectivamente. Isto nos permite confirmar na saída d programa que os objetos são construídos de dentro para fora e destruídos na ordem inversa, de fora para dentro (i.e. os objetos membro de Date são destruídos depois do objeto Empioyee que os contém).

Um objeto membro não necessita ser inicializado explicitamente através de um inicializador de membro. S um inicializador de membro não é fornecido, o construtor default do objeto membro será chamado implicitamente Os valores, se houver algum, estabelecidos pelo construtor default podem ser substituídos por funções *set.* Entretan to, para inicialização complexa, essa abordagem pode exigir trabalho e tempo adicionais significativos.

```
84 // Fig. 7.4: empiyi.h
85 // Declaração da classe Employee.
86 /1 Funções membro definidas em emplyl.cpp
```

Fig. 7.4 Usando inicializadores para objetos membro **emplyi h** (parte 1 de 2).


```
87 #ifndef EMPLY1H
88 #define EMPLY1H
89
90 #include "datel.h"
91
92 class Employee
93 public:
94 Employee( char char int, int, int, int, int, int );
95 void print() const;
96 '.-EmployeeO; II fornecido para confirmar a ordem de destruição
97 private:
```

```
98 char firstName[ 25 );
99 char lastName[ 25 J;
100 const Date birthDate;
101 const Date hireDate;
102 };
103
104 #endif
```

Fig. 7.4 Usando inicializadores para objetos membro - emplyl . h (parte 2 de 2).

Erro comum de programação 7.6

Não definir um construtor default para a classe de um objeto membro, quando nenhum inicializador de membro éfornecido para aquele objeto membro, é um erro de sintaxe.

```
105 //Fig. 7.4: emplyl.cpp
106 //Definições das funções membro para a classe Employee.
107 #include <iostream>
108
109 using std::cout;
110 using std::endl;
111
112 #include <cstring>
iado 113 #include “emplyl.h”
114 #include ‘datel.h”
115
116 Employee::Employee( char *fname, char *lname,
117 int bmonth, int bday, int byear,
118 int hxnonth, int hday, int hyear)
-ibros 119 : birthDate( bmonth, bday, byear ),
o).A 120 hireDate( hmonth, hday, hyear
bjeto 121
dado. 122 /I copia fname para firstName e assegura que ele cabe
(i.e. 123 int length strlen( fname );
124 length =( length < 25 ? length : 24 );
o Se 125 strncpy( firstName, fname, length );
• 126 firstName[ length ] =
ente. 127
‘etan- 128 II copia lname para lastNaxne e assegura que ele cabe
129 length = strlen( lname );
130 length =( length < 25 ? length : 24 );
131 strncpy( lastName, lname, length );
132 lastName[ length ] =
133
134 cout « “Construtor do objeto Employee:
135 « firstName « , · « lastName « endi;
```

Fig. 7.4 Usando inicializadores para objetos membro - emplyl . cpp (parte 1 de 2).


```
136 }
137
138 void Employee: :print() const
139
140 cout « lastNaxne « ·,· « firstName « “\nContratado:
141 hireDate.print
142 cout « ·Data de nascimento: “;
143 birthDate.printO;
144 cout « endl;
145
146
147 1/Destruidor: fornecido para confirmar a ordem de destruição
148 Employee: :-Employee()
149
150 cout « “Destruidor do objeto Employee:
151 « lastName « ·,· « firstName « endi;
152
```

Fig. 7.4 Usando inicializadores para objetos membro - emplyl . cpp (parte 2 de 2).

Dica de desempenho 7.2

___Inicialize explicitamente objetos membro através de inicializadores de membros. Isto elimina o overhead *da “dupla*

inicialização" de objetos membro - uma vez quando o construtor default do objeto membro for chamado e novamente quando funções set forem usadas para inicializar o objeto membro.

```
153 //Fig. 7.4: figO7O4.cpp
154 //Demonstrando a composição: um objeto com objetos membro.
155 #include <iostream>
156
157 using std::cout;
158 using std: :endl;
159
160 #include "emplyl.h"
161
162 int main ()
163
164 Employee e( "José", "Silva", 7, 24, 1949, 3, 12, 1988 );
165
166 cout «  '\n';
167 e.printO;
168
169 cout « "\nTesta o construtor de Date com valores inválidos:\n";
170 Date d( 14, 35, 1994 ); // valores inválidos de Date
171 cout « endl;
172 return 0;
173 }
```

Fig. 7.4 Usando inicializadores para objetos membro - figo7 04 - cpp (parte 1 de 2).

```
Construtor do objeto Date para a data 7/24/1949
Construtor do objeto Date para a data 3/12/1988
Construtor do objeto Employee: José Silva
Silva, José Contratado: 3/12/1988 Data de nascimento: 7/24/1949
```


Fig. 7.4 Usando inicializadores para objetos membro - figO7 04 - cpp (parte 2 de 2).

Observação de engenharia de software 7.8

___Se uma classe tem como membro um objeto de outra classe, tornar esse objeto membro public *não viola o encapsulamento e a ocultação dos membros* **private** *daquele objeto membro.*

Note a chamada para a função membro print de Date na linha 52. Muitas funções membro de classes em C++ não exigem nenhum argumento. Isso ocorre porque cada função membro contém um *handie* implícito (na forma de um - ponteiro) - para o objeto sobre o qual ela opera. Discutiremos o ponteiro implícito - chamado this - na Seção 7.5. head Nesta primeira versão de nossa classe Employee (por facilidade de programação), usamos dois arrays de 25 *ofor* caracteres (firstName e lastName) para representar o primeiro e o último nomes do Employee. Esses arrays podem ser um desperdício de espaço para nomes mais curtos que 24 caracteres (lembre-se de que um caractere em cada array está destinado para o caractere nulo terminal, '\O ',do *string).* Fora isso, nomes maiores que 24 caracteres devem ser truncados para caber nestes arrays de caracteres. Mais à frente neste capítulo, apresentaremos outra versão da classe Employee que cria dinamicamente a quantidade exata de espaço para manter o primeiro e o último nome. Também poderíamos usar dois objetos string para representar os nomes. A classe string da biblioteca padrão é apresentada em detalhes no Capítulo 19.

7.4 Funções friend e classes friend

Um função **`friend de uma classe`** é definida fora do escopo daquela classe, mas ainda tem o direito de acessar membros private da classe (e protected. como veremos no Capítulo 9, "Herança"). Uma função ou uma classe inteira podem ser declaradas como friends de outra classe.

Usar funções friend melhora o desempenho. Aqui é mostrado um exemplo mecânico de como funciona uma função friend. Mais à frente no livro, funções friend serão usadas para sobrecarregar operadores para uso com objetos de classes e para criar classes de iteradores. Os objetos de uma classe de iteradores são usados sucessivamente para selecionar itens ou executar uma operação sobre os itens em um objeto de uma classe contêiner (ver Seção 7.9). Os objetos de classes contêineres são capazes de armazenar itens. O uso de friends freqüentemente é apropriado quando

uma função membro não puder ser usada para certas operações, como veremos no Capítulo 8, “Sobrecarga de operadores”.
Para declarar uma função como um friend de uma classe, preceda o protótipo da função na definição da classe com a palavra-chave friend. Para declarar a classe ClassTwo como um friend da classe ClassOne. coloque uma declaração da forma
friend class ClassTwo;
na definição da classe ClassOne.
Observação de engenharia de software 7.9
___Embora os protótipos para funções **friend** *apareçam na definição da classe, ainda assim* **friends** *não são funções membro.*

LI

Testa o construtor de Date com	valores	inválidos:
Mês 14 inválido. Inicializa mês	com 1.	
Dia 35 inválido. Inicializa dia	com 1.	
Construtor do objeto Date para	a data	1/1/1994
Destruidor do objeto Date para	a data	1/1/1994
Destruidor do objeto Employee:	Silva,	José
Destruidor do objeto Date para	a data	3/12/1988
Destruidor do objeto Date para	a data	7/24/1949

Observação de engenharia de software 7.10
___As noções de acesso a membros private, protected *e* public *não são relevantes para as declarações de relações* friend, *de modo que as declarações de relações* friend *podem ser colocadas em qualquer lugar na definição da classe.*
Boa prática de programação 7.2
Coloque todas as declarações friend *no início da classe, logo depois do cabeçalho da classe, e não as preceda com quaisquer especificadores de acesso a membros.*
A condição de friend é concedida, não tomada; i.e., para a classe B ser friend da classe A, a classe A deve declarar explicitamente a classe B como sua friend. Além disso, a condição de friend não é nem simétrica nem transitiva: i.e., se a classe A é friend da classe B e a classe B é friend da classe C, você não pode deduzir que a classe B é friend da classe A (repetindo, a condição friend não é simétrica), que a classe C é friend da classe B, ou que a classe A é friend da classe C (repetindo, friend não é transitiva).
Observação de engenharia de software 7.11
Algumas pessoas na comunidade de OOP consideram que a noção de friend *corrompe a ocultação de informações e enfraquece o valor da abordagem de projeto orientado a objetos.*
A Fig. 7.5 demonstra a declaração e uso da função friend setx para inicialização do membro de dados private x da classe count. Note que a declaração de friend aparece primeiro (por convenção) na declaração da classe, mesmo antes de as funções membro public serem declaradas, O programa da Fig. 7.6 demonstra as mensagens produzidas pelo compilador quando a função não-friend cannotSetX é chamada para modificar o membro de dados private x. As Figs. *7.5* e 7.6 têm por objetivo apresentar a “mecânica” do uso de funções friend
exemplos práticos do uso de funções friend aparecerão nos capítulos mais à frente.

```
1 II Fig. 7.5: figO7O5.cpp
2 //Friends podem acessar membros private de uma classe.
3 #include <iostream>
4
5 using std: :cout;
6 using std::endl;
7
8 /l Classe Count modificada
9 class Count
10 friend void setX( Count &,int ); /l Declaração de friend
11 public:
12 Count() { x = 0; } II Construtor
13 void print() const { cout « x « eridl; } II saída
```

```
14 private:
15 int x; // membro de dados
16
17
18 // Pode modificar dados private de Count porque
19 // setX é declarada como uma função friend de Count
20 void setX ( Count &c, int val
21
22 c.x = val; // válido: setX é função friend de Count
23
24
25 int main ()
26
```

Fig. 7.5 friendS podem acessar membros private de uma classe (parte 1 de 2).


```
27 Count counter;
28
29 cout « 'counter.x após instanciação:
30 counter.print ();
31 cout « "counter.x após chamada à função friend setX:
32 setX( counter, 8 ); // inicializa x com uma função friend
33 counter.print();
34 return 0;
35 }
```

Fig. 7.5 friendS podem acessar membros private de uma classe (parte 2 de 2).

Note que a função setX (linha 20) *é* uma função ao estilo da programação em C, uma função isolada - ela não é uma função membro da classe Count. Por essa razão, quando setX é invocada para o objeto counter, usamos o comando na linha 32

setX(counter, 8); *// setX com uma função friend*

que recebe counter como um argumento em vez de usar um *handle* (tal como o nome do objeto) para chamar a função, como em

`counter.setX( 8 );`

Como mencionamos, a Fig. 7.5 é um exemplo mecânico da construção `friend`. Normalmente, seria apropriado definir a função **setX** como uma função membro da classe Count.

Observação de engenharia de software 7.12

Como C++ é uma linguagem híbrida, é comum ter-se uma mistura de dois tipos de chamadas de função em um programa efreqüentemente lado a lado - chamadas ao estilo de C, que passam dados primitivos ou objetos para funções, e chamadas no estilo próprio de C++, que passam funções (ou mensagens) para objetos.

```
1 // Fig. 7.6: fig07_06.cpp
2 // Funções não-friend / não-membro não podem
3 // acessar dados private de uma classe.
4 #include <iostream>
5
6 using std::cout;
7 using std: :endl;
8
9 // Classe Count modificada
10 class Count {
11 public:
12 Count() { x = 0; } // construtor
13 void print() const { cout « x « endl; } // saída
14 private:
15 int x; // membro de dados
```

Fig. **7.6** Funções não-friend / não-membro não podem acessar membros de classe private (parte 1 de 2).

counter.x	após	instanciação: O
counter.x	após	chamada à função friend setX: 8


```
16
17
18 // Função tenta modificar dados private de Count,
19 // mas não pode porque não é uma função friend de Count.
20 void cannotSetX( Count &c, int vai
21
22 c.x = vai; // ERRO: 'Count: :x' não é acessível
23 }
24
25 int main Q
26
27 Count counter;
28
29 cannotSetX( counter, 3 ); // cannotSetx não é uma função friend
30 return 0;
31
```

Mensagens de erro do compilador Borland C++ com linha de comando

```
Borland C++ 5.5 for Win32 Copyright (c) 1993, 2000 Borland
Fig0706 .cpp:
Error E2247 FigO7_06.cpp 22: Count::x' is not acessibie in
function cannotSetX(Count &,int)
*** 1 errors in Compile ***
```

Mensagens de erro do compilador Microsofi Visual C++

```
Compiiing...
FigO 7_06 .cpp
D: \books\2000\cpphtp3\exampies\Ch07\FigO7 06\FigO7 06. cpp (22) error C2248:
'x' : cannot access private member deciared in
class Count
D: \books\2000\cpphtp3\examples\Ch07\Fig0706\
Fig0706.cpp(15) : see deciaration of 'x
Error executing cl.exe.
test.exe - 1 error(s), O warning(s)
```

Fig. 7.6 Funções não-friend / não-membro não podem acessar membros de classe **private** (parte 2 de **2).**

É possível especificar funções sobrecarregadas como friends de uma classe. Cada função sobrecarregada com intenção de ser um friend deve ser explicitamente declarada na definição da classe como um friend da classe.

7.5 Usando o ponteiro this

4 Todo objeto tem acesso ao seu próprio endereço através de um ponteiro chamado this. O ponteiro this de um objeto não faz parte do objeto propriamente dito - exemplo: o tamanho do ponteiro this não é computado no resultado de uma operação sizeof sobre o objeto. Em vez disso, o ponteiro this é passado para o objeto (pelo compilador) como primeiro parâmetro implícito em toda chamada de função membro não-static para o objeto (membros static são discutidos na Seção 7.7).

O ponteiro this é implicitamente usado para prover referência tanto para os membros de dados como para as funções membro de um objeto; também pode ser usado explicitamente. O tipo do ponteiro this depende do tipo do

objeto e de se a função membro em que this é usado é declarada const ou não. Em uma função membro não- constante da classe Employee, o ponteiro this tem o tipo Employee * const (um ponteiro constante para um um objeto de Employee). Em uma função membro constante da classe Employee. o ponteiro this tem o tipo de dado const Employee * const (um ponteiro constante para um objeto Employee que é constante).

No momento, mostraremos um exemplo simples do uso explícito do ponteiro this: mais tarde, neste capítulo e no Capítulo 8, mostraremos alguns exemplos significativos e sutis de uso de this. Toda função membro nãostatic tem acesso ao ponteiro this para o objeto para o qual o membro está sendo invocado.

Dica de desempenho 7.3

___Por motivos de economia de memória e armazenamento, existe só uma cópia de cada função membro por classe e essa função membro é invocada para todos os objetos daquela classe. Cada objeto, por outro lado, tem sua própria cópia dos membros de dados da classe.

A Fig. 7.7 demonstra o uso explícito do ponteiro this para habilitar uma função membro da classe Test a imprimir o dado privado x de um objeto de Test.

```
1 II Fig. 7.7: figO7O7.cpp
2 // Usando o ponteiro this para referenciar membros de objetos.
3 #include <iostreain>
4
5 using std::cout;
6 using std::endl;
7
8 class Test
9 public:
10 Test( int =O ); // construtor default
11 void print() const;
12 private:
13 int x;
14 };
15
16 Test::Test( int a ) { x= a; } II construtor
17
18 void Test: :print() const II () em volta de *this obrigatórios
19
20 cout « -x = « x
21 « \n this->x = « this->x
22 « \n(*this) .x = " « ( *this ) .x « endl;
23
24
25 int ntain()
26 {
27 Test test0bject( 12 );
28
29 testobject.printO;
30
31 return 0;
32 )
```

Fig. 7.7 Usando o ponteiro this.

	x = 12
this->x 12	
(*this)	.x = 12

Para fins de ilustração, a função membro print na Fig. 7.7 primeiro imprime x diretamente. A seguir, print usa 2 duas notações diferentes para acessar x através do ponteiro this - o operador seta (->) do ponteiro this e o 2 operador ponto (.) do ponteiro this derreferenciado.

Note os parênteses ao redor de *this quando ele é usado com o operador de seleção de membro ponto (.). 3(Os parênteses são necessários porque o operador ponto tem precedência mais alta que o operador . Sem os parên- 3 teses, a expressão 3•

3

*thjsx -

Fi

seria avaliada como se estivesse entre parênteses, como segue:

*(this.x

o que é um erro de sintaxe, pois o operador ponto não pode ser usado com um ponteiro.

Erro com um de programação 7.7
Tentar usar o operador de seleção de membro (.) com um ponteiro para um objeto é um erro de sintaxe - o operador de seleção de membro ponto só pode ser usado com um objeto ou com uma referência para um objeto.
Um uso interessante do ponteiro this é evitar que um objeto seja atribuído a si próprio. Como veremos no Capítulo 8, "Sobrecarga de operadores", a atribuição a si próprio pode produzir erros sérios quando os objetos contiverem ponteiros para memória alocada dinamicamente.
Outro uso do ponteiro this é habilitar chamadas a funções membro em cascata. A Fig. 7.8 ilustra retornar uma referência para um objeto Time para habilitar chamadas em cascata de funções membro da classe Time. Cada uma das funções membro setTime. setHour, setMinute e setSecond retorna *this com um tipo de retorno Time **&.**

```
1 // Fig. 7.8: time6.h
2 // Encadeando chamadas a funções membro.
3
4 // Declaração da classe Time.
5 // Funções membro definidas em time6.cpp
6 #ifndef TIME6H
7 #define TIME6H
8
9 class Time
10 public:
11 Time( int =0, int =0, int =0 ); // construtor default
12
13 // funções de inicialização
14 Time &setTime( int, int, int ); // inicializa hour, minute, second
15 Time &setHour( int ); // inicializa hour
16 Time &setMinute( int ); // inicializa minute
17 Time &setSecond( int ); // inicializa second
18
19 // funções para obter dados (normalmente declaradas const)
20 int getHour() const; // retorna hour
21 int getMinute() const; // retorna minute
22 int getSecond() const; // retorna second
23
24 // funções print (normalmente declaradas const)
25 void printMilitary() const; // imprime o tempo militar
```

Fig. 7.8 Encadeando chamadas a funções membro - time6 . h (parte 1 de 2).


```
void printStandard() const; // imprime o tempo padrão
private:
int hour; // 0 - 23
int minute; // 0 - 59
int second; // 0 - 59
```

Fig. 7.8 Encadeando chamadas a funções membro - time 6. h (parte 2 de 2).

Por que a técnica de retornar *this como uma referência funciona? O operador ponto **(.)** se associa da esquelua para a direita, de modo que a expressão
t.setHour(18) .setNinute(30) .setSecond(22);
primeiro avalia t. setHour **(** 18 **)** e então retorna uma referência para o objeto t como o valor desta chamada da função. A expressão restante é então interpretada como
t.setNinute(30) .setSecond(22);
A chamada t. setMinute **(** 30 é executada e retorna o equivalente de t. A expressão restante é interpretada como
t.setSecond(22)
Note que as chamadas
t.setTime(20, 20, 20) .printStandardO;
também usam o recurso de cascateamento. Essas chamadas devem aparecer nesta expressão nessa ordem, porque printStandard. como definida na classe, não retorna uma referência para t. Colocar a chamada para **printStandard,** no comando precedente, antes da chamada para setTime. resulta em um erro de sintaxe.

```
// Fig. 7.8: time6.cpp
// Definições de funções membro para a classe Time.
#include <iostream>
38 using std: :cout;
39
40 #include "time6.h"
```

```
// Função construtor para inicializar dados private.
// Chama função membro setTime para inicializar variáveis.
// Valores default são O (ver definição da classe)
Time::Time( int hr, int mm, int sec
{ setTime( hr, mi sec ); }
48 // Inicializa os valores de hour, minute e second.
49 Time &Time::setTime( int h, int m, int s
setHour ( h );
setMinute ( m );
setSecond( s );
```

b usa 3 eo

arên 26

```
27
28
29
30
```
31 };
```
32
33 #endif
```

xe -

i *um*

ítulo
Tem

mar ada

a de

```
34
35
36
37
```

```
41
42
43
44
45
46
47
```

50 {
```
51
52
53
```

Fig. 7.8 Encadeando chamadas a funções membro - time6 **.** cpp (parte 1 de 2).


```
54 return *thjs; // habilita o encadeamento 1C
```
55 } 1C

```
56 li
57 // Inicializa o valor de hour 1
58 Time &Time::setHour( int h ) 1]
59 { 1]
60 hour=(h>=0&&h<24)?h:0; 11
61 1]
62 return *this; // habilita o encadeamento 1]
63 } 1
64
65 // Inicializa o valor de minute 11
66 Time &Time: :setMinute( int m
67
68 minute = ( m >= O && m < 60 ) ? m : 0;
69
70 return *this; // habilita o encadeamento
71
72
73 // Inicializa o valor de second
74 Time &Time::setSecond( int s
75 {
76 second = ( s >= O && s < 60 ) ? s : 0;
77
78 return *this; 1/ habilita o encadeamento
79 }
80
81 // Obtém o valor de hour
82 int Time; :getHour() const { return hour;
83
84 // Obtém o valor de minute
85 int Time::getMinute() const return minute;
86
87 // Obtém o valor de second 7
88 int Time: :getSecond() const { return second;
89
90 // Exibe o tempo no formato militar: HH:MM
91 void Time: :printMilitary() const (p
92 { C
93 cout « ( hour < 10 ? "0" : "" ) « hour «
94 « ( minute < 10 ? "0" : "" ) « minute;
95 }
96
97 // Exibe o tempo no formato padrão: HH:4:SS da manhã (ou tarde)
98 void Time: :printStandard() const
99 {
100 cout« ( ( hour == O hour == 12 ) ? 12 : hour % 12
101 « ":" « ( minute < 10 ? "0" : "" ) « minute
102 ":" « ( second < 10 ? "0" : "" ) « second -
103 « ( hour < 12 ? " da manhã" ; ' da tarde" ); fu
104} m
```

Fig. 7.8 Encadeando chamadas a funções membro - time 6 . cpp (parte 2 de 2).

```
105 // Fig. 7.8: figO7OB.cpp O
106 // Encadeando chamadas a funções membro P(
107 1/ em conjunto com o ponteiro this ar
```

Fig. 7.8 Encadeando chamadas a funções membro - time 6 . cpp (parte 1 de 2). te,


```
108 #include <iostream>
```

```
109
110 using std::cout;
111 using std::endl;
112
113 #include "time6.h"
114
115 int main O
116 {
117 Time t;
118
119 t.setHour( 18 ) .setMinute( 30 ) .setSecond( 22 );
120 cout « "Hora militar: ";
121 t.printNilitaryO;
122 cout « "\nHora padrão:
123 t.printStandard
124
125 cout « "\n\nNova hora padrão:
126 t.setTime( 16, 20, 20 ).printStandardO;
127 cout « endl;
128
129 return 0;
130 }
Hora militar: 18:30
Hora padrão: 6:30:22 da tarde
Nova hora padrão: 4:20:20 da tarde
```

Fig. 7.8 Encadeando chamadas a funções membro - figo7 08 .cpp (parte 2 de 2).

7.6 Alocação dinâmica de memória com os operadores new e delete

Os operadores new e delete fornecem um meio mais agradável de executar a alocação dinâmica de memória
(para qualquer tipo primitivo ou definido pelo usuário) do que com as chamadas das funções malloc e free de C.
Considere o seguinte código

TypeName * typeNamePtr;

Em ANSI/C, para criar dinamicamente um objeto do tipo TypeName, você faria

typeNamePtr = malloc(sizeof(TypeName));

Isso exige uma chamada da função malloc e o uso explícito do operador sizeof. Em versões de C anteriores ao ANSI/C, você também teria que fazer uma coerção do ponteiro retornado por malloc com (TypeName *) . A função malloc não fornece qualquer método para inicializar o bloco de memória alocado. Em C++, você simplesmente escreve

typeNaxnePtr = **new TypeName;**

O operador new cria automaticamente um objeto do tamanho apropriado, chama o construtor do objeto e retorna um ponteiro do tipo correto. Se new não conseguiu encontrar espaço, ele retorna um ponteiro O em versões de C++ anteriores ao padrão ANSI/ISO. [Nota: no Capítulo 13, mostramos-lhe como lidar com insucessos de **new** no contexto do padrão ANSI/ISO para C++. Em particular, mostraremos como **new** "dispara" uma "exceção" e mostrare

mos como "capturar" aquela exceção e tratá-la]. Para destruir o objeto e liberar o espaço usado para este objeto em C++, você deve usar o operador delete. como segue:

delete typeNamePtr;

C++ permite que você forneça um *inicializador* para um objeto criado por new, como em:

```
double *thingptr =riew double ( 3.14159 );
```

que inicializa um objeto double recém-criado com 3 .14159.
Um array inteiro de 10 elementos pode ser criado e atribuído a arrayPtr como segue:

```
int *arrayptr =new int[ 10 ];
```

Este array pode ser deletado com o comando

```
delete [ ] arrayPtr;
```

Como veremos, usar new e delete, em vez de malloc e free. oferece também outros benefícios. Em particular, **new** invoca o construtor e delete invoca o destruidor da classe.

Erro comum de programação 7.8
Misturar o estilo **new** *e* **delete** *com o estilo* malJ.oc *e* **free** *de alocação dinâmica de memória dinâmica é um erro de lógica: o espaço criado por* malloc *não pode ser liberado por* **delete:** *os objetos criados por* **new** *não podem ser deletados por* **free.**
Erro comum de programação 7.9
Usar **delete** *em vez de* **delete [ )** *para arrays pode levar a erros de lógica durante a execução. Para evitar problemas, o espaço criado como um array deveria ser apagado com o operador* **delete [1** *e o espaço criado como um elemento individual deveria ser apagado com o operador* **delete.**
Boa prática de programação 7.3
Como C++ inclui C, programas em C++ podem conter memória criada por **malloc** *e liberada por* **free** *e objetos criados por* **new** *e deletados por* delete. *E melhor usar somente* **new** *e* **delete.**

7.7 Membros de classe static

Cada objeto de uma classe tem sua própria cópia de todos os membros de dados da classe. Em certos casos, todos os objetos de uma classe deveriam compartilhar só uma cópia de uma variável. Uma variável de classe static é usada por essa e outras razões. Uma variável de classe static representa informações "de âmbito de toda a classe" (i.e., uma propriedade da classe, não de um objeto específico da classe). A declaração de um membro static começa com a palavra-chave static.
Para motivar a necessidade de termos dados static - válidos para toda a classe - usaremos um exemplo de videogame. Suponhamos que temos um videogame com Marcianos e outras criaturas do espaço. Cada Marciano tende a ser valente e disposto a atacar outras criaturas do espaço quando o Marciano sabe que existem pelo menos cinco Marcianos presentes. Se menos de cinco Marcianos estiverem presentes, cada Marciano se torna covarde. Assim, cada Marciano necessita saber o numeroDeMarcianos. Poderíamos dotar cada instância da classe Marciano com numeroDeMarcianos como um membro de dados. Se assim fizermos, então todo Marciano terá uma cópia separada do membro de dados e toda vez que criarmos um novo Marciano teremos que atualizar o membro de dados numeroDeMarcianos em todos os objetos Marciano. Isso desperdiça espaço com as cópias redundantes e desperdiça tempo para atualizar as cópias separadas. Em vez disso, declaramos numeroDeMarcianos como static. Isso torna numeroDeMarcianos um dado conhecido por toda a das

to em se. Todo Marciano pode ver o nuineroDeMarcianos como se ele fosse um membro de dados de Marciano. mas só uma cópia do membro estático numeroDeMarcianos é mantida por C++. Economiza espaço. Economizamos tempo fazendo com que o construtor de Marciano incremente o numeroDeMarcianos estático. Como só existe uma cópia, não temos que incrementar cópias separadas de numeroDeMarcianos para cada objeto Marciano.
jDica de desempenho 7.4
___ Use membros de dados s tati c *para economizar memória quando uma única cópia dos dados é suficiente.*
Embora membros de dados static possam se parecer com variáveis globais, membros de dados static têm escopo de classe. Membros static podem ser public, private ou protected. Membros de dados static *devem* ser inicializados *uma vez* (e só uma vez) em escopo de arquivo. Os membros de dados public static de uma classe podem ser acessados por qualquer objeto daquela classe, ou então eles podem ser acessados através do nome da classe usando o operador de resolução de escopo binário. Os membros private e protected static de uma classe devem ser acessados através de funções membro public da classe ou através de friends da classe. Os membros static de uma classe existem mesmo quando não existem objetos daquela classe. Para acessar um membro public static de uma classe, quando não existem objetos daquela classe, simplesmente prefixe o
Lrticu- nome do membro de dados da classe com o nome da classe e o operador binário de resolução de escopo *(:* **:).** Para acessar um membro de classe public ou private quando não existem objetos daquela classe, deve ser providen ciad uma função membro static e a função deve ser chamada prefixando-se o seu nome com o nome da classe e o operador de resolução de escopo binário.
- O programa da Fig. **7.9** demonstra o uso de um membro de dados private static e de uma função *dinô-* membro public static. O membro de dados count é inicializado com zero no escopo de arquivo com o *jetos* comando

```
int Employee: :count =0;
```

O membro de dados count mantém uma contagem do número de objetos da classe Employee que foram *e* instanciados. Quando existem objetos da classe Employee, o membro count pode ser referenciado por qualquer
função membro de um objeto Employee - neste exemplo, count é referenciado tanto pelo construtor como pelo
destruidor.
- *Erro comum de programação 7.10*
ipor É um erro de sintaxe incluir a palavra-chave static *na definição de uma variável de classe* static *em escopo de arquivo.*

Fig. 7.9 Usando um membro de dados **static** para manter uma contagem do número de objetos de uma classe - **employl** .h (parte 1 de 2).

osos ic é asse"
Ltic
lo de .cipelo D se stântudo mos paço mos das

1	// Fig. 7.9: employl.h		
2	// Uma classe Employee		
3	#ifndef **EMPLOY1H**		
4	**#define EMPLOY1H**		
5			
6	**class Employee** {		
7	public:		
8	Employee(const char*, const char*)	//	construtor
9	**-EmployeeO;**	//	**destruidor**
10	const char *getFirstNe() const;	//	retorna o primeiro nome
11	const char *getTastNaxp.e() const;	//	retorna o último nome
12			
13	// função membro static		


```
14 static int getCount(); // retorna # de objetos instanciados
15
16 private:
17 char *fjrstName;
18 char *lastName;
19
20 // membro de dados static
21 static int count; // número de objetos instanciados
22 };
23
24 #endif
```

Fig. 7.9 Usando um membro de dados static para manter uma contagem do numero ae orjetos cie uma classe - employl .h (parte 2 de 2).

```
25 // Fig. 7.9: employl.cpp
26 1/ Definições de funções membro para a classe Employee
27 #include <iostream>
28
29 using std: :cout;
30 using std::endl;
31
32 #include <cstring>
33 #include <cassert>
34 #include "employl.h"
35
36 // Inicializa o membro de dados static
37 int Employee::count =0;
38
39 // Define a função membro static que retorna
40 II o número de objetos Employee instanciados.
41 int Employee::getCount() { return count;
```

```
42
43 II Construtor aloca dinamicamente o espaço para
44 // o primeiro e o último nomes e usa strcpy para copiar
45 // o primeiro e o último nomes para o objeto
46 Employee: :Employee( const char *fjrst, const char *last
47 {
48 firstName =new char[ strlen( first ) + 1 ];
49 assert( firstName !=o ) ; 1/ assegura-se de que a memória foi alocada
50 strcpy( firstName, first );
51
52 lastName = new char[ strlen( last ) ÷ 1 ];
53 assert( lastName != O ); // assegura-se de que a memória foi alocada
54 strcpy( lastName, last );
55
56 ++count; // incrementa contador de empregados static
57 cout « Construtor para Employee « firstName
58 « ' « lastName « ' chamado." « endl;
59 }
60
61 II Destruidor desaloca a memória alocada dinamicamente
62 Employee: :-.Employee()
63 {
64 cout« "-.Employee() chamado para ' « firstName
65 « ' « lastName « endl;
```

Fig. 7.9 Usando um membro de dados static para manter uma contagem do número de objetos de uma classe - employl .cpp (parte 1 de 2).


```
66 delete [] firstName; // recupera a memória
67 delete [) lastName; // recupera a memória
68 --count; // decrementa contador de empregados static
69
70
71 // Retorna primeiro nome de Employee
72 const char *Employee::getFirstName() const
73
74 // Const antes do tipo de retorno evita que o cliente modifique
75 // dados private. Os clientes devem copiar o string retornado antes que o
76 // destruidor delete a memória para evitar um ponteiro indefinido.
77 return firstName;
78
79
80 // Retorna último nome de Employee
81 const char *Employee::getLastName() const
82
83 // Const antes do tipo de retorno evita que o cliente modifique
84 // dados private. Os clientes devem copiar o string retornado antes que o
85 // destruidor delete a memória para evitar um ponteiro indefinido.
86 return lastName;
87
```

Fig. 7.9 Usando um membro de dados static para manter uma contagem do número de objetos de uma classe - employl .cpp (parte 2 de 2).

```
88 // Fig. 7.9: figO7O9.cpp
89 // Programa para testar a classe Employee
90 #include <iostream>
91
92 using std::cout;
93 using std:endl;
94
95 #include "employl.h"
96
```

```
97 int main()
98
99 cout « Número de Employees antes da instanciação é
100 « Employee::getCountO) « endl; // usa nome da classe
101
102 Employee *elPtr =new Employee( “Simone’, “Bianchi”
103 Employee *e2ptr =new Employee( “Roberto”, ‘Schmidt” );
104
105 cout « “Número de Employees após a instanciação é
106 « elPtr->getCountO;
107
108 cout« ‘\n\nEmployee 1:
109 « e1Ptr->getFirstName
110 « -« elPtr->getLastName()
111 « “\nEmployee 2:
112 « e2Ptr->getFirstName()
113 -« e2Ptr->getLastName() « “\n\n”;
114
115 delete elPtr; // recupera a memória
116 elPtr =0;
117 delete e2Ptr; // recupera a memória
118 e2Ptr =0;
```

Fig. 7.9 Usando um membro de dados static para manter uma contagem do número de objetos de uma classe - figo7 09. cpp (parte 1 de 2).


```
119 As
120 cout « ‘Número de Employees após deleção é -Em]
121 « Employee::getCount() « endi; dei
122
123 return O;
124)
Número de Employees antes da instanciação é O
Construtor para Employee Simone Bianchi chamado.
Construtor para Entployee Roberto Schmidt chamado.
Número de Employees após a instanciação é 2 No
Employee 1: Simone Bianchi ca
Employee 2: Roberto Schmidt um
-Employee() chamado para Simone Bianchi pai
-Emp1oyee() chamado para Roberto Schmidt um
Número de Employees após deleção é O de
```

Fig. 7.9 Usando um membro de dados static para manter uma contagem do número de objetos de me uma classe - figO7 09. cpp (parte 2 de 2).

Quando não existem objetos da classe Employee, o membro count ainda pode ser referenciado, mas somente tes
através de uma chamada para a função membro static getCount, como segue:

```
Employee: :getCount()
```

Nesse exemplo, a função getCount é usada para determinar o número de objetos de Employee atualmente err instanciados. Note que, quando não existir nenhum objeto instanciado no programa, é usada a chamada à função ex Employee: : getCount O. Porém, quando existirem objetos instanciados, a função getCount pode ser chamada através de um dos objetos, como mostrado no comando nas linhas 105 e 106

```
cout « “Número de Employees após a instanciação é
« elPtr->getCountO);
ei
```

Note que as chamadas e2Ptr->getCount O e Employee: : getCount () produzem o mesmo resultado. coi

Observação de engenharia de software 7.13 m *___Algumas organizações têm em seus padrões de engenharia de software a norma de que todas as chamadas afunções membro* **static** *sejam feitas usando o nome da classe e não o* handle *do objeto.* ou dei

Uma função membro pode ser declarada static se ela não acessa membros de dados e funções membro não- ge
static da classe. Diferentemente de funções membros não-static, uma função membro static não tem um ari

ponteiro this porque membros de dados static e funções membros static existem independentemente de foi quaisquer objetos de uma classe, de
Erro comum de programação 7.11
Referir-se ao ponteiro **this** *dentro de uma função membro* **static** *é um erro de sintaxe.* **7.**
Erro comum de programação 7.12 cl
Declarar uma função membro **static** *como* **const** *é um erro de sintaxe, de*
pil
Observação de engenharia de soflware 7.14
___ *Os membros de dados* static *e as funções membro* static *de uma classe existem e podem ser usados ainda que nenhum objeto daquela classe tenha sido instanciado.* sai

•e-LLs) soiuqn p oapnd oqqi,, ‘o
OIniid3 ou opd Doqq!q P PS SSID sowaitpnis ,,so1uqB,, ‘J oIn1Jd3 ou (iq1id) ps ssp rndçud issou sowiu so5!Ms sns soiisqi sopip p sodji 1uwsi3Jd swm jniumjdwi i.rnd sssp sn opiwioid o ‘++D Wj IpTUIJpUJ OA!WU OJiUJ Um p pupnnb zini iin.iix p o3ido i sw ‘sowp os sopp p sonjdo sfn, ‘SoAUuu soJui p o5ou oidwx on u3tÍqns JopndwoD op JAflh1d tp oxij oqunwii o owo situ iuinbui ip soiwjd PAJSUS flJ!uuj p ‘ouod ou w ‘sprnnx ops Sflp!TTUUd so5ijdo SWT piUJJpUT ojz iod O1STAJp sm ‘oInp9uJ O1STATp ‘OP5fl3JId!flflUl ‘o5tiqns ‘o5!p P so5ido uijp ++3 u uç p o5ou n ‘o1dwx JOd SOj31p SSS qos spiitwd os nb *SdQ5*

-vJ?do sn *sopvp p ovvjusaida,* umn :tfs no ‘soou snp nqou iuuq oiuisqi sopnp p odu wn JopulndwoD wn W ‘osT3Jd p ouçwjsis 1A;U wnp opunw
op sQou p sopow os sp ‘niuss w soiusq sopp p sodn p sojdwx sopoi os sojno 1t.p eqnop uT 0W03 sodii JP!SU03 n flAiJDdSJd ou mn w DOA ‘flJO *:SJI%J* souwnuonsnb ws u-r SouJiD ‘inb y .I1aI opunw op sowuwjJodwo3 SOPDUO3 p ‘SOppoUT no ‘SQ5lWiXOJdU 9S OS 1UW

-IPaI ++J OwOD o5wnoid p sunnuj od sopiiuoj soAjliwud sopup p sodj so nb oisnb V
•saJ1DnJD p snqJm nb ‘suap o suodif

o omo s!lfl sunnuq niud opnbpnu uwioi osj sujip sinun p SIOfA J 3UJO siq s p iijsv saiPi p ounFuoD O opunw ou p wn Ç sopiudwoD sopiuuj uisq
os ‘sJop1ndwoD sop in.ioTw u ‘xtp odu op SaIOI1A 1UTp od uussi (ç) oip wn ‘($) Jjçp p tun ‘oInsnujw z um ‘oInDsnww wn omo ‘wniusjdj sp nb sw.rnc so wo npnu w w3arnd s oçu sgpd s1s spq oio wo soz sun p sgipud os nuwjuwiou iwt.p SIO{1A owixoid mim wt.p p'
•e-[qnop tuid oJ!p1pJA owsw .pai opuntu op oiiui wn p oou up
oi5ntuixojd iwn os iuwi sJopindwoD tu *WÇ mil p* on5ou n ‘wiss’ iwjqoid w ou sonwiuw
SOJT3U! SJ •0IaUODUT opninsi 11111 iiznpojd p pipijiqissod i opuinput ‘iujnbw p upudp
opotu wnp p puods I?u!nb?w *(Mofiaio)* oinois p wn ‘oiiui sp oj n ojnc wn

p OPtfflSJ o S sgqiq + I sQqIq .0IAJ1U! o oprn!w!I is pod siiq p iuinbpw mim
w ;u ‘oidWx JOd tU SOpflitUtj utnsiq iuwjmiuou os sJopnrndmoD w su-ç ‘njn3iund w
•3!Ww11w w O.llTU! Wfl nb o uwtinx ou Jopndwo3 tun ui smii ‘t3fltuWw w wiiui oJwpu wn p op5ou uui ç UJA nb O •;u OATITtUUd odij o pisuo ojuijsq opnp p odn wn nb O
sïwmojd p O1UtU!AIOAUSp p ossod
o iuioqpw pod nb 1uoT3!p OP5flZ!IflWJoJ mn wuioj ‘usqo ON npmJnnJ1s tunuisqns
ou •stpmp sirnp swil s uiunp ipmunnis onwioid n ounnb mp m Íoq o5uw miuri

siUv (sIuv) sopisq sopp p sodt r ousou i ipmpisuo sou od omuzquwoj V sopip iqos so
-ou snwni mz!I1imuJoJ nosiDaid on5iwtuojd p sumnuiI P pupiunmo n ‘OJip BSu .tUqU!UJU3
sodj spnbip

sopíqo iu sg5iui s (sssp ‘i) sopip p sodi sOAOu niJc ++J w mJwud pnp!AJin V sopp sop mDu1JOdUJi n nutunn ++3 soÍqo u opniuuo p opjs o ++ woc tpnw oS!A iss
SOAOU sopp p sodij soudçud sns ucm saopw
-uojd so iuid 1!’J!P soAfliwud sopp 3p sodii sunii 95 w31s1x] SflJ3,, oS sopip 50 *S3Q0* su nb op suus
-s.I3uJ souw,, opus OwoD so)siA os sopnp so ‘opow J3nb1unb j 1innx3 wusjoad sumuioid so 3nb so3n su jiwodns

u.rud WSIX sopup so suunui *SflSS3* sou uziuJuo o5numu.mojd 3p suunuq snp uuoiuw V
n oSJA uuin uu3s31dn ++3 t113 soíqo n upuu3uo o5nwniod

V o5uwn.mojd 3p suunui suiuni wsix inb od ozmu nmn uss .uin oipud OSIA !u9 uwn 31s1x3 ON waiusn siopuwuioid so niud 3111TU3AUO3 OSIA utun ITIIJD J3AU Ofli 3p WunUII uuin 3j3 Ounj V
pnw ou ssup upnbup u3Jqpd
3uJJ31ui nb psp ‘uwisis op osam o iujn umas upniisqns no upu.m3Tu .is 3pod *(npvina* 3 ***Op5d2su*** 3 *S3Q5*

-ul3do suns uwn uTu3woldwT 3nb utun owo Iu) .mujnDTiud ssp mim p o5uu3w31dwT u 3nb u3!J!11!s osi sqpnp ssp upu3d3p 3nb o!pç)3 l3AaIDs3 W3A3 Oi!U sounsn so ‘upuu3w3jdwi 3SS uwn 011103 3 squip so J3UOD 1313nb uiussod soijnnsn 50 uioqtu *(sadj rnvu pvJJsq’I* . sjuv) *sowisqo sopvp ap soda* sopuulwou3p sopup p sodii w3up ++D w sssup 3 *‘sopvp 2p op5vjsqv p* opumuq o3u1u3w3IdwJ
uns p 3u3w31uopu3d3pui 3ssuj3 nwn 3 3pupquUO!DuflJ u J3A3JDS3Q nus u oJi3umud ‘.ImIu3 o oJi3wud w3plo ou sopnuzowio woÍ3s sopop 3p su3li so ou sopop p suii sopnoio waioj opuunb ‘3nb J3nb

-u3ws31dwls 3u3ip **O** opou3w31dwi uII1!d o otuo iqns olTss3D3u ou nq1id ssop otun p 31U31P w (sopop p suannjs,, ‘çj olnlldoD J3A SOO3fl3U3 suisij no) suuo wo suporn3w3ldw! ms 31u3m1130J w3pod soqid s’i ows3w op

o5ow3w3Idtu! O S3TU3!13 SII3S Op J3U0DS3 oqjid 35SO nwn 10113 3pod opnwnoid **O**
IVNV1DOdd O14O3 ++3 **OXt’**

7.8.1 Exemplo: tipo de dado abstrato array

nen- Discutimos arrays no Capítulo 4. Um array não é muito mais que um ponteiro e algum espaço na memória. Este dem recurso primitivo para executar operações com arrays aceitável se o programador for cauteloso e pouco exigente. sua Existem muitas operações que seriam interessantes de se executar com arrays, mas que não estão disponíveis em *ipos* C++. Com as classes de C++, o programador pode desenvolver um ADT array que seja melhor do que “arrays crus”. s de A classe array pode fornecer muitos recursos novos e úteis, tais como
era- . Verificação de intervalo de validade de subscritos.
face
. Um intervalo arbitrário de subscritos em vez de ter que começar com O.
Não ..
o A . Atribuiçao de arrays.
. Comparação de arrays.
)rtar
res- . Operações de entradalsaída para arrays.
gra- . Arrays que conhecem seus tamanhos.
dos . Arrays que se expandem dinamicamente para acomodar mais elementos.
etos
Criamos nossa própria classe array no Capítulo 8, “Sobrecarga de operadores”, e estudamos a classe equivalente da ; no- biblioteca padrão (chamada vector) no Capítulo 20.
bem C++ tem um conjunto pequeno de tipos primitivos. As classes estendem a linguagem de programação básica. não
ar o *Observação de engenharia de software 7. 15*
um ___ *O programador pode criar novos tipos através do mecanismo de classes. Esses novos tipos podem ser* tica *projetados para serem usados tão convenientemente quanto os tipos primitivos. Deste modo, C+ + é uma* : em linguagem extensível. *Embora a linguagem seja fácil de estender com estes novos tipos, a linguagem* o de *básica propriamente dita não é mutável.*
iodo
irs Novas classes criadas em ambientes C++ podem ser propriedades individuais, de grupos pequenos ou de empre ça sas. As classes também podem ser colocadas em bibliotecas de classe padrão, dirigidas para uma distribuição em larga escala. Isto não promove padrões necessariamente, embora de fato estejam surgindo padrões . O verdadeiro stes valor de C++ pode ser percebido somente quando são usadas bibliotecas de classe substanciais e padronizadas ulo, para desenvolver novas aplicações. O ANSI (American National Standards Institute) e a ISO (Intemational Standards
são Organization) desenvolveram uma versão padrão de C++ que inclui uma biblioteca de classes padrão. O leitor bits que aprende C++ e a programação orientada a objetos estará preparado para tirar proveito dos novos tipos de 110 **o** desenvolvimento de software rápido e orientado a componentes, tornados possíveis com bibliotecas cada vez mais abundantes e ricas.
eal int
har

do 7.8.2 Exemplo: tipo de dado abstrato *string*

era- C++ é uma linguagem intencionalmente sucinta, que fornece aos programadores somente os recursos crus necesção, sários para construir uma ampla gama de sistemas (considere-a uma ferramenta para fabricar ferramentas). A são linguagem foi projetada para minimizar sobrecargas que comprometem o desempenho. C++ é apropriada tanto a do para a programação de aplicativos como para a programação de sistemas . esta última impõe aos programas idos exigências extraordinárias de desempenho. Certamente, teria sido possível incluir um tipo de dado primitivo ador *string* entre os tipos primitivos oferecidos por C++. Em vez disso, a linguagem foi projetada para incluir mecanispria mos para criar e implementar tipos de dados abstratos *string* através de classes. Desenvolveremos nosso próprio tulo ADT

string no Capítulo 8. O padrão ANSI/ISO inclui uma classe string que discutiremos em detalhes no Capítulo 19.

Implementar uma classe *proxy* exige vários passos (Fig. 7. 10). Primeiro, criamos a definição da classe e os

arquivos de implementação para a classe cujos dados private queremos esconder. Nossa classe exemplo, que
queue. chamaremos de Implementation. é mostrada na Fig. 7. 10, linhas 1 a 12. A classeproxy Interface é mostra-
la para da na Fig. 7. 10, linhas 1 3 a 41 , e o programa de teste e sua saída são mostrados na Fig. 7. 1 O, linhas 42 a 61.
e todos A classe de Implementation fornece um único membro de dados private. chamado value (este é o)utador dado que queremos esconder do cliente), um construtor para inicializar value e as funções setValue e getValue.
o que

```
ompre- 1 II Fig. 7.10: implementation.h
-nafila 2 II Arquivo de cabeçalho para a classe Implementation
retirar
- 4 class Implementation {
ns sao
5 public:
iliaeo 6 Implementation( int v ) { value =
7 void setValue( int v ) { value =
ns que 8 int getValue() const { return value; }
na fila 9
querem 10 private:
aquele 11 int value;
primei- 12 };
```

ntação **Fig. 7.10** Implemeritando uma classe *proxy*- `implementation.h`

```
esteve
lar esta 13 // Fig. 7.10: interface.h
clientes 14 // Arquivo de cabeçalho para interface.cpp
erações 15 class Implementation; // declaração antecipada da classe
nificar 16
17 class Interface
10. 18 public:
oCapi- 19 Interface( int );
20 void setValue( int ); // mesma interface pública que a
21 int getValue() const; // classe Implementation
22 -InterfaceO;
23 private:
24 Implementation *ptr; // requer a declaração
25 // antecipada acima
õesde 26 };
```

Offio **Fig. 7.10** Implementando uma classe *proxy*- **interface.h**

```
pítulo 4
27 II Fig. 7.10: interface.cpp
ier.Um 28 // Definição da classe Interface
itemde 29 #include "interface.h
sepode 30 #include "implementation.h"
pessoas 31
)erando 32 Interface::Interface( int v
têineres 33 : ptr ( new Implementation( v ) ) { }
35 // chama a função setValue de Implementation
36 void Interface::setValue( int v ) { ptr->setValue( v );
```

```
37
38 // chama a função getValue de Implementation
39 int Interface::getValue() const { return ptr->getValue(); }
40
ietánas 41 Interface: :Interface() { delete ptr; }
a classe
sda sua Fig. 7.10 Implementando uma classe proxy- interface.cpp
```


Criamos uma definição de classe *proxy* com uma interface public idêntica à da classe Implementation. O único membro private da classe *proxy* é um ponteiro para um objeto da classe Implementation. Usar um ponteiro desta maneira permite que nós escondamos do cliente os detalhes de implementação da classe **Implementation.**

```
42 /1 Fig. 7.10: figO7lO.cpp
43 II Ocultando os dados private de urna classe com uma classe proxy.
44 #include <iostream>
45
46 using std::cout;
47 using std::endl;
48
49 #include "interface.h"
50
51 int main O
52 {
53 Interface i ( 5 );
54
55 cout« "Interface contém: « i.getValue()
56 « o antes de setValue « endl;
57 i.setValue( 10 );
58 cout« "Interface contém: « i.getValue()
59 « após setValue" « endl;
60 return 0;
61 }
Interface contém: 5 antes de setValue
Interface contém: 10 após setValue
```

Fig. 7.10 Implementando uma classe *proxy*- figo7_l0 .cpp.

A classe Interface. na parte 2 da Fig.7.lO, é a classeproxy para a classe Implementation. Note que a única menção à classe proprietária Impleinentation na classe Interface é na declaração de ponteiro (linha 24). Quando uma definição de classe (tal como a da classe Interface) usa só um ponteiro para outra classe (tal como para a classe Implementation), o arquivo de cabeçalho de classe para aquela outra classe (que normalmente revelaria os dados private daquela classe) não precisa ser obrigatoriamente incluído com #include. Você pode simplesmente declarar aquela outra classe como um tipo de dados, usando uma *declaração de classe antecipada* - *f orward* (linha *15),* antes de o tipo ser usado no arquivo.

O arquivo de implementação contendo as funções membro para a classeproxv Interface (Fig.7.lO, parte 3) é o único arquivo que inclui o arquivo de cabeçalho implementation h, que contém a classe **Implementation.** O arquivo interface cpp (Fig. 7.10, parte 3) é fornecido ao cliente como um arquivo objeto pré-compilado, junto com o arquivo de cabeçalho interface h, que inclui os protótipos de funções dos serviços fornecidos pela classe *proxy.* Como o arquivo interface cpp toma-se disponível para o cliente somente como o código objeto compilado, o cliente não pode ver as interações entre a classe *proxy* e a classe proprietária.

O programa na parte 4 da Fig. 7.10 testa a classe Interface. Note que só o arquivo de cabeçalho para a classe Interface é incluído em main - não existe nenhuma menção à existência de uma classe separada denominada Implementation. Deste modo, o cliente nunca vê os dados private da classe Implementation, nem o código cliente pode se tornar dependente do código de Implementation.

7.11 (Estudo de caso opcional) Pensando em objetos: programando as classes para o simulador de elevador

Nas seções "Pensando em objetos", nos finais dos Capítulos 2 a *5,* projetamos nosso simulador de elevador e, no Capítulo 6, começamos a programar o simulador em C++. No corpo do Capítulo 7, discutimos os recursos restantes

de C++ de que necessitamos para implementar um simulador de elevador completo e que funcione. Discutimos técnicas de administração dinâmica de objetos, usando new e delete para criar e destruir objetos, respectivamente. Também discutimos a composição, um recurso que nos permite criar classes que contêm objetos de outras classes como membros de dados. A composição nos habilita a criar uma classe Building que contém um objeto Scheduler, um objeto Clock. um objeto Elevator e dois objetos Floor; uma classe Elevator que contem um objeto de cada uma das classes ElevatorButton, Door e Be].1; e uma classe Floor que contém objetos FloorButton e Light. Também discutimos como usar membros de classes static, membros de classes const e a sintaxe de inicialização de membros em construtores. Nesta seção, continuamos a implementar nosso sistema de elevador em C++ usando estas técnicas. No fim desta seção, apresentamos um programa simulador de elevador completo em C++ (aproximadamente 1000 linhas de código) e um *walkthrough* detalhado do código. Na seção "Pensando em objetos" no fim do Capítulo 9, completamos nosso estudo de caso do simulador de elevador incorporando herança ao simulador de elevador; naquele ponto, apresentamos somente o código C++ adicional necessário para implementarmos a herança.

Uma i'isão geral da implementação da simulação de elevador

Nossa simulação de elevador é controlada por um objeto da classe Building, que contém dois objetos da classe Floor e um objeto de cada uma das classes Elevator, Clock e Scheduler. Este relacionamento composto foi mostrado no diagrama de classes da UML (Fig. 2.44) apresentado na seção "Pensando em objetos" no fim do Capítulo 2. O relógio simplesmente registra a hora atual em segundos e é incrementado uma vez por segundo pelo prédio. O *scheduler* é responsável por escalonar a chegada de pessoas a cada andar.

Após cada tiquetaqueada do relógio, o prédio atualiza o *scheduler* com a hora atual (através da função membro processTime da classe Scheduler). O *scheduler* compara esta hora com a próxima hora de chega- da escalonada para pessoas em cada andar. Se está escalonada a chegada de uma pessoa a um andar, o *scheduler* verifica se o andar está ou não ocupado chamando a função membro isOccupied da classe Floor. Se esta chamada retorna true. existe uma pessoa no andar naquele momento, de modo que o *scheduler* invoca sua função delayArrival para retardar em um segundo a próxima hora de chegada de uma pessoa para aquele andar.

Se o andar está vazio (i.e., a chamada retorna false), o *scheduler* cria um novo objeto da classe Person e aquela pessoa entra no andar apropriado. A pessoa então invoca a função membro pressButton da classe FloorButton. O botão do andar, por sua vez, invoca o método suninionElevator da classe Elevator.

O prédio também atualiza o elevador com a hora atual em segundos após cada tiquetaqueada do relógio. Quando recebe a hora atualizada, o elevador primeiro verifica seu estado atual (se está "se movendo" ou "para- do"). Se o elevador está se movimentando entre os andares, mas não está escalonado para chegar em um andar .flo naquela hora, o elevador simplesmente exibe na tela a direção em que está se movendo. Se o elevador está se ne movendo e a hora atual coincide com a próxima hora de chegada escalonada, o elevador pára, desliga o botão do e elevador, toca a campainha e avisa ao andar que chegou (através da função membro elevatorArrived da *a* classe Floor). Em resposta, o andar desliga o botão de andar e liga a luz. O elevador então abre a porta, o que rte permite que a pessoa no elevador saia e a pessoa no andar entre. O elevador então fecha a porta e determina se o outro andar necessita de atendimento. Se o outro andar precisa de atendimento, o elevador começa a se mover sse para aquele andar.

vo Se o elevador não está se movendo quando recebe a hora atualizada do prédio, o elevador determina qual

- andar precisa ser atendido. Se o andar atual precisa ser atendido (i.e., uma pessoa pressionou o botão no andar atual em que o elevador está), o elevador toca a campainha, avisa o andar que o elevador chegou e abre a porta .A pessoa na. que está no andar entra no elevador e aperta o botão para começar a movimentar o elevador para o outro andar. Se o a a outro andar necessita de atendimento (i.e., uma pessoa apertou o botão no outro andar), o elevador começa a se mover para aquele andar.

A implementação da simulação de elevador

Nas seções "Pensando em objetos" anteriores, reunimos muitas informações sobre nosso sistema. Usamos estas informações para criar um projeto orientado a objetos para nossa simulação de elevador e representamos este projeto usando a UML. Agora, já discutimos toda a tecnologia de programação orientada a objetos em C++ que é necessária para implementar uma simulação que funcione. O restante desta seção contém nossa implementação em C++ e um *walkthrough* detalhado do código.

fies

Nosso programa acionador (Fig. 7. 1 1) inicialmente pede ao usuário que digite o período de tempo para o qual

a simulação deve ser executada (linhas 15 e 16). A chamada para cm .ignore na linha 17 instrui o *stream* cm a ignorar o caractere de retorno do carro *(return)* que o usuário digitar após o inteiro durante a execução. Isto remove o caractere *return* do *stream* de entrada. O acionador cria, então, o objeto bui lding (linha 1 9) e invoca sua função membro runSimulation, passando como parâmetro a duração especificada pelo usuário (linha 23). O acionador também exibe na tela mensagens indicando ao usuário quando a simulação inicia (linha 21) e termina (linha 24).

```
1 II Figure 7.11
2 II Acionador para a simulação
3 #include <iostream>
4
5 using std: :cout;
6 using std: :cin;
7
8 using std::endl;
9
10 #include "building.h"
11
12 int main()
13
14 int duration; II duração da simulação em segundos
15
16 cout « "Digite o tempo de simulação: ;
17 cm » duration;
18 cin.ignore(); // ignora caractere return
19 Building building; // cria o prédio
20
21 cout « endl « "INÍCIO DA SIMULAÇÃO DO ELEVADOR
22 « endl « endl;
23 building.runSimulation( duration ); // começa a simulação
24 cout « FIM DA SIMULAÇÃO DO ELEVADOR ;" « endl;
25
26 return 0;
27
```

Fig. 7.11 Acionador para a simulação do elevador.

De acordo com nosso diagrama de classes (Fig. 2.44), a classe Building é composta por objetos de diversas outras classes. O arquivo de cabeçalho de Building, apresentado na Fig. 7.12, reflete esta composição (linhas 46 a 50). A classe Building é composta de dois objetos Floor (floori e floor2), um objeto Elevator (elevator), um objeto Clock (clock) e um objeto Scheduler (scheduler).

```
28 // building.h
29 // Definição da classe Building.
30 #ifndef BUILDINGH
31 #define BUILDINGH
32
33 #include "elevator.h"
34 #include "floor.h"
35 #include clock.h"
36 #include "scheduler.h"
37
38 class Building {
```

Fig. 7.12 Arquivo de cabeçalho da classe **Building** (parte 1 **de** 2).


```
ual 39
na 40 public:
ve 41 Building() ; II construtor
io 42 -Building() ; II destruidor
dor 43 void runSimulation( int );// executa simulação pelo tempo especificado
44
). 45 private:
46 Floor floori; /1 objeto floori
47 Floor floor2; II objeto floor2
48 Elevator elevator; // objeto elevator
49 Clock clock; II objeto clock
```

```
50 Scheduler scheduler; 1/ objeto scheduler
51 };
52
53 #endif II BUILDINGH
```

Fig. 7.12 Arquivo de cabeçalho da classe Building (parte 2 de 2).

O arquivo de implementação para a classe Building é mostrado na Fig. 7. 13. O construtor está nas linhas 64 a 69. Na lista de inicialização de membros (linhas 65 a 68), são chamados construtores para muitos dos objetos dos quais a classe Building é composta, com os argumentos apropriados. FLOOR1 e FLOOR2 (mencionados nas linhas 65 e 66) são constantes definidas na classe Floor (linhas 821 e 822).

```
54 II building.cpp
55 // Definições de funções membro para a classe Building.
56 #include <iostream>
57
58 using std::cout;
59 using std::cin;
60 using std::endl;
61
62 #include 'building.h"
63
64 Building::Building() // construtor
65 : floorl( Floor::FLOOR1, elevator ),
66 floor2( Floor::FLOOR2, elevator ),
67 elevator( floori, floor2 ),
68 scheduler( floori, floor2
sas 69 { cout « "prédio criado" « endi;
46 70
or 71 Building::Building() // destruidor
72 { cout « "prédio destruido" « endl;
73
74 // controla a simulação
75 void Building: :runSimulation( int totalTime
76 {
77 int currentTime =0;
79 while ( currentTime < totalTime
80 clock.tickO;
81 currentTime =clock.getTime();
82 cout « "HORA: - « currentTime « endl;
83 scheduler.processTime( currentTime );
84 elevator.processTime( currentTime );
85 cin.getO; // pára a cada segundo para usuário ler informações
86 }
87 }
```

Fig. 7.13 Arquivo de implementação da classe Building.

A principal funcionalidade da classe Building está em sua função membro runSixnulation (linhas 87), que executa um laço até que a quantidade de tempo especificada tenha decorrido. Em cada iteraçã building instrui o clock a incrementar sua hora em um segundo, enviando a mensagem tick para c].c (linha 80). A seguir, o building obtém a hora de clock chamando a função membro getTime (linha 81 currentTime é então enviado através de mensagens processTiine para o scheduler e para o elevat nas linhas 83 e 84, respectivamente. Finalmente, adicionamos uma chamada a cm . get (linha 85) para qi. usuário suspenda temporariamente a rolagem dos dados de saída na tela, a fim de visualizar a saída da simula para o próximo segundo de tempo simulado, antes de pressionar a tecla *enter* para retomar a rolagem dos dado saída na tela.

Clock é uma classe simples que não é composta por qualquer outro objeto. O arquivo de cabeçalho pa classe Clock é mostrado na Fig. 7. 14 e sua implementação na Fig. 7. 1 5. Um objeto da classe Clock pode rece mensagens para incrementar time através da função membro tick, que está prototipada na linha 98 e implement nas linhas 1 22 e 123. A hora atual torna-se disponível para outros objetos através da função membro getTime. linhas 99, 125 e 126. Observe que getTime é const.

```
88 // clock.h
```

```
89 // Definição da classe Clock.
90 #ifndef CLOCKH
91 #define CLOCKH
92
93 class Clock
94
95 public:
96 ClockO; // construtor
97 '-ClockO; // destruidor
98 voidtick // incrementa relógio em um segundo
99 int getTime() const; // retorna hora atual do relógio
100
101 private:
102 int time; // hora do relógio
103
104
105 #endif // CLOCKH
```

Fig. 7.14 Arquivo de cabeçalho da classe Clock.

```
106 // clock.cpp
107 // Definição de funções membro para a classe Clock.
108 #include <iostream>
109
110 using std::cout;
111 using std: :endl;
112
113 #include "clock.h"
114
115 Clock::Clock() // construtor
116 : time( O
117 { cout « "relógio criado" « endl;
118
119 Clock: :-Clock() // destruidor
120 { cout « "relógio destruído" « endl; 1
121
122 void Clock::tick() // incrementa hora em 1
```

Fig. 7.15 Arquivo de implementação da classe Clock (parte 1 de 2).


```
74a 123 { time++; }
jo, o 124
ock 125 int Clock::getTirne() const // retorna hora atual
[].o 126 { return time;
tor
```

ue o **Fig. 7.15** Arquivo de implementação da classe Clock (parte 2 de 2).

ação

)s de A classe Scheduler (Fig. 7. 16) é responsável por criar objetos da classe Person em momentos gerados aleatoriamente e por colocar estes objetos nos andares apropriados. A interface public lista a função membro
ara a processTime, que recebe como seu argumento a hora atual (linha 1 39). O arquivo de cabeçalho também lista
eber diversas funções utilitárias private (que discutimos em seguida) que executam as tarefas requeridas pela função
itada membro processTime.
nas

```
127 // scheduler.h
128 // Definição da classe Scheduler
129 #ifndef SCHEDULERH
130 #define SCHEDULERH
```

```
131
132 class Floor; // declaração antecipada (forward)
133
134 class Scheduler {
135
136 public:
137 Scheduler( Floor &, Floor & ) ; // construtor
138 -Scheduler() ; // destruidor
139 void processTime( int ); // inicializa hora do scheduler
140
141 private:
142 // escalona chegada a um andar
143 void scheduleTime( const Floor & );
144
145 // retarda chegada em um andar
146 void delayTime( const Floor & );
147
148 // cria nova pessoa; coloca no andar
149 void createNewPerson( Floor & );
150
151 // processa chegada de uma pessoa a um andar
152 void handleArrivals( Floor &, int );
153
154 int currentClockTime;
155
156 Floor &floorlRef;
157 Floor &floor2Ref;
158
159 int floorlArrivalTime;
160 int floor2ArrivalTime;
161
162
163 #endif // SCHEDULERH
```

Fig. 7.16 Arquivo de cabeçalho da classe Scheduler.

A Fig. 7.17 lista o arquivo de implementação para a classe Scheduler. A função membro processTime (linhas
222 a 232) delega a maior parte de suas responsabilidades a funções utilitárias menores dentro da classe, O construtor da classe Scheduler (linhas 178 a 189) primeiro "semeia" o gerador de números pseudo-aleatórios com um

número baseado na hora atual do mundo real (linha 1 83). Isto faz com que o gerador de números aleatórios produza uma série diferente de números cada vez que o programa é executado. A classe Scheduler então chama a função utilitária scheduleTime (linhas 194 a 207) uma vez para cada um dos dois andares (linhas 187 e 188). Esta função membro calcula uma hora de chegada pseudo-aleatória (neste caso, um número aleatório no intervalo de 5 a 20, inclusive) para o primeiro objeto Person em cada andar.

```
164 // scheduler.cpp
165 // Definições de funções membro para a classe Scheduler.
166 #include <iostream>
167
168 using std: :cout;
169 using std::endl;
170
171 #include <cstdlib>
172 #include <ctime>
173
174 #include scheduler.h'
175 #include "floor.h"
176 #include "person.h"
177
178 // construtor
179 Scheduler::Scheduler( Floor &firstFloor, Floor &secondFloor
```

```
: currentClockTime( 0 ), floorlRef( firstFloor ),
floor2Ref( secondFloor
{
srand( time( 0 ) ); // semente p/gerador de números aleatórios
cout « scheduler criado" « endi;

II escalona primeiras chegadas para andar 1 e andar 2
scheduleTime ( floorlRef );
scheduleTime( floor2Ref );
}

Scheduler: :Scheduler() II destruidor
{ cout « "scheduler destruído" « endl;

II escalona chegada em um andar
void Scheduler::scheduleTime( const Floor &floor
{
int floorNuinber floor.getNumber();
int arrivalTime = currentClockTime + ( 5 + rand() % 16 );

floorNumber == Floor: :FLOOR1 ?
floorlArrivalTime = arrivalTime
floor2ArrivalTime = arrivalTime;

cout « "(scheduler escalona próxima pessoa para andar
« floorNuinber « -na hora -« arrivalTime « ')'
« endl;
}

// reescalona chegada em um andar
void Scheduler::delayTime( const Floor &floor
{
int floorNumber = floor.getNumberQ;
```

Fig. 7.17 Arquivo de implementação da classe Scheduler (parte 1 de 2).


```
luza 213
Ição 214 int arrivalTime ( floorNumber == Floor: :FLOOR1 ) ?
Esta 215 ++floorlArrivalTinie : ++floor2ArrivalTime;

cout « '(scheduler retarda próxima pessoa para andar
« floorNumber « até a hora -« arrivalTime « )
« endi;
1

II fornece a hora para o scheduler
void Scheduler::processTime( int time

currentClockTime = time; // record time

II processa chegadas no andar 1
handleArrivals( floorlRef, currentClockTime );

II processa chegadas no andar 2
handleArrivals( floor2Ref, currentClockTime );
}

II cria nova pessoa e a coloca no andar especificado
void Scheduler: :createNewPerson( Floor &floor
{
int destinationFloor =
floor.getNumber() Floor::FLOOR1 ?
```

```
239 Floor: :FLOOR2 : Floor: :FLOOR1;
240
241 II cria nova pessoa
242 Person *newpersonptr =new Person( destinationFloor )
243
244 cout « "scheduler cria pessoa
245 « newPersonPtr->getlD() « endl;
246
247 II coloca pessoa no andar adequado
248 newPersonPtr->stepOntoFloor( floor );
249
250 scheduleTime( floor ); 1/ escalona próxima chegada
251 }
252
253 II processa chegadas em um andar específico
254 void Scheduler::handleArrivals( Floor &floor, int time
255
256 int floorNumber =floor.getNunibero;
257
258 int arrivalTime =( floorNumber ==Floor: :FLOOR1 )?
259 floorlArrivalTime : floor2ArrivalTime;
260
261 if ( arrivalTime ==time
262
263 if ( floor.isOccupied() ) // verifica se andar está ocupado
264 delayTime( floor );
265 else
266 createNewPerson( floor
267 }
268
```

Fig. 7.17 Arquivo de implementação da classe Scheduler (parte 2 de 2).

Em nossa simu1aço, o building informa a hora atual ao scheduler a cada segundo, através da função mcm- bro de scheduler processTime (linhas 222 a 232). O diagrama de seqüência na Fig. 4.27 modelou a seqüência de atividades que ocorre em resposta a esta mensagem e nossa implementação reflete este modelo. Quando a função membro processTime é invocada, scheduler chama a função utilitária handleArrivals para cada andar (linhas 228 a 231). Esta função utilitária compara a hora atual (time, fornecida por building) com a próxima hora de chegada escalonada para aquele andar (linha 261). Se a hora atual coincide com a hora de chegada para o andar e se o andar está atualmente ocupado (linha 263), scheduler chama a função utilitária delayTime para retardar a próxima chegada escalonada em um segundo (linha 264). Se o andar está desocupado, o scheduler invoca a função utilitária createNewPerson (linha 266), que cria um novo objeto da classe Person usando o operador new (linha 242). O scheduler então envia a este novo objeto da classe Person a mensagem stepOntoFloor (linha 248). Assim que a pessoa tiver entrado no andar, o scheduler calcula a próxima hora de chegada de uma pessoa àquele andar chamando a função utilitária scheduleTime (linha 250). Examinamos a implementação de todas as classes que compõem a parte controladora da simulação; agora, examinemos as classes que compõem a parte "mundo" da simulação. A classe Beil, como a classe Clock, não é composta por outros objetos. A interface public da classe Beli, como definida em seu arquivo de cabeçalho na Fig. 7. 18, consiste em um construtor, um destruidor e a função membro ringBell. As implementações destas funções (linhas 292 e 293, 295 e 296, e 298 e 299, respectivamente, na Fig. 7. 1 9) simplesmente enviam mensagens para a tela.

```
269 II bell.h
270 II Definição da classe Beli.
271 #ifndef BELLH
272 #define BELLH
273
274 class Beil {
275
276 public:
277 BeilO; II construtor
278 Bell 1/ destruidor
279 void ringBell() const; II toca a campainha
280
281
```

```
282 #endif II BELLH
```
Fig. 7.18 Arquivo de cabeçalho da classe Beli.
```
283 II bell.cpp
284 II Definições de funções membro para a classe Beil.
285 #include <iostreani>
286
287 using std::cout;
288 using std::endl;
289
290 #include bell.h"
291
292 Bell::Bell() II construtor
293 { cout « 'campainha criada" « endl;
294
295 Bell::Bell() // destruidor
296 { cout « "campainha destruída" « endi;
297
298 void Bell::ringBell() const II toca campainha
299 { cout « "elevador toca a campainha" « endl;
```
Fig. 7.19 Arquivo de implementação da classe Beli.

1- A classe Light (Figs. 7.20 e 7.21) expõe duas funções membro em sua interface public. além do construtor e do n- destruidor. A função membro turnOn simplesmente liga a luz, ajustando o membro de dados on para true (linhas
) a 335 a 339). A função membro turnOff (linhas 341 a 345) desliga a luz, ajustando o membro de dados on para
da false.
a
da
```
e 300 II light.h
r 301 II Definição da classe Light.

▪
▪ 302 #ifndef
303 #define LIGHT H
m 304 -
ra 305 class Light {
306
ra, 307 public:
)é 308 Light( const char *) ; // construtor
ig. 309 -Light() ; II destruidor
es 310 void turnOn() ; II liga a luz
Ia. 311 void turnOffO; II desliga a luz
312
313 private:
314 bool on; II true se ligada; false se desligada
315 const char *ne; /1 em que andar a luz está ligada
316 };
317
318 #endif II LIGHTH
```
Fig. 7.20 Arquivo de cabeçalho da classe Light.
```
319 II light.cpp
320 II Definições de funções membro para a classe Light.
```
321 #include <iostream>
```
322
323 using std: :cout;
324 using std: :endl;
325
326 #include 'light.h"
```

```
327
328 Light::Light( const char *string ) // construtor
329 : on( false ), name( string
330 { cout « name « " luz criada" « endl;
331
332 Light::-.Light() II destruidor
333 { cout « name « " luz destruída" « endl;
334
335 void Light::turnOn() II liga a luz
336
337 on = true;
338 cout « name « " liga a luz" « endl;
339 1
340
341 void Light::turnOff() II desliga a luz
342 {
343 on =false;
344 cout « name « "desliga a luz" « endl;
345 }
```

Fig. 7.21 Arquivo de implementação da classe Light.

A classe Door (Figs. 7.22 e 7.23) desempenha um papel importante em nossa simuIaço de elevador. É o objeto **door** que avisa ao passageiro do elevador para sair; door também avisa à pessoa que está esperando no andar que entre no elevador. Estas ações são executadas pela função membro openDoor da classe Door. Você vai notar que a função membro openDoor recebe quatro argumentos (linhas 361 a 362 e 390 a 392). O primeiro é um ponteiro para o objeto da classe Person que ocupa o elevador. O segundo argumento é um ponteiro para o objeto da classe Person que está esperando no andar. Os dois argumentos restantes são referências para o objeto da classe Floor apropriado e para o objeto elevator.

```
346 II door.h
347 II Definição da classe Door.
348 #ifndef DOORH
349 #define DOORH
350
351 class Person; // declaração antecipada
352 class Floor; // declaração antecipada
353 class Elevator; 1/ declaração antecipada
354
355 class Door {
356
357 public:
358 DoorO; II construtor
359 -.Dooro; II destruidor
360
361 void openDoor( Person *const, Person *const,
362 Floor &,Elevator & );
363 void closeDoor( const Floor & );
364
365 private:
366 bool open; 1/ aberta ou fechada
367
368
369 #endif II DOORH
```

Fig. 7.22 Arquivo de cabeçalho da classe Door.

A classe Door é um objeto composto da classe Elevator para implementar esta composição, o arquivo de cabeçalho da classe Elevator precisa conter a linha

```
#include "door .h"
```

A classe Door usa uma referência a um objeto da classe Elevator (linha 362). Para declarar a classe Elevator. de modo a permitir que a classe Door use esta referência, poderíamos colocar a seguinte linha no arquivo de cabeçalho da classe Door:

#include "elevator .h"

Assim, o arquivo de cabeçalho para a classe Elevator incluiria o arquivo de cabeçalho para a classe Door. e vice- versa. O pré-processador não seria capaz de resolver tais diretivas #include e geraria um erro fatal devido a este *problema de inclusão circular.*

Para evitar este problema, colocamos uma declaração antecipada *(forward)* da classe Elevator no arquivo de cabeçalho da classe Door (linha 353). Esta declaração antecipada diz ao pré-processador que queremos fazer referência a objetos da classe Elevator em nosso arquivo, mas que a definição da classe Elevator está fora do arquivo. Note que também fazemos declarações antecipadas para as classes Person e Floor (linhas 351 e *352),* de modo que podemos usar estas classes no protótipo para a função membro openDoor.

A Fig. 7.23 lista o arquivo de implementação para a classe Door. Nas linhas 378 a 380, incluímos os arquivos de cabeçalho para as classes Person. Floor e Elevator. Estas diretivas #include correspondem às nossas declarações antecipadas no cabeçalho; e os arquivos de cabeçalho que estão sendo incluídos contêm os protótipos de funções obrigatórios de que necessitamos para sermos capazes de invocar as funções membro apropriadas destas classes. Quando a funçào membro openDoor (linhas 389 a 409) é chamada, ela primeiro verifica se a portajá não está aberta. A porta verifica se o ponteiro para a pessoa no elevador (passengerPtr) não é zero (linha 400). Se este ponteiro não é zero, existe uma pessoa no elevador que precisa sair. A pessoa é avisada para sair do elevador através da mensagem exitElevator (linha 401 ***).*** A porta elimina o objeto da classe Person que estava andando no elevador através do operador delete (linha 402).

```
370 II door.cpp
371 II Definições de funções membro para a classe Door.
372 #include <iostream>
373
374 using std: :cout;
375 using std: :endl;
376
377 #include “door.h”
378 #include “person.h”
379 #include ‘floor.h”
380 #include “elevator.h”
381
382 Door: :Door() II construtor
383 : open( false
384 { cout « “porta criada” « endl;
385
386 Door::-Door() // destruidor
387 { cout « “porta destruída” « endi;
388
389 II abre a porta
390 void Door: :openDoor( Person *const passengerPtr,
391 Person *const nextPassengerPtr,
392 Floor &currentFloor, Elevator &elevator
393 {
394 if ( !open )
395 open = true;
396
397 cout « “elevador abre a porta no andar
398 « currentFloor.getNuiTlber() « endl;
399
400 if ( passengerPtr ! O ) {
401 passengerPtr->exitElevator( currentFloor, elevator )
402 delete passengerPtr; /1 passageiro sai da simulação
403 }
404
405 if ( nextPassengerPtr !O
406 nextPassengerPtr->enterElevator
407 elevator, currentFloor )
408 }
409 }
410
411 II fecha a porta
412 void Door::closeDoor( const Floor &currentFloor
413 {
```

Fig. 7.23 Arquivo de implementação da classe Door (parte 1 de 2).

to A Fig. 7.23 lista o arquivo de implementação para a classe Door. Nas linhas 378 a 380, incluímos os arquivos ue de cabeçalho para as classes Person, Floor e Elevator. Estas diretivas #include correspondem às nossas ue declarações antecipadas no cabeçalho; e os arquivos de cabeçalho que estão sendo incluídos contêm os protótipos de ro funções obrigatórios de que necessitamos para sermos capazes de invocar as funções membro apropriadas destas se classes.
)r Quando a função membro openDoor (linhas 389 a 409) é chamada, ela primeiro verifica se a portajá não está aberta. A porta verifica se o ponteiro para a pessoa no elevador (passengerPtr) não é zero (linha 400). Se este ponteiro não é zero, existe uma pessoa no elevador que precisa sair. A pessoa é avisada para sair do elevador através da mensagem exitElevator (linha 40 1 ***).*** A porta elimina o objeto da classe Person que estava andando no elevador através do operador delete (linha 402).

```
370 II door.cpp
371 II Definições de funções membro para a classe Door.
372 #include <iostream>
373
374 using std: :cout;
375 using std::endl;
376
377 #include 'door.h"
378 #include "person.h'
379 #include "floor.h"
380 #include "elevator.h"
381
382 Door::Door() II construtor
383 : open( false
384 { cout « "porta criada" « endi;
385
386 Door::Door() // destruidor
387 { cout « "porta destruída" « endl;
388
389 II abre a porta
390 void Door: :openDoor( Person *const passengerPtr,
391 Person *const nextPassengerPtr,
392 Floor &currentFloor, Elevator &elevator
393
394 if ( !open ) {
395 open = true;
396
397 cout « "elevador abre a porta no andar
398 « currentFloor.getNumber() « endl;
399
400 if ( passengerPtr != 0 ) {
401 passengerPtr->exitElevator( currentFloor, elevator )
402 delete passengerPtr; // passageiro sai da simulação
403 }
404
405 if ( nextPassengerPtr != 0
406 nextPassengerPtr->enterElevator
e- 407 elevator, currentFloor );
te 408 }
409 }
vo 410
er 411 II fecha a porta
io 412 void Door::closeDoor( const Floor &currentFloor )
), 413 {
```

Fig. 7.23 Arquivo de implementação da classe Door (parte 1 de 2).

1

```
414 if (open) {
415 open = false;
416 cout « elevador fecha a porta no andar
417 « currentFloor.getNuniber() « endi;
418 }
419 }
```

Fig. 7.23 Arquivo de implementação da classe Door (parte 2 de 2).

Assim que o passageiro sai do elevador, a porta verifica o ponteiro para a pessoa que está esperando no andar (nextPassengerPtr) para ver se aquele ponteiro não é zero (linha 405). Se o ponteiro não é zero (i.e., existe uma pessoa esperando para entrar no elevador), a pessoa é autorizada a entrar no elevador através da função membro enterElevator da classe Person (linhas 406 e 407). A função membro closeDoor (linhas 412 a 419) simplesmente verifica se a porta está aberta e, se estiver, a fecha.

Algumas pessoas no sistema usam um objeto da classe ElevatorButton (Figs. 7.24 e 7.25) para fazer o elevador começar a se mover para o outro andar. A função membro pressButton (linhas 460 a 466) primeiro ajusta o atributo pressed do botão do elevador para true e então envia a mensagem prepareToLeave para o elevator. A função membro resetButton simplesmente ajusta o atributo pressed para false.

```
420 II elevatorButton.h
421 II Definição da classe ElevatorButton.
422 #ifndef ELEVATORBUTTONH
423 #define ELEVATORBUTTONH
424
425 class Elevator; II declaração antecipada
426
427 class ElevatorButton {
428
429 public:
430 ElevatorButton( Elevator & ); II construtor
431 -'ElevatorButton() ; // destruidor
432
433 void pressButtonQ; II aperta o botão
434 void resetButtonO; II desliga o botão
435
436 private:
437 bool pressed; 1/ estado do botão
438 Elevator &elevatorRef; II referência para o elevador do botão
439
440
441 #endif II ELEVATORBUTTONH
```

Fig. 7.24 Arquivo de cabeçalho da classe ElevatorButton.

Fig. 7.25 Arquivo de implementação da classe ElevatorButton (parte 1 de 2).

442	II elevatorButton.cpp:	
443	II Definição de funções	membro para a classe ElevatorButton.
444	4include <iostream>	
445		
446	using std::cout;	
447	using std: :endl;	
448		
449	#include "elevatorButton.h"	
450	#include "elevator.h"	


```
451
452 II construtor
453 ElevatorButton: :ElevatorButton( Elevator &elevatorHandle
454 : pressed( false ), elevatorRef( elevatorHandle
455 { cout « "botão do elevador criado « endl;
```

```
456
457 ElevatorButton : : -ElevatorButton() II destruidor
458 { cout « "botão do elevador destruído" « endl;
459
460 void ElevatorButton::pressButton() II aperta o botão
461
462 pressed = true;
463 cout « botão do elevador avisa elevador que se prepare para sair'
464 « endl;
465 elevatorRef.prepareToLeave( true );
466 }
467
468 void ElevatorButton::resetButton() II desliga o botão
469 { pressed = false; }
```

Fig. 7.25 Arquivo de implementação da classe ElevatorButton (parte 2 de 2).

A classe FloorButton (Figs. 7.26 e 7.27), através de sua interface public. expõe as mesmas funções membro que a classe ElevatorButton. A função membro public pressButton chama o elevador através da mensagem summonElevator. O botão do andar é desligado através de uma chamada para a função membro `resetButton`.

```
470 II floorButton.h
471 II Definição da classe FloorButton.
472 #ifndef FLOORBUTTONH
473 #define FLOORBUTTONH
474
475 class Elevator; II declaração antecipada
476
477 class FloorButton
478
479 public:
480 FloorButton( const int, Elevator & ); II construtor
481 FloorButton II destruidor
482
483 void pressButton() ; // aperta o botão
484 void resetButton // desliga o botão
485
486 private:
487 const int floorNuinber; // número do andar do botão
488 bool pressed; II estado do botão
489
490 II referência ao elevador do botão
491 Elevator &elevatorRef;
492
493
494 #endif II FLOORBUTTONH
```

Fig. 7.26 Arquivo de cabeçalho da classe FloorButton.


```
495 II floorButton.cpp
496 II Definições de funções membro para a classe FloorButton.
497 #include <iostream>
498
499 using std::cout;
500 using std::endl;
501
502 #include "floorButton.h
503 #include "elevator.h"
504
505 II construtor
506 FloorButton: :FloorButton( const int numner,
507 Elevator &elevatorHandle
```

```
508 : floorNumber( nurnber ), pressed( false ),
509 elevatorRef( elevatorHandle
510
511 cout « "botão do andar -« floorNuniber « -criado"
512 « endl;
513 }
514
515 FloorButton : : FloorButton() // destruidor
516
517 cout « "botão do andar -« floorNumber « -destruído"
518 « endl;
519 )
520
521 II aperta o botão
522 void FloorButton: :pressButton()
523
524 pressed = true;
525 cout « "botão do andar -« floorNuniber
526 « -chama o elevador" « endl;
527 elevatorRef.suinmonElevator( floorNumber );
528 }
529
530 II desliga o botão
531 void FloorButton: :resetButton()
532 { pressed = false; )
```

Fig. 7.27 Arquivo de mplementação da classe FloorButton.

O arquivo de cabeçalho para a classe Elevator (Fig. 7.28) é o mais complexo em nossa simulação. A classe Elevator expõe cinco funções membro (além de seu construtor e destruidor) em sua interface public. A função membro processTime permite que o edifício envie a hora atualizada do relógio para o elevador. A função membro summonElevator permite que um objeto Person envie uma mensagem para o elevador para solicitar seu serviço. As funções membro passengerEnters e passengerExits habilitam os passageiros a entrar e sair do elevador e a função membro prepareToLeave habilita o elevador a executar quaisquer tarefas necessárias, antes de começar a se mover para um outro andar. Declaramos o objeto elevatorButton como public, de modo que um objeto da classe Person possa acessar diretamente o elevatorButton. Uma pessoa geralmente não interage com a campainha ou com a porta (a menos que aquela pessoa seja um técnico de manutenção do elevador). Portanto, declaramos os objetos beil e door na seção private da definição da classe.

Funções utilitárias são incluídas nas linhas 558 a 5ó1. A classe Elevator também define uma série de valores private static const (linhas 564 a *566).* Estes valores são declarados static porque contêm informação que é usada por todos os objetos da classe Elevator; estes valores não devem ser modificados nunca, de modo que eles também são declarados como const.

```
541


II elevator.h
II Definição da classe Elevator. #ifndef ELEVATORH
#define ELEVATORH
#include "elevatorButton.h"


#include "door.h" #include "bell .


542 class Floor;
543 class Person;


544


545 class Elevator
546
```

```
public:
Elevator( Floor &, Floor & );
-Elevator();
void surninonElevator ( int ); oid prepareToLeave ( bool ); void processTime( int ); void
passengerEnters( Person *void passengerExits();
ElevatorButton elevatorButton;
private:
void processPossibleArrival() void processPossibleDeparture(); void arriveAtFloor( Floor
& ); void move();


int currentBuildingClockTime; bool moving;
int direction;
int currentFloor;
int arrivalTime;
bool floorlNeedsService; bool floor2NeedsService;
Floor &floorlRef;
Floor &floor2Ref;
Person *passengerptr;
Door door;
Beil bell;


II declaração antecipada II declaração antecipada
II construtor
II destruidor
II pede atendimento para andar
II prepara para sair II fornece hora ao elevador


const ) ;// embarca um passageiro II sai um passageiro II note objeto public


II direção para cima II direção para baixo


II estado do elevador
II direção atual
II posição atual
II hora de chegar a um andar II indicador de atendimento de floori II indicador de
atendimento de floor2
II referência a floori
II referência a floor2 II ponteiro para passageiro atual
II objeto door
II objeto bell
```

Fig. 7.28 Arquivo de cabeçalho da classe Elevator.

As linhas *568* a *58* 1 do arquivo de cabeçalho de Elevator contêm membros de dados private adicionais. Note que para cada um dos objetos da classe Floor são fornecidos *handies* de referência (linhas 576 e *577),* enquanto

```
533
534
535
536
537
538
539
540
```

```
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
```

```
II hora de se mover entre andares static const int ELEVATOR TRAVEL TIME; static const
int UP;
static const int DOWN;
```

```
II hora atual
```

A
A
ra
ar

n
1.
m
la

```
#endif II ELEVATORH
```


um ponteiro *é* usado para o objeto passageiro (linha 578). Usamos um ponteiro para o passageiro porque este *handie* vai precisar mudar sempre que um objeto da classe Person entra ou sai do elevador. Preferimos *handies* de referência para os objetos Floor.

Usamos a UML para modelar muitas das atividades e colaborações associadas com a classe Elevator (ver Figs. 3.31, 3.32 e 5.37); nosso código para a classe Elevator (Fig. 7.29) implementa a informação contida nestes modelos. O construtor de Elevator tem uma extensa lista de inicializadores de membros (linhas 602 a 607). Você lembrará, de nossa definição da classe ElevatorButton (Fig. 7.24), que um objeto daquela classe requer um *handie* para um objeto da classe Elevator como um argumento para seu construtor. Providenciamos este *handie* em nossa lista de inicialização de membros derreferenciando o ponteiro this do objeto elevator (linha 602). Alguns compiladores geram um aviso nesta linha porque o objeto elevator ainda não foi completamente inicializado.

```
585 II elevator.cpp
586 II Definições de funções membro para a classe Elevator.
587 #include <iostream>
588
589 using std: :cout;
590 using std: :endl;
591
592 #include 'elevator.h
593 #include 'person.h"
594 #include "floor.h"
595
596 const int Elevator::ELEVATORTRAVELTINE =5;
597 const int Elevator::UP =O;
598 const int Elevator: :DOWN =1;
599
600 II construtor
601 Elevator: :Elevator( Floor &firstFloor, Floor &secondFloor
602 : elevatorButton( *this ) ,currentBuildingClockTime( O )
603 moving( false ) ,direction( UP ),
604 currentFloor( Floor::FLOOR1 ), arrivalTime( O ),
605 floorlNeedsService ( false ) , floor2NeedsService ( false ),
606 floorlRef( firstFloor ), floor2Ref( secondFloor ),
607 passengerPtr( O
608 { cout « elevador criado" « endi;
609
610 Elevator::-.Elevator() // destruidor
611 { cout « elevador destruído" « endi;
612
613 II fornece hora para o elevador
614 void Elevator::processTime( int time
615 {
616 currentBuildingClockTime =time;
617
618 if ( moving
619 processpossibleArrival O;
620 else
621 processPossibleDeparture();
622
623 if ( !moving
624 cout « "elevador parado no andar
625 « currentFloor « endi;
626 }
627
```

Fig. 7.29 Arquivo de implementação da classe Elevator (parte 1 de 4).


```
628 II quando elevador está se movendo, determina se ele deve parar
629 void Elevator: :processPossibleArrival()
630 {
631 II se elevador chega ao andar de destino
632 if ( currentBuildingClockTime arrivalTime )
633
634 currentFloor =/1 atualiza andar atual
635 ( currentFloor ==Floor: :FLOOR1 ?
```

```
636 Floor::FLOOR2 : Floor::FLOOR1 );
637
638 direction =II atualiza direção
639 ( currentFloor ==Floor: :FLOOR1 ? UP : DOWN );
640
641 cout « elevador chega no andar -
642 « currentFloor « endl;
643
644 arriveAtFloor( currentFloor ==Floor: :FLOOR1 ?
645 floorlRef : floor2Ref );
646
647 return;
648 }
649
650 II elevador está se movendo
651 cout « 'elevador se movendo
652 « ( direction ==UP ? para cima' : para baixo ) « endl;
653 )
654
655 II determina se o elevador deve se mover
656 void Elevator: :processPossibleDeparture()
657
658 II este andar precisa ser atendido?
659 bool currentFloorNeedsService
660 currentFloor ==Floor::FLOOR1 ?
661 floorlNeedsService : floor2NeedsService;
662
663 II outro andar precisa ser atendido?
664 bool otherFloorNeedsService =
665 currentFloor ==Floor::FLOOR1 ?
666 floor2NeedsService : floorlNeedsService;
667
668 II atende a este andar (se necessário)
669 if ( currentFloorNeedsService ){
670 arriveAtFloor( currentFloor Floor::FLOOR1 ?
671 floorlRef : floor2Ref );
672
673 return;
674 }
675
676 II atende a outro andar (se necessário)
677 else prepareToLeave( otherFloorNeedsService );
678 }
679
680 II chega em um andar específico
681 void Elevator: :arriveAtFloor( Floor& arrivaiFloor
682 {
683 moving =false; II desliga estado
684
685 cout « "elevador desliga o botão" « endl;
```

Fig. 7.29 Arquivo de implementação da classe Elevator (parte 2 de 4).

•(17 ep e.rnd) 2OAT essI3 p o5iuewepdwi ep OA!flbJV 6L 6!

```
opuTs SO epod op?AeTe o enb pu o stie //
eooi; : ;etoou Iooi::1ooT ==.2ooT.ueJnD IL
=1oo.t.s-çt. OL
} 6EL
( 5UTAe- ooq )eAe'o1e1€ded: :oAe p-ro 8EL
1?PU op JTS es-xdeid // LEL
9EL
{ O =I:4d1e5uessd ) pio ÇEL
opuTs se oitefessd o nb iope-a o s-r // TEL
```

{ *EL*
TPUe » OOTUflD)> *IEL*
« pu ou .xopAee ou wua „» *OEL*
Oaiteb<-2aJe5uessd » „ossed,, » noz *6L*
8 *L*
1duosed = d15uesswd *LL*
oi-r5ssd // *9L*
} *SL*
(juosad suo uosje ptoi *L*
oita5ssd um te // *EL*
{ *IL*
:anl; = eQTAJeSspeeM2oot; : en eDTIXeSSPeNT1OOT; *OL*
é Too'i.: :Joo == iooi; *6!L*
op-rxdod o;uuzpue ep ioptput eZÇUT // 8tL
} *LIL*
(too ut p-ro *91L*
:topaie OTad oueuxtpue TzTos // 91L
IL
{ *EIL*
esi = eDTAJeSSPeeN,IOOT :esTw = DTAIeSSP9NTOOI IL
. nioo1.::oot. == JOOT.Ue1flD TIL
o;uawTpUeE ep opiput -rsep 'otiuoo ose // eSte *OIL*
(eDTAeSSPeeN2OOTeT4O)aAeio1e1daJd *60L* (TAeSSPeeN1OOI.U9IJflDI) t *80L*
1pu o2no o -ç s-wdid // LOL
o:uuITpue ep stDeJd ou ese as // *90L*
OL
!9DTAJeSSP9eNI,2001 : tA1Sp9JOOf *OL*
'Ioo1::ooT. == 1OOTU11flQ *LOL*
= aDtAJeSSP9eNOOT2eT.O ooq OL o:uewTpue p stDazd ipu oi;no // TOL
OOL
!eDtAIeSSPeN1OOTJ : 9YtAeSSPeaNI2OOTJ 669
G IOO'I. :JooT == 1ooT.ue11nD 869
= 9DtAJeSSp9eNIOOIUe11fl T°°q L69
ouemTpue: ep swed pu e;se // 969
969
(s-rt '1OOITAtI t69
'duosed2ooTJ '.x;dJa5uassd)oouedooop E69
6 9
! IdUOS19dJOOI uosled 169
no5eT4D JOPA91e O 9flb ptrn o STA9 // 069
689
! rre5u.x"rTeq 889
L89
989
IVNVIOOId OVOD ++3 ZO


```
744 thisFloor.elevatorLeaving();
745
746 door.closeDoor( thisFloor );
747
748 if ( leaving ) /1 sai, se necessário
749 moveO;
750 )
751
752 void Elevator: :move() /1 vai para um andar específico
753 {
754 moving = true; II muda estado
755
756 II escalona hora de chegada
757 arrivalTime = currentBuildingClockTime +
758 ELEVATORTRAVEL TIME;
759
```

```
760 cout « “elevador começa a se mover
761 « ( direction == DOWN ? “para baixo : para cima )
762 « “para o andar
763 « ( direction == DOWN ? ‘1’ : ‘2’
764 « „(chega na hora « arrivalTime « )‘
765 « endl;
766
```

Fig. 7.29 Arquivo de implementação da classe Elevator (parte 4 de 4).

O building invoca a função membro processTime (linhas 613 a 626) da classe Elevator, passando como um parâmetro a hora atual da simulação time. Esta função membro atualiza o membro de dados currentBuiJ.dingClockTime com a hora atual da simulação (linha 616) e então verifica o valor do membro de dado motion (linha 618). Se o elevador está se movendo, elevator invoca sua função utilitária processPossibleArrival (linha 619). Se o elevador não está se movendo, elevator invoca sua função utilitáriaprocessPossibleDeparture (linha 621). Se o elevador ainda não está se movendo depois de deter- minar se ele deveria chegar ao andar atual ou partir para o outro andar, elevator envia para a tela uma mensagem indicando que ele está parado no currentFloor (linhas 623 a 625).

A função processPossibleArrival determina se o elevador precisa parar de se mover comparando o currentBuildingClockTime ao arrivalTime calculado (linha 632). Se está na hora do elevador chegar a um andar particular, elevator atualiza currentFloor (linhas 634 a 636) e direction (linhas 638 e 639). O elevator chama então sua função utilitária arriveAtFloor para executar as tarefas necessárias na chegada. A função utilitária processPossibleDeparture determina se o elevador precisa ou não começar a se mover para atender a um outro andar. O código determina se o andar atual ou o outro andar necessita do atendimento do elevador (linhas 658 a 666). Se o andar atual precisa ser atendido, o elevador chama sua função utilitária arriveAtFloor para o andar atual (linhas 670 e 67 1). Caso contrário, a função utilitária prepareToLeave é chamada (linha 677) e o elevador vai se mover para o outro andar, se aquele andar necessita de atendimento.

A função utilitária arriveAtFloor executa as tarefas necessárias para o elevador chegar a um andar específico. Esta função utilitária primeiro pára o elevador, ajustando a variável membro moving para false (linha 683), a seguir desliga o elevatorButton (linha 686) e toca a campainha (linha 688). Um ponteiro temporário para um objeto da classe é então declarado para guardar um *handie* para um objeto Person que poderia estar esperando no andar. Este ponteiro recebe o valor de retorno da chamada à função membro elevatorArrived daquele andar (linha 691).

O elevator abre a porta chamando a função membro openDoor da classe Door, passando como parâmetro um *handie* para o passageiro atual, um *handie* para a pessoa que está esperando no andar, um *handie* para o andar onde o elevador chegou e um *handie* para o próprio elevator (linhas 693 e 694). Elevator novamente determina se algum dos andares necessita de atendimento (linhas 696 a 704). Se o andar atual não necessita de atendimento pelo elevador, elevator se prepara para sair para o outro andar (linha 709) e sai se o outro andar necessita de atendimento. Caso contrário, elevator desliga o indicador de atendimento para o andar atual (linhas 71 1 e 712).

A função membro summonElevator permite que outros objetos solicitem serviços do elevador. Quando invocada, a função membro summonElevator recebe um número de andar como argumento e ajusta o indicador de serviço para true (linhas 719 e 720).

A função membro passengerEnters recebe um ponteiro para um objeto da classe Person como seu único argumento (linha 724) e atualiza o *handie* passengerPtr de elevator, de modo que este aponte para o novo passageiro (linha 727).

A função membro passengerExi ts simplesmente zera o *handie* pas sengerPtr, indicando assim que o passageiro saiu do elevador (linha *735).*

A função membro prepareToLeave recebe um argumento do tipo bool que indica se o elevator deve sair do andar atual (linha 738). O elevador notifica o andar atual de que está saindo enviando uma mensagem elevatorLeaving para o andar (linha 744). O elevador então fecha a porta (linha 748) e, se for o caso, começa a se mover chamando a função utilitária move, que ajusta o membro de dados moving para true (linha 754). Ela então calcula a hora de chegada ao destino do elevator usando o valor static const ELEVATOR_TRAVEL_TI (linhas 757 e *758).* Finalmente, ela escreve na saída a direção na qual está se movendo, o andar de destino e a hora de chegada (arrivalTime) prevista (linhas 760 a 765).

Nossa definição da classe Floor (Fig. 7.30) contém uma mistura de maneiras de associar objetos de outras classes com objetos Floor. Primeiro, usamos uma referência como um *handie* para eJ.evator (linha 804) - isto ***é*** apropriado porque este *handie* sempre se refere ao mesmo elevator. Também temos um ponteiro como um *handie* para um objeto Person (linha 805) - este *handie* vai mudar sempre que uma pessoa entra no andar ou sai do andar para entrar no elevador. Finalmente, temos objetos compostos, incluindo um objeto public floorButton (linha 800) e um objeto private light (linha 806). Declaramos o objeto floorButton como public para permitir que objetos da classe Person acessem o floorButton diretamente.’ A definição da classe Floor também contém os membros de dados static const FLOOR1 e FLOOR2 (linhas 798 e 799). Usamos estas constantes em lugar dos verdadeiros números dos andares; inicializamos estes membros de dados const no arquivo de implementação (linhas 821 e 822). Normalmente, membros de dados

const de uma classe devem ser inicializados na lista de inicialização de membros do construtor. No caso especial de membros de dados static const, eles são inicializados em escopo de arquivo.

```
767 II floor.h
768 II Definição da classe Floor.
769 #ifndef FLOORH
770 #define FLOORH
771
772 #include “floorButton.h”
773 #include “light.h”
774
775 class Elevator; /1 declaração antecipada
776 class Person; /1 declaração antecipada
777
778 class Floor {
779
780 public:
781 Floor( int, Elevator & ); /1 construtor
782 -Floor 1/ destruidor
783 bool isOccupied() const; II retorna true se o andar está ocupado
784 int getNumber() const; /1 retorna número do andar
785
786 II passa um handle para nova pessoa entrando no andar
787 void personArrives ( Person *const );
788
```

Fig. 7.30 Arquivo de cabeçalho da classe Floor (parte 1 de 2).

, Uma pessoa normalmente não tem permissão para interagir com a luz em um andar (a menos que seja um técnico de manutenção). Portanto, o objeto light é declarado na seção private da definição da classe.

795
796
797

```
II avisa o andar que o elevador void elevatorLeaving();


static const int FLOOR1; static const int FLOOR2; FloorButton floorButton;
private:
const int floorNumber;
Elevator &elevatorRef;
Person *occupantptr;
Light light;
#endif II FLOORH


II objeto floorButton
II o nuuero do andar
II ponteiro para o elevador
II ponteiro para a pessoa no andar
II objeto light
```

Fig. 7.30 Arquivo de cabeçalho da classe Floor (parte 2 de 2).

A Fig. 7.3 1 contém o arquivo de implementação para a classe Floor. A função membro isOccupied da classe Floor (linhas 836 a 838) retorna um valor bool que indica se existe ou não uma pessoa esperando no andar. Para determinar se uma pessoa está esperando, verificamos se o occupantPtr não é zero (linha 838). Se o occupantPtr é zero, não há nenhuma pessoa esperando no andar. A função membro getNumber devolve o valor da variável membro floorNumber (linha 841). A função membro personArrives recebe um ponteiro para o objeto Person que está entrando no andar. A este ponteiro é atribuído o membro de dados private
occupantPtr.

```
// floor.cpp
// Definições de funções membro para a classe Floor. #include <iostreain>

using std::cout;
using std::endl;

#include "floor.
#include "person.h"
#include "elevator.

const int Floor::FLOOR1 =1;
const int Floor::FLOOR2 2;

// construtor
Floor::Floor(int nuniber, Elevator &elevatorHandle : floorButton( nunber, elevatorHandle
),
floorNumber( nunber ),elevatorRef( elevatorHandle )
occupantPtr ( 0 ),
light( floorNumber ==1 ? "floor 1" : "floor 2" { cout « "andar «« floorNutnber « 'criado"
« endl;
```



789
790
791
792
793
794

```
// avisa o andar que o elevador chegou Person *elevatorArrived()
```



```
está saindo

// avisa o andar que a pessoa está saindo do andar void personBoardingElevator();
```

798
799
800
801
802
803
804
805
806
807

```
808
809


810
811
812
813
814
815
816
817
818
819
820
821
0822
823
824
825
826
827
828
829
830
831
832


II destruidor
```

Fig. 7.31 Arquivo de implementação da classe Floor (parte 1 de 2).


```
833 Floor: :‘Floor()
834 { cout « “andar « floorNuxnber « " destruído’ « endi;
835
836 II deterxnina se o andar está ocupado
837 bool Floor: :isOccupied() const
838 { return ( occupantPtr != o ) ; }
839
840 II retorna o número deste andar
841 int Floor: :getNumber() const { return floorNumber; }
842
843 II passa pessoa para o andar
844 void Floor: :personArrives( Person *const personPtr
845 { occupantPtr = personPtr; }
846
847 II avisa o andar que o elevador chegou
848 Person *Floor: :elevatorArrived()
849 {
850 II desliga o botão no andar, se necessário
851 cout « “andar ' « floorNumber
852 « ₀desliga o botão” « endl;
853 floorButton.resetButton();
854
855 light.turnOn();
856
857 return occupantPtr;
858 }
859
860 II diz ao andar que o elevador está saindo
```

```
861 void Floor: :elevatorLeaving() { light.turnOff() ; }
862
863 II notifica o andar que urna pessoa está saindo dele
864 void Floor: :personBoardingElevator() { occupantPtr O; }
```

Fig. 7.31 Arquivo de implementação da classe Floor (parte 2 de 2).

A função membro elevatorArrived (linhas 847 a 858) desliga o objeto floorButton do andar 1inha liga a luz e retoma o *handie* occupantPtr (linha 857). A função membro elevatorLeaving desliga a 1 (linha 861). Finalmente, a função membro personBoardingElevator zera o occupantPtr, indicando q a pessoa saiu do andar (linha 864).

Os elementos do arquivo de cabeçalho para a classe Person (Fig. 7.32) deveriam parecer familiares a e' altura. A função membro getlD retorna o ID único do objeto Person. As funções membro stepontoFloo enterElevator e exitElevator formam o resto da interface publie de Person. Usamos uma variável classe private static personCount para registrar quantos objetos da classe Person foram criados. Tar bém declaramos os atributos ID e destinationFloor como membros de dados private const.

```
865 II person.h
866 II Definição da classe Person
867 #ifndef PERSONH
868 #define PERSONH
869
870 class Floor; /1 declaração antecipada
871 class Elevator; /1 declaração antecipada
872
873 class Person
874
```

Fig. 7.32 Arquivo de cabeçalho da classe Person (parte 1 de 2).


```
875 public:
876 Person( const int ); /1 construtor
877 -PersonQ; 1/ destruidor
878 int getlD() const; II devolve o ID da pessoa
879
880 void stepOntoFloor( Floor & );
881 void enterElevator( Elevator &, Floor & );
882 void exitElevator( const Floor &, Elevator & ) const;
883
884 private:
885 static int personCount; II número total de pessoas
886 const int ID; /1 número de ID único da pessoa
887 const int destinationFloor; II número do andar de destino
888 };
889
890 #endif II PERSONH
```

Fig. 7.32 Arquivo de cabeçalho da classe Person (parte 2 de 2).

A imp1ementaço da classe Person (Fig. 7.33) começa com o construtor (linhas 905 a 907), que recebe um único const int como seu argumento. Este representa o andar de destino do objeto Person. Usamos este valor na saída de dados de nossa simulação. O destruidor (linhas 909 a 914) exibe uma mensagem indicando que uma pessoa saiu do elevator.

```
891 II person.cpp
892 II Definições de funções membro para a classe Person.
893 #include <iostreani>
894
895 using std::cout;
896 using std: :endl;
897
898 #include "person.h"
899 #include "floor.h"
900 #include "elevator.h"
901
```

```
902 II inicializa o membro static personCount
903 int Person: :personCount 0;
904
905 Person: :Person( const int destFloor ) // construtor
906 : ID( ++personCount ), destinationFloor( destFloor
907 {)
908
909 Person::Person() // destruidor
910 {
911 cout « "pessoa " « ID « " sai da simulação no andar
912 « destinationFloor « " (destruidor de pessoa invocado)"
913 « endl;
914
915
916 int Person::getlD() const { return ID; } II obtém o ID
917
918 II pessoa entra em um andar
919 void Person::stepontoFloor( Floor& floor
920 {
921 II avisa o andar que urna pessoa está chegando
```

Fig. 7.33 Arquivo de implementação da classe Person (parte 1 de 2).


```
922 cout « "pessoa " « ID « " entra no andar
923 « floor.getNumber() « endi;
924 floor.personArrives( this );
925
926 II aperta botão no andar
927 cout « pessoa " « ID
928 « " aperta o botão de andar no andar
929 « floor.getNumber() « endi;
930 floor.floorButton.pressButton();
931 }
932
933 II pessoa entra no elevador
934 void Person: :enterElevator( Elevator &elevator, Floor &floor
935 {
936 floor.personBoardingElevator() ; II pessoa sai do elevador
937
938 elevator.passengerEnters( this ); 1/ pessoa entra no elevador
939
940 II aperta botão no elevador
941 cout « pessoa " « ID
942 " aperta o botão do elevador" « endi;
943 elevator.elevatorButton.pressButton();
944 }
945
946 II pessoa sai do elevador
947 void Person: :exitElevator(
948 const Floor &floor, Elevator &elevator ) const
949
950 cout « "pessoa " « ID « " sai do elevador no andar
951 « floor.getNumber() « endl;
952 elevator.passengerExits();
953 }
```

Fig. 7.33 Arquivo de implementação da classe Person (parte 2 de 2).

A função membro stepOntoFloor (linhas 918 a 931) primeiro notifica o andar de que a pessoa chegou, enviando ao andar uma mensagem personArrives (linha 924). A pessoa então chama o método pressButton de floorButton (linha 930), que chama o elevador. A função membro enterElevator primeiro avisa o andar que a pessoa está embarcando no elevador, enviando a mensagem personBoardingElevator (linha 936). A pessoa envia a mensagem passengerEnters para avisar o elevador que a pessoa está entrando (linha 938). A pessoa então envia a mensagem pressButton para o objeto elevatorButton, para que o elevador comece a se mover para

o outro andar (linha 943). A função membro exitElevator envia para a saída uma mensagem, indicando que a pessoa está saindo do elevador, e então envia a mensagem passengerExits para o elevator.
Completamos agora a implementação efetiva da simulação do elevador que apresentamos no Capítulo 2. O Capítulo 8 não contém uma seção "Pensando em objetos". No Capítulo 9, discutimos herança em C++ e a aplicamos ao nosso elevador.

Resumo

. A palavra-chave const especifica que um objeto não é modificável.
. O compilador em C++ não permite chamadas de funções membro não-corist para objetos const.
. Uma tentativa de uma função membro const de uma classe de modificar um objeto daquela classe é um erro de sintaxe.
. Uma função é especificada como const tanto em sua declaração como em sua definição.

Uma função membro const pode ser sobrecarregada com uma versão não-corist. A escolha de qual função membro sobrecarregada usar é feita pelo compilador verificando se o objeto foi declarado const ou não.
. Um objeto const deve ser inicializado - inicializadores de membros devem ser fornecidos no construtor de uma classe quando aquela classe contém membros de dados const.
. Classes podem ser compostas por objetos de outras classes.
. Objetos membro são construídos na ordem em que eles estão listados na definição de classe e antes que sejam construídos os objetos da classe que os contém.
. Se um inicializador de membro não é fornecido para um objeto membro, o construtor default do objeto membro é chamado.
. Uma função friend de uma classe é uma função definida fora daquela classe e que tem o direito de acessar todos os membros da classe.
. Declarações friend podem ser colocadas em qualquer lugar na definição da classe.
. O ponteiro this é usado implicitamente para fazer referência tanto para as funções membro não-static como para os membros de dados não-static do objeto.
. Cada função membro não-static tem acesso ao endereço de seu objeto usando a palavra-chave this.
. O ponteiro this pode ser usado explicitamente.
. O operador new aloca espaço para um objeto, executa o construtor do objeto e retorna um ponteiro do tipo correto. Para liberar o espaço para este objeto, use o operador delete.
. Um array de objetos pode ser alocado dinamicamente com new, como em

```
int *ptr = new int[ 100 ];
```

que aloca um array de 100 inteiros e atribui a localização do começo do array a ptr. O array de inteiros precedente pode ser deletado com o comando

```
delete [] ptr;
```

. Um membro de dados static representa informações "com âmbito em toda a classe" (i.e., uma propriedade da classe, não um objeto). A declaração de um membro static começa com a palavra-chave static.
. Membros de dados static têm escopo de classe.
. Membros static de uma classe podem ser acessados através de um objeto daquela classe ou através do nome da classe, usando o operador de resolução de escopo (se o membro for public).
. Uma função membro pode ser declarada static se ela não acessar membros não-static da classe. Diferentemente de funções membro não-static, uma função membro static não tem um ponteiro this. Isto acontece porque membros de dados static e funções membro static existem independentemente de quaisquer objetos de uma classe.
. Classes normalmente escondem seus detalhes de implementação dos clientes das classes. Isto é chamado de ocultação de informações.
. As pilhas são conhecidas como estruturas de dados "último a entrar, primeiro a sair" (LIFO) - o último item inserido na pilha é o primeiro item retirado da pilha.
. Descrever a funcionalidade de uma classe independentemente de sua implementação é chamado de abstração de dados e as classes em C++ definem os chamados tipos de dados abstratos (ADT5).
. C++ aumenta a importância atribuída aos dados. A atividade primária em C++ é criar novos tipos de dados (i.e., classes) e expressar as interações entre objetos daqueles tipos de dados.
. Tipos de dados abstratos são maneiras de representar noções do mundo real, com algum nível satisfatório de precisão, dentro de um sistema de computador.
. Um tipo de dados abstrato na realidade captura duas noções, quais sejam: uma representação de dados e as operações que são permitidas sobre esses dados.

1

. C++ é uma linguagem extensível. Embora a linguagem seja fácil de estender com estes novos tipos, a linguagem básica propriamente dita não é mutável.

. C++ é uma linguagem intencionalmente sucinta, que fornece aos programadores somente os recursos básicos necessários para construir uma ampla gama de sistemas. A linguagem foi projetada para minimizar impactos negativos sobre o desempenho.
. Itens são retirados de uma fila na ordem "primeiro a entrar, primeiro a sair" (FIFO) - o primeiro item inserido na fila é o 7.1
primeiro item retirado da fila. 7.2
. Classes contêiner (também chamadas de classes coleção) são projetadas para manter coleções de objetos. Uma classe contêiner normalmente fornece serviços tais como inserção, retirada, pesquisa, classificação e verificação de se um item pertence à classe e assim por diante.
. É comum se associar objetos iterador - ou simplesmente iteradores - com classes contêiner. Um iterador é um objeto que retorna o próximo item de uma coleção (ou executa alguma ação sobre o próximo item de uma coleção).
. Fornecer aos clientes de sua classe uma classe *proxy,* que conhece somente a interface public da sua classe, habilita os clientes a usar serviços da sua classe sem dar ao cliente acesso a detalhes de implementação da sua classe.

. O única membro **private da** *classeproxy é* um ponteiro para um objeto da classe cujos dados private queremos ocultar.
. Quando uma definição de classe usa só um ponteiro para outra classe, o arquivo de cabeçalho da classe para aquela outra 79 classe (que normalmente revelaria os dados private daquela classe) não precisa ser incluído com #include. Você pode simplesmente declarar aquela outra classe como um tipo de dados, usando uma declaração antecipada *(forward)* de classe antes de o tipo ser usado no arquivo.

. O arquivo de implementação contendo as funções membro para a classe *proxy é* o único arquivo que inclui o arquivo de cabeçalho para a classe cujos dados private gostaríamos de ocultar.

. O arquivo de implementação é fornecido para o cliente como um arquivo objeto pré-compilado, junto com o arquivo de cabeçalho que inclui os protótipos de funções para os serviços fornecidos pela classe pro.sy.

Terminologia 7.1

classe friend
classe *proxy*
composição
construtor
construtor de objeto membro
construtor default
contêiner
declaração antecipada *(fonvard)* de classe destruidor
destruidor default
encadeamento de chamadas a funções membro escopo de classe
especificadores de acesso a membro função friend
função membro const
função membro static
inicializador de membro
inserir (operação de fila)
inserir (push) (operação de pilha)
iterador
linguagem extensível
membro de dados static

objeto const
objeto hospedeiro
objeto membro
objetos dinâmicos
operações de um ADT
operador de resolução de escopo binário (: :) operador de seleção de membro **(.)** operador delete
operador delete []

operador new
operador new []
operador ponteiro de seleção de membro **(->)** ponteiro this
primeiro a entrar, primeiro a sair (FIFO) programação baseada em objetos representações de dados
retirar (operação de fila)
retirar (pop) (operação de pilha) tipo de dados abstrato (ADT) tipo de dados abstrato queue (fila) tipo de dados abstrato stack (pilha)
último a entrar, primeiro a sair (LIFO)

Tem
declar

Erri

7.3
7.4
7.5
7.6

7.8

7.10

7.11
7.12

Bo

7.2
7.3
7.4

Di

7.1
7.2
7.3
7.4

OL

7.1

7.2

Terminologia de "Pensando em objetos"
declaração antecipada *(forward)* problema da inclusão circular

Erros comuns de programação
7.1 Definir como const uma função membro que modifica um membro de dados de um objeto é um erro de sintaxe.

7.2 Definir como const uma função membro que chama uma função membro não-corist da classe sobre a mesma instância da classe é um erro de sintaxe.
7.3 Invocar uma função membro não-const sobre um objeto const é um erro de sintaxe.
7.4 Tentar declarar um construtor ou destruidor como const é um erro de sintaxe.
7.5 Não definir um inicializador de membro para um membro de dados const é um erro de sintaxe.
7.6 Não definir um construtor default para a classe de um objeto membro, quando nenhum inicializador de membro é fornecido para aquele objeto membro, é um erro de sintaxe.
7.7 Tentar usar o operador de seleção de membro (.)com um ponteiro para um objeto é um erro de sintaxe - o operador de seleção de membro ponto ó pode ser usado com um objeto ou com uma referência para um objeto.
7.8 Misturar o estilo new e delete com o estilo malloc e free de alocação dinâmica de memória dinâmica é um erro de lógica: o espaço criado por malloc não pode ser liberado por **delete:** os objetos criados por new não podem ser deletados por free.
7.9 Usar delete em vez de delete [ ] para arrays pode levar a erros de lógica durante a execução. Para evitar problemas, o espaço criado como um array deveria ser apagado com o operador dele te [J e o espaço criado como um elemento individual deveria ser apagado com o operador delete.
7.10 E um erro de sintaxe incluir a palavra-chave static na definição de uma variável de classe static em escopo de arquivo.
7.11 Referir-se ao ponteiro this dentro de uma função membro s tatic é um erro de sintaxe.
7.12 Declarar uma função membro static como const é um erro de sintaxe.

Boas práticas de programação

7.1 Declare como const todas as funções membro que não necessitam modificar o objeto corrente, de forma que você possa usá-las sobre um objeto const se necessitar.
7.2 Coloque todas as declarações friend no início da classe, logo depois do cabeçalho da classe, e não as preceda com quaisquer especificadores de acesso a membros.
7.3 Como C++ inclui C, programas em C++ podem conter memória criada por malloc e liberada por free e objetos criados por new e deletados por delete. E melhor usar somente new e delete.
7.4 Depois de deletar memória dinamicamente alocada, reinicialize o ponteiro que referenciava aquela posição de memória com O. Isso desconecta o ponteiro do espaço anteriormente alocado na memória livre disponível para o programa.

Dicas de desempenho

7.1 A declaração de variáveis e objetos const não só é uma prática efetiva de engenharia de software - ela também pode melhorar o desempenho porque os sofisticados compiladores otimizadores atuais podem executar certas otimizações com constantes que não podem ser executadas com variáveis.
7.2 Inicialize explicitamente objetos membro através de inicializadores de membros. Isto elimina o *overhead* da "dupla inicialização" de objetos membro - uma vez quando o construtor default do objeto membro for chamado e novamente quando funções *sei* forem usadas para inicializar o objeto membro.
7.3 Por motivos de economia de memória e armazenamento, existe só uma cópia de cada função membro por classe e essa função membro é invocada para todos os objetos daquela classe. Cada objeto, por outro lado, tem sua própria cópia dos membros de dados da classe.
7.4 Use membros de dados static para economizar memória quando uma única cópia dos dados é suficiente.

Observações de engenharia de software

7.1 Declarar um objeto como cons t ajuda a seguir o princípio do mínimo privilégio. As tentativas de modificar o objeto são detectadas durante a compilação, em vez de causar erros durante a execução.
7.2 Usar const é crucial para o projeto de classes, o projeto de programas e a codificação adequados.

7.3 Uma função membro const pode ser sobrecarregada com uma versão não-const. A escolha de qual função membro sobrecarregada a ser usada é feita pelo compilador, baseado no fato de o objeto ser const ou não.
7.4 Um objeto const não pode ser modificado por atribuição; por isso, ele deve ser inicializado. Quando um membro de dados de uma classe é declarado const, um *inicializador de membro* deve ser usado para fornecer ao construtor o valor inicial do membro de dados de um objeto da classe.
7.5 Membros de classe constantes (objetos const e "variáveis" const) devem ser inicializados com a sintaxe de inicializador de membro; não são permitidas atribuições.
7.6 **E** uma boa prática declarar como const todas as funções membro de uma classe que não modificam o objeto sobre o qual elas operam. Ocasionalmente, isto será uma anomalia porque você não tem nenhuma intenção de criar objetos const daquela classe. Entretanto, declarar como const tais funções membro oferece um benefício: se você, inadvertidamente, modificar o objeto naquela função membro, o compilador emitirá uma mensagem de erro de sintaxe.
7.7 A forma mais comum de reutilização de software é a *composição,* na qual uma classe tem objetos de outras classes como membros.
7.8 Se uma classe tem como membro um objeto de outra classe, tornar esse objeto membro public não viola o encapsulamento e a ocultação dos membros private daquele objeto membro.
7.9 Embora os protótipos para funções friend apareçam na definição da classe, ainda assim friends não são funções membro.

7.10 As noções de acesso a membros private. protected e public não são relevantes para as declarações de relações friend. de modo que as declarações de relações friend podem ser colocadas em qualquer lugar na definição da classe.
7.1 1 Algumas pessoas na comunidade de OOP consideram que a noção de friend corrompe a ocultação de informações e enfraquece o valor da abordagem de projeto orientado a objetos.
7.12 Como C++ é uma linguagem híbrida, é comum ter-se uma mistura de dois tipos de chamadas de função em um programa e freqüentemente lado a lado - chamadas ao estilo de C, que passam dados primitivos ou objetos para funções, e chamadas no estilo próprio de C++, que passam funções (ou mensagens) para objetos.
7.13 Algumas organizações têm em seus padrões de engenharia de software a norma de que todas as chamadas a funções membro static sejam feitas usando-se o nome da classe e não *o handie* do objeto.
7.14 Os membros de dados static e as funções membro static de uma classe existem e podem ser usados ainda que nenhum objeto daquela classe tenha sido instanciado.
7.15 O programador pode criar novos tipos através do mecanismo de classes. Esses novos tipos podem ser projetados para serem usados tão convenientemente quanto os tipos primitivos. Desse modo, C++ é uma linguagem extensível. Embora a linguagem seja fácil de estender com estes novos tipos, a linguagem básica propriamente dita não é mutável.

Dicas de teste e depura ção

7.1 Sempre declare funções membro como const se elas não modificarem o objeto. Isso pode ajudar a eliminar muitos bugs.
7.2 As linguagens como C++ são como "alvos móveis", pois evoluem. E provável que mais palavras-chave sejam incorpora- das à linguagem. Evite usar palavras muito importantes, tais como "object", como identificadores. Embora "object" não seja atualmente uma palavra-chave em C++, no futuro ela o poderá ser. Assim, futuros compiladores poderiam tornar inoperante o código existente.

Exercícios de auto-revisão

7.1 Preencha os espaços em branco em cada um dos seguintes itens:
a) A sintaxe de _____ é usada para inicializar membros constantes de uma classe.
b) Uma função não-membro deve ser declarada como um _____ de uma classe para ter acesso aos membros de dados private daquela classe.
e) O operador aloca dinamicamente memória para um objeto de um tipo especificado e retorna um daquele tipo.
d) Um objeto constante deve ser **;** não pode ser modificado depois de ser criado.
e) Um membro de dados _____ representa informações "de toda a classe".

O Funções membro de um objeto têm acesso a um "ponteiro para si mesmo" para o objeto, chamado de ponteiro

g) A palavra-chave especifica que um objeto ou variável não é modificável depois de ser inicializado.
h) Se um inicializador de membro não for fornecido paraiim objeto membro de uma classe, é chamado de _____ do objeto.
i) Uma função membro pode ser declarada static se não acessar membros da classe _____

j) Objetos membro são construídos _____ do objeto de classe que os inclui.
k) O operador recupera memória previamente ameada por new.

Ache o(s) erro(s) em cada um dos seguintes itens e explique como corrigi-lo(s).
a) class Exemplo
de public:

```
or Exemplo( int y 10 ) { dado = y;
int obtemDadolncrementado() const { return ++dado;
static int getCount()
o cout « "Dado é - « dado « endl;
E5 return count;
r- }
private:
int data;
static int count;
b) char *string;
string = new char[ 20 ];
free ( string );
```

Respostas aos exercícios de auto-revisão

7.1 a) inicializador de membro. b) friend. c)new, ponteiro. d) inicializado. e) static. O this. g) const. h) construtor default. i) não-static. j) antes. k) delete.
7.2 a) Erro: a definição de classe para Exemplo tem dois erros. O primeiro ocorre na função obtemDadolncrementado.
e A função é declarada const, mas ela modifica o objeto. Correção: para corrigir o primeiro erro, remova a palavra-chave const da definição de obtemDadolncremeritado.
a Erro: o segundo erro acontece na função getCount. Esta função é declarada static, assim não é permitido a acessar qualquer membro não-static da classe.
Correção: para corrigir o segundo erro, remova a linha de saída de dados da definição de getCount.
b) Erro: a memória alocada dinamicamente por new é liberada pela função free. da Biblioteca Padrão de C. Correção: use o operador delete de C++ para liberar a memória. A alocação dinâmica de memória ao estilo de C não deve ser misturada com os operadores new e delete de C++.

Exercícios

7.3 Compare e contraste a alocação dinâmica de memória com os operadores new e delete de C++ com a alocação

dinâmica de memória com as funções malloc e free da Biblioteca Padrão de C.
7.4 Explique a noção de friend em C++. Explique os aspectos negativos de frierid conforme descritos no texto.
7.5 Uma definição correta da classe Time pode incluir os dois construtores seguintes? Se não, explique por quê.

```
Time ( int h O , int m O , int s O );
Time();
```

7.6 O que acontece quando um tipo de retomo, mesmo void, é especificado para um construtor ou destruidor?
7.7 Crie uma classe Date com os seguintes recursos:
a) Enviar a data para a saída em diversos formatos, tais como

```
DDD YYYY
```

DD/MM/YY

```
14 de junho de 1992
```

b) Use construtores sobrecarregados para criar objetos Date inicializados com datas nos formatos especificados no item (a).
e) Crie um construtor de Date que lê a data do sistema usando as funções do arquivo de cabeçalho <ctime> da biblioteca padrão e inicializa os membros de Date.
No Capítulo 8, seremos capazes de criar operadores para testar a igualdade de duas datas e para comparar datas para determinar se uma data é anterior ou posterior à outra.

7.8 Crie uma classe ContaDePoupanca. Use um membro de dados static para conter a taxaDeJurosAnual para cada um dos poupadores. Cada membro da classe contém um membro de dados private saldoDaPoupanca indicando a quantia que o poupador atualmente tem em depósito. Forneça uma função membro calculeRendimentoMensal que calcula o rendimento mensal multiplicando o saldo pela taxaDeJurosAnual dividida por 1 2; este rendimento deve ser somado ao saldoDaPoupanca. Forneça uma função membro static modifiqueTaxaDeJuros que incializa a variável static taxaDeJurosAnual com um novo valor. Escreva um programa de teste para testar a classe ContaDePoupanca. Instancie dois objetos diferentes contaDePoupanca. poupadori e poupador2. com saldos de $2000.00 e $3000.00, respectivamente. Inicialize taxaDeJurosAnual com 6%, então calcule o rendimento mensal e imprima os novos saldos para cada um dos poupadores. Inicialize então a taxaDeJurosAnual com 8% e calcule o rendimento do próximo mês e imprima o novo saldo para cada um dos poupadores.
7.9 Crie uma classe chamada ConjuntoDelnteiros. Cada objeto de classe ConjuntoDelnteiros pode manter inteiros no intervalo O a 100. Um conjunto é representado internamente como um array de uns e zeros. O elemento de array a

i] é 1 se o inteiro *i* está no conjunto. O elemento de array a [j] *é* O se o inteiroj não está no conjunto. O construtor default
inicializa o conjunto com o chamado "conjunto vazio", i.e., um conjunto cuja representação de array contém toda ela zeros.
Providencie funções membro para as operações comuns sobre conjuntos. Por exemplo, forneça uma função membro uniaoDeCojuntosDelnteiros que cria um terceiro conjunto que é a união teórica de dois conjuntos existentes (i.e., a um elernentu do array do terceiro conjunto é atribuído 1 se esse elemento é 1 em um ou ambos os conjuntos existentes e a um elemento do array do terceiro conjunto é atribuído O se esse elemento é O em cada um dos conjuntos existentes).
Forneça uma função membro intersecaoDeConjuntosDelnteiros que cria um terceiro conjunto que é a interseção teórica de dois conjuntos existentes (i.e., um elemento do terceiro array do conjunto é inicializado com O se esse elemento

é O em um ou ambos os conjuntos existentes, e um elemento do array do terceiro conjunto é inicializado com 1 se esse elemento
é 1 em cada um dos conjuntos existentes).
Forneça uma função membro insiraElemento que insere um novo inteiro *k* para um conjunto (inicializando a [k 1 com 1). Forneça uma função membro retiraElemento que exclui o inteiro *m* (inicializando a [m] com 0).
Forneça uma função membro imprimaConjurito que imprime um conjunto como uma lista de números separados por espaços. Imprima somente aqueles elementos que estão presentes no conjunto (i.e., sua posição no array tem um valor 1). Imprima -- para um conjunto vazio.
Forneça uma função membro elgualA que determina se dois conjuntos são iguais.
Forneça um construtor adicional para receber cinco argumentos inteiros, que pode ser usado para inicializar um objeto conjunto. Se você quiser fornecer menos de cinco elementos para o conjunto, usem argumentos default de -1 para os outros.
Agora, escreva um programa de teste para testar sua classe ConjuntoDelnteiros. Instancie vários objetos de ConjuntoDelnteiros. Certifique-se de que todas as suas funções membro funcionem corretamente.
7.10 Seria perfeitamente razoável para a classe Time da Fig. 7.8 representar o tempo internamente como o número de segundos desde a meia-noite, em vez dos três valores inteiros hour. minute e second. Os clientes poderiam usar os mesmos métodos públicos e obter os mesmos resultados. Modifique a classe Time da Fig. 7.8 para implementar Time como o número de segundos decorridos desde a meia-noite e mostre que não existe nenhuma mudança visível de funcionalidade para os clientes da classe.

Entrada/saída com streams em C++

Objetivos
• Entender como usar a entradalsaída orientada a objetos com streams de C++.
. • Ser capaz de formatar entradas e saídas.
• Entender a hierarquia de classes de E/S com streams.
• Entender como fazer entrada/saída com objetos de tipos definidos pelo usuário.
• Ser capaz de criar manipuladores de streams definidos pelo usuário.
• Ser capaz de determinar o sucesso ou o fracasso de opera ções de entradalsaída.
• Ser capaz de vincular o stream de saída ao stream de entra da. Consciência... não parece em si dividida em pedacinhos
Um "rio" ou uma "corrente'" são metáforas utilizadas para descrevê-la mais naturalmente.
William James
Todas as notícias que podem ser impressas2.
Adolph S. Ochs

N. de R.: Stream, no originaL
2 N. de R.: Lema do jornal The New York Times

638 C++ COMO PROGRAMAR
Visão Geral
11.1 Introdução
11.2 Streams
11.2.1 Arquivos de cabeçalho da biblioteca iostream
11.2.2 Classes e objetos de entrada/saída com streams
11.3 Saída com streams
11.3.1 Operador de inserção em stream
11.3.2 Encadeando operadores de inserção/extração do streani
11.3.3 Saída de variáveis char*
11.3.4 Saída de caracteres com a função membro put; encadeando putS
11.4 Entrada com streams
11.4.1 Operador de extração do stream
11.4.2 Funções membro get e getline
11.4.3 Funções membro peek, putback e ignore de istreaxn
11.4.4 EIS segura quanto ao tipo
11.5 E/S não-formatada com read, gcount e write

11.6 Manipuladores de streams
11.6.1 Base do stream de inteiros: dec, oct, hex e setbase
11.6.2 Precisão em ponto flutuante (precision, setprecision)
11.6.3 Largura de campo (setw, width)
11.6.4 Manipuladores definidos pelo usuário
11.7 Estados de formato do stream
11.7.1 Indicadores de estado de formato
11.7.2 Zeros à direita e pontos decimais (ios: : showpoint)
11.7.3 Alinhamento (ios: : left, ios: : right, ios: internal)
11.7.4 Preenchimento (fui, setf iii)
11.7.5 Base do stream de inteiros (ios: :dec, ios: :oct, ios: :hex, ios: : showbase)
11.7.6 Números em ponto flutuante; notação científica (ios: : scientific, ios: : fixed)
11.7.7 Controle de maiúsculas/minúsculas (ios: : uppercase)
11.7.8 Inicializando e reinicializando os indicadores de formato (fiags, setiosfiags,
resetiosfiags)
11.8 Estados de erro do stream
11.9 Vinculando um stream de saída a um stream de entrada
Resu,no. Terminologia Erros comuns de programação Boas práticas de programação Dica de desempenho . Dica de portabilidade. Observações de engenharia de software Exercícios de auto-revisão• Respostas aos exercícios de auto-revisão • Exercícios

CAPÍTULO 11 - ENTRADA/SAÍDA COM STREAMS EM C++ 639
11.1 Introdução
As bibliotecas padrão de C++ fornecem um extenso conjunto de recursos de entrada/saída. Este capítulo discute uma gama suficiente de recursos para executar as operações de EIS mais comuns e avaliar os demais recursos. Alguns dos recursos apresentados aqui foram discutidos anteriormente no texto, mas esse capítulo fornece uma discussão mais completa dos recursos de entrada/saída de C++.
Muitos dos recursos de E/S descritos aqui são orientados a objetos. O leitor deve achar interessante ver como
tais recursos são implementados. Este estilo de E/S faz uso de outras características de C++, tais como referências, sobrecarga de funções e sobrecarga de operadores.
Como veremos, C++ usa E/S segura quanto ao tipo. Cada operação de EIS é automaticamente executada de uma maneira sensível ao tipo dos dados. Se uma função de EIS foi adequadamente definida para tratar um tipo de dado particular, então aquela função é chamada para tratar aquele tipo de dado. Se não existe uma correspondência entre o tipo real dos dados e uma função para manipular aquele tipo de dado, é gerada uma indicação de erro de compilação. Desse modo, dados impróprios não podem se mover “furtivamente” através do sistema (como pode acontecer em C - uma brecha em C que permite alguns erros bastante sutis e freqüentemente estranhos).

Os usuários podem especificar E/S de tipos definidos pelo usuário, como também de tipos padrão. Esta
extensibilidade é um dos recursos mais valiosos de C++.
Boa prática de programação 11.1
Use exclusivamente a forma de EIS de C+ + em programas em C+ +, apesar do fato de que a EIS ao estilo
de C está disponível para os programadores de C++.
Observação de engenharia de software 11.1
_____ O estilo de EIS de C+ + é seguro quanto ao tipo.
Observação de engenharia de soflware 11.2
_____ C+ + possibilita um tratamento comum da EIS de tipos primitivos e de tipos definidos pelo usuário. Este tipo de "comunalidade "facilita o desenvolvimento de software em geral e a reutilização de software em particular
11.2 Streams
A E/S em C++ ocorre em streams de bytes. Um stream é simplesmente uma seqüência de bytes. Em operações de entrada, os bytes fluem de um dispositivo (por exemplo: um teclado, uma unidade de disco ou uma conexão de rede) para a memória principal. Em operações de saída, os bytes fluem da memória principal para um dispositivo (por exemplo: uma tela de monitor, uma impressora, uma unidade de disco ou uma conexão de rede).
O aplicativo associa significados aos bytes. Os bytes podem representar caracteres ASCII, formato interno de dados brutos, imagens gráficas, voz digitalizada, vídeo digital ou qualquer outro tipo de informações que um aplicativo possa requerer.
O trabalho dos mecanismos de E/S do sistema é mover bytes de dispositivos para a memória, e vice-versa, de uma maneira consistente e confiável. Tais transferências envolvem freqüentemente movimento mecânico, tal como a rotação de um disco ou uma fita, ou o próprio bater nas teclas em um teclado. O tempo que estas transferências tomam normalmente é enorme, se comparado ao tempo que o processador leva para manipular dados internamente. Desse modo, operações de E/S exigem planejamento e afinação cuidadosa, para garantir o máximo desempenho.
C++ fornece tanto recursos de E/S "de baixo nível" como de "alto nível". Recursos de E/S de baixo nível (i.e., E/S não-formatada) especificam tipicamente que algum número de bytes deve simplesmente ser transferido de um dispositivo para a memória ou da memória para um dispositivo. Em tais transferências, cada byte é o item de interesse. Tais recursos de baixo nível fornecem alta velocidade, transferências de grande volume, mas estes recursos não são particularmente convenientes para as pessoas.
As pessoas preferem uma visão de nível mais alto da E/S (i.e, EIS formatada), na qual os bytes são agrupados
em unidades significativas, tais como inteiros, números de ponto flutuante, caracteres, strings e tipos definidos pelo

usuário. Estes recursos orientados a tipos são satisfatórios para a maioria das operações de EIS, exceto para processamento de grandes volumes de arquivos.

Dica de desempenho 11.1
_____ Use E/S não-formatada para obter melhor desempenho no processamento de arquivos de grande volume.

11.2.1 Arquivos de cabeçalho da biblioteca iostream

A biblioteca iostream de C÷+ fornece centenas de recursos de EIS. Vários arquivos de cabeçalho contêm partes da interface da biblioteca.

A maioria dos programas em C++ incluem o arquivo de cabeçalho <iostream>. que declara serviços básicos necessários para todas as operações de E/S com streams. O arquivo de cabeçalho <iostreain> define os objetos cm, cout, cerr e clog, que correspondem ao stream padrão de entrada, o stream padrão de saída, o stream padrão de erros sem buffer e o stream padrão de erros com buifer, respectivamente. São fornecidos tanto os serviços de EIS não-formatada como formatada.

O cabeçalho <iomanip> declara serviços úteis para executar operações de processamento de arquivos, com os chamados manipuladores de streams parametriados.

O cabeçalho <fstreaxn> declara serviços importantes para operações de processamento de arquivo controladas pelo usuário. Usamos este cabeçalho nos programas de processamento de arquivos do Capítulo 14.

As implementações de C÷+ geralmente contêm outras bibliotecas de EIS relacionadas, que fornecem recursos
específicos do sistema, tais como controlar dispositivos especiais para EIS de áudio e vídeo.

11.2.2 Classes e objetos de entrada/saída com streams

A biblioteca ios tream contém muitas classes para tratar uma grande variedade de operações de E/S. As classes de iostream suportam as operações de entrada em stream. As classes de ostream suportam operações de saída com streams. As classes de istream de suportam tanto as operações de entrada com streams como de saída com streams.

A classe istream e a classe ostream são derivadas por herança simples da classe base ios. A classe
iostream é derivada através de herança múltipla, tanto da classe istream como da classe ostrearn. Estas relações de herança são resumidas na Fig. 11.1.

jos
istream ostream
ios tream

Fig. 11.1 Parte da hierarquia de classes de EIS com streams.

A sobrecarga de operadores fornece uma notação conveniente para executar entrada/saída. O operador de deslocamento à esquerda («)é sobrecarregado para designar saída com stream e é chamado de operador de inserção no stream. O operador de deslocamento à direita (») é sobrecarregado para designar entrada com stream e é chamado de operador de extração do stream. Estes operadores são usados com os objetos stream padrão cm, cout, cerr e

clog e comumente com objetos stream definidos pelo usuário.
O objeto pré-definido cm é uma instância da classe istream e se diz que é “vinculado” (ou conectado) ao dispositivo de entrada padrão, que normalmente é o teclado. O operador de extração de stream (»), como usado no comando seguinte, faz com que um valor para a variável inteira grade (assumindo-se que grade foi declarada como uma variável int) seja lido de cm para a memória:

cm » grade; // os dados “fluem” na direção
// das setas, para a direita
Note que a operação de extração de stream é “suficientemente esperta” para “saber” qual é o tipo dos dados. Assumindo-se que grade foi corretamente declarada, nenhuma informação de tipo adicional necessita ser especificada para uso com o operador de extração de stream (como, a propósito, é o caso no estilo de EIS de C).
O objeto predefinido cout é uma instância da classe ostream e se diz que é “vinculado” ao dispositivo de saída padrão, normalmente ateia do monitor de vídeo. O operador de inserção em stream («), conforme usado no comando seguinte, faz com que o valor da variável inteira grade seja enviado da memória para o dispositivo padrão de saída:
cout « grade; II os dados “fluem” na direção
// das setas, para a esquerda
Note que o operador de inserção em stream é “suficientemente esperto” para “saber” o tipo de grade (assumindo- se que ela foi corretamente declarada), de modo que nenhuma informação de tipo adicional precisa ser especificada para uso com o operador de inserção em stream.
O objeto pré-definido cerr é uma instância da classe ostream e se diz que é “vinculado” ao dispositivo de erro padrão. As saídas para o objeto cerr não são colocadas em um buifer. Isto significa que cada inserção no stream cerr faz com que sua saída apareça imediatamente; isto é apropriado para notificar prontamente um usuário sobre erros.
O objeto pré-definido clog é uma instância da classe ostream e também se diz ser “vinculado” ao dispositivo de erro padrão. As saídas para clog são colocadas em um buifer. Isto significa que cada inserção em clog pode fazer com que sua saída seja mantida em um buifer até o buifer estar cheio ou até ser esvaziado.
O processamento de arquivos em C++ usa as classes ifstreain para executar operações de entrada em arquivos, ofstream para operações de saída em arquivos e fstream para operações de entrada/saída em arquivos. A classe ifstream herda de istream, a classe ofstream herda de ostream e a classe fstream herda de iostream. As várias relações de herança das classes relacionadas a EIS são resumidas na Fig. 11.2. Existem muitas classes mais na hierarquia completa de classes de E/S com streams suportads na maioria das instalações, mas as classes mostradas aqui fornecem a grande maioria dos recursos de que a maioria dos programadores necessita. Veja o manual de referência da biblioteca de classes para seu sistema de C++ para obter mais

informações sobre o processamento de arquivos.

Fig. 11.2 Parte da hierarquia de classes de EIS com streams com as classes essenciais para processamento de arquivos.

11.3 Saída com streams

A classe ostream de C++ fornece a possibilidade de executar saída formatada e não-formatada. Os recursos para saída incluem: saída dos tipos de dados padrão com o operador de inserção em stream; saída de caracteres com a função membro put; saída não-formatada com a função membro wri te (Seção 11.5); saída de inteiros nos formatos decimal, octai e hexadecimal (Seção 11.6.1); saída de valores de ponto flutuante com várias precisões (Seção 11.6.2), com pontos decimais forçados (Seção 11.7.2), em notação científica e em notação fixa (Seção 11.7.6); saída de dados alinhados em campos com larguras designadas de campo (Seção 11.7.3); saída de dados em campos preenchidos com caracteres especificados (Seção 11.7.4); e saída de letras maiúsculas em notação científica e notação hexadecimal (Seção 11.7.7).

	ios
	- istream ostream
. ifstrea m	iost eam ofstream
	fstream

11.3.1 Operador de inserção em stream

A saída com streams pode ser executada com o operador de inserção em siream, i.e., o operador « sobrecarregado. O operador « é sobrecarregado para saída de itens de dados de tipos primitivos, saída de strings e para a saída de valores do tipo ponteiro. A Seção 11.9 mostra como sobrecarregar « para fazer a saída de itens de dados de tipos definidos pelo usuário. A Fig 11.3 demonstra a saída de um string usando um único comando de inserção em stream. Múltiplos comandos de inserção podem ser usados como na Fig. 11.4. Quando é executado este programa, produz a mesma saída que o programa anterior.

```
1 II Fig. 11.3: figll_03.cpp
2 II Enviando um string para a saída usando inserção no stream.
3 #include <iostream>
4
5 using std::cout;
6
7 int main()
8
9 cout « “Bem-vindo a C++’\n’;
```

```
10
11 return O;
12 }
```
Bem-vindo a C++!
Fig. 11.3 Enviando um string para a saída usando inserção no stream.
```
1 II Fig. 11.4: figIIO4.cpp
2 II Enviando um string para a saída usando duas inserções no stream.
3 #include <iostream>
4
5 using std: :cout;
6
7 int main()
8{
9 cout « “Bem-vindo a
10 cout « “C++!\n”;
11
12 return O;
13 }
```
Bem-vindo a
Fig. 11.4 Enviando um string para a saída usando duas inserções no stream.

O efeito da seqüência de escape \n (nova linha) também pode ser obtido com o manipulador de stream, endi (fim de linha), como na Fig. 11.5. O manipulador de stream endi gera um caractere nova linha e, além disso, esvazia o buffer de saída (i.e., faz com que a saída do buffer seja executada imediatamente, mesmo que o buifer não esteja cheio). O buffer de saída também pode ser esvaziado simplesmente com

```
cout « flush;
```

Os manipuladores de stream são discutidos em detalhes na Seção 11.6.
Expressões podem ser enviadas para a saída conforme mostrado na Fig. 11.6.

Boa prática de programação 11.2
Quando enviar expressões para a saída, coloque-as entre parênteses, para prevenir problemas de precedência de operadores entre os operadores na expressão e o operador «.

```
1 II Fig. 11.5: figIIO5.cpp
2 II Usando o manipulador de stream endl.
3 #include <iostream>
4
5 using std: :cout;
6 using std::endl;
7
8 int main()
9
10 cout « “Bem-vindo a “;
11 cout « “C++!”;
```

```
12 cout « endi; II manipulador de stream endi (fim de linha)
13
14 return O;
15
```

```
1. // Fig. 11.6: figllO6.cpp
1/ Enviando valores de expressões para a saída.
#include <iostreani>
using std::cout;
using std::endl;
i.nt main()
cout « “47 mais 53 é
II parênteses não necessários; usados para maior clareza
cout « ( 47 + 53 ); II expressão
cout « endl;
turn O;
53é100
(51i5’
eam ando valores de expressões para a saída.
est
ÇÇet
‘ando operadores de inserção/extração do stream
Darregados « e » podem, cada um deles, ser usados de uma forma encadeada,
conforme
7.
```

Bem-vindo a C++I 1	
Fig. 11.5	Usando o manipulador de stream endl.


```
1 II Fig. 11.7: figll_07.cpp
2 II Encadeando o operador « sobrecarregado.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 int main()
9 {
10 cout « “47 mais 53 é “ « ( 47 + 53 ) « endi;
11
12 return 0;
13
47 mais 53 é 100
```

Fig. 11.7 Encadeando o operador « sobrecarregado.
As inserções múltiplas em stream na Fig. 11.7 são executadas como se elas tivessem sido escritas:
cout « “47 mais 53 é “) « (47 + 53)) « endi);
(i.e., « se associa da esquerda para a direita). Este tipo de encadeamento de operadores de inserção em stream é permitido porque o operador « sobrecarregado retorna uma referência para seu objeto operando da esquerda, i.e., cout. Deste modo, a expressão entre parênteses mais à esquerda
cout « “47 mais 53 é
envia o string de caracteres especificado para a saída e retorna uma referência para cout. Isto permite que a expressão entre parênteses do meio seja avaliada como
cout « (47 + 53
que envia para a saída o valor inteiro 100 e retorna uma referência para cout. A expressão entre parênteses mais à direita é então avaliada como
cout « endl
que envia um caractere nova linha para a saída, esvazia cout e retorna uma referência para cout. Este último retorno não é usado.
11.3.3 Saída de variáveis char*
Na EIS no estilo de C, o programador precisa fornecer informações sobre os tipos dos dados. C++ determina tipos de dados automaticamente - uma agradável melhoria em relação a C. Mas, às vezes, isso atrapalha. Por exemplo, sabemos que um string de caracteres é do tipo char *. Suponha que queiramos imprimir o valor daquele ponteiro, i.e., o endereço do primeiro caractere daquele string na memória. Mas o operador « foi sobrecarregado para imprimir dados do tipo char* como um string terminado com caractere nulo. A solução é fazer a coerção do ponteiro para void * (isto deveria ser feito para qualquer variável ponteiro que o programador deseje enviar para a saída como um endereço). A Fig. 11.8 demonstra a impressão de uma variável char * em ambos os formatos, de string e endereço. Note que o endereço é impresso como um valor hexadecimal (base 16). Falaremos mais sobre como controlar as bases de números nas Seções 11.6.1, 11.7.4, 11.7.5 e 11.7.7. Nota: a saída do programa na Fig. 11.8 pode ser diferente de um compilador para outro.


```
1 II Fig. 11.8: figllO8.cpp
2 II Imprimindo o endereço armazenado em uma variável char *
3 #include <iostream>
4
5 using std::cout;
6 using std: :endl;
7
8 int main()
9 {
10 char *string = “teste”;
11
```

```
12 cout « “Valor do string é: “ « string
13 « “\nValor de static_cast< void * >( string ) é:
14 « static_cast< void * >( string ) « endl;
15 return 0;
16 }
Valor do string é: teste
LValor de static cast< void *>( string ) é: 0046C070
```

Fig. 11.8 Imprimindo o endereço armazenado em uma variável char*.

11.3.4 Saída de caracteres com a função membro put; encadeando puts

A função membro put faz a saída de um único caractere, como em

```
cout.put( ‘A’ );
```

que exibe A na tela. As chamadas para put podem ser encadeadas como em

```
cout.put( ‘A’ ).put( ‘\n’ );
```

que envia para a saída a letra A seguida por um caractere nova linha. Como com «, o comando precedente é executado dessa maneira porque o operador ponto é associado da esquerda para a direita e a função membro put retorna uma referência para o objeto ostream que recebeu a mensagem put (chamada de função). A função put pode ser também chamada com uma expressão equivalente a um valor de código ASCII, como em cout. put (65), que também envia A para a saída.

11.4 Entrada com streams

Agora, iremos considerar o stream de entrada. Isso pode ser feito com o operador de extração de stream, i.e., o operador » sobrecarregado. Tal operador normalmente ignora caracteres em branco (tais como espaços em branco, marcas de tabulação e novas linhas) no stream de entrada. Mais tarde, veremos como mudar esse comportamento. O operador de extração de stream retorna zero (falso) quando é encontrado o fim de arquivo em um stream; caso contrário, o operador de extração de stream retorna uma referência para o objeto que recebeu a mensagem de extração (por exemplo, cm na expressão cm » grade). Cada stream contém um conjunto de bits de estado, usados para controlar o estado do stream (i.e., formatação, definição de estados de erro, etc.). A extração de stream faz com que o failbit do stream seja ligado (colocado em 1), se dados do tipo errado foram fornecidos como entrada, e faz com que o badbit do stream seja ligado se a operação falhar. Logo veremos como testar estes bits depois de uma operação de EIS. As Seções 11.7 e 11.8 discutem os bits de estado do stream em detalhes.

11.4.1 Operador de extração do stream

Para ler dois inteiros, use o objeto cm e o operador de extração de stream sobrecarregado, como na Fig. 11.9. Note que operações de extração de stream também podem ser encadeadas.

A precedência relativamente alta dos operadores » e « pode causar problemas. Por exemplo, o programa

da Fig. 11.10 não compilará corretamente sem os parênteses em torno da expressão condicional. O leitor deve cuidar disso.

```
1 II Fig. 11.9: figllO9.cpp
2 II Calculando a soma de dois inteiros udos do teclado
3 II com cm e o operador de extração de streani.
4 #include <iostream>

5

6 using std::cout;
7 using std: :cin;
8 using std: :endl;
```

Fig. 11.10 Evitando um problema de precedência entre o operador de inserção em stream e o operador condicional (parte 1 de 2).


```
9
10 int main()
11 {
12 int x, y;
13
14 cout « “Digite dois inteiros: “;
15 cin»x»y;
16 cout « “Soma de “ « x « “ e “ « y « “ é:
17 «(x+y)«endl;
18
19 return 0;
20 }
```

```
1
‘4
ç
```

Digite dois inteiros: 30 92 Soma de 30 e 92 é: 122	
Fig. 11.9 Calculando a soma de dois inteiros udos do teclado com cm e o operador de	extraçã o de
stream.	
1 II Fig. 11.10: figlllo.cpp	
2 II Evitando um problema de precedência entre o operador de	
3 II inserção em stream e o operador condicional.	
4 II Requer parênteses em torno da expressão condicional.	
5 #include <iostrealrL>	
6	

```
7 using std: :cout;
8 using std::cin;
9 using std::endl;
10
11 int main()
12
13 int x, y;
14
15 cout « Digite dois inteiros:
16 cin»x»y;
17 cout « x « ( x = y ? “ é” : “ não é”
18 « “ igual a “ « y « endl;
19
20 return 0;
21
```


Fig. 11.10 Evitando um problema de precedência entre o operador de inserção em stream e o operador condicional (parte 2 de 2).

Erro comum de programação 11.1

Tentar ler de um ostream (ou qualquer outro stream somente de saída).

Erro comum de programação 11.2

Tentar escrever em um istream (ou qualquer outro stream somente de entrada).

Erro comum de programação 11.3

Não usar parênteses, para forçar a precedência apropriada, quando usar os operadores de precedencia relativamente mais alta para inserção em stream («) ou extração de stream (»).

Um meio popular para ler uma série de valores da entrada é usar a operação de extração de stream na condição de continuação do laço em um laço while. A extração retorna falso (O) quando o fim de arquivo é encontrado. Considere o programa da Fig. 11.1 1, que acha a nota mais alta em uma prova. Assuma que o número de notas não é conhecido com antecedência e que o usuário digitará fim de arquivo para indicar que todas as notas já foram digitadas. A condição do while, (cm » grade), se torna O (interpretado como false) quando o usuário digitar fim de arquivo.

Dica de portabilidade 11.1

Quando informar ao usuário sobre como terminar a entrada de dados pelo teclado, peça ao usuário que
“digite fim de arquivo para terminar a entrada” em vez de solicitar <ctrl>-d (UNJX e Macintosh) ou <ctrl>-z (PC e VAX).

Na Fig. 11.11. cm » grade pode ser usada como uma condição, porque a classe

base ios (da qual istream é herdado) fornece um operador de coerção sobrecarregado que converte um stream em um ponteiro do tipo void ‘. O valor do ponteiro retornado é O (false) se um erro aconteceu enquanto se tentava ler um valor ou o indicador de fim de arquivo foi encontrado. O compilador pode usar o operador de cast void * implicitamente.

11.4.2 Funções membro get e getline

A função membro get sem parâmetros fornece como entrada um caractere do stream designado (mesmo que este seja um espaço em branco) e retorna este caractere como o valor da chamada da função. Esta versão de get retorna EOF quando o fim de arquivo é encontrado no stream.

```
1 II Fig. 11.11: figlll.cpp
2 1/ Operador de extração de stream retornando false no fim de arquivo.
3 #include <iostream>
4
```

Fig. 11.11 Operador de extração de stream retornando false quando encontra fim de arquivo (parte 1 de 2).

Digite dois inteiros: 7 5 7 não é igual a 5			
Digite	dois	inteiros: 8 8	
8 é igual a		8	


```
5 using std::cout;
6 using std::cin;
7 using std::endl;
8
9 int main()
10
11 int grade, highestGrade = -1;
12
13 cout « “Digite nota (digite fim de arquivo para terminar): “;
14 while ( cm » grade
15 if ( grade > highestGrade
16 highestGrade = grade;
17
18 cout « “Digite nota (digite fim de arquivo para terminar):
19
20
21 cout « “\n\nNota mais alta é: “ « highestGrade « endi;
22 return 0;
23
Digite nota (digite fim de arquivo para terminar): 67
Digite nota (digite fim de arquivo para terminar): 87
Digite nota (digite fim de arquivo para terminar): 73
Digite nota (digite fim de arquivo para terminar): 95
```

```
Digite nota (digite fim de arquivo para terminar): 34
Digite nota (digite fim de arquivo para terminar): 99
Digite nota (digite fim de arquivo para terminar):
Nota mais alta é: 99
```

Fig. 11.11 Operador de extração de stream retornando false quando encontra fim de arquivo (parte 2 de 2).

A Fig. 11. 12 demonstra o uso das funções membro eof e get com o stream de entrada cm e da função membro put com o stream de saída cout. O programa primeiro imprime o valor de cm . eof () , i.e., false (0, na saída), para mostrar que ainda não encontrou o fim de arquivo em cm. O usuário digita uma linha de texto e aperta a tecla Enter, seguida por fim de arquivo (<ctrl>-z em sistemas compatíveis com o IBM PC, <ctrl>-d em sistemas UNIX e Macintosh). O programa lê cada caractere e o envia para a saída em cout, usando a função membro put. Quando o fim de arquivo é encontrado, o while termina e cm . eof () - agora true - é novamente impresso (1 na saída), para mostrar que o fim de arquivo foi “ligado” em cm. Note que este programa usa a versão da função membro get de istream que não aceita nenhum argumento e retorna o caractere que está sendo lido.

A função membro get com um argumento de referência para caractere recebe como entrada o próximo caractere do stream de entrada (mesmo que este seja um espaço em branco) e armazena-o no parâmetro do tipo caractere. Esta versão de get retorna O quando o fim de arquivo é encontrado; caso contrário, esta versão de get retorna uma referência para o objeto istream para o qual a função membro get está sendo invocada.

```
1 // Fig. 11.12: figlll2.cpp
2 // Usando as funções membro get, put e eof.
3 #include <iostream>
4
5 using std::cout;
6 using std::cin;
7 using std::endl;
8
9 int main()
10
```

Fig. 11.12 Usando funções membro get, put e eof (parte 1 de 2).


```
11 char e;
12
13 cout « “Antes da leitura, cin.eof () é “ « cin.eof O
14 « \nDigite urna frase seguida por um fim de arquivo:\n;
15
16 while ( ( c = cin.getO ) != EOF
17 cout.put( e );
18
19 cout « “\nEOF neste sistema é: “ « c;
```

```
20 cout « “\nApós a leitura, cin.eof() é “ « cin.eof() « endl;
21 return 0;
22
Antes da leitura, cin.eof() é O
Digite uma frase seguida por um fim de arquivo:
Testando as funções membro get e putAZ
Testando as funções membro get e put
EOF neste sistema é: -1
Após a leitura, cin.eof() é 1
```

Fig. 11.12 Usando funções membro get, put e eof (parte 2 de 2).

Uma terceira versão da função membro get aceita três parâmetros - um array de caracteres, um limite de tamanho e um delimitador (com valor default \n'). Esta versão lê caracteres do stream de entrada. Ela lê um caractere a menos que o número máximo de caracteres especificado e termina, ou termina assim que o delimitador é lido. Um caractere nulo é inserido para terminar o string de entrada no array de caracteres usado como um buifer pelo programa. O delimitador não é colocado no array de caracteres, mas permanece no stream de entrada (o delimitador será o próximo caractere a ser lido). Deste modo, o resultado de um segundo get sucessivo é uma linha vazia, a menos que o caractere delimitador seja removido do stream de entrada. A Fig. 11.13 compara a entrada usando cm com a extração de stream (que lê caracteres até que um espaço em branco seja encontrado) e a entrada com ci get. Note que a chamada a cm . get não especifica um caractere delimitador, de modo que o default '\n' é usado.

```
1 II Fig. 11.13: figlll3.cpp
2 II Comparando leitura de strings com cm e cin.get.
3 #include <iostrearn>
4
5 using std: :cout;
6 using std::cin;
7 using std: :endl;
8
9 int main()
10 (
11 const int SIZE = 80;
12 char bufferl[ SIZE ], buffer2[ SIZE ];
13
14 cout « “Digite urna frase:\n”;
15 cm » bufferl;
16 cout « “\nO string lido com cm era:\n”
17 « bufferl « “\n\n”;
18
19 cin.get( buffer2, SIZE );
20 cout « “O string lido com cin.get era:\n”
21 « buffer2 « endl;
```

Fig. 11.13 Comparando a leitura de um string usando cm com extração do stream e a leitura com cm. get (parte 1 de 2).

```
1
22
23 return O;
24
Digite uma frase:
Comparando leitura de strings com cm e cin.get
O string lido com cm era:
Comparando
O string lido com cin.get era:
leitura de strings com cm e cin.get
```
Fig. 11.13 Comparando a leitura de um stririg usando cm com extração do stream e a leitura com cm . get (parte 2 de 2).

A função membro getlmne opera como a terceira versão da função membro get e insere um caractere nulo depois da linha no array de caracteres. A função getline remove o delimitador do stream (i.e., lê o caractere e o descarta), mas não armazena o mesmo no array de caracteres. O programa da Fig. 11.14 demonstra o uso da função membro getline para ler uma linha de texto.

```
1 /1 Fig. 11.14: figlll4.cpp
2 II Leitura de caracteres com a função membro getlmne.
3 #include <iostream>
4
5 using std::cout;
6 using std::cin;
7 using std: :endl;
8
9 int main()
10
11 const SIZE = 80;
12 char buffer[ SIZE );
13
14 cout « Digite uma frase:\n;
15 cin.getline( buffer, SIZE );
16
17 cout « “\nA frase digitada é:\n” « buffer « endl;
18 return 0;
19
```

Fig. 11.14 Entrada de caracteres com a função membro getline

11.4.3 Funções membro peek. putback e ignore de istream

A função membro ignore ignora um número especificado de caracteres (o default é um caractere) ou termina quando encontra um delimitador especificado (o delimitador default é EOF, que faz com que ignore salte para o fim do arquivo quando estiver lendo um arquivo).

Digite	uma frase:	
Usando a função membro		getline
A frase	digitada é:	
Usando a função membro		getline

A função membro putback coloca o último caractere lido de um stream de entrada por um get de volta naquele stream. Esta função é útil para aplicativos que varrem streams de entrada procurando um campo com um caractere inicial específico. Quando esse caractere é lido da entrada, o aplicativo coloca esse caractere de volta no stream, para que o caractere possa ser novamente lido do stream, junto com os demais dados.
A função membro peek retoma o próximo caractere de um stream de entrada, mas não remove o caractere do stream.
114.4 EIS segura quanto ao tipo
C++ oferece EIS segura quanto ao tipo. Os operadores « e » são sobrecarregados para aceitar itens de dados de tipos específicos. Se são processados dados inesperados, vários indicadores de erro são “ligados”para que o usuário possa testar se uma operação de E/S teve sucesso ou falhou. Desta maneira, o programa “permanece” no controle. Discutimos estes indicadores de erro na Seção 11.8.
11.5 EIS não-formatada com read, gcount e write
A entrada/saída não-frrmatada é executada com as funções membro read e write. Cada uma dessas funções recebe como entrada, ou envia para a saída, um certo número de bytes para ou de um array de caracteres na memória. Estes bytes não são formatados de nenhuma forma. Eles simplesmente são udos da entrada ou enviados para a saída como bytes brutos. Por exemplo, a chamada
char buffer[J = “FELIZ ANIVERSÁRIO”;
cout.write(buffer, 10);
envia para a saída os primeiros 10 bytes de buffer (inclusive caracteres nulos, que fariam a saída com cout e « terminar). Como um string de caracteres é avaliado como o endereço de seu primeiro caractere, a chamada
cout.write(‘ABCDEFGHIJKLMNOPQRSTUVWXYZ”, 10);
exibe os primeiros 10 caracteres do alfabeto.
A função membro read lê um número especificado de caracteres para um array de caracteres. Se o número de caracteres udos é menor que o especificado, failbit é “ligado”. Logo veremos como determinar se failbit foi “ligado” (ver Seção 11.8). A função membro gcount informa o número de caracteres udos pela última operação de entrada.
A Fig. 11.15 demonstra as funções membro de istream read e gcount,e a função membro write de ostream. O programa recebe como entrada 20 caracteres (de uma seqüência de entrada mais longa) para o array de caracteres buffer com o uso de read, determina o número de caracteres fornecidos como entrada com

gcount e envia para a saída os caracteres em buffer usando write.

Fig. 11.15 E/S não-formatada com as funções membro read, gcount e write (parte 1 de 2).

1	// Fig. 11.15: figlll5.cpp		
2	// E/S não-formatada	com	read, gcount e write.
3	#include <iostream>		
4			
5	using std::cout;		
6	using std::cin;		
7	using std::endl;		
8			
9	int main()		
10	(		
11	const int SIZE =	80;	
12	char buffer[SIZE	];	

A função membro putback coloca o último caractere lido de um stream de entrada por um get de volta naquele stream. Esta função é útil para aplicativos que varrem streams de entrada procurando um campo com um caractere inicial específico. Quando esse caractere é lido da entrada, o aplicativo coloca esse caractere de volta no stream, para que o caractere possa ser novamente lido do stream, junto com os demais dados.
A função membro peek retoma o próximo caractere de um stream de entrada, mas não remove o caractere do stream.

11.4.4 EIS segura quanto ao tipo

C++ oferece EIS segura quanto ao tipo. Os operadores « e » são sobrecarregados para aceitar itens de dados de tipos específicos. Se são processados dados inesperados, vários indicadores de erro são “ligados”para que o usuário possa testar se uma operação de E/S teve sucesso ou falhou. Desta maneira, o programa “permanece” no controle. Discutimos estes indicadores de erro na Seção 11.8.

11.5 EIS não-formatada com read, gcount e write

A entrada/saída não-formatada é executada com as funções membro read e write. Cada uma dessas funções recebe como entrada, ou envia para a saída, um certo número de bytes para ou de um array de caracteres na memória. Estes bytes não são formatados de nenhuma forma. Eles simplesmente são lidos da entrada ou enviados para a saída como bytes brutos. Por exemplo, a chamada

```
char buffer [] = “FELIZ ANIVERSÁRIO’;
cout.write(buffer, 10);
```

envia para a saída os primeiros 10 bytes de buffer (inclusive caracteres nulos, que fariam a saída com cout e « terminar). Como um string de caracteres é avaliado como o endereço de seu primeiro caractere, a chamada

```
cout.write ( “ABCDEFGHIJKLMNOPQRSTUVWXYZ”, 10 );
```

exibe os primeiros 10 caracteres do alfabeto.
A função membro read lê um número especificado de caracteres para um array de caracteres. Se o número de caracteres lidos é menor que o especificado, failbit é “ligado”. Logo veremos como determinar se failbit foi “ligado” (ver Seção 11.8). A função membro gcount informa o número de caracteres lidos pela última operação de entrada.
A Fig. 11.15 demonstra as funções membro de istream read e gcount,e a função membro write de ostrea!n. O programa recebe como entrada 20 caracteres (de uma seqüência de entrada mais longa) para o array de caracteres buffer com o uso de read. determina o número de caracteres fornecidos como entrada com gcount e envia para a saída os caracteres em buffer usando write.

Fig. 11.15 EIS não-formatada com as funções membro read, gcount e write (parte 1 de 2).

```
1  II Fig. 11.15: figlll5.cpp
2  /1 E/S não-formatada com  read, gcount e write.
3  #include <iostream>
4
5  using std: :cout;
6  using std: :cin;
7  using std: :endl;
8
9  int main()
10 {
11 const int SIZE = 80;
```

1 2	char buffer[SIZE];	


```
13
14 cout « “Digite urna frase:\n”;
15 cin.read( buffer, 20 );
16 cout « “\nA frase digitada é:\n’;
17 cout.write( buffer, cin.gcount() );
18 cout « endi;
19 return 0;
20
Digite urna frase:
Usando as funções membro read, write e gcount
A frase digitada é:
Usando as funções me
```

Fig. 11.15 EIS não-formatada com as funções membro read, gcount e write (parte 2 de 2).

11.6 Manipuladores de streams

C++ fornece vários manipuladores de streams que executam tarefas de formatação. Os manipuladores de streams 4 fornecem recursos tais como definição de larguras de campo, definição de precisões, definição e redefinição de
indicadores de formato, definição do caractere de preenchimento de campo, esvaziamento de streams, inserção de um caractere nova linha no stream de saída, inserção de um caractere nulo no stream de saída e desconsideração de espaços em branco no stream de entrada. Estes recursos são descritos nas seções seguintes.

11.6.1 Base do stream de inteiros: dec, oct, hex e setbase

Inteiros são normalmente interpretados como valores decimais (base 10). Para mudar a base em que os inteiros são interpretados em um stream, insira o manipulador hex para definir a base como hexadecimal (base 16) ou insira o manipulador oct para definir a base como octal (base 8). Insira o manipulador de stream dec para redefinir a base do stream como decimal.

A base de um stream pode também ser mudada pelo manipulador de stream setbase, que aceita um parâmetro inteiro de valor 10, 8 ou 16 para definir a base. Como setbase aceita um parâmetro, ele é chamado de manipulador de stream parametrizado. Usar setbase ou qualquer outro manipulador parametrizado exige a inclusão do arquivo de cabeçalho <iomanip>. A base do stream permanece a mesma até que ela seja explicitamente mudada. A Fig. 11.16 mostra o uso dos manipuladores de stream hex. oct, dec e setbase.

```
1 II Fig. 11.16: figll_16.cpp
2 // Usando os manipuladores de stream hex, oct, dec e setbase.
3 #include <iostrearn>
4
5 using std::cout;
```

```
6 using std::cin;
7 using std::endl;
8
9 #include <iomanip>
10
11 using std::hex;
12 using std: :dec;
13 using std::oct;
14 using std::setbase;
15
```

Fig. 11.16 Usando os manipuladores de streamhex, oct, dec e setbase (parte 1 de 2).

```
Digite um n(imero decimal: 20
20 em hexadecimal é: 14
20 em octal é: 24
20 em decimal é: 20
```

Fig. 11.16 Usando os manipuladores de stream hex, oct, dec e setbase (parte 2 de 2).

11.6.2 Precisão em ponto flutuante (precision, setprecision)

Podemos controlar a precisão de números em ponto flutuante, i.e., o número de dígitos à direita do ponto decimal, usando ou o manipulador de stream setprecision ou a função membro precision. Uma chamada a qualquer um destes define a precisão para todas as operações de saída subseqüentes, até a próxima chamada para definição de precisão. A função membro precisiori sem parâmetro retorna a definição da precisão atual. O programa da Fig. 11.17 usa tanto a função membro precision como o manipulador setprecision para imprimir uma tabela mostrando a raiz quadrada de 2 com precisões variando de O a 9.

```
4


5
6
7
8

10

14

16
```

17
18
19
20
21

using std::cout; using std: :cin; using std::endl;

std: :ios;
std: : setiosflags; std: : setprecision;

double root2 = sqrt(2.0); int places

CAPÍTULO 11 - ENTRADA/SAÍDA COM STREAMS EM C++ 653

16 int main()
17 {
18 int n;
19
20 cout « “Digite um número decimal:
21 cm » n;
22
23 cout « n « “ em hexadecimal é:
24 « hex « n « ‘\n
25 « dec « n « em octal é:
26 « oct « n « ‘\n’
27 « setbase(10) « n « “ em decimal é:
28 « n « endl;
29
30 return 0;
31 }

1 II Fig. 11.17: figlll7.cpp
2 II Controlando a precisão de valores em
3 #include <iostream>

ponto

flutuante

9 #include <iomanip>

11 using
12 using

13 using

15 #include <cmath>

int main()

Fig. 11.17 Controlando a precisão de valores em ponto flutuante (parte 1 de 2).


```
22 cout « setiosflags( ios::fixed
23 « “Raiz quadrada de 2 com precisões de 0 a
24 « “Precisão inicializada pela
25 « “função membro precision:” « endl;
26
27 for ( places = 0; places <= 9; places++
28 cout.precision( places );
29 cout « root2 « ‘\n’;
30
31
32 cout « “\nPrecisão inicializada pelo
33 « “manipulador setprecision:\n”;
34
35 for ( places = 0; places <= 9; places++
36 cout « setprecision( places ) « root2 « ‘\n’;
37
38 return 0;
39
Raiz quadrada de 2 com precisões de O a 9.
Precisão inicializada pela função membro precision:
1 ê
1.4
1.41
1.414
1.4142
1.41421
1.414214
1.4142136
1.41421356
1.414213562
Precisão inicializada pelo manipulador setprecision:
1
1.4
1.41
1.414
1.4142
1.41421
```

1.414214
1.4142136
1.41421356
1.414213562
Fig. 11.17 Controlando a precisão de valores em ponto flutuante (parte 2 de 2).
11.6.3 Largura de campo (setw, width)
A função membro setw de ios define a largura de campo (ie., o número de posições de caracteres com que um valor deveria ser mostrado na saída ou o número de caracteres com que deveria ser lido) e retorna a largura anterior. Se os valores processados são menores que a largura de campo, caracteres de enchimento são inseridos como preenchimento. Um valor maior que a largura especificada não será truncado - o número completo será impresso.
Erro comum dc programação 11.4
A definição de setw se aplica somente à próxima inserção ou extração; posteriormente, setw é implicitainente inicializada com 0, i.e., os valores de saída simplesmente serão tão grandes quanto eles necessi

tam ser A função setw sem parâmetro retorna a definição atual. É um erro de lógica assumir que a definição de setw se aplica a todas as saídas subseqüentes.
Erro comum de programação 11.5
Quando não se fornecer um campo suficientemente grande para tratar a saída, a impressão da saída será tão grande quanto necessária, provavelmente tornando a saída difícil de ser lida.
A Fig. li . 18 demonstra o uso da função membro setw tanto com entradas como com saídas. Note que, na leitura para um array char, o número máximo de caracteres udos será um a menos do que a largura, porque é reservada uma posição para o caractere nulo que será colocado no string de entrada. Lembre-se de que a extração do stream termina quando espaços em branco não-iniciais são encontrados. O manipulador de stream setw também pode ser usado para definir a largura dos campos. Nota: quando o usuário é solicitado a fornecer uma entrada, o usuário deve digitar uma linha de texto e apertar Enter seguido por fim de arquivo (<ctrl>-z em sistemas compatíveis com o IBM PC, <ctrl>-d em sistemas UNIX e Macintosh). Nota: ao se ler qualquer coisa que não seja um array char, width e setw são ignoradas.

```
1 II figlll8.cpp
2 II Demonstrando a função membro width
3 #include <iostream>
4
5 using std::cout;
6 using std::cin;
7 using std: :endl;
8
9 int main()
```

```
10
cout « “Digite uma frase:\n”;
cin.width( 5 );
while ( cm » string
cout.width( w++ );
cout « string « endi;
cin.width( 5 );
23 return 0;
24
Digite uma frase:
Este programa testa a função membro width
Es te
prog
rama
tes t

‘4,

int w = 4;
char string[ 10 );

11
12
13
14
15
16
17
18
19
20
21
22

a
a
funç
ao
memb
ro
widt
h
```

Fig. 11.18 Demonstrando a função membro width.

11.6.4 Manipuladores definidos pelo usuário

Os usuários podem criar seus próprios manipuladores de stream. A Fig. 11.19 mostra a criação e uso dos novos manipuladores de stream beli, ret (retorno de carro). tab e endLine. Os usuários também podem criar seus próprios manipuladores de stream parametrizados consulte os manuais de sua instalação para instruções sobre como fazê-lo.

```
1 II Fig. 11.19: figll_19.cpp
2 // Criando e testando manipuladores de strealTL
3 II não-parametrizados, definidos pelo usuário.
4 #include <iostream>
5
6 using std: :ostream;
7 using std::cout;
8 using std::flush;
9
10 // manipulador bell (usando a sequência de escape \a)
11 ostream& bell( ostream& output ) { return output « \a';
12
13 // manipulador ret (usando a seqüência de escape \r)
14 ostream& ret( ostream& output ) { return output « '\r';
15
16 II manipulador tab (usando a seqüência de escape \t)
17 ostream& tab( ostream& output ) { return output « '\t';
18
19 II manipulador endLine (usando a seqüência de escape \n
20 // e a função membro flush)
21 ostream& endLine( ostream& output
22
23 return output « '\n' « flush;
24
25
26 int main()
27
28 cout « "Testando o manipulador tab:" « endLine
29 « 'a' « tab « 'b' « tab « 'c' « endLine
30 « "Testando os manipuladores ret e beli:"
31 « endLine «
32 cout « bell;
33 cout « ret « "-' « endLine;
34 return 0;
35
Testando o manipulador tab:
a b c
Testando os manipuladores ret e bell:
```

Fig. 11.19 Criando e testando manipuladores de stream não-parametrizados

definidos pelo usuário.
11.7 Estados de formato do stream
Vários indicadores de formato especificam os tipos de formatação a serem executados durante as operações de EI S com streams. As funções membro setf, unsetf e flags controlam as definições destes indicadores.

11.7.1 Indicadores de estado de formato
Cada um dos indicadores de estado de formato mostrados na Fig. 11.20 (e alguns que não são mostrados) é definido como uma enumeração na classe ios e explicado nas próximas seções.

[Indicador de estado de formato Descrição

ios: skipws Ignora espaços em branco em um stream de entrada.
ios: left Justifica à esquerda a saída em um campo. Caracteres de preenchimento
aparecem à direita, se necessário.
ios: right Justifica à direita a saída em um campo. Caracteres de preenchimento aparecem à esquerda, se necessário.
ios: internal Indica que o sinal de um número deve ser alinhado à esquerda em um
campo e a magnitude do número deve ser alinhada à direita nesse mes m campo (i.e., caracteres de preenchimento aparecem entre o sinal e o número).
ios: : dec Especifica que inteiros devem ser tratados como valores decimais (base
10).
ios: : oct Especifica que inteiros devem ser tratados como valores octais (base 8).
ios: hex Especifica que inteiros devem ser tratados como valores hexadecimais (base 16).
ios: showbase Especifica que a base de um número deve ser mostrada à frente do nú mer na saída (um o à esquerda para octais; um Ox ou OX para hexadecimais).
ios: : showpoint Especifica que números em ponto flutuante devem ser mostrados na sa íd com um ponto decimal. Isto é normalmente usado com ios: fixed
para garantir um certo número de dígitos à direita do ponto decimal.
ios: : uppercase Especifica que letras maiúsculas (i.e., X e A até F) devem ser usadas nos
inteiros hexadecimais e que E maiúsculo deve ser usado ao se represen ta um valor em ponto flutuante na notação científica.
ios: : showpos Especifica que números positivos e negativos devem ser precedidos por
um sinal + ou -, respectivamente.

ios: : scientific Especifica que a saída de um valor em ponto flutuante deve ser mostrada
na notação científica.
ios: : fixed Especifica que a saída de um valor em ponto flutuante deve ser mostrada
na notação de ponto fixo, com o número especificado de dígitos à direita
do ponto decimal.
Fig.11.20 Indicadores de estado de formato.
Estes indicadores podem ser controlados pelas funções membro flags, setf e unsetf, mas muitos programadores de C++ preferem usar manipuladores de streams (ver Seção 11.7.8). O programador poder usar a operação “ou” sobre bits, 1, para combinar várias opções em um único valor long (ver Fig. 11.23). Chamar a função membro flags para um stream e especificar as opções combinadas através de um “ou” define as novas opções para aquele streum e retorna um valor long contendo as opções anteriores. Este valor freqüentemente é salvo, para que flags possa ser chamada com o valor salvo para restabelecer as opções anteriores para o stream.
A função flags deve especificar um único valor que represente as definições de todos os indicadores. A
função setf de um argumento, por outro lado, especifica um ou mais indicadores combinados através de “ou” e faz o “ou” deles com as definições de indicadores já existentes, para formar um novo estado de formato.

O manipulador de stream parametrizado setiosflags executa as mesmas funções que a função membro setf. O manipulador de stream resetiosflags executa as mesmas funções que a função membro unsetf. Para usar qualquer um destes manipuladores de stream, não esqueça de incluir <iomanip> em seu programa.
O indicador skipws indica que » deve ignorar espaços em branco em um stream de entrada. O comportamento default de » é ignorar espaços em branco. Para mudar isto, use a chamada unsetf (ios: : skipws) . O manipulador de stream ws também pode ser usado para especificar que espaços em branco devem ser ignorados.
11.7.2 Zeros à direita e pontos decimais (ios: : showpoint)
O indicador showpoint é inicializado para forçar um niimero em ponto flutuante a ser mostrado na saída com seu ponto decimal e zeros à direita. Um valor em ponto flutuante de 79 » O será impresso como 79 se showpoint não estiver inicializado e como 79.000000 (ou tantos zeros à direita quanto os especificados pela precisão atual) com showpoint inicializado. O programa na Fig. 11.21 mostra o uso da função membro setf para inicializar o indicador showpoint para controlar os zeros à direita e a impressão do ponto decimal para valores de ponto flutuante.
```
1 II Fig. 11.21: figll2l.cpp
2 II Controlando a impressão de zeros à direita e de
3 II pontos decimais para valores em ponto flutuante.
4 #include <iostream>
```

```
5
6 using std: :cout;
7 using std::endl;
8
9 #include <iomanip>
10
11 using std: :ios;
12
13 int main()
14
15 cout « “Antes de ajustar o indicador ios::showpoint\n”
16 « “9.9900 é impresso como: “ « 9.9900
17 « “\n9.9000 é impresso como: “ « 9.9000
18 « ‘\n9.0000 é impresso como: “ « 9.0000
19 « “\n\nDepois de ajustar o indicador ios::showpoint\n”;
20 cout.setf( ios::showpoint );
21 cout « “9.9900 é impresso como: “ « 9.9900
22 « “\n9.9000 é impresso como: “ « 9.9000
23 « “\n9.0000 é impresso como: “ « 9.0000 « endi;
24 return 0;
25 }
```

Fig. 11.21 Controlando a impressão de zeros à direita e de pontos decimais com valores float.

Antes	de	ajustar o indicador ios::showpoint		
9.9900	é	impresso	como: 9.99	
9.9000	é	impresso	como: 9.9	
9.0000	é	impresso	como: 9	
Depois	de	ajustar	o indicador ios:	: showpoint
9.9900	é	impresso	como: 9.99000	
9.9000	é	impresso	como: 9.90000	
9.0000	é	impresso	como: 9.00000	

11.7.3 Alinhamento (ios: :left, ios: :right, ios: :internal)

Os indicadores left e right possibilitam que campos sejam alinhados (oujustificados) à esquerda com caracteres de preenchimento à direita, ou alinhados à direita com caracteres de preenchimento à esquerda, respectivamente. O caractere a ser usado para preenchimento é o especificado pela função membro Liii ou pelo manipulador de stream parametrizado setfiiJ. (veja a Seção 11.7.4). A Fig. 11.22 mostra o uso dos manipuladores setw, setiosflags e resetiosfJ.ags e das funções membro setf e unsetf para controlar o alinhamento de dados inteiros à esquerda e à direita em um campo.

```
1 // Fig. 11.22: figll22.cpp
2 // Alinhamento à esquerda e alinhamento à direita.
3 #include <iostream>
4
5 using std: :cout;
6 using std: :endl;
7
8 #include <iomanip>
9
10 using std::ios;
11 using std::setw;
12 using std: :setiosflags;
13 using std::resetiosflags;
14
15 int main()
16
17 int x = 12345;
18
19 cout « “O default é justificado à direita:\n”
20 « setw(10) « x « “\n\nUSANDO FUNÇÕES MEMBRO’
21 « “\nUsa setf para ajustar ios::left:\n” « setw(10);
22
23 cout.setf( ios::left, ios::adjustfield );
24 cout « x « “\nUsa unsetf para restaurar default:\n”;
25 cout.unsetf( ios::left );
26 cout « setw( 10 ) « x
27 « “\n\nUSANDO MANIPULADORES DE STREAM PARAMETRIZADOS”
28 « “\nUsa setiosfiags para ajustar ios::ieft:\n”
29 « setw( 10 ) « setiosflags( ios::left ) « x
30 « \nUsa resetiosflags para restaurar default:\n”
31 « setw( 10 ) « resetiosflags( ios::left
32 « x « endl;
33 return 0;
34 )
```

```
O default é justificado à direita:
12345
USANDO FUNÇÕES MEMBRO
Usa setf para ajustar ios::left:
```

12345
Usa unsetf para restaurar defauit:
12345
USANDO MANIPULADORES DE STREAM PARAMETRIZADOS
Usa setiosflags para ajustar ios::left:
12345
Usa resetiosflags para restaurar default:
12345
Fig. 11.22 Alinhamento à esquerda e alinhamento à direita.

O indicador internal indica que o sinal de um número (ou a base, quando o indicador ios: : showbase está inicializado; veja a Seção 11.7.5) deve ser alinhado à esquerda dentro de um campo, a magnitude do número deve ser alinhada à direita e os espaços intervenientes devem ser preenchidos com o caractere de enchimento. Os indicadores left, right e internal estão contidos no membro de dados estático ios: : adjustfield. O argumento ios: : adjustfield deve ser fornecido como o segundo argumento para setf ao se inicializar os indicadores de alinhamento left. right ou internal. Isto possibilita a setf garantir que somente um dos três indicadores de alinhamento é inicializado (eles são mutuamente exclusivos). A Fig. 11.23 mostra o uso dos manipuladores de stream setiosflags e setw para especificar espaçamento interno. Note o uso do indicador ios: : showpos para forçar a impressão do sinal de mais.

```
1 II Fig. 11.23: figll_23.cpp
2 // Imprimindo um inteiro com espaçamento
3 // interno e forçando o sinal de mais.
4 #include <iostream>
5
6 using std: :cout;
7 using std: :endl;
8
9 #include <iomanip>
10
11 using std::ios;
12 using std: :setiosflags;
13 using std::setw;
14
15 int main()
16
17 cout « setiosflags( ios::internal 1 ios::showpos
18 « setw( 10 ) « 123 « endl;
19 return 0;
20 }
```

```
+ 123
```

Fig. 11.23 Imprimindo um inteiro com espaçamento interno e forçando o sinal de mais.

11.7.4 Preenchimento (fill, setfill)

Afunção membro fill especifica o caractere de enchimento a ser usado com campos justificados; se nenhum valor for especificado, são usados espaços para o preenchimento. A função fili retorna o caractere de preenchimento anterior. O manipulador setfill também inicializa o caractere de preenchimento. A Fig. 11.24 demonstra o uso da função membro fill e do manipulador setfill para controlar a definição e redefinição do caractere de enchimento.

Fig. 11.24 Usando a função membro fill e o manipulador setfill para mudar o caractere de preenchimento para campos maiores do que os valores que estão sendo impressos (parte 1 de 2).

1	II Fig. 11.24: figll_24.cpp			
2	// Usando a função membro fill e	o manipulador		
3	// setfill para mudar o caractere	de preenchiment o		
4	// para campos maiores do que os	valores que estão	sen do	impress os.
5	#include <iostream>			
6				
7	using std::cout;			
8	using std::endl;			

O indicador internal indica que o sinal de um número (ou a base, quando o indicador ios: : showbase está inicializado; veja a Seção 11.7.5) deve ser alinhado à esquerda dentro de um campo, a magnitude do número deve ser alinhada à direita e os espaços intervenientes devem ser preenchidos com o caractere de enchimento. Os indicadores left, right e internal estão contidos no membro de dados estático ios: adustfie1d. O argumento jos: : adustfield deve ser fornecido como o segundo argumento para setf ao se inicializar os indicadores de alinhamento left, right ou internal. Isto possibilita a setf garantir que somente um dos três indicadores de alinhamento é inicializado (eles são mutuamente exclusivos). A Fig. 11.23 mostra o uso dos manipuladores de stream setíosflags e setw para especificar espaçamento interno. Note o uso do indicador ios: :

showpos para forçar a impressão do sinal de mais.

```
1 1/ Fig. 11.23: figll_23.cpp
2 // Imprimindo um inteiro com espaçamento
3 /1 interno e forçando o sinal de mais.
4 #include <iostreani>

5

6 using std::cout;
7 using std::endl;

8

9 #include <iomanip>
10

11 usíng std: :ios;
12 using std::setiosflags;
13 using std::setw;

14

int main()

{

1

cout « setiosflags( ios::internal 1 ios::showpos « setw( 10 ) « 123 « endl;
return 0;
```

Fig. 11.24 Usando a função membro fill e o manipulador setfíll para mudar o caractere de preenchimento para campos maiores do que os valores que estão sendo impressos (parte 1 de 2).

```
15
16
17
18
19
20
```

+ 123

Fig. 11.23 imprimindo um inteiro com espaçamento interno e forçando o sinal de mais.

11.7.4 Preenchimento (f iii, setf iii)

Afunção membro fi 11 especifica o caractere de enchimento a ser usado com campos justificados; se nenhum valor for especificado, são usados espaços para o preenchimento. A função f iii retorna o caractere de preenchimento anterior. O manipulador setfill também inicializa o caractere de preenchimento. A Fig. 11.24 demonstra o uso da função membro f ii]. e do manipulador setfill para controlar a definição e redefinição do caractere de enchimento.

cO°
se
que sao

1	1/ Fig. 11.24: figll_24.cpp			
2	// Usando a função membro fili e	o manipulador		
3	1/ setfíll para mudar o caractere	de preenchimento		
4	/1 para campos maiores do que os	valores que estão	send o	impresso s.
5	#include <iostream>			
6				
7	using std::cout;			
8	using std::endl;			


```
9
10 #include <iomanip>
11
12 using std::ios;
13 using std: :setw;
14 using std: :hex;
15 using std::dec;
16 using std::setfilJ.;
17
18 int main()
```

```
19 ( «
20 int x 10000;
21
22 cout « x « " impresso como int justificado à direita e à esquerda\n"
23 « "e como hex com justificação interna.\n"
24 « Usando o caractere de preenchimento default (espaço) : \n';
25 cout.setf( ios::showbase );
26 cout « setw( 10 ) « x « '\ri;
27 cout.setf( ios::left, ios::adjustfield );
28 cout « setw( 10 ) « x « \n';
29 cout.setf( ios: :internal, ios: :adjustfield );
30 cout « setw( 10 ) « hex « x;
31
32 cout « "\n\nUsando vários caracteres de preenchimento:\n;
33 cout.setf( ios::right, ios::adjustfield );
34 cout.fill( '*'
35 cout « setw( 10 ) « dec « x « '\n';
36 cout.setf( ios::left, ios::adjustfield );
37 cout « setw( 10 ) « setfill( '% ) « x « \n';
38 cout.setf( ios::internal, ios::adjustfield );
39 cout « setw( 10 ) « setfill( ' ) « hex « x « endi;
40 return 0;
41
10000 impresso como int justificado à direita e à esquerda
e como hez com justificação interna.
Usando o caractere de preenchimento default (espaço)
10000
10000
Ox 2710
Usando vários caracteres de preenchimento:
*****10000
10000%%%%%
0xAA271O
```

Fig. 11.24 Usando a função membro fiJJ. e o manipulador setfill para mudar o caractere de preenchimento para campos maiores do que os valores que estão sendo impressos (parte 2 de 2).

11.7.5 Base do streamde inteiros (ios: :dec, ios: :oct, ios: :hex, ios: :showbase)

O membro estático ios: : basefield (usado de forma semelhante a ios: : adjustfield com setf) inclui os bits indicadores ios: : oct, ios: : hex e ios: dec para especificar que inteiros devem tratados como valores octais, hexadecimais e decimais, respectivamente. As inserções no stream são, por default, decimais, se nenhum destes bits estiver "Jigado". O default para extrações do síream é processar os dados na forma em que são fornecidos - inteiros começando com O são tratados como valores octais, inteiros que começam com Ox ou OX são

tratados como valores hexadecimais, e todos os outros valores inteiros são tratados como decimais. Uma vez que uma base particular tenha sido especificada para um stream, todos os inteiros naquele stream serão processados com aquela base, até uma nova base ser especificada, ou até o fim do programa.

Inicialize o indicador showbase para forçar a base de um valor inteiro a ser mostrada na saída. Os números decimais são mostrados na saída normalmente, números octais são mostrados na saída com um O à esquerda, e números hexadecimais são mostrados na saída com um Ox à esquerda ou um ox à esquerda (o indicador uppercase determina qual a opção escolhida; veja a Seção 11.7.7). A Fig. 11.25 demonstra o uso do indicador showbase para forçar que um inteiro seja impresso nos formatos decimal, octal e hexadecimal.

```
1 // Fig. 11.25: figll25.cpp
2 // Usando o indicador ios: :showbase
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
1
8 #include <iomanip>
9
10 using std::ios;
11 using std: :setiosflags;
12 using std::oct;
13 using std::hex;
14
15 int main()
16 {
17 int x = 100;
18
19 cout « setiosflags(
20 « “Imprimindo inteiros precedidos por sua nase:\n’
21 « x « ‘\n’
22 « oct « x « \n’
23 « hex « x « endi;
24 return 0;
25 )
```

```
Imprimindo inteiros precedidos por sua base:
100
0144
0x64
```

Fig. 11.25 Usando o indicador ios: : showbase.

11.7.6 Números em ponto flutuante; notação científica (ios: : scientific, ios: : fixed)

O indicador ios: : scientific e o indicador ios: : fixed estão contidos no membro de

dados estático
ios: : floatfield (estes indicadores são usados de maneira semelhante a ios: : adustfie1d e
ios: : basefield em setf). Estes indicadores controlam o formato de saída dos números em ponto flutuante.
O indicador scientific força a saída de um número em ponto flutuante no formato científico. O indicador
fixed força um número em ponto flutuante a ser mostrado na saída com um número específico de dígitos (conforme especificado pela função membro precision) à direita do ponto decimal. Sem estes indicadores "ligados".
o valor do número em ponto flutuante determina o formato de saída.
A chamada cout. setf (0, ios: : floatfield) restabelece o formato default para a saída de números em ponto flutuante. A Fig. 11.26 demonstra a exibição de números em ponto flutuante, nos formatos fixo e científi L

co, usando a setf de dois argumentos com ios: : floatfield. O formato do expoente em notação científica pode ser diferente de um compilador para outro.

```
1 II Fig. 11.26: figll26.cpp
2 II Exibindo valores em ponto flutuante nos
3 // formatos default do sistema, científico e fixo.
4 #include <iostream>
5
6 using std: :cout;
7 using std::endl;
8 using std::ios;
9
10 int main()
11
12 double x = .001234567, y = 1.946e9;
13
14 cout « "Exibido no formato default:\n"
15 « x « '\t' « y « '\n';
16 cout.setf( ios::scientific, ios::floatfield );
17 cout « "Exibido no formato científico:\n"
18 « x « '\t' « y « '\n';
19 cout.unsetf( ios::scientific );
20 cout « "Exibido no formato default após unsetf:\n"
21 « x « '\t' « y « '\n';
22 cout.setf( ios::fixed, ios::floatfield );
23 cout « "Exibido no formato fixo:\n"
24 « x « '\t' « y « endl;
25 return 0;
26
Exibido no formato default:
0.00123457 l.946e+009
```

Exibido no formato científico:
1. 234567e-003 1. 946000e+009
Exibido no formato default após unsetf:
0.00123457 1.946e+009
Exibido no formato fixo:
0.001235 1946000000.000000
Fig. 11.26 Exibindo valores em ponto flutuante nos formatos default, científico e fixo do sistema.

11.7.7 Controle de maiúsculas/minúsculas (ios: : uppercase)

O indicador ios: : uppercase força a saída de uma letra maiúscula X ou E com inteiros hexadecimais ou com valores em ponto flutuante na notação científica, respectivamente (Fig. 11.27). Quando inicializado, o indicador ios: : uppercase faz com que todas as letras em um valor hexadecimal sejam maiúsculas.

```
1 // Fig. 11.27: figll27.cpp
2 // Usando o indicador ios: :uppercase
3 #include <iostream>
4
5 using std: :cout;
6 using std: :endl;
```

Fig. 11.27 Usando o indicador ios: : uppercase (parte 1 de 2).


```
7
8 #include <iomanip>
9
10 using std::setiosflags;
11 using std::ios;
12 using std::hex;
13
14 int main()
15 {
16 cout « setiosflags( ios::uppercase
17 « “Imprimindo letras maiúsculas em expoentes na\n”
18 « “notação científica e em valores hexadecimais:\n”
19 « 4.345e10 « ‘\n’ « hex « 123456789 « endi;
20 return 0;
21 }
```

```
Imprimindo letras maiúsculas em expoentes na
notação científica e em valores hexadecimais:
4. 345E+010
75BCD15
```

Fig. 11.27 Usando o indicador ios: :uppercase (parte 2 de 2).

11.7.8 Inicializando e reinicializando os indicadores de formato (flags. setiosflags.
resetiosflags)

A função membro flags sem um argumento simplesmente retorna (como um valor long) as definições atuais dos indicadores de formato. A função membro flags com um argumento long inicializa os indicadores de formato conforme especificado pelo argumento e retorna as definições anteriores dos indicadores. Quaisquer indicadores de formato não-especificados no parâmetro são “desligados”. Note que as definições iniciais dos indicadores podem diferir para cada sistema, O programa da Fig. 11.28 demonstra o uso da função membro flags para inicializar um novo estado de formato e salvar o estado de formato anterior e, depois, restaurar as definições de formatos originais.

```
1 II Fig. 11.28: figll_28.cpp
2 // Demonstrando a função membro flags.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7 using std::ios;
8
9
10 int main()
11 {
12 int i = 1000;
13 double d = 0.0947628;
14
15 cout « “O valor da variável flags é:
16 « cout.flags()
17 « ‘\nlmprime int e double no formato original:\n”
18 « i « ‘\t’ « d « “\n\n”;
19 long originalFormat =
20 cout.flags( ios::oct ios::scientific );
21 cout « “O valor da variável flags é:
```

Fig. 11.28 Demonstrando a função membro flags (parte 1 de 2).

1

```
22 « cout.flags()
23 « \nlmprime int e double em um novo formato\n”
24 « especificado usando a função membro flags:\n”
25 « i « ‘\t’ « d « “\n\n”;
26 cout.flags( originalFormat );
27 cout « O valor da variável flags é:
28 « cout.flags()
29 « “\nlmprime int e double novamente no
30 « i « ‘\t’ « d « endi;
31 return 0;
```

32

Fig. 11.28 Demonstrando a função membro flags (parte 2 de 2).
A função membro setf inicializa os indicadores de formato fornecidos em seu argumento e retorna as definições anteriores dos indicadores como um valor long, como em
long previousFlagSettings =
cout.setf(ios: :showpoint 1 ios: :showpos);
A função membro setf com dois parâmetros long, como em
cout.setf(ios: :left, ios: :adjustfield);
primeiro “limpa” os bits de ios: : adjustfield e depois inicializa o indicador ios: : left. Esta versão de setf é usada com os campos de bits associados com ios: : basef ield (representados por ios: : dec, ios: : oct e ios: :hex), ios: :floatfield (representados por ios: :scientific e ios: :fixed) e ios: : adjustfield (representados por ios:: left, ios: : right e ios: : internal).
A função membro unsetf reinicializa os indicadores especificados e retorna o valor dos indicadores antes
de serem reinicializados.
11.8 Estados de erro do stream
O estado de um stream pode ser testado através de bits da classe ios - a classe base para as classes istreain. ostream e iostream que estamos usando para EIS.
O eofbit é “ligado” para um stream de entrada quando é encontrado o fim de arquivo. Um programa pode
usar a função membro eof para determinar se o fim de arquivo foi encontrado em um stream depois de uma
tentativa de extrair dados além do fim do stream. A chamada
cin.eof O

formato original: \n”

er

:15
ro

as

retorna verdadeiro se o fim de arquivo foi encontrado em cm e falso caso contrário.

O valor da variável flags é: O	
Imprime int e double no formato original:	
1000 0.0947628	

O valor da variável flags é: 4040	
Imprime int e double em um novo formato	
especificado usando a função membro flags:	
1750 9.476280e-002	
O valor da variável flags é: O	
Imprime int e double novamente no formato original:	
1000 0.0947628	

O failbit é “ligado” para um siream quando um erro de formato acontece no stream. Por exemplo, ocorre um erro de formato quando o programa está lendo inteiros e um caractere diferente de dígito é encontrado no stream de entrada. Quando ocorre um erro como este, os caracteres não são perdidos. A função membro fail informa se uma operação de stream falhou; normalmente é possível se recuperar tais erros.

O badbit é “ligado” para um stream quando acontece um erro que resulta na perda de dados. A função membro bad informa se uma operação de stream falhou. Tais erros sérios normalmente não são recuperáveis.

O goodbit é “ligado” para um stream se nenhum dos bits eofbit. failbit ou badbit estão “ligados” para o stream.

A função membro good retorna true se as funções bad, fail e eof retornaram false. As operações de EIS somente devem ser executadas em streams com o estado “bom”, ou seja, com o goodbit “ligado”.

A função membro rdstate retorna o estado de erro do stream. Uma chamada cout. rdstate. por exempio, retornaria o estado do stream, que poderia então ser testado por um comando switch que examinaria ios: :eofbit, ios: :badbit, ios: : failbit e ios: :goodbit. O meio preferido detestar o estado de
um stream é usar as funções membro eof, bad, fail e good - usar estascfunções não exige que o programador esteja familiarizada com bits de estado particulares.

A função membro clear é normalmente usada para restabelecer o estado de um stream para “bom”, de forma que a E/S possa continuar naquele stream. O argumento defauit para clear é ios: : goodbit, de modo que o comando
cin.clearO;
limpa cm e “liga” goodbit para o stream. O comando
cin.clear(ios::failbit
inicializa o failbit. O usuário pode querer fazer isto quando estiver executando uma entrada com cm com um tipo definido pelo usuário e encontrar um problema. O nome clear parece impróprio neste contexto, mas é correto.

O programa da Fig. 11.29 ilustra ouso das funções membro rdstate, eof, fail. bad. good e clear. Nota: os valores realmente exibidos podem variar de um compilador para outro.

```
1 II Fig. 11.29: figll_29.cpp
2 // Testando estados de erro.
3 #include <iostream>
```

```
4
5 using std::cout;
6 using std: :endl;
7 using std::cin;
8
9 int main()
10
11 int x;
12 cout « “Antes de uma operação de entrada com problema:
13 « “\ncin.rdstate(): “ « cin.rdstate()
14 « “\n cin.eof O: “ « cin.eof O
15 « “\n cin.fail() : “ « cin.fail()
16 « “\n cin.badO: “ « cin.bad()
17 « “\n cin.goodO: “ « cin.good()
18 « “\n\nEspera receber um inteiro, mas digite um caractere:
19 cm » x;
20
21 cout « “\nApós uma operação de entrada com problema:”
22 « “\ncin.rdstateO: “ « cin.rdstate()
```

Fig. 11.29 Testando estados de erro (parte 1 de 2).


```
23 « ‘\n cin.eof: “ « cin.eof O
24 « “\n cin.fail() : “ « cin.fail()
25 « ‘\n cin.badO: “ « cin.bad()
26 « \n cin.good() : ‘ « cin.good() « “\n\n”;
27
28 cin.clearO;
29
30 cout « “Após cin.clearO”
31 « “\ncin.failO: “ « cin.fail()
32 « “\ncin.goodO: “ « cin.good() « endl;
33 return 0;
34
ia
le Antes de uma operação de entrada com problema:
cin.rdstateO: O
cin.eofO: O
le cin.failO: O
lo cin.bad() : O
cin.goodO: 1
Espera receber um inteiro, mas digite um caractere: A
Após uma operação de entrada com problema:
cin.rdstateO: 2
cin.eof: O
cin.failO: 1
```

cin.badO: O
cin.goodO: O
Após cin.clear()
cin.failO: O
cin.good(): 1
Fig. 11.29 Testando estados de erro (parte 2 de 2).
A função membro operator! retorna true se ou o badbit está “ligado”, ou o failbit está “ligado” ou ambos estão “ligados”. A função membro operador void* retorna false (O) se ou o badbit está “ligado”, ou o failbit está “ligado” ou ambos estão “ligados”. Essas funções são úteis no processamento de arquivos, quando uma condição verdadeiro/falso está sendo testada sob o controle de uma estrutura de seleção ou de uma estrutura de repetição.
11.9 Vinculando um stream de saída a um stream de entrada
Aplicativos interativos envolvem geralmente um istream para entrada e um ostream para saída. Quando a mensagem de prompt aparece na tela, o usuário responde digitando os dados apropriados. Obviamente, o prompt precisa aparecer antes de a operação de entrada ocorrer. Com a buiferização da saída, os dados de saída aparecem somente quando o buffer encher, quando a saída for esvaziada explicitamente pelo programa ou, automaticamente, no fim do programa. C++ fornece a função membro tie para sincronizar (i.e., “amarrar uma à outra”) a operação de um istream e um ostream. para garantir que a saída apareça antes de sua subseqüente entrada. A chamada
cin.tie(&cout);
amarra cout (um ostream) a cm (um istream). Na realidade, esta chamada em particular é redundante. porque C++ executa esta operação automaticamente para criar um ambiente padrão de entrada/saída do usuário. O

usuário, contudo, amarra explicitamente entre si outros pares de istream/ostream. Para desamarrar um stream
de entrada, inputStream. de um stream de saída, use a chamada
inputStream.tie(O);
Resumo
As operações de EIS são executadas de uma maneira sensível ao tipo dos dados.
• A E/S em C++ acontece em streams de bytes. Um stream é simplesmente uma seqüência de bytes.
Os mecanismos de E/S do sistema movem bytes dos dispositivos para a memória e vice-versa, de uma maneira eficiente e confiável.
C++ fornece recursos de EIS “de baixo nível” e “alto nível”. Os recursos de EIS de baixo nível especificam que um certo número de bytes deve ser transferido de um dispositivo para a memória ou da memória para um dispositivo. A E/S de alto
nível é executada com bytes agrnpados em unidades com significado, tais como inteiros, números em ponto flutuante, caracteres, strings e tipos definidos pelo usuário.

• C++ fornece tanto operações de E/S não-formatadas como formatadas. As transferências de E/S não-formatadas são rápidas, mas processam dados brutos difíceis para as pessoas usarem. A E/S formatada de dados processa os mesmos em unidades com significado, mas exige tempo de processamento extra que pode impactar negativamente transferências de grandes volumes de dados.
• A maioria dos programas em C++ incluem o arquivo de cabeçalho <iostream>, que declara todas as operações de EIS com streams.
• O cabeçalho <iomanip> declara a entrada/saída formatada com manipuladores de streams parametrizados.
• O cabeçalho <fstream> declara as operações de processamento de arquivos.
• A classe istream suporta as operações de entrada com streams.
• A classe ostream suporta as operações de saída com streams.
• A classe iostream suporta tanto operações de entrada com streams como a saída com streams.
• A classe istream e a classe ostream são derivadas por herança simples da classe base ios.
• A classe iostream é derivada por herança múltipla tanto da classe istream e como da classe ostream.
• O operador de deslocamento à esquerda («) é sobrecarregado para especificar saída com streams e é chamado de operador de inserção em stream.
• O operador de deslocamento à direita (») é sobrecarregado para especificar entrada com stream e é chamado de operador de extração de stream.
• O objeto cm de istream é associado ao dispositivo padrão de entrada, normalmente o teclado.
• O objeto cout da classe ostream é associado ao dispositivo de saída, normalmente a tela.
• O objeto cerr da classe ostream é associado ao dispositivo de erro padrão. Os dados enviados para saída em cerr não são colocados em um buffer; cada inserção em cerr aparece imediatamente na saída.
• O manipulador de stream endi envia um caractere nova linha e esvazia o buifer de saída.
• O compilador de C++ determina tipos de dados automaticamente para entrada e saída.
• Endereços são exibidos em formato hexadecimal por default.
• Para imprimir o endereço armazenado em uma variável ponteiro, faça a coerção do ponteiro para void*.
• A função membro put envia um caractere para a saída. As chamadas para put podem ser encadeadas.

• A leitura de um stream é executada com o operador de extração de stream ». Este operador ignora, automaticamente, caracteres em branco no stream de entrada.
• O operador » retorna false quando o fim de arquivo é encontrado em um

stream.
• Extração de um stream faz failbit ser “ligado” para dados de entrada impróprios e badbit ser “ligado” se a operação

falha.

• Uma série de valores pode ser fornecida como entrada usando-se o operador de extração de stream no cabeçalho de um laço while. A extração retorna O quando o fim de arquivo é encontrado.
• A função membro get sem argumentos recebe um caractere do stream de entrada e retorna este caractere; EOF é retornado se o fim de arquivo é encontrado no stream.
• A função membro get com um argumento do tipo referência para char recebe um caractere do stream de entrada . EOF é retornado quando o fim de arquivo é encontrado; caso contrário, o objeto istream para o qual a função membro get está sendo invocada é retomado.
• A função membro get com três argumentos - um array de caracteres, um limite de tamanho e um delimitador (com valor default nova linha) - lê caracteres do stream de entrada até um máximo de “limite de tamanho - 1” caracteres e termina, ou termina quando o delimitador é lido. O string de entrada é terminado com um caractere nulo. O delimitador não é colocado no array de caracteres, mas permanece no stream de entrada.
• A função membro getline opera como a função membro get de três argumentos. A função getline remove o delimitador
do stream de entrada, mas não o armazena no string.
• A função membro ignore ignora o número especificado de caracteres (o default é 1) no síream de entrada; ela termina se o delimitador especificado é encontrado (o delimitador default é EOF).
• A função membro putback coloca o último caractere obtido por um get de um stream de entrada de volta naquele stream.
• A função membro peek retorna o próximo caractere do stream de entrada, mas não remove o caractere do stream.
)
• C++ oferece EIS segura quanto ao tipo. Se dados inesperados são processados pelos operadores « e», vários indicadores são ligados para que o usuário possa fazer testes e determinar se uma operação de E/S teve sucesso ou falhou.
• A E/S não-formatada é executada com as funções membro read e write. Elas recebem da entrada e colocam na memória ou enviam da memória para a saída um certo número de bytes, iniciando a partir de um endereço de memória especificado. Eles são recebidos como entrada ou enviados para a saída como bytes brutos, sem formatação.
• A função membro gcount retorna o número de caracteres fornecidos como entrada pela operação read anterior naquele stream.

• A função membro read recebe como entrada um número especificado de

caracteres para um array de caracteress. failbit é “ligado” se são lidos menos caracteres que o número especificado de caracteres.
• Para mudar a base em que inteiros são mostrados na saída, use o manipulador hex para definir a base como hexadecimal (base 16) ou oct para definir a base como octal (base 8). Use o manipulador dec para redefinir a base como decimal. A base permanece a mesma até ser explicitamente mudada.
• O manipulador de tream parametrizado setbase também define a base para saída de inteiros. setbase aceita um parâmetro inteiro de valor 10, 8 ou 16 para definir a base.
• A precisão de valores em ponto flutuante pode ser controlada usando-se ou o manipulador de stream setprecision ou a função membro precision. Ambos inicializam a precisão para todas as operações subseqüentes de saída até a próxima chamada de inicialização de precisão. A função membro precision sem argumento retorna o valor da precisão atual.
• Os manipuladores parametrizados exigem a inclusão do arquivo de cabeçalho <ioxnanip>.
• A função membro width inicializa a largura do campo e retorna a largura anterior. Os valores menores que o campo são preenchidos com caracteres de enchimento. A definição da largura do campo somente se aplica para a próxima inserção ou extração; posteriormente, o comprimento do campo é implicitamente definido com O (valores subseqüentes serão mostrados

na saída tão grandes quanto necessário). Valores maiores que um campo são impressos em sua totalidade. A função width sem argumento retorna a definição atual da largura. O manipulador setw também inicializa a largura.
• Para a entrada, o manipulador de stream setw estabelece um tamanho máximo de string; se um string maior é digitado, a linha maior é quebrada em pedaços não maiores que o tamanho designado.
• Os usuários podem criar seus próprios manipuladores de streams.
• As funções membro setf. unsetf e flags controlam as definições de indicadores.

• O indicador skipws indica que » deve ignorar espaços em branco no stream de entrada. O manipulador de stream ws também salta por cima de espaços iniciais em branco em um stream de entrada.
• Os indicadores de formato são definidos como uma enumeração na classe ios.
• As funções membro flags e setf controlam indicadores de formato, mas muitos programadores de C++ preferem usar manipuladores de stream. A operação “ou” sobre bits, 1 pode ser usada para combinar várias opções em único valor long. Chamar a função membro flags para um stream e especificar estas opções combinadas por um ou define as opções para aquele stream e retorna um valor long contendo as opções anteriores. Este valor é freqüentemente salvo para que flags possa ser chamada com este valor salvo para restabelecer as opções do anteriores para o stream.
• A função flags deve especificar um único valor representando as definições de todos os indicadores. A função setf com um parâmetro, por outro lado,

automaticamente tome “ou” todos os indicadores especificados com as definições de indicadores existentes, para formar um novo estado de formato.
• O indicador showpoint é definido para forçar um número de ponto flutuante a ser mostrado na saída com um ponto decimal e um número de dígitos significativos especificado pela precisão.
• Os indicadores left e right fazem com que os campos sejam alinhados à esquerda com caracteres de preenchimento à direita, ou alinhados à direita com caracteres de preenchimento à esquerda.
• O indicador internal indica que o sinal de um número (ou base quando o indicador ios : showbase está “ligado”) deve ser alinhado à esquerda dentro de um campo, a magnitude deve ser alinhada à direita e os espaços intervenientes devem ser preenchidos com o caractere de enchimento.
• ios: adjustfield contém os indicadores left. right e internal.
• A função membro f iii especifica o caractere de enchimento a ser usado com campos alinhados left. right e internal (o default é espaço): o caractere de preenchimento anterior é retomado. O manipulador de stream setfill também inicializa o caractere de enchimento.
• O membro estático ios: basefield inclui os bits oct, hex e dec para especificar que inteiros devem tratados como valores octais, hexadecimais e decimais, respectivamente. O default para a saída de inteiros é decimal se nenhum destes bits estiver “ligado”; as extrações de stream processam os dados na forma em que os dados são fornecidos.
• Inicialize o indicador showbase para forçar a saída da base de um valor inteiro.
• O membro de dados estático ios: : floatfield contém os indicadores scientific e fixed. Inicialize o indicador scientific para mostrar um número de ponto flutuante na saída no formato científico. Inicialize o indicador fixed para mostrar um número de ponto flutuante na saída com a precisão especificada pela função membro precision.
• A chamada cout. setf (O, ios: : floatfield) restabelece o formato default para exibir números de ponto flutuante.
• Inicialize o indicador uppercase para forçar um X ou E maiúsculo a ser mostrado na saída com inteiros hexadecimais ou com valores de ponto flutuante em notação científica, respectivamente. Quando inicializado, o indicador ios : uppercase faz com que todas as letras em um valor hexadecimal estejam em maiúsculas.
• A função membro flags sem parâmetro retorna o valor long das definições atuais dos indicadores de formato. A função membro flags com um parâmetro long inicializa os indicadores de formato especificados pelo parâmetro e retoma as definições de indicadores anteriores.
• A função membro setf define os indicadores de formato em seu parâmetro e retorna as definições anteriores de indicadores como um valor long.

• A função membro setf (long setbits, long resetBits) “desliga” os bits de resetBits e então inicializa os bits em setBits.

• A função membro unsetf reinicializa os indicadores designados e retorna o valor dos indicadores antes de serem reinicializados.
• O manipulador de stream parametrizado setiosflags executa as mesmas funções que a função membro flags.
• O manipulador de strea,n parametrizado resetiosflags executa as mesmas funções que a função membro unsetf.
• O estado de um stream pode ser testado através de bits na classe íos.
• O eofbit é “ligado” para um streatn de entrada quando o fim de arquivo é encontrado durante uma operação de entrada. A função membro eof é usada para determinar se o eofbit foi ligado.
• O failbit é “ligado” para um stream quando ocorre um erro de formato no siream. Nenhum caractere é perdido. A função membro fail informa se uma operação com stream falhou; normalmente, é possível se recuperar tais erros.
• O badbit é “ligado” para um siream quando ocorre um erro que resulta em perda de dados. A função membro bad informa se uma operação com stream falhou. Tais falhas sérias normalmente não são recuperáveis.
• A função membro good retorna verdadeiro se as funções bad. fai]. e eof retomaram falso. Operações de EIS devem ser executadas somente sobre streams “bons”.
• A função membro rdstate retorna o estado de erro do siream.
• A função membro clear é usada normalmente para restabelecer o estado de um ,oreain para “bom”, para que a E/S possa continuar naquele strearn.
• C++ fornece a função membro tie para sincronizar operações em istream e ostreaiu. a fim de garantir que saídas apareçam antes das entradas subseqüentes.

Terminologia

arquivo de cabeçalho padrão <iomanip> formatação na memória
badbit função membro bad
caractere de preenchimento função membro clear
caractere default de enchimento (espaço) função membro eof
caracteres em branco função membro fail
cerr função membro f iii
cm função membro flags
classe fstream função membro flush
classe ifstream função membro gcount
classe ios função membro get
classe iostream função membro getline
classe istream função membro good
classe ofstreain função membro ignore
classe ostream função membro operador’
clog função membro operator void*
cout função membro peek
EIS formatada função membro precision
EIS não-formatada função membro put
E/S segura quanto a tipo função membro putback
endi função membro rdstate

entrada em stream função membro read
eofbit função membro setf
estados de formato função membro tie
failbit função membro unsetf
fim de arquivo função membro write

672 C++ CoMo PROGRAMAR

função membro ws indicadores de formato ios: :adjustfield

ios: :basefield ios: : fixed ios: floatfield ios: : internal ios: :scientific

ios: : showbase ios: : showpoint ios: showpos justificado à direita justificado à esquerda

largura
largura de campo
maiúsculas
manipulador s tream manipulador de stream dec manipulador de stream flush

Erros comuns de programação

manipulador de stream hex manipulador de stream oct manipulador de stream parametrizado

manipulador de stream resetiosflags manipulador de stream setbase
manipulador de stream setfill manipulador de stream setios flags manipulador de stream setprecision

manipulador de stream setw operador de extração de stream (») operador de inserção em stream («) precisão default preenchimento

saída em siream
skipws
streams definidos pelo usuário streams predefinidos O mais à esquerda (octal) Ox ou OX mais à esquerda (hexadecimal)

11.1 Tentar ler de um ostream (ou qualquer outro stream somente de saída).
11.2 Tentar escrever em um istream (ou qualquer outro stream somente de entrada).
11.3 Não usar parênteses para forçar a precedência apropriada, quando usar os operadores de precedência relativamente mais alta para inserção em stream («) ou extração de stream (»).
11.4 A definição de setw se aplica somente à próxima inserção ou extração;

posteriormente. setwé implicitamente inicializada com O, i.e., os valores de saída simplesmente serão tão grandes quanto eles necessitam ser. A função setw sem parâmetro
retorna a definição atual. E um erro de lógica assumir que a definição de setw se aplica a todas as saídas subseqüentes.
11.5 Quando não se fornecer um campo suficientemente grande para tratar a saída, a impressão da saída será tão grande quanto necessária, provavelmente tornando a saída difícil de ser lida.

Boas práticas de programação

11.1 Use exclusivamente a forma de EIS de C++ em programas em C++, apesar do fato de que a EIS ao estilo de C está disponível para os programadores de C ++.
11.2 Quando enviar expressões para a saída, coloque-as entre parênteses, para prevenir problemas de precedência de operadores entre os operadores na expressão e o operador «.

Dica de desempenho

11.1 Use EIS não-formatada para obter melhor desempenho no processamento de arquivos de grande volume.

Dica de portabilidade

11.1 Quando informar ao usuário sobre como terminar a entrada de dados pelo teclado, peça ao usuário que “digite fim de arquivo para terminar a entrada” em vez de solicitar <ctrl>-d (UNIX e Macintosh) ou <ctrl>-z (PC e VAX).

Observações de engenharia de software

11.1 O estilo de EIS de C++ é seguro quanto ao tipo.
11.2 C++ possibilita um tratamento comum da EIS de tipos primitivos e de tipos definidos pelo usuário. Este tipo de “comunalidade” facilita o desenvolvimento de software em geral e a reutilização de software em particular.

CAPÍTULO 11 - ENTRADA/SAÍDA COM STREAMS EM C++ 673 [
Exercícios de auto-revisão
11.1 Responda a cada um dos seguintes itens:
a) Operadores de streum sobrecarregados são freqüentemente definidos como funções de uma classe.
b) Os bits de alinhamento do formato que podem ser inicializados incluem _____, _____ e _____
c) A entrada/saída em ocorre como _____ de bytes.

d) Os manipuladores de stream parametrizados e podem ser usados para ligar e desligar indicadores de estado de formato.
e) A maioria dos programas em C++ deve incluir o arquivo de cabeçalho que contém informações básicas necessárias para todas as operações EIS com stream,v.
f) As funções membro _____ e ligam e desligam indicadores de estado de formato.
g) O arquivo de cabeçalho contém as declarações requeridas para executar formatação “na memória’.
h) Quando usar manipuladores parametrizados, deve ser incluído o arquivo de cabeçalho
i) O arquivo de cabeçalho contém as declarações requeridas para o processamento de arquivos controlado
pelo usuário.
j) O manipulador de stream _____ insere um caractere nova linha na saída do stream e esvazia o stream de saída.
k) O arquivo de cabeçalho é usado em programas que misturam os estilos de E/S de C++ e de C.
1) A função membro ostream _____ é usada para executar saída não-formatada.
m) Operações de entrada são sustentadas pela classe _____
n) A saída do stream de erro padrão é dirigida ou para o _____ ou para o objeto do stream.
o) As operações de saída são suportadas pela classe _____
p) O símbolo para o operador de inserção em streams é _____
q) Os quatro objetos que correspondem aos dispositivos padrão do sistema são _____, _____ e
r) O símbolo para o operador de extração do stream é_____
s) Os manipuladores de siream _____, _____ e especificam que inteiros devem ser exibidos em formato octal, hexadecimal e decimal, respectivamente.
t) A precisão default para exibir valores de ponto flutuante é _____
u) Quando inicializado, o indicador _____ faz com que números positivos sejam mostrados com um sinal de mais.
11.2 Afirme se os seguintes itens são verdadeiros ou falsos. Se a resposta for falso, explique por quê.
a) A função membro de stream flags () com um argumento long inicializa a variável de estado de flags com seu argumento e retoma seu valor anterior.
b) O operador de inserção em streams « e o operador de extração de streams » são sobrecarregados para tratar todos os tipos de dados padrão - inclusive endereços de strings e de memória (somente inserção em streams) - e todos os tipos de dados definidos pelo usuário.
e) A função membro de stream flags () sem argumentos desliga todos os bits de indicadores na variável de estado de flags.
d) O operador de extração de stream » pode ser sobrecarregado com uma função operador que aceita uma referência para istream e uma referência para

um tipo definido pelo usuário como parâmetros e retoma um referência para istream.
e) O manipulador de stream ws ignora (salta) os espaços em branco iniciais em um stream de entrada.
f) O operador de inserção em stream « pode ser sobrecarregado com uma função operador que aceita uma referência a is tream e uma referência a um tipo definido pelo usuário como parâmetros e retorna uma referência para is tream.
g) A entrada com o operador de extração de streams » sempre ignora caracteres em branco iniciais no stream fornecido como entrada.
h) Os recursos de entrada e saída são fornecidos como parte de C++.
i) A função membro de stream rdstate () retorna o estado atual do stream.
j) O streanl cout normalmente é conectado à tela do monitor.
k) A função membro de stream good() retorna verdadeiro se as funções membrobadO, fail () e eof retornam falso.
1) O stream cm normalmente é conectado à tela do monitor.
m) Se um erro não-recuperável ocorre durante uma operação com stream, a função membro bad retornará verdadeiro.
n) A saída de cerr não é posta em um buffer e a saída de clog é posta em um buifer.
o) Quando o indicador ios: showpoint é inicializado, valores em ponto flutuante são forçados a serem impressos com o default de seis dígitos de precisão - desde que o valor da precisão não tenha sido mudado, caso em que os valores em ponto flutuante são impressos com a precisão especificada.
p) A função membro put de ostream envia para a saída o número especificado de caracteres.
q) Os manipuladores de stream dec, oct e hex somente afetam a próxima operação de saída de inteiros.
r) Quando mostrados na saída, os endereços de memória são exibidos como inteiros lorig por default.

11.3 Para cada um dos seguintes itens, escreva um único comando que execute a tarefa indicada.
a) Envie para a saída o string “Digite seu nome:
b) Inicialize um indicador que faz com que o expoente em notação científica e as letras em valores hexadecimais sejam impressas em letras maiúsculas.
e) Envie para a saída o endereço da variável string do tipo char*.
d) Inicialize um indicador de forma que a impressão de valores em ponto flutuante seja feita na notação cientitica.
e) Envie para a saída o endereço da variável integerPtr do tipo int*.
f) Inicialize um indicador de forma que, quando valores inteiros forem enviados para a saída, a base do inteiro seja mostrada para valores nas bases octal e hexadecimal.
g) Envie para a saída o valor apontado por floatPtr. do tipo float*.
h) Use uma função membro de stream para definir o caractere de preenchimento

como ‘*‘ para imprimir em campos com largura maior que os valores sendo mostrados. Escreva um comando separado para fazer isto com um manipulador de stream.
i) Envie para a saída os caracteres ‘0’ e K em um só comando, usando a função put de ostream.
j) Obtenha o valor do próximo caractere no stream de entrada sem extraí-lo do stream.
k) Forneça como entrada um caractere único para a variável e de tipo char. usando a função membro get de istream de dois modos diferentes.
1) Receba como entrada e descarte os próximos seis caracteres do stream de entrada.
m) Use a função membro read de istream para ler 50 caracteres para o array linha do tipo char.
n) Leia 10 caracteres para o array de caracteres nome. Pare de ler caracteres se o delimitador . for encontrado. Não remova o delimitador do stream de entrada. Escreva outro comando que executa esta tarefa e remove o delimitador da entrada.
o) Use a função membro gcount de istream para determinar o número de caracteres fornecidos como entrada para o array de caracteres linha pela última chamada à função membro read de istream e envie para a saída essa quantidade de caracteres usando a função membro write de ostream.
p) Escreva comandos separados para esvaziar a saída do stream usando uma função membro e um manipulador de r stream.
q) Envie para a saída os seguintes valores: 124, 18.376, ‘Z’, 1000000 e “String”.
r) Imprima a definição da precisão atual usando uma função membro.
s) Receba como entrada um valor inteiro para a variável int meses e um valor em ponto flutuante para a variável float percentual.
t) Imprima 1. 92, 1 . 925 e 1 . 9258 com 3 dígitos de precisão usando um manipulador.
u) Imprima o inteiro 100 em octal, hexadecimal e decimal usando manipuladores de stream.
v) Imprima o inteiro 100 em decimal, octal e hexadecimal usando um manipulador de stream único para mudar a base.
w) Imprima 1234 alinhado à direita em um campo de 10 dígitos.
x) Leia caracteres para o array de caracteres linha até o caractere ‘z ser encontrado, até um máximo de 20 caracteres (incluindo o caractere nulo terminal). Não extraia o caractere delimitador do stream.
y) Use as variáveis inteiras x e y para especificar a largura do campo e a precisão usada para exibir o valor double 87.4573 e exiba o valor.
11.4 Identifique o erro em cada um dos comandos seguintes e explique como corrigi-lo.
a) cout« “Valor de x <= y é: “ « x < y;
b) O comando seguinte deveria imprimir o valor inteiro de ‘e
cout « ‘c’;
e) cout « “Um string entre aspas”;
11.5 Para cada um dos seguintes itens, mostre a saída.

a) eout « “12345” « endi;
cout.width(5);
cout.fill(‘*‘
cout « 123 « endi « 123;
b) cout « setw(10) « setfill(‘$‘) « 10000;
e) cout « setw(8) « setprecision(3) « 1024.987654;
d) cout « setiosflags(ios: :showbase) « oct « 99
« endi « hex « 99;
e) cout « 100000 « endi
« setiosflags(ios::showpos) « 100000;
f) cout « setw(10) « setprecision(2) «
« setiosflags(ios::scientifie) « 444.93738;

CAPÍTULO 11 - ENTRADA/SAÍDA COM STREAMS EM C++ 675
Respostas aos exercícios de auto-revisão
11.1 a) friend b) ios: :left, ios: :right e ios: :internal. c) streams.
d) setiosflags, resetiosflags. e) iostream. f) setf, unsetf.
g) strstreain. h) iomanip. i) fstream. j) endi. k) stdiostream.
1) write. m) istream. fl) cerr ou clog. O) ostream. p) «.
q) cm, cout, cerr e clog. r) ». s) oct, hex. dec. t) seis dígitos de precisão.
u) ios: :showpos.
11.2 a) Verdadeiro.
b) Falso. Os operadores de inserção em stream e extração de stream não são sobrecarregados para todos os tipos definidos pelo usuário. O programador de uma classe deve fornecer especificamente as funções de operador sobrecarregadas
para sobrecarregar os operadores de streams a serem usados com cada tipo definido pelo usuário.
c) Falso. A função membro de stream flags () sem argumentos simplesmente retoma o valor atual da variável de estado de flags.
d) Verdadeiro.
e) Verdadeiro.
f) Falso. Para sobrecarregar o operador de inserção em stream «, a função de operador sobrecarregada deve receber uma referência a ostream e uma referência a um tipo definido pelo usuário como argumentos e retornar uma referência para ostream.
g) Verdadeiro. A menos que ios: : skipws esteja “desligado”.
h) Falso. Os recursos de EIS de C++ são fornecidos como parte da Biblioteca Padrão C++. A linguagem C++ não contém recursos para entrada, saída ou processamento de arquivos.
i) Verdadeiro.
j) Verdadeiro.
k) Verdadeiro.
1) Falso. O stream cm é conectado à entrada padrão do computador, que é normalmente o teclado.
m) Verdadeiro.

n) Verdadeiro.
o) Verdadeiro.
p) Falso. A função membro put de ostream coloca na saída o caractere recebido como seu único argumento.
q) Falso. Os manipuladores de streams dec. oct e hex inicializam o estado do formato de saída para inteiros para a base especificada até a base ser novamente mudada ou o programa terminar.
r) Falso. Os endereços de memória são exibidos no formato hexadecimal por default. Para exibir endereços como inteiros long, o endereço deve sofrer uma coerção para um valor long.
11.3 a) cout « “Digite seu nome:
b) cout.setf(ios: :uppercase);
c) cout « (void *) string;
d) cout. setf (ios: : scientific, ios: : floatfield);
e) cout « integerPtr;
f) cout « setiosflags(ios: :showbase);
g) cout « *f].oatptr;
h) cout.fill(‘*‘);
cout « setfill (*‘
i) cout.put(‘O’).put(‘K’);
j) cin.peek
k) c = cin.get
cin.get(c);
1) cin.ignore(6);
m)cin.read(linha, 50);
n) cin.get(nome, 10, ‘.‘
cin.getline(nome, 10, .‘);
o) cout.write(linha, cin.gcount());
p) cout.flush()
cout « flush;
q) cout « 124 « 18.376 « 1000000 « Z « “String”;
r) cout « cout.precisionQ;
s) cm » meses » percentual;
t) cout « setprecision(3) « 1.92 « ‘\t’
« 1.925 « ‘\t « 1.9258;
u) cout « oct « 100 « hex « 100 « dec « 100;

v) cout « 100 « setbase(8) « 100 « setbase(16) « 100;
w)cout « setw(10) « 1234;
x) cin.get(linha, 20, ‘z’
y) cout « setw(x) « setprecision(y) « 87.4573;
11.4 a) Erro: a precedência do operador «é mais alta que a precedência de <=. o que faz com que o comando seja avaliado impropriamente e provoca um erro de compilação.
Correção: para corrigir o comando, adicione parênteses em tomo da expressão x

<= y.
Este problema acontecerá com qualquer expressão que usa operadores de precedência mais baixa que o operador « se a expressão não for colocada entre parênteses.
b) Erro: em C++, caracteres não são tratados como inteiros pequenos como são em C.
Correção: para imprimir o valor numérico para um caractere no conjunto de caracteres do computador, o caractere deve sofrer uma coerção para um valor inteiro, como segue:
cout « int(‘c’);
e) Erro: os caracteres aspas não podem ser impressos em um string a menos que uma seqüência de escape seja usada. Correção: imprima o string em um dos modos seguintes:
cout « ‘“‘ « “Um string entre aspas” «
cout « “\“Um string entre aspas\”;
11.5 a) 12345
**123
123
b) $$$$$10000
c) 1024.988
d) 0143
0x63
e) 100000
+100000
f) 4.45e+02
Exercícios
11.6 Escreva um comando para cada um dos seguintes itens:
a) Imprima o inteiro 40000 alinhado à esquerda em um campo de 15 dígitos.
b) Leia um string para a variável array de caracteres estado.
c) Imprima 200 com e sem sinal.
d) Imprima o valor decimal 100 no formato hexadecimal, precedido por Ox.
e) Leia caracteres para o array s até o caractere ‘p’ ser encontrado e até um máximo de 10 caracteres (incluindo o caractere nulo terminal). Extraia o delimitador do stream de entrada e descarte-o.
f) Imprima 1 .234 em um campo de 9 dígitos com zeros à esquerda.
g) Leia um string da forma “caracteres” da entrada padrão. Armazene o string no array de caracteres s. Elïmine as aspas do stream de entrada. Leia um máximo de 50 caracteres (incluindo o caractere nulo terminal).
11.7 Escre a um programa para testar a entrada de alores inteiros no formato decimal, octal e hexadecimal. Envie para a saída
4 cada inteiro lido pelo programa em todos os três formatos. Teste o programa com os seguintes dados de entrada: 10,010, 0x lO.
1 1.11 Escreva um programa que imprime valores de ponteiros usando coerções para todos os tipos de dados inteiros. Quais imprimem valores estranhos? Quais causam erros?
11.9 Escreva um programa para testar os resultados de imprimir o valor inteiro

12345 e o valor de ponto flutuante 1 .2345
em vários tamanhos de campo. O que ocorre quando os valores são impressos em campos que contêm menos dígitos que O
valores?
11.10 Escreva um programa que imprime o valor 100. 453627 arredondado para o mais próximo dígito, décimo, centésimo, milésimo e décimo-milésimo.
11.11 Escreva um programa que recebe um string pelo teclado e determina o comprimento do string. Imprima o string usando duas vezes o seu comprimento como a largura do campo.
11.12 Escreva um programa que converte temperaturas Fahrenheit inteiras de O até 212 graus para temperaturas Celsius em ponto flutuante com 3 dígitos de precisão. Use a fórmula
celsius = 5.0 / 9.0 * (fahrenheit - 32)

caractere é um caractere de espaço em branco. Cada caractere deve ser lido usando a função membro get de istream. Quando é encontrado um caractere que não é um espaço em branco, o manipulador skipwhite termina seu trabalho colocando o caractere de volta no stream de entrada e retornando uma referência para istream.
Teste o manipulador criando uma função main em que o indicador ios: : skipws não é inicializado, de forma que o operador de extração de strearn não ignora automaticamente espaços em branco. Então, teste o manipulador com um stream de entrada que tem um caractere precedido por espaços em branco. Imprima o caractere que foi inserido para confirmar que não foi lido um caractere de espaço em branco.

Gabaritos
Objetivos
• Ser capaz de usar gabaritos de funções para criar um grupo
de funções relacionadas (sobrecarregadas).
• Ser capaz de distinguir entre gabaritos de funções e fun çõe gabarito.
• Ser capaz de usar gabaritos de classes para criar um grupo
de tipos relacionados.
• Ser capaz de distinguir entre gabaritos de classe e classes
gabarito.
• Entender como sobrecarregar funções gabarito.
• Entender os relacionamentos entre gabaritos, friends, he- L
rança e membros estáticos.
Por trás daquele padrão externo as formas sombrias ficam mais
claras a cada dia.
E sempre a ínesma forma, apenas muito numerosa
Charlotte Perkins Gilman
The Yellow Wallpaper
Se você conseguir deslizar entre os parâmetros dos céus e da terra, deslize.
O Corão
Um labirinto poderoso! mas não sem um plano.
Alexander Pope

12

Visão geral
12.1 Introdução
12.2 Gabaritos de função
12.3 Sobrecarregando funções gabarito
12.4 Gabaritos de classe
12.5 Gabaritos de classe e parâmetros não-tipo
12.6 Gabaritos e herança
12.7 Gabaritos e friends
12.8 Gabaritos e membros static
Resumo . Terminologia . Erros comuns de programação Dicas de desempenho.
Observações de engenharia de software . Dica de teste e depura ção •
Exercícios de auto-revisão Respostas aos exercícios de auto- revisão •
Exercícios
12.1 Introdução
Neste capítulo, discutimos um dos recursos mais poderosos de C++, ou seja, os

gabaritos. Os gabaritos nos possibilitam especificar, com um único segmento de código, uma gama inteira de funções relacionadas (sobrecarregadas) - chamadas de funções gabarito - ou uma gama inteira de classes relacionadas - chamadas de classes gabarito.
Podemos escrever um único gabarito de função para uma função de classificação de arrays e então fazer com que C++ gere automaticamente funções gabarito separadas que classificam um array int, classificam um array float, classificam um array de strings, e assim por diante.
Discutimos gabaritos de função no Capítulo 3. Para benefício daqueles leitores que saltaram aquele tópico, apresentamos uma discussão e um exemplo adicionais neste capítulo.
Podemos escrever um gabarito de classe único para uma classe pilha e então fazer com que C++ gere classes gabaritos separadas, tais como uma classe pilha de int, uma classe pilha de float, uma classe pilha de string. e assim por diante.
Note a distinção entre gabaritos de função e funções gabarito: os gabaritos de função e os gabaritos de classe são como estênceis com os quais nós traçamos formas; as funções gabarito e as classes gabarito são como os traçados separados que tem todos a mesma forma, mas poderiam ter sido desenhados, por exemplo, com cores diferentes.
Observação de engenharia de software 12.1
_____ Os gabaritos são um dos recursos mais poderosos de C+ + para a reutilização de software.
Neste capítulo, apresentamos exemplos de um gabarito de função e um gabarito de classe. Também consideraremos as relações entre gabaritos e outras características de C++, tais como sobrecarga, herança, friends e membros static.
O projeto e os detalhes dos mecanismos de gabarito discutidos aqui se baseiam no trabalho de Bjarne Stroustrup apresentado no seu artigo Parameterized Tvpes for C+ + e publicado nos Proceedings ofthe USENIX C+ + Conference, ocorrido em Denver, Cobrado, em outubro de 1988.
Este capítulo pretende ser somente uma breve introdução ao rico e complexo tópico de gabaritos. O Capítulo 20, “A biblioteca padrão de gabaritos (STL)”, o maior capítulo deste livro, apresenta um tratamento detalhado das classes gabarito contêineres, iteradores e algoritmos da STL. O Capítulo 20 contém dezenas de exemplos de código real baseados em gabaritos - ilustrando técnicas de programação com gabaritos mais sofisticadas que as usadas aqui no Capítulo 12.

12.2 Gabaritos de função
Funções sobrecarregadas são normalmente usadas para executar operações semelhantes sobre tipos de dados diferentes. Se as operações são idênticas para cada tipo, isto pode ser executado mais compacta e convenientemente usando-se gabaritos de função. O programador escreve uma única definição do gabarito da função. Baseado nos tipos de argumentos fornecidos explicitamente

ou inferidos das chamadas para esta função, o compilador gera funções separadas, no código objeto, para tratar cada tipo de chamada apropriadamente. Em C, esta tarefa pode ser executada usando-se macros criadas com a diretiva #define do pré-processador (ver Capítulo 17, “O pré- processador”). Porém, macros apresentam a possibilidade de sérios efeitos colaterais e não possibilitam ao compilador executar uma verificação de tipo. Os gabaritos de função fornecem uma solução compacta como as macros, mas possibitam uma verificação de tipo completa.

® Dica de teste e depura ção 12.1

Gaba ritos dei unção, como macros, possibilitam a reutilização de software. Mas, diferentemente de macros, gabaritos de função ajudam a eliminar muitos tipos de erros por causa do escrutínio da verificação de tipo completa de C++.

Todas as definições de gabaritos de função começam com a palavra-chave template seguida por uma lista de parâmetros de tipo formais para o gabarito de função, incluso entre os sinais de maior e menor (< e >); cada parâmetro formal de tipo deve ser precedido pela palavra-chave class ou typename, como em

```
template< class T >
```

ou

```
template< typenaine ElementType >
```

ou

```
template< class BorderType, class FillType >
```

Os parâmetros de tipo formais de uma definição de gabarito são usados (como seriam com parâmetros de tipos primitivos ou tipos definidos pelo usuário) para especificar os tipos dos parâmetros da função, especificar o tipo de retorno da função e para declarar variáveis dentro da função. A definição da função vem a seguir e é definida como qualquer outra função. Note que a palavra-chave class ou typename usada para especificar parâmetros de tipo do gabarito de função na realidade significa “qualquer tipo primitivo ou tipo definido pelo usuário”.

Erro comum de programação 12.1

Não colocar class (ou typename) antes de cada parâmetro formal de tipo de um gabarito de função.

Examinemos o gabarito de função printArray na Fig. 12.1. O gabarito de função é usado no programa completo daFig. 12.2.

```
1 template< class T >
2 void printArray( const T *array, const int count
3
4 for ( int i = O; i < count; i++
5 cout « array[ i 1 «
6
7 cout « endl;
8
```

Fig. 12.1 Um gabarito de função.

O gabarito de função printArray declara um parâmetro formal único T (T poderia ser qualquer identificador válido) para o tipo do array a ser impresso pela função

printArray: T * é chamado de parâmetro de tipo. Quando o compilador encontrar uma chamada para a função printArray no código-fonte do programa, o tipo do primeiro parâmetro de printArray é substituído por T na definição do gabarito e C ++ cria uma função gabarito completa para imprimir um array do tipo de dados especificado. Então, a função recém-criada é compilada. Na Fig. 12.2, três funções printArray são instanciadas - uma espera um array int. uma espera um array double e uma espera um array char, Por exemplo, a instanciação para o tipo int é:

```
void printArray( const int *array, const int count
for ( int i = O; i < count; i++ )
cout « array[ i ] «
cout « endl;
```

Cada parâmetro de tipo formal, em uma definição de gabarito de função, deve normalmente aparecer na lista de parâmetros da função pelo menos uma vez. O nome de um parâmetro de tipo formal pode ser usado somente uma vez na lista de parâmetros de um cabeçalho de gabarito. Nomes de parâmetros de tipo formais entre funções gabarito não precisam ser únicos.

A Fig. 12.2 ilustra o uso do gabarito de função printArray. O programa começa por instanciar int array a, array double b e char array c, de tamanhos 5, 7, e 4, respectivamente. A seguir, cada um dos arrays é impresso chamando printArray - uma vez com um primeiro parâmetro a do tipo int* , uma vez com um primeiro parâmetro b do tipo double* e uma vez com um primeiro parâmetro c do tipo char*. A chamada

```
printArray( a, aCount );
```

por exemplo, faz com que o compilador infira que T é int e instancie uma função gabarito printArray cujo parâmetro de tipo T é int. A chamada

```
printArray( b, bCount );
```

faz com que o compilador infira que T é double e instancie uma segunda função gabarito printArray cujo parâmetro de tipo T é double. A chamada

```
printArray( c, cCount );
```

faz com que o compilador infira que T é char e instancie uma terceira função gabarito printArray cujo parâmetro de tipo T é char.

```
1 II Fig 12.2: figl2O2.cpp
2 // Usando funções gabarito
3 #include <iostream>
4
5 using std: :cout;
6 using std: :endl;
7
8 template< class T >
9 void printArray( const T *array, const int count )
10 {
11 for ( int i = o; i < count; i++
12 cout « array[ i ] «
13
14 cout « endi;
```

15 }
Fig. 12.2 Usando funções gabarito (parte 1 de 2).

CAPÍTULO 12 - GABARITOS 683

```
16
17 int main()
18 {
19 const int aCount = 5, bCount = 7, cCount = 4;
20 inta[aCount)={1,2,3,4,5};
21 double b[ bCount J = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
22 char c[ cCount ] = “ALÔ”; //quarta posição reservada para o caractere nulo
23
24 cout « “Array a contém:” « endi;
25 printArray( a, aCount ); II função gabarito para inteiros
26
27 cout « “Array b contém:” « endi;
28 printArray( b, bCount ); // função gabarito para doulies
29
30 cout « “Array c contém:” « endl;
31 printArray( c, cCount ); // função gabarito para caracteres
32
33 return 0;
34 }
Array a contém:
12345
Array b contém:
.1 2.2 3.3 4.4 5.5 6.6 7.7
Array c contém:
ALÔ
```

Fig. 12.2 Usando funções gabarito (parte 2 de 2).

Neste exemplo, o mecanismo de gabarito poupa o programador de ter que escrever tres runçoes sonrecarregauas separadas, com protótipos

```
void printArray(const int “, const int);
void printArray(const double ‘, const int);
void printArray(const char , const int);
```

as quais usam todas o mesmo código, exceto pelo tipo T.

Dica de desempenho 12.1

f Gabaritos certamente oferecem os benefícios de reutilização de software. Mas tenha em mente que múltiplas cópias de funções gabarito e classes gabarito ainda são instanciadas em um programa, apesar do fato de que o gabarito é escrito somente uma vez. Estas cópias podem consumir memória considerável.

12.3 Sobrecarregando funções gabarito

Funções gabarito e sobrecarga estão intimamente relacionadas. As funções relacionadas geradas a partir de um gabarito de função têm todas o mesmo nome, de modo que o compilador usa a resolução de sobrecarga para invocar a função apropriada.

Um gabarito de função pode ser sobrecarregado de vários modos. Podemos fornecer outros gabaritos de função que especificam o mesmo nome da função, mas parâmetros da função diferentes. Por exemplo, o gabarito da função printArray da Fig. 12.2 poderia ser sobrecarregado com outro gabarito da função printArray com parâmetros adicionais lowsubscript e highSubscript para especificar a parte do array a ser impressa (ver Exercício 12.4).

Um gabarito de função pode também ser sobrecarregado fornecendo outras funções não-gabarito com o mesmo nome da função, mas com argumentos de função diferentes. Por exemplo, o gabarito de função printArray da Fig. 12.2 poderia ser sobrecarregado com uma versão não-gabarito que imprime especificamente um array de string de caracteres em um formato organizado, tabular, de coluna (ver Exercício 12.5).

Erro comum de programação 12.2

Se um gabarito é invocado com um tipo de classe definido pelo usuário e se esse gabarito usa operadores (como +, <, etc.) com objetos do tipo daquela classe, então aqueles operadores devem ser sobrecarregados! Esquecer de sobrecarregar tais operadores provoca erros, porque o com pilador naturalmente, ainda gera chamadas às funções operador sobrecarregado apropriadas, apesar do fato de que estas funções não estão presentes.

O compilador executa um processo de correspondência para determinar qual função chamar quando uma função é invocada. Primeiro, o compilador tenta achar e usar uma correspondência precisa, na qual os nomes da função e os tipos de parâmetro coincidem exatamente com aqueles da chamada da função. Se isto falha, o compilador verifica se está disponível um gabarito de função que pode ser usado para gerar uma função gabarito com uma correspondência precisa de nome da função e tipos de parâmetros. Se tal gabarito de função é encontrado, o compilador gera e usa a função gabarito apropriada.

Erro comum deprogramnação 12.3

O compilador executa um processo de correspondência para determinar qualfunção chamar quando uma função é invocada. Se nenhuma correspondência pode ser encontrada ou se o processo de correspondência produz correspondências múltiplas, é gerado um erro de compilação.

12.4 Gabaritos de classe

É possível se compreender o que é uma pilha (uma estrntura de dados na qual inserimos itens em uma ordem e recuperamos os mesmos na ordem inversa, ou seja, “último a entrar, primeiro a sair”), independentemente do tipo dos itens que estão sendo colocados na pilha. Mas quando chega o momento de efetivamente instanciar uma pilha, um tipo de dados deve ser especificado. Isto cria uma oportunidade maravilhosa para a reutilização de software. Necessitamos dos meios de descrever a noção de uma pilha genericamente e instanciar classes que são versões desta classe genérica para tipos específicos. Este recurso é fornecido por gabaritos de classe em C++.

Observação de engenharia de software 12.2

_____ Os gabaritos de classe incentivam a reutilização de software, possibilitando que sejam instanciadas versões de classes genéricas para tipos espec(ficos.
Gabaritos de classe são chamados de tipos parametrizados porque exigem um ou mais parâmetros de tipo para especificar como personalizar um gabarito de uma “classe genérica” para formar um gabarito de classe específico.
O programador que deseja produzir diversas classes gabarito simplesmente escreve uma definição de gabarito de classe. Toda vez que o programador necessita de uma nova instanciação específica para um tipo, o programador usa uma notação concisa, simples e o compilador escreve o código-fonte para a classe gabarito de que o programa- dor necessita. Um gabarito de classe Stack, por exemplo, poderia assim se tornar a base para criar muitas classes Stack (tais como “Stack de double”, “Stack de int”. “Stack de char”. “ Stack de Employee’. etc.) usadas em um programa.
Note a definição do gabarito de classe Stack na Fig. 12.3. Parece com uma definição de classe convencional, a não ser pelo fato de ser precedida pelo cabeçalho (linha 6)
template< class T >
para especificar que esta é uma definição de gabarito de classe com o parâmetro de tipo T indicando o tipo da classe de Stack a ser criada. O programador não necessita usar especificamente o identificador T - qualquer identificador

pode ser usado, O tipo de elemento a ser armazenado nessa Stack é mencionado somente genericamente como T, em todo o cabeçalho de classe de Stack e nas definições das funções membro. Mostraremos em breve como T torna-se associado com um tipo específico, tal como double ou int. Existem duas restrições para tipos de dados não-primitivos usados com esta Stack: eles devem ter um construtor default e devem suportar o operador de atribuição. Se um objeto da classe usada com esta Stack contém memória alocada dinamicamente, o operador de atribuição deve ser sobrecarregado para aquele tipo, como mostrado no Capítulo 8.

```
1 II Fig. 12.3: tstackl.h
2 // Gabarito de classe Stack
3 #ifndef TSTACK1H
4 #define TSTACK1H
5
6 template< class T >
7 class Stack
8 public:
9 Stack( int = 10 ); // construtor default (tamanho da pilha 10)
10 -‘Stack() { delete [] stackPtr; ) II destruidor
11 bool push( const T& ); II insere um elemento na pilha
12 bool pop( T& ); II retira um elemento da pilha
13 private:
```

```
14 int size; II quantidade de elementos na pilha
15 int top; II posição do elemento do topo da pilha
16 T *stackptr; II ponteiro para a pilha
17
18 bool isEmpty() const { return top == -1; } II funções
19 bool isFull() const ( return top == size - 1; } II utilitárias
20
21
22 // Construtor com tamanho default 10
23 template< class T >
24 Stack< T >::Stack( int s
25
26 size = s > O ? s : 10;
27 top = -1; // inicialmente, Stack está vazia
28 stackPtr = new T[ size ]; // aloca espaço para elementos
29 }
30
31 II Insere uni elemento na pilha
32 // retorna 1 se bem-sucedida, O caso contrário
33 template< class T >
34 bool Stack< T >::push( const T &pushValue
35 {
36 if ( ‘isFull() ) {
37 stackPtr[ ++top ] pushValue; II coloca item em Stack
38 return true; II push bem-sucedida
39 1
40 return false; II push malsucedida
41
42
43 // Retira um elemento da pilha
44 template< class T >
45 bool Stack< T >: :pop( T &popValue
46 {
47 if ( !isEmpty() ) {
48 popValue = stackPtr[ top- ]; II remove item de Stack
49 return true; // pop bem-sucedida
50
```

Fig. 12.3 Demonstrando o gabarito de classe Stack - tstackl .h (parte 1 de 2).


```
51 return false; II pop malsucedida
52 }
53
54 #endif
```

Fig. 12.3 Demonstrando o gabarito de coasse Stack - tstackl .h parte 2 ae 2).

Agora consideremos o programa (função main) que testa o funcionamento do

gabarito de classe Stack (ver a saída na Fig. 12.3). O programa de teste começa por instanciar o objeto doubleStack, de tamanho 5. Esse objeto é declarado como sendo da classe Stack< double> (pronunciado “Stack de double”). O compilador associa o tipo double com o parâmetro de tipo T no gabarito, para produzir o código-fonte para uma classe Stack do tipo double. Embora o programador não veja este código-fonte, ele é incluído no código-fonte e compilado.

```
55 II Fig. 12.3: figl2O3.cpp
56 II Programa de teste para o gabarito Stack
57 #include <iostream>
58
59 using std: :cout;
60 using std: :cin;
61 using std: :endl;
62
63 #include “tstackl.h”
64
65 int main()
66 {
67 Stack< double > doubleStack( 5 );
68 double f 1.1;
69 cout « “Inserindo elementos em doubleStack\n”;
70
71 while ( doubleStack.push( f ) ) { II sucesso: true retornado
72 cout«f« ‘
73 f + 1.1;
74 }
75
76 cout « “\nStack está cheia. Não pode inserir « f
77 « “\n\nRetirando elementos de doubleStack\n”;
78
79 while ( doubleStack.pop( f ) ) II sucesso: true retornado
80 cout « f « ,
81
82 cout « “\nStack está vazia. Não pode retirar\n”;
83
84 Stack< int > intStack;
85 inti=1;
86 cout « “\nlnserindo elementos em intStack\n”;
87
88 while ( intStack.push( i ) ) { II sucesso: true retornado
89 cout « i «
90
91 }
92
93 cout « “\nStack está cheia. Não pode inserir “ « i
```

```
94 « “\n\nRetirando elementos de intStack\n”;
95
96 while ( intStack.pop( i ) ) II sucesso: true retornado
97 cout«i« ‘
```
Fig. 12.3 Demonstrando o gabarito de classe Stack - figl2_03 . cpp (parte 1 de 2).


```
98
99 cout « \nStack está vazia. Não pode retirar\n”;
100 return 0;
101
Inserindo elementos em doubleStack 1
1.1 2.2 3.3 4.4 5.5
Stack está cheia. Não pode inserir 6.6
Retirando elementos de doubleStack
5.5 4.4 3.3 2.2 1.1
Stack está vazia. Não pode retirar
Inserindo elementos em intStack
1 2 3 4 5 6 7 8 9 10
Stack está cheia. Não pode inserir 11
Retirando elementos de intStack
10 9 8 7 6 5 4 3 2 1
Stack está vazia. Não pode retirar
```
Fig. 12.3 Demonstrando o gabarito de classe Stack - figl2_03 . cpp (parte 2 de 2).

O programa de teste então insere, sucessivamente, os valores double 1. 1,2.2,3.3,4.4 e 5.5 em doublestack.

O laço de push termina quando o programa de teste tenta inserir um sexto valor em doubleStack (que já está
cheia, porque foi criada para manter um máximo de cinco elementos).

O programa de teste agora retira os cinco valores da pilha (note na Fig. 12.3 que os valores são retirados na ordem “último a entrar, primeiro a sair”), O programa de teste tenta retirar um sexto valor, mas doubleStack agora está vazio, assim o laço de pop termina.

Em seguida, o programa de teste instancia a pilha de inteiros intStack com a declaração

```
Stack< int > intStack;
```

(pronunciada “intStack é um Stack de int”). Como nenhum tamanho foi especificado, o tamanho assume o valor default 10, conforme especificado no construtor default (linha 24). Uma vez mais, o programa de teste executa um laço de inserção de valores em intStack até que ela esteja cheia e, então, executa iterações de retirada de valores de intStack até que ela esteja vazia. Uma vez mais, os valores são retirados na ordem “último a entrar, primeiro a sair”.

As definições de funções membro fora da classe começam, cada uma, com o

cabeçalho (linha 23)
template< class T >
Então, cada definição se assemelha a uma definição convencional de função, exceto pelo fato de que o tipo de elemento de Stack é sempre listado genericamente como o parâmetro de tipo T. O operador de resolução de escopo binário é usado com o nome do gabarito de classe Stack< T > para amarrar cada definição de função membro ao escopo do gabarito de classe. Neste caso, o nome de classe é Stack< T >. Quando doubleStack é instanciado como sendo do tipo Stack< double >, o constructor de Stack usa new para criar um array de elementos do tipo double para representar a pilha (linha 28). O comando
stackPtr = new T[size];
na definição do gabarito de classe de Stack é gerado pelo compilador na classe gabarito Stack< double > como
stackPtr = new double[size];

Note que o código na função main da Fig. 12.3 é quase idêntico a ambas as manipulações de doubleStack na metade superior de main e as manipulações de intStack na metade inferior de main. Isto nos apresenta uma outra oportunidade para usar um gabarito de função. A Fig. 12.4 usa o gabarito de função testStack para executar as mesmas tarefas que main na Fig. 12.3 - insere uma série de valores em uma Stack< T > e retira os valores de uma Stack< T >. O gabarito de função testStack usa o parâmetro de tipo formal T para representar o tipo de dados armazenados na Stack< T >. O gabarito de função aceita quatro parâmetros - uma referência para um objeto do tipo Stack< T >, um valor do tipo T que será o primeiro valor inserido no Stack< T >, um valor do tipo T usado para incrementar os valores inseridos no Stack< T > e um string de caracteres do tipo const char
* que representa o nome do objeto Stack< T > para fins de saída de dados. A função main, agora, simplesmente instancia um objeto do tipo Stack< double > chamado doubleStack e um objeto do tipo Stack< int > chamado intStack e usa estes objetos nas linhas 42 e 43
testStack(doubleStack, 1.1, 1.1, ‘doubleStack’);
testStack(intStack, 1, 1, “intStack”);
Note que a saída da Fig. 12.4 corresponde precisamente à saída da Fig. 12.3.
1 II Fig. 12.4: figl2O4.cpp
2 II Programa de teste para o gabarito Stack.
3 II A função main usa um gabarito de função para
4 // manipular objetos do tipo Stack< T >.
5 #include <iostream>
6
7 using std: :cout;
8 using std: :cin;
9 using std: :endl;
10

```
#include “tstackl .h’
II Gabarito de função para
template< class T >
void testStack(
Stack< T > &theStack,
T value,
T increment,
const char *stackName
cout « “\Inserindo elementos em « stackName « \I;
while ( theStack.push( value ) ) { II sucesso: true retornado cout « value «
value += increment;
cout « ‘\nPilha está cheia. Não pode inserir « value
« “\n\nRetirando elementos de « stackName « ‘\n’;
while ( theStack.pop( value ) ) II sucesso: true retornado cout « value «
cout « “\nPilha está vazia. Não pode retirar \n”;
int main()

manipular Stack< T >
II referência ao Stack< T >
// valor inicial a ser inserido
// incremento para valores subseqüentes
// nome do objeto Stack < T >

11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
```

34
35
36
37
38

Fig. 12.4 Passando um objeto gabarito Stack para um gabarito de função (parte 1 de 2).

39 Stack< double > doubleStack(5);
40 Stack< int > intStack;
41
42 testStack(doubleStack, 1.1, 1.1, “doubleStack”);
43 testStack(intStack, 1, 1, “intStack”);
44
45 return 0;
46 }
Inserindo elementos em doubleStack
1.1 2.2 3.3 4.4 5.5
Pilha está cheia. Não pode inserir 6.6
Retirando elementos de doubleStack
5.5 4.4 3.3 2.2 1.1
Pilha está vazia. Não pode retirar
Inserindo elementos em intStack
1 2 3 4 5 6 7 8 9 10
Pilha está cheia. Não pode inserir 11
Retirando elementos de intStack
10 9 8 7 6 5 4 3 2 1
Pilha está vazia. Não pode retirar
Fig. 12.4 Passando um objeto gabarito Stack para um gabarito de função (parte 2 de 2).

12.5 Gabaritos de classe e parâmetros não-tipo

O gabarito de classe Stack da seção anterior usou somente parâmetros de tipo no cabeçalho do gabarito. Também
é possível usar parâmetros não-tipo; um parâmetro não-tipo pode ter um argumento default e o parâmetro não-tipo
é tratado como const. Por exemplo, o cabeçalho do gabarito poderia ser modificado para aceitar um parâmetro
int elements. como segue:

```
template< class T, int elements > //note parâmetro não-tipo
```

Então, um declaração tal como

```
Stack< double, 100 > mostRecentSalesFigures;
```

iria instanciar durante a compilação) uma classe gabarito Stack de 100 elementos chamaaa mostRecentSalesFigures de valores double: esta classe gabarito seria do tipo Stack< double, 100

>. O cabeçalho de classe pode então conter um membro de dados private com uma declaração de array tal como
T stackHolder[elements]; // array para guardar o conteúdo da pilha
Dica de desempenho 12.2
Quando é possívelfazê-lo, especificar o tamanho de uma classe contêiner (tal como uma classe array ou
uma classe pilha) durante a compilação (possivelmente através de um parâmetro de gabarito não-tipo
para o tamanho) elimina o overhead de criar o espaço dinamicamente com new durante a execução.
Observação de engenharia de software 12.3
Quando é possível fazê-lo, especificar o tamanho de um contêiner durante a compilação (possivelmente através de um parâmetro de gabarito não-tipo para o tamanho) evita a possibilidade de um erro potencialmente fatal durante a execução se new ficar impossibilitado de obter a memória necessária.

Nos exercícios, será pedido a você para usar um parâmetro não-tipo para criar um gabarito para a classe Array desenvolvida no Capítulo 8, “Sobrecarga de operadores”. Este gabarito possibilitará a objetos Array serem instanciados com um número especificado de elementos de um tipo especificado durante a compilação, em vez de criar espaço dinamicamente para os objetos Array durante a execução.
Uma classe para um tipo específico, o qual não corresponde a um gabarito de classe comum, pode ser fornecida para redefinir o gabarito da classe para aquele tipo. Por exemplo, um gabarito de classe Array pode ser usado para instanciar um array de qualquer tipo. O programador pode optar por assumir o controle do instanciamento da classe Array de um tipo específico, tal como Marciano. Isto é feito simplesmente formando a nova classe com um nome de classe Array< Marciano>.
12.6 Gabaritos e herança
Os gabaritos e a herança se relacionam de vários modos:
• Um gabarito de classe pode ser derivado de uma classe gabarito.
• Um gabarito de classe pode ser derivado de uma classe não-gabarito.
• Uma classe gabarito pode ser derivada de um gabarito de classe.
• Uma classe não-gabarito pode ser derivada de um gabarito de classe.
12.7 Gabaritos e friends
Vimos que funções e classes inteiras podem ser declaradas como friends de classes não-gabarito. Com gabaritos de classe, os tipos óbvios de friends podem ser declarados. A relação de friend pode ser estabelecida entre um gabarito de classe e uma função global, uma função membro de outra classe (possivelmente uma classe gabarito) ou até uma classe inteira (possivelmente uma classe gabarito). As notações exigidas para estabelecer estas relações de friends podem ser incômodas.
Dentro de um gabarito de classe para a classe X que foi declarado com
template< class T > class X

uma declaração de friend da forma
friend void f 10;
torna a função fi um friend de toda classe gabarito instanciada a partir do gabarito de classe precedente. Dentro de um gabarito de classe para a classe X que foi declarado com
template< class T > class X
uma declaração de friend da forma
friend void f2(X< T > &);
para um tipo particular T, tal como float, torna a função f2 (X< float> &) um friend somente de X< float>. Dentro de um gabarito de classe, você pode declarar que uma função membro de outra classe é um friend
de qualquer classe gabarito gerada a partir do gabarito de classe. Simplesmente nomeie a função membro da outra

classe usando o nome de classe e o operador de resolução de escopo binário. Por exemplo, dentro de um gabarito de classe para a classe x que foi declarado com
template< class T > class X
uma declaração de friend da forma
friend void A::f40;
torna a função membro f4 da classe A um friend de toda classe gabarito instanciada a partir do gabarito de classe precedente.
Dentro de um gabarito de classe para a classe X que foi declarado com
template< class T > class X
uma declaração de friend da forma
frierid void C < T >: :f5(X< T > &);
para um tipo particular T, tal como float, torna a função membro
C < float >::f5(X< float > &
uma função friend somente da classe gabarito X < float >
Dentro de um gabarito de classe para a classe X que foi declarado com
template< class T > class X
uma segunda classe Y pode ser declarada com
friend class Y;
tornando toda função membro da classe Y um friend de toda classe gabarito produzida a partir do gabarito de classe para X.
Dentro de um gabarito de classe para a classe X que foi declarado com
template< class T > class X
uma segunda classe Z pode ser declarada com
friend class Z< T >;
então, quando uma classe gabarito é instanciada com um tipo particular para T, tal como float, todos os membros de class Z< float > tornam-se friends da classe gabarito X< float >.

12.8 Gabarítos e membros static

E os membros de dados static ? Lembre que, com uma classe não-gabarito,

uma cópia de um membro de dados static é compartilhada entre todos os objetos da classe e o membro de dados static deve ser inicializado em escopo de arquivo.
Cada classe gabarito instanciada a partir de um gabarito de classe tem sua própria cópia de cada membro de dados static do gabarito de classe; todos os objetos daquela classe gabarito compartilham aquele membro de dados static. E como com membros de dados static de classes não-gabarito, membros de dados static de classes gabarito devem ser inicializados em escopo de arquivo. Cada classe gabarito obtém sua própria cópia das funções membro estáticas do gabarito de classe.

692 C++ COMO PROGRAMAR

• Gabaritos nos possibilitam especificar uma gama de funções relacionadas (sobrecarregadas) - chamadas de funções gabarito
- ou uma gama de classes relacionadas - chamadas de classes gabarito.
• Para usar gabaritos de função, o programador escreve uma única definição de gabarito de função. Com base nos tipos dos argumentos fornecidos em chamadas para esta função, C÷+ gera funções separadas para tratar de cada tipo de chamada apropriadamente. Elas são compiladas junto com o resto do código-fonte de um programa.
• Todas as definições de gabarito de função começam com a palavra-chave template seguida por parâmetros de tipo formais para o gabarito de função incluso entre sinais de maior e menor (< e >); cada parâmetro formal deve ser precedido pela
palavra-chave class (ou typename). A palavra-chave class (ou typename) é usada para especificar parâmetros de tipo de gabaritos de função significando “qualquer tipo primitivo ou um tipo definido pelo usuário”.
• Os parâmetros de tipo formais de definições de gabaritos são usados para especificar os tipos dos parâmetros para a função, o tipo de retorno da função e para declarar variáveis na função.
• O nome de um parâmetro de tipo formal pode ser usado somente uma vez na lista de parâmetros de um cabeçalho de gabarito. Os nomes de parâmetro de tipo formais de gabaritos de funções não necessitam ser únicos.
• Um gabarito de função pode ser ele próprio sobrecarregado de vários modos. Podemos fornecer outros gabaritos de função que especificam o mesmo nome de função, mas parâmetros de função diferentes. Um gabarito de função pode também ser
sobrecarregado fornecendo-se outras funções não-gabarito com o mesmo nome de função, mas parâmetros de função diferentes.
• Gabaritos de classe fornecem os meios para descrever uma classe genericamente e instanciar classes que são versões específicas desta classe genérica para um tipo.
• Gabaritos de classe são chamados de tipos de parametrizados; eles requerem parâmetros de tipo para especificar como personalizar um gabarito de classe genérico para formar uma classe gabarito específica.

• O programador que deseja usar classes gabarito escreve um gabarito de classe. Quando o programador necessita de uma nova classe para um tipo específico, o programador usa uma notação concisa e o compilador escreve o código-fonte para a classe gabarito.
• Uma definição de gabarito de classe se parece com uma definição de classe convencional, a não ser que ela é precedida por template< class T > (ou template< typeriaxne T >), para indicar que esta é uma definição de gabarito de classe
com o parâmetro de tipo T indicando o tipo da classe a ser criada. O tipo T é mencionado ao longo do cabeçalho de classe e das definições de funções membro, como um nome de tipo genérico.
• As definições de funções membro fora da classe começam cada uma com o cabeçalho template< class T > (ou template< typename T>) . Então, cada definição de função se assemelha a uma definição de função convencional, a não ser que os
dados genéricos na classe são sempre listados genericamente como parâmetros do tipo T. O operador de resolução de escopo binário é usado com o nome do gabarito de classe para amarrar cada definição de função membro ao escopo do gabarito de
classe, como em ClassName< T>.
• É possível se usar parâmetros não-tipo no cabeçalho de um gabarito de classe.

• Uma classe para um tipo específico pode ser fornecida para sobrescrever o gabarito de classe para aquele tipo.
• Um gabarito de classe pode ser derivado de uma classe gabarito. Um gabarito de classe pode ser derivado de uma classe não- gabarito. Uma classe gabarito pode ser derivada de um gabarito de classe. Uma classe não-gabarito pode ser derivada de um gabarito de classe.
• Funções e classes inteiras podem ser declaradas como friends de classes não-gabarito. Com gabaritos de classe, os tipos óbvios de friends possíveis podem ser declarados. A relação de friend pode ser estabelecida entre um gabarito de classe
e uma função global, uma função membro de outra classe (possivelmente uma classe gabarito) ou até uma classe inteira (possivelmente uma classe gabarito).
• Cada classe gabarito instanciada de um gabarito de classe tem sua própria cópia de cada membro de dados static do gabarito de classe; todos os objetos daquela classe gabarito compartilham aquele membro de dados static. E, como com
os membros de dados static de classes não-gabarito, membros de dados static de classes gabarito devem ser inicializados em escopo de arquivo.

Resumo

• Cada classe gabarito obtém uma cópia das funções membro static do gabarito

de classe.

CAPÍTULO 12 - GABARITOS 693

Terminologia
argumento de gabarito
classe gabarito
declaração de gabarito de função
definição de gabarito de função
friend de um gabarito
função gabarito
função membro de classe gabarito
função membro static de um gabarito de classe função membro static de uma classe gabarito gabarito de classe
gabarito de função
membro de dados static de um gabarito de classe membro de dados static de uma classe gabarito
Erros comuns de programação

nome de gabarito
nome de gabarito de classe
palavra-chave class em um parâmetro de tipo de gabarito palavra-chave template
parâmetro de gabarito
parâmetro de tipo em um cabeçalho de gabarito parâmetro não-tipo em um cabeçalho de gabarito parâmetro de tipo formal em um cabeçalho de gabarito
sinais de menor e maior (< e>)
sobrecarregando uma função gabarito
template<class T>
tipo parametrizado
typename

12.1 Não colocar class (ou typenalne) antes de cada parâmetro formal de tipo de um gabarito de função.
12.2 Se um gabarito é invocado com um tipo de classe definido pelo usuário e se esse gabarito usa operadores (como ==. <=, etc.) com objetos do tipo daquela classe, então aqueles operadores devem ser sobrecarregados! Esquecer de sobrecarregar tais operadores provoca erros, porque o compilador, naturalmente, ainda gera chamadas às funções operador sobrecarregado apropriadas, apesar do fato de que estas funções não estão presentes.
12.3 O compilador executa um processo de correspondência para determinar qual função chamar quando uma função é invocada. Se nenhuma correspondência pode ser encontrada ou se o processo de correspondência produz correspondências múltiplas, é gerado um erro de compilação.
Dicas de desempenho
12.1 Gabaritos certamente oferecem os benefícios de reutilização de software.

Mas tenha em mente que múltiplas cópias de funções gabarito e classes gabarito ainda são instanciadas em um programa, apesar do fato de que o gabarito é escrito somente uma vez. Estas cópias podem consumir memória considerável.
12.2 Quando é possível fazê-lo, especificar o tamanho de uma classe contêiner (tal como uma classe array ou uma classe pilha) durante a compilação (possivelmente através de um parâmetro de gabarito não-tipo para o tamanho) elimina o overhead de criar o espaço dinamicamente com new durante a execução.
Observações de engenharia de software
12.1 Os gabaritos são um dos recursos mais poderosos de C++ para a reutilização de software.
12.2 Os gabaritos de classe incentivam a reutilização de software, possibilitando que sejam instanciadas versões de classes genéricas para tipos específicos.
12.3 Quando é possível fazê-lo, especificar o tamanho de um contêiner durante a compilação (possivelmente através de um parâmetro de gabarito não-tipo para o tamanho) evita a possibilidade de um erro potencialmente fatal durante a execução se new ficar impossibilitado de obter a memória necessária.
Dica de teste e depura ção
12.1 Gabaritos de função, como macros, possibilitam a reutilização de software. Mas, diferentemente de macros, gabaritos de função ajudam a eliminar muitos tipos de erros por causa do escrutínio da verificação de tipo completa de C++.
Exercícios de auto-revisão
12.1 Responda a cada um dos seguintes itens com verdadeiro ou falso. Para aqueles que são falsos, mostre por quê.
a) Uma função friend de um gabarito de função deve ser uma função gabarito.
b) Se várias classes gabarito são geradas a partir de um único gabarito de classe com um único membro de dados static, cada uma das classes gabarito compartilha uma cópia única do membro de dados static do gabarito de

classe.

c) Uma função gabarito pode ser sobrecarregada por outra função gabarito com o mesmo nome de função.
d) O nome de um parâmetro de tipo formal pode ser usado somente uma vez na lista de parâmetros de tipo formais da
definição do gabarito. Os nomes de parâmetros de tipo formais entre definições de gabaritos devem ser únicos.
e) As palavras-chave class e typenaine. como usadas com um parâmetro de tipo de um gabarito, significam especificamente ‘qualquer tipo de classe definido pelo usuário”.
12.2 Preencher os espaços em branco em cada um dos seguintes itens:
a) Gabaritos nos possibilitam especificar, com um segmento de código único, uma gama inteira de funções relacionadas chamadas de _____ ou uma

gama inteira de classes relacionadas chamadas de _____
b) Todas as definições de gabaritos de função começam com a palavra-chave seguida por uma lista de parâmetros formais do gabarito de função inclusos entre _____
c) As funções relacionadas geradas a partir de um gabarito de função têm todas o mesmo nome, de modo que o compilador usa resolução de para invocar a função apropriada.
d) Gabaritos de classes são também chamados de tipos
e) O operador é usado com um nome de classe gabarito para amarrar cada definição de função membro ao escopo do gabarito de classe.
f) Como com os membros de dados static de classes não-gabarito, membros de dados static de classes gabarito também devem ser inicializados em escopo de
Respostas aos exercícios de auto-revisão
12.1 a) Falso. Poderia ser uma função não-gabarito. b) Falso. Cada classe gabarito terá uma cópia do membro de dados static. c) Verdadeiro. d) Falso. Os nomes de parâmetro de tipo formais entre funções gabarito não precisam ser únicos.
e) Falso. A palavra-chave class, neste contexto, também permite um parâmetro de tipo de um tipo primitivo.
12.2 a) funções gabarito, classes gabarito. b) template, sinais de menor e maior (< e >). c) sobrecarga. d) parametrizados. e) resolução de escopo binário. f) arquivo.
Exercícios
12.3 Escreva um gabarito de função bubbleSort baseado no programa de classificação da Fig. 5.15. Escreva um programa de teste que lê da entrada, classifica e envia para a saída um array int e um array float.
12.4 Sobrecarregue o gabarito de função printArray da Fig. 12.2 de forma que ele aceite dois parâmetros do tipo inteiro adicionais, quais sejam int lowSubscript e int highSubscript. Uma chamada a esta função imprimirá somente a parte indicada do array. Valide lowSubscript e highSubscript: se um deles estiver fora do intervalo ou se highSubscript for menor que ou igual a lowSubscript. a função sobrecarregada printArray deve retornar O; caso contrário, printArray deve retornar o número de elementos impressos. A seguir, modifique main para testar ambas as versões de printArray com os arrays a, b, e c. Não deixe de testar todos os recursos de ambas as versões de printArray.
12.5 Sobrecarregue o gabarito de função printArray da Fig. 12.2 com uma versão não-gabarito que especificamente imprime um array de strings de caracteres em formato organizado. tabular. por colunas.
12.6 Escreva um gabarito de função simples para a função predicado elgualA que compara seus dois argumentos com o operador de igualdade (==) e retorna true se eles forem iguais e false se eles não forem iguais. Use este gabarito de função em um programa que chama elgualA somente com diversos tipos primitivos. Agora, escreva, que uma versão separada do programa que chama e IgualA com um tipo de classe definido pelo usuário, mas não sobrecarrega o operador de igualdade. O que acontece quando você tenta executar este programa? Agora sobrecarregue o operador de igualdade (com a função

operador operator==). O que acontece, agora, quando você tenta executar este programa?
12.7 Use um parâmetro não-tipo numeroDeElementos e um parâmetro de tipo TipoDeElemento para ajudar a criar um gabarito para a classe Array que desenvolvemos no Capítulo 8, “Sobrecarga de operadores”. Este gabarito possibilitará a objetos Array serem instariciados com um número especificado de elementos de um tipo de elemento especificado durante a compilação.
12.8 Escreva um programa com o gabarito de classe Array. O gabarito pode instanciar um Array de quaisquer tipos de
elementos. Redefina o gabarito com uma definição específica para um Array de elementos float (class Array< float>).
O programa de teste deve demonstrar a instanciação de um Array de ints a partir do gabarito e deve mostrar que uma tentativa
de instanciar um Array de floats usa a definição fornecida em class Array< float >.

1

694 C++ COMO PROGRAMAR

c) Uma função gabarito pode ser sobrecarregada por outra função gabarito com o mesmo nome de função.
d) O nome de um parâmetro de tipo formal pode ser usado somente uma vez na lista de parâmetros de tipo formais da definição do gabarito. Os nomes de parâmetros de tipo formais entre definições de gabaritos devem ser únicos.
e) As palavras-chave class e typename, como usadas com um parâmetro de tipo de um gabarito, significam especificamente “qualquer tipo de classe definido pelo usuário”.
12.2 Preencher os espaços em branco em cada um dos seguintes itens:
a) Gabaritos nos possibilitam especificar, com um segmento de código único, uma gama inteira de funções relacionadas chamadas de _____ ou uma gama inteira de classes relacionadas chamadas de _____
b) Todas as definições de gabaritos de função começam com a palavra-chave seguida por uma lista de parâmetros formais do gabarito de função inclusos entre _____
c) As funções relacionadas geradas a partir de um gabarito de função têm todas o mesmo nome, de modo que o compi lado usa resolução de para invocar a função apropriada.
d) Gabaritos de classes são também chamados de tipos
e) O operador é usado com um nome de classe gabarito para amarrar cada definição de função
membro ao escopo do gabarito de classe.
O Como com os membros de dados static de classes não-gabarito, membros de dados static de classes gabarito também devem ser inicializados em escopo de
Respostas aos exercícios de auto-revisão
12.1 a) Falso. Poderia ser uma função não-gabarito. b) Falso. Cada classe

gabarito terá uma cópia do membro de dados static. c) Verdadeiro. d) Falso. Os nomes de parâmetro de tipo formais entre funções gabarito não precisam ser únicos. e) Falso. A palavra-chave class, neste contexto, também permite um parâmetro de tipo de um tipo primitivo.
12.2 a) funções gabarito, classes gabarito. b) template, sinais de menor e maior (< e >). e) sobrecarga. d) parametrizados. e) resolução de escopo binário. f) arquivo.
Exercícios
12.3 Escreva um gabarito de função bubbleSort baseado no programa de classificação da Fig. 5.15. Escreva um programa de teste que lê da entrada, classifica e envia para a saída um array int e um array float.
12.4 Sobrecarregue o gabarito de função printArray da Fig. 12.2 de forma que ele aceite dois parâmetros do tipo inteiro adicionais, quais sejam int lowSubscript e int highSubscript. Uma chamada a esta função imprimirá somente a parte indicada do array. Valide lowSubscript e highSubscript; se um deles estiver fora do intervalo ou se highSubscript for menor que ou igual a lowSubscript, a função sobrecarregada printArray deve retornar O; caso contrário. printArray deve retornar o número de elementos impressos. A seguir, modifique main para testar ambas as versões de printArray com os arrays a, b, e c. Não deixe de testar todos os recursos de ambas as versões de priritArray.
12.5 Sobrecarregue o gabarito de função printArray da Fig. 12.2 com uma versão não-gabarito que especificamente imprime um array de strings de caracteres em formato organi7ado, tabular, por colunas.
12.6 Escreva um gabarito de função simples para a função predicado elgualA que compara seus dois argumentos com o operador de igualdade (==) e retoma true se eles forem iguais e false se eles não forem iguais. Use este gabarito de função em um programa que chama elgualA somente com diversos tipos primitivos. Agora, escreva, que uma versão separada do programa que chama elgualA com um tipo de classe definido pelo usuário, mas não sobrecarrega o operador de igualdade. O que acontece quando você tenta executar este programa? Agora sobrecarregue o operador de igualdade (com a função operador operator==). O que acontece, agora, quando você tenta executar este programa?
12.7 Use um parâmetro não-tipo numeroDeElemeritos e um parâmetro de tipo TipoDeElemento para ajudar a criar um gabarito para a classe Array que desenvolvemos no Capítulo 8, “Sobrecarga de operadores”. Este gabarito possibilitará a objetos Array serem instanciados com um número especificado de elementos de um tipo de elemento especificado durante a compilação.
12.8 Escreva um programa com o gabarito de classe Array. O gabarito pode instanciar um Array de quaisquer tipos de
elementos. Redefina o gabarito com uma definição específica para um Array de elementos float (class Array< float>).
O programa de teste deve demonstrar a instanciação de um Array de ints a partir do gabarito e deve mostrar que uma tentativa
de instanciar um Array de floats usa a definição fornecida em class Array< float >.

12.9 Qual a diferença entre os termos “gabarito de função” e “função gabarito”?
12.10 O que se parece mais como um estêncil - um gabarito de classe ou uma classe gabarito? Explique sua resposta.
12.11 Qual é a relação entre gabaritos de função e sobrecarga?
12.12 Por que você optaria por usar um gabarito de função em vez de uma macro?
12.13 Que problema de desempenho pode resultar do uso de gabaritos de função e gabaritos de classe?
12.14 O compilador executa um processo de correspondência para determinar qual função gabarito chamar quando uma função
é invocada. Em que circunstâncias uma tentativa de fazer uma correspondência pode provocar um erro de compilação?
12.15 Por que é apropriado chamar um gabarito de classe de um tipo parametrizado?
12.16 Explique por que você poderia usar o comando
Array< Employee > workerList(100);
em um programa em C++.
12.17 Revise sua resposta para o Exercício 12.16. Agora, por que você poderia usar o comando
Array< Empioyee > workerList;
em um programa em C++?
12.18 Explique o uso da seguinte notação em um programa em C++:
template< ciass T > Array< T >: :Array(int s)
12.19 Por que você poderia tipicamente usar um parâmetro não-tipo com um gabarito de classe para um contêiner tal como um array ou uma pilha?
12.20 Descreva como fornecer uma classe para um tipo específico para sobrescrever o gabarito de classe para aquele tipo.
12.21 Descreva a relação entre gabaritos de classe e herança.
12.22 Suponha que um gabarito de classe tem o cabeçalho
tempiate< ciass Ti > ciass Ci
Descreva os relacionamentos do tipo friend estabelecidos ao se colocar cada uma das seguintes declarações friend dentro deste cabeçalho de gabarito de classe. Identificadores começando com “f “são funções, identificadores começando com “C” são classes e identificadores começando com “T” podem representar quaisquer tipos (i.e., tipos primitivos ou tipos de classe).
a) friend void f 1Q;
b) friend void f2(Ci< Ti > &)
c) friend void C2::f4( );
d) friend void C3< Ti >: :f5(Ci< Ti > &)
e) friend ciass C5;
f) friend class C6< Ti >;
12.23 Suponha que o gabarito de classe Empregado tem um membro de dados static contagem. Suponha que tres classes gabarito são instanciadas a partir do gabarito de classe. Quantas cópias do membro de dados static existirão? Como

será restringido o uso de cada um (se for o caso)?

13
Tratamento de exceções

Objetivos
• Usar try, throw e catch para detectar, indicar e tratar exceções, respectivamente.
• Processar exceções não-capturadas e inesperadas.
• Ser capaz de processar falhas de new.
• Usar auto_ptr para prevenir perdas de memória.
• Entender a hierarquia padrão de exceções.
Nunca esqueço um rosto, mas no seu caso vou abrir uma exce ção.

Groucho (Julius Henry) Marx

Nenhuma regra é tão geral que não admita exceções. Robert Burton, The Anatomy of Melancholy
É do senso comum pegar um método e experimentá-lo. Sefalhar admita francamente e tente outro. Mas acima de tudo, tente alguma coisa.
Franklin Delano Roosevelt.
Oh! Jogue fora a pior parte disso e deixe a parte mais pura com a outra metade.
William Shakespeare
Se eles estão correndo e não olham aonde estão indo, Tenho que sair de algum lugar e apanhá-los. Jerome David Salinger
E ao se desculpar várias vezes de uma falha não piore essa falha pela desculpa.

William Shakespeare.

1

Errar é humnano, perdoar é divino.
Alexander Pope, An Essay on Criticism

CAPfTULO 13 - TRATAMENTO DE EXCEÇÕES 697

Visão geral
13.1 Introdução
13.2 Quando o tratamento de exceções deve ser usado
13.3 Outras técnicas de tratamento de erros
13.4 Fundamentos do tratamento de exceções em C++: try, throw, catch
13.5 Um exemplo simples de tratamento de exceção: divisão por zero
13.6 Disparando uma exceção

13.7 Capturando uma exceção
13.8 Disparando novamente uma exceção
13.9 Especificações de exceção
13.10 Processando exceções inesperadas
13.11 Desempilhando a pilha
13.12 Construtores, destruidores e o tratamento de exceções
13.13 Exceções e herança
13.14 Processando falhas de new
13.15 A classe auto_ptr e a alocação dinâmica de memória
13.16 Hierarquia de exceções da biblioteca padrão
Resumo• Terminologia Erros comuns de programação • Boas práticas de programação. Observações de engenharia de software Dicas de desempenho Dicas de teste e depura ção Exercícios de auto-revisão Respostas aos exercícios de auto-revisão • Exercícios

13.1 Introdução

Neste capítulo, introduzimos o tratamento de exceções. A estensibilidade de C++ pode aumentar substancialmente a quantidade e os tipos de erros que podem acontecer. Os recursos apresentados aqui possibilitam aos programadores escreverem programas mais claros, mais robustos e tolerantes a falhas. Sistemas recentes desenvolvidos com estas e/ou técnicas semelhantes têm reportado resultados positivos. Também mencionaremos quando o tratamento de exceções não deve ser usado.

O estilo e os detalhes do tratamento de exceções apresentados neste capítulo estão baseados no trabalho de Andrew Koenig e Bjarne Stroustrup apresentado em seu artigo “Exception Handling for C++ (revised)”, publicado nos Proceedings ofthe USENIX C++ Conference, ocorrida em São Francisco, em abril de 1990.

O código de tratamento de erro varia em natureza e entre sistemas de software, dependendo de se o aplicativo de software é ou não um produto para lançamento comercial. Os produtos comerciais tendem a conter muito mais código de tratamento de erros do que o software “mais informal”.

Existem muitos meios populares de se lidar com erros. Mais comumente, o código de tratamento de erro está misturado no meio do código de um sistema. Os erros são tratados nos lugares do código onde os erros podem acontecer. A vantagem desta abordagem é que um programador que estiver lendo o código pode ver o processamento de erro na vizinhança imediata do código e determinar se foi implementada a verificação apropriada de erro.

O problema com este esquema é que o código, de certo modo, se torna “poluído” com o processamento de erros. Se torna mais difícil para um programador preocupado com o aplicativo em si ler o código e determinar se o mesmo está funcionando corretamente. Isto torna mais difícil de se entender e manter o código.

Alguns exemplos comuns de exceções são uma falha de new em obter uma quantidade solicitada de memória, um subscrito de array fora dos limites, estouro em operações aritméticas, divisão por zero e parâmetros de função inválidos.

Os recursos de tratamento de exceções de C++ possibilitam ao programador remover o código de tratamento de erros da “linha principal” de execução de um programa. Isso melhora a legibilidade e a possibilidade de modificar o programa. Com o estilo de C++ de tratamento de exceções, é possível se capturar todos os tipos de exceções, capturar todas as exceções de um certo tipo ou capturar todas as exceções de tipos relacionados. Isso torna os programas mais robustos, reduzindo a probabilidade de que erros não sejam capturados por um programa. O tratamento de exceções é fornecido para possibilitar aos programas capturar e tratar erros, em vez de deixá-los acontecer e sofrer as conseqüências. Se o programador não fornece um meio de tratamento para um erro fatal, o programa termina.

O tratamento de exceções foi projetado para lidar com erros síncronos, tal como uma tentativa de dividir por zero (isso acontece quando o programa executa a instrução dividir). Com tratamento de exceções, antes de o programa executar a divisão, ele confere o denominador e “dispara” uma exceção se o denominador for zero.

O tratamento de exceções não foi projetado para lidar com situações assíncronas, tais como término de EIS em disco, chegada de mensagem pela rede, diques de mouse, etc.; essas são mais bem tratadas através de outros meios, tal como interrupções de processamento.

O tratamento de exceções é usado em situações em que o sistema pode recuperar o erro que causou a exceção. O procedimento que faz a recuperação é chamado de tratador de exceção. O tratamento de exceções é tipicamente usado quando o erro será tratado por uma parte diferente do programa (i.e., um escopo diferente) daquela que detectou o erro. Um programa que conduz um diálogo interativo com um usuário não deve usar exceções para processar erros de entrada.

O tratamento de exceções é especialmente apropriado a situações em que o programa não será capaz de se recuperar, mas precisa fazer uma “limpeza final” organizada e então terminar “elegantemente”.

Boa prática de programação 13.1

Use exceções para erros que devem ser processados em um escopo diferente daquele em que ocorrem. Use outros meios de tratamento de erros para erros que serão processados no escopo em que acontecem.

Boa prática de programação 13.2

Evite usar tratamento de exceções para fins diferentes do tratamento de erros, pois isso pode reduzir a clareza do programa.

Existe outra razão para se evitar usar técnicas de tratamento de exceções para o controle de programas convencionais. O tratamento de exceções foi projetado para processamento de erros, que é uma atividade infreqüente, comumente usada porque um programa está para terminar. Em vista disso, não se espera que os autores de compiladores C++ implementem o tratamento de exceções com o tipo de desempenho otimizado que se espera para o código regular de

aplicativos.
Dica de desempenho 13.1
f Embora seja possível se usar o tratamento de exceções para propósitos diferentes do tratamento de erros, isso pode reduzir o desempenho do programa.
Dica de desempenho 13.2
f ‘ O tratamento de exceções é geralmente implementado em compiladores de tal maneira que quando não ocorrem exceções, pouco ou nenhum overhead é imposto pela presença do código de tratamento de exceções. Quando ocorrem exceções, elas incorrem em overhead durante a execução. Certamente, a presença de código de tratamento de exceções faz o programa consumir mais memória.
Observação de engenharia de software 13.1
_____ O fluxo de controle com estruturas de controle convencionais é geralmente mais claro e eficiente do que com exceções.

Erro comum de programação 13.1

Uma outra razão segundo a qual exceções podem ser perigosas como uma alternativa ao fluxo normal de controle é que a pilha é desempilhada e os recursos alocados antes da ocorrência da exceção podem não ser liberados. Este problema pode ser evitado com uma programação cuidadosa.

O tratamento de exceções ajuda a melhorar a tolerância a falhas de um programa. Torna-se “mais agradável” escrever código de processamento de erros e, assim, é mais provável que os programadores o façam. Também se torna possível capturar exceções de vários modos, tal como por tipo, ou até especificar que exceções de qualquer tipo devem ser capturadas.
A maioria dos programas escritos atualmente suporta somente uma “única thread” (fluxo) de execução. O multithreading vem recebendo grande atenção em sistemas operacionais recentes, como Windows NT, OS/2 e várias versões do Unix. As técnicas discutidas neste capítulo se aplicam até para programas que usam multithreading, embora não discutamos especificamente programas que usam multithreading.
Mostraremos como lidar com exceções “não-capturadas”. Consideraremos como são tratadas exceções inesperadas. Mostraremos como exceções relacionadas podem ser representadas por classes de exceções derivadas de uma classe base de exceção comum.
Os recursos de tratamento de exceções de C++ estão se tornando amplamente usados, como resultado do padrão C++. A padronização é especialmente importante para grandes projetos de software, nos quais dezenas ou até centenas de pessoas trabalham em componentes separados de um sistema, e tais componentes necessitam interagir para o sistema global funcionar corretamente.

Observação de engenharia de software 13.2

O tratamento de exceções é bem adequado para sistemas com componentes desenvolvidos separadamente. O tratamento de exceções facilita a combinação dos componentes. Cada componente pode executar sua própria detecção de exceção, separada do tratamento das exceções em outro escopo.

o tratamento de exceções pode ser visto como um outro meio de retornar o controle de uma função ou sair de um bloco de código. Normalmente, quando acontece uma exceção, ela será tratada por um invocador da função geradora da exceção, por um invocador daquele invocador, ou quão longe para trás na cadeia de chamadas for necessário se ir para achar um tratador para aquela exceção.

13.2 Quando o tratamento de exceções deve ser usado

O tratamento de exceções deve ser usado somente para processar situações excepcionais, apesar do fato de que não existe nada que impeça o programador de usar exceções como uma alternativa para o controle do programa; processar exceções para componentes de programa que não estão preparados para tratar aquelas exceções diretamente; processar exceções de componentes de software tais como funções, bibliotecas e classes que, provavelmente, serão amplamente usados e onde não faz sentido que esses componentes tratem suas próprias exceções; e em projetos de grande porte para tratar o processamento de erros de uma maneira uniforme em todo o projeto.

Boa prática de programação 13.3

Use técnicas de tratamento de erros convencionais em lugar do tratamento de exceções para um processamento de erros direto e local, no qual um programa pode facilmente lidar com seus próprios erros.

Observação de engenharia de software 13.3

Ao lidar com bibliotecas, o chamador dafirnção de biblioteca provavelmente terá em mente umprocessainento de erro especifico para uma exceção gerada na função de biblioteca. E improvável que uma função de biblioteca execute um processamento de erro que satisfaça às necessidades particulares de todos os usuários. Portanto, exceções são meios apropriados para tratar erros produzidos por funções de bibliotecas.

13.3 Outras técnicas de tratamento de erros
Apresentamos uma variedade de modos de lidar com situações excepcionais

anteriormente a este capítulo. O texto a seguir resume estas e outras técnicas úteis.

• Use assert para testar erros de codificação e projeto. Se uma asserção for false. o programa termina e o código deve ser corrigido. Isto é útil durante a depuração do programa.

• Simplesmente ignore as exceções. Isso seria devastador para produtos de software lançados para o público em geral, ou para software de finalidade especial - necessário para missões críticas. Mas para seu próprio software, desenvolvido para seus próprios fins, é bastante comum se ignorar muitos tipos de erros.

• Aborte o programa. Isso, é claro, evita que um programa execute até a conclusão e produza resultados incorretos. Realmente, para muitos tipos de erros isto é apropriado, especialmente para erros não-fatais que possibilitam que um programa execute até a conclusão, talvez levando o programador a pensar que o programa funcionou corretamente. Aqui, também, tal estratégia é imprópria para aplicativos destinados a missões críticas. Os aspectos relacionados aos recursos também são importantes aqui. Se um programa obtém um recurso, o programa deveria liberar normalmente aquele recurso antes do término do programa.

Erro comum de programação 13.2

Abortar um programa pode deixar um recurso em um estado em que outros programas não são capazes de
acessar o mesmo, e conseqüentemente o programa teria uma assim chamada “perda de recurso

• Inicialize algum indicador de erro. O problema com esta solução é que os programas não podem verificar estes indicadores de erro em todos os pontos em que os erros poderiam ser problemáticos.

• Teste a condição de erro, emita uma mensagem de erro e chame exit para passar um código de erro apropriado para o ambiente do programa.

• Use setjump e longjump. Estas funções da biblioteca <csetjmp> possibilitam ao programador especificar um desvio imediato para fora de chamadas de funções profundamente aninhadas, de volta para um tratador de erro. Sem setjump /longjump. um programa deve executar vários returns para sair das chamadas de funções profundamente aninhadas. Isso poderia ser usado para desviar para algum tratador de erro. Mas elas são perigosas, porque elas desempilham a pilha sem chamar destruidores para objetos automáticos. Isso pode levar a sérios problemas.

• Certos tipos específicos de erros têm recursos dedicados para o seu tratamento. Por exemplo, quando new falha na alocação de memória, pode fazer com que uma função new handier seja executada para tratar o erro. Esta função pode ser trocada fornecendo-se um nome de função como argumento para setnewhandler. Discutimos a função set newhandler em detalhes na Seção 13.14.

13.4 Fundamentos do tratamento de exceções em C++: try, throw, catch

O tratamento de exceções em C++ se destina a situações em que a função que descobre um erro está impossibilitada de tratá-lo. Tal função disparará (throw)

uma exceção. Não existe nenhuma garantia de que existirá “qualquer coisa lá fora”, i.e., um tratador de exceção especificamente preparado para processar aquele tipo de exceção. Se existir, a exceção será capturada e tratada. Se não existir nenhum tratador de exceção para aquele tipo particular de exceção, o programa termina.
O programador coloca dentro de um bloco try o código que pode gerar um erro que produzirá uma exceção. O bloco try é seguido por um ou mais blocos catch. Cada bloco catch especifica o tipo de exceção que ele pode capturar e tratar. Cada bloco catch contém um tratador de exceção. Se a exceção corresponde ao tipo do parâmetro em um dos blocos catch, o código daquele bloco catch é executado. Se nenhum tratador for encontrado, é chamada a função terminate. a qual, por default. chama a função abort.
Quando uma exceção é disparada, o controle do programa sai do bloco try e pesquisa os blocos catch em
busca de um tratador apropriado (logo discutiremos o que torna um tratador “apropriado”). Se nenhuma exceção foi

disparada no bloco try, os tratadores de exceção para aquele bloco são saltados e o programa retoma a execução depois do último bloco catch.
Podemos especificar as exceções que uma função dispara. Como opção, podemos especificar que uma função
não disparará nenhuma exceção.
A exceção é disparada em um bloco try na função, ou a exceção é disparada a partir de uma função chamada diretamente ou indiretamente a partir do bloco try. O ponto em que o throw é executado é chamado de ponto de disparo. Este termo também é usado para descrever a própria expressão throw. Depois que uma exceção é disparada, o controle não pode retornar ao ponto de disparo da mesma.
Quando ocorre uma exceção, é possível passar informações para o tratador de exceção, a partir do ponto em
que ocorreu a exceção. Essa informação é o tipo do próprio objeto disparado ou informações colocadas no objeto disparado.
O objeto disparado é tipicamente um string de caracteres (para uma mensagem de erro) ou um objeto de uma classe. O objeto disparado leva informações para o tratador de exceção que processará aquela exceção.

Observação de engenharia de software 13.4

Uma idéia-chave do tratamento de exceções é que a parte de um programa ou sistema que tratará a exceção pode ser bastante diferente ou distante da parte do programa que descobriu e gerou a situação excepcional.

13.5 Um exemplo simples de tratamento de exceção: divisão por zero

Agora, iremos considerar um exemplo simples de tratamento de exceções. A Fig. 13.1 usa try, throw e catch para descobrir que ocorreu uma divisão por zero, indicar uma exceção de divisão por zero e tratar uma exceção de divisão por zero.

1 II Fig. 13.1: figl3Ol.cpp
2 II Um exemplo simples de tratamento de exceção.
3 // Verificando uma exceção de divisão por zero.
4 #include <iostream>

5

6 using std::cout;
7 using std::cin;
8 using std: :endl;

9

II Classe DivideByZeroException a ser usada no tratamento de 1/ exceção para disparar uma exceção em caso de divisão por zero. class DivideByZeroException
public:
DivideByZeroException O
message(‘tentou dividir por zero”
const char *what() const { return message;
private:
const char *message;

// Definição da função quociente. Demonstra o disparo de uma // exceção quando uma exceção de divisão por zero é encontrada. double quotient(int numerator, int denominator

if (denominator == O
throw DivideByZeroExceptionO;

28 return static_cast< double > (numerator) / denominator;

10
11
12
13
14
15
16

```
17
18
19
20
21
22
23
24
25
26
27
```

Fig. 13.1 Um exemplo simples de tratamento de exceção com divisão por zero (parte 1 de 2).


```
29 }
30
31 // Programa de teste
32 int main()
33
34 int numberi, number2;
35 double result;
36
37 cout « “Digite dois inteiros (fim de arquivo para terminar) :
38
39 while ( cm » nuxnberl » number2
40
41 // o bloco try engloba o código que pode disparar
42 II uma exceção e o código que não deve ser executado
43 II se ocorrer uma exceção
44 try{
45 result = quotient( numberi, number2 );
46 cout « “O quociente é: « result « endl;
47
48 catch ( DivideByZeroException ex ) { II exception handler
49 cout « “Ocorreu uma exceção: “ « ex. what () « ‘\n
50
51
52 cout « “\nDigite dois inteiros (fim de arquivo para terminar)
53
54
55 cout « endl;
56 return 0; // termina normalmente
57
```

Fig. 13.1 Um exemplo simples de tratamento de exceção com divisão por zero (parte 2 de 2).

Agora, considere o programa de teste em main. Note a declaração “localizada” de numberl e nuniber2.

O programa contém um bloco try (linha 44) que engloba o código que pode disparar uma exceção. Note que a divisão real, que pode causar o erro, não é explicitamente listada dentro do bloco try. Em vez disso, a chamada para a função quotient (definida na linha 23) invoca o código que tenta a divisão real. A função quotient realmente dispara o objeto de exceção divisão por zero, como veremos em seguida. Em geral, os erros podem aparecer através de código explicitamente mencionado no bloco try. através de chamadas a uma função ou até através de chamadas de função profundamente aninhadas iniciadas por código no bloco try.

O bloco try é imediatamente seguido por um bloco catch contendo o tratador de exceção para o erro divisão por zero. Em geral, quando uma exceção é disparada dentro de um bloco try. a exceção é capturada por um bloco catch que especifica o tipo apropriado que corresponde à exceção disparada. Na Fig. 13.1, o bloco catch especifica que ele capturará objetos de exceção do tipo DivideByZeroException; este tipo corresponde ao tipo do objeto associado à função quotient. O corpo desse tratador de exceção imprime a mensagem de erro retornada pela chamada da função what. Tratadores de exceção podem ser muito mais elaborados que esse.

Digite dois O quociente	inteiros (fim de é: 14.2857	arquivo	para terminar):	10 0	7	
Digite dois Ocorreu uma	inteiros (fim de exceção: tentou	arquivo dividir	para terminar): por zero	10 0	O	
Digite dois O quociente	inteiros (fim de é: 3.66667	arquivo	para terminar):	33 9		
Digite dois	inteiros (fim de	arquivo	para terminar):			

Se, ao ser executado, o código em um bloco try não dispara uma exceção, então todos os tratadores catch imediatamente após o bloco try são saltados e a execução continua na primeira linha de código depois dos tratadores catch; na Fig. 13.1, é executado um comando return que retorna O, indicando término normal.

Agora, iremos examinar as definições da classe DivideByzeroException e da função quotient. Na função quotient, quando o comando if determina que o denominador é zero, o corpo do comando if executa um comando throw que especifica o nome do construtor para o objeto de exceção. Isto faz com que um objeto da classe DivideByZeroException seja criado. Este objeto será capturado

pelo comando catch (que especifica o tipo DivideByzeroException) depois do bloco try. O construtor para a classe DivideByZeroException simplesmente aponta o membro de dados message para o string “tentou dividir por zero”. O objeto disparado é recebido no parâmetro especificado no tratador catch (nesse caso, o parâmetro ex) e a mensagem é impressa lá através de uma chamada à função what O.

Boa prática de programação 13.4

Associar cada tipo de erro durante a execução a um objeto de exceção apropriadamente nomeado melhora a clareza do programa.

13.6 Disparando uma exceção

A palavra-chave throw é usada para indicar que uma exceção aconteceu. Isto é chamado de disparar uma exceção.

Um throw normalmente especifica um operando (um caso especial que discutiremos não especifica nenhum ope rando) O operando de um throw pode ser de qualquer tipo. Se o operando é um objeto, chamamo-lo de objeto de exceção. O valor de qualquer expressão pode ser disparado em vez de um objeto. E possível se disparar objetos não
voltados ao tratamento de erro.

Onde uma exceção é capturada? Ao ser disparada, a exceção será capturada pelo tratador de exceção mais
próximo (do bloco try do qual a exceção foi disparada) que especifique o tipo apropriado. Os tratadores de exceção
para um bloco try são listados imediatamente após o bloco try.

• Como parte do disparo de uma exceção, uma cópia temporária do operando do throw é criada e inicializada.

A Esse objeto então inicializa o parâmetro no tratador de exceção. O objeto temporário é destruído quando o tratador
de exceção completa a execução e o controle sai dele. F

Observação de engenharia de software 13.5

_____ Se é necessário passar informações sobre o erro que causou uma exceção, tal informação pode ser colocada no objeto disparado. O tratador catch então vai conter um nome de parâmetro através do qual as infi.rmações podem ser referenciadas.

Observação de engenharia de software 13.6

_____ Um objeto pode ser disparado sem conter informações a serem passadas; nesse caso, o mero conhecimento de que uma exceção desse tipo foi disparada pode fornecer informações suficientes para o tratador fazer seu trabalho corretamente.

Quando uma exceção é disparada, o controle sai do bloco try corrente e continua em um tratador catch apropriado (se existir um) após aquele bloco try. E possível que o ponto de disparo esteja em um escopo profundamente aninhado dentro de um bloco try; o controle ainda seguirá para o tratador catch. E também possível que o ponto de throw possa estar em uma chamada de função profundamente aninhada; ainda neste caso, o controle seguirá para o tratador catch.

Um bloco try pode parecer não conter nenhuma verificação de erro e não incluir

nenhum comando throw. mas o código referenciado no bloco try certamente pode fazer com que código de verificação de erro nos construtores seja exectitado. O código em um bloco try poderia executar indexação de arrays em um objeto de classe array cuja função membro operator [1 é sobrecarregada para disparar uma exceção quando ocorre um erro de subscrito fora do intervalo. Qualquer chamada de função pode invocar código que pode disparar uma exceção ou chamar
4 outra função que dispara uma exceção.

Embora uma exceção possa terminar a execução do programa, não é obrigatório fazê-lo. Porém, uma exceção
termina o bloco em que a mesma aconteceu.
Erro comum de programação 13.3
Exceções deveriam ser disparadas somente dentro de um bloco try. Uma exceção disparada fora de um
bloco try provoca uma chamada a terminate.
Erro comum de programação 13.4
É possível se disparar uma expressão condicional. Mas seja cuidadoso, porque as regras de promoção podem fazer com que o valor retornado pela expressão condicional seja de um tipo diferente daquele que você esperava. Por exemplo, ao disparar um int ou um double a partir da mesma expressão condicional, a expressão condicional converterá o int em um double. Então, o resultado sempre será capturado por um catch com um parâmetro double, em vez de, às vezes, capturar double (para um double realmente pretendido) e às vezes capturar int.
13.7 Capturando uma exceção
Os tratadores de exceção estão contidos em blocos catch. Cada bloco catch começa com a palavra-chave catch seguida por parênteses contendo um tipo (indicando o tipo de exceção que este bloco catch trata) e um nome de parâmetro opcional. Este é seguido por chaves delimitando o código de tratamento de exceções. Quando uma exceção é capturada, o código no bloco catch é executado.
O tratador catch define seu próprio escopo. Um catch especifica entre parênteses o tipo do objeto a ser capturado. O parâmetro em um tratador catch pode ter nome ou não. Se o parâmetro tem nome, ele pode ser referenciado no tratador. Se o parâmetro não tem nome, i.e., só um tipo é listado para fins de comparação com o tipo de objeto disparado, então não são levadas informações do ponto de disparo até o tratador; só o controle passa do ponto de disparo para o tratador. Para muitas exceções, isto é aceitável.
IErro comum de programação 13.5
Especificar uma lista de parâmetros catch separados por vírgulas é um erro de sintaxe.
Uma exceção cujo tipo de objeto disparado corresponde ao tipo do parâmetro no cabeçalho do catch faz com que o bloco catch. i.e., o tratador de exceção para exceções daquele tipo, seja executado.
O tratador catch que captura uma exceção é o primeiro listado depois do bloco

try atualmente ativo que corresponde ao tipo do objeto disparado. As regras de correspondência serão discutidas em breve.
Uma exceção que não é capturada provoca uma chamada para terminate que, por default, termina o programa chamando abort. E possível se especificar um comportamento personalizado, estabelecendo que outra função seja executada, fornecendo-se o nome daquela função como parâmetro em uma chamada da função setterminate.
Um catch seguido por parênteses incluindo reticências
catch (...)
significa capturar todas as exceções.
Erro comum de programação 13.6
Colocar catch (. . .) antes de outros blocos catch impediria que aqueles blocos fossem executados;
catch (. . .) deve ser colocado por último na lista de tratadores que segue um bloco try.
Observação de engenharia de software 13.7
_____ Um ponto fraco na técnica de capturar exceções com catch . . . é que normalmente você não pode ter certeza de qual é o tipo da exceção. Outro ponto fraco é que, sem um parâmetro nomeado, não existe
nenhum modo de se referir ao objeto de exceção dentro do tratador de exceção.

É possível que nenhum tratador corresponda a um objeto disparado em particular. Isso faz com que a procura por uma correspondência continue no próximo bloco try mais externo. A medida que tal processo continua, pode eventualmente ser determinado que não existe nenhum tratador no programa que corresponda ao tipo do objeto disparado; neste caso, é chamada a função terminate, a qual, por default, chama a função abort.
Os tratadores de exceção são pesquisados em ordem, procurando uma correspondência apropriada. O primeiro tratador que corresponde ao tipo de objeto disparado é executado. Quando aquele tratador termina de executar, o controle continua com o primeiro comando depois do último bloco catch, i.e., o primeiro comando depois do último tratador de exceção para aquele bloco try.
E possível que vários tratadores de exceção representem uma correspondência aceitável para o tipo da exceção que foi disparada. Neste caso, é executado o primeiro tratador de exceção que corresponde ao tipo de exceção. Se vários tratadores apresentam uma correspondência, e se cada um destes trata a exceção diferentemente, então a ordem dos tratadores afetará a maneira como a exceção é tratada.
E possível que vários tratadores catch possam conter um tipo de classe que corresponda ao tipo de um objeto disparado em particular. Isto pode ocorrer por várias razões. Primeiro, pode haver um tratador catch (.
“captura tudo”, que capturará qualquer exceção. Segundo, por causa das hierarquias de herança, é possível que um objeto de uma classe derivada possa ser capturado por qualquer tratador que especifique o tipo da classe derivada ou por tratadores que especifiquem os tipos de quaisquer classes base daquela

classe derivada eliminar.
Erro comum de programação 13.7
Colocar um catch que captura um objeto de uma classe base antes de um catch que captura um objeto de uma classe derivada daquela classe base é um erro de lógica. O catch da classe base capturaria todos os objetos de classes derivadas daquela classe base e, desse modo, o catch da classe derivada nunca seria executado.
® Dica de teste e depura ção 13.1
O programador determina a ordem em que os tratadores de exceção são listados. Esta ordem pode afetar C(»flO exceções originadas naquele bloco try são manipuladas. Se você estiver obtendo um comportamento inesperado no tratamento de exceções do seu programa, pode ser porque um bloco catch anterior
está interceptando e tratando as exceções antes que possam alcançar seu tratador catch planejado.
Às vezes, um programa pode processar muitos tipos de exceções intimamente relacionados. Em vez de fornecer classes de exceção e tratadores catch separados para cada uma, um programador pode fornecer uma única classe de exceção e um tratador catch único para um grupo de exceções. A medida que acontece cada exceção, o objeto de exceção pode ser criado com dados privados diferentes. O tratador catch pode examinar estes dados privados para identificar o tipo da exceção.
Quando ocorre uma correspondência? O tipo de parâmetro do tratador catch corresponde ao tipo do objeto
disparado se
• eles são realmente do mesmo tipo.
• o tipo de parâmetro do tratador catch é uma classe base public da classe do objeto disparado.
• o parâmetro do tratador é de um tipo ponteiro ou referência para uma classe base e o objeto disparado é de um tipo ponteiro ou referência para uma classe derivada.
• o tratador catch é da forma catch (. .
Erro comum de programação 13.8
Colocar um tratador de exceção com um tipo de argumento void* antes de tratadores de exceção com outros tipos de ponteiro provoca um erro de lógica. O tratador void* capturaria todas as exceções de tipos ponteiro, de modo que os outros tratadores nunca seriam executados. Somente catch (. . .) deve virapóscatch(void *)
É necessária uma correspondência de tipo de exata. Nenhuma promoção ou conversão são executadas quando se estiver procurando um tratador de exceção, exceto conversões de classe derivada para classe base.

É possível se disparar objetos const. Neste caso, o tipo de parâmetro do tratador catch deve também ser
declarado const.

Se nenhum tratador é encontrado para uma exceção, o programa termina. Embora isto possa parecer a coisa
certa a ser feita, não é o que os programadores estão acostumados a fazer. Em vez disso, em geral os erros simplesmente acontecem e então a execução do programa continua, possivelmente “mancando”.
Um bloco try seguido por vários catches se assemelha a um comando switch. Não é necessário usar break para sair um tratador de exceção de modo a passar por cima os tratadores de exceção restantes. Cada bloco catch define um escopo distinto, enquanto que todos os casos em um comando switch estão contidos dentro do escopo do switch.
Erro comum de programação 13.9
Colocar um ponto-e-vírgula depois de um bloco try, ou depois de qualquer tratador catch (exceto o
último catch) em seguida a um bloco try é um erro de sintaxe.
Um tratador de exceção não pode acessar objetos automáticos definidos dentro de seu bloco try, porque quando uma exceção ocorre o bloco try termina e todos os objetos automáticos dentro do bloco try são destruídos antes de o tratador começar a ser executado.
O que acontece quando uma exceção ocorre em um tratador de exceção? A exceção original que foi capturada é oficialmente tratada quando o tratador de exceção começa a executar. Assim, exceções que acontecem em um tratador de exceção necessitam ser processadas fora do bloco try em que a exceção original foi disparada.
Os tratadores de exceção podem ser escritos de vários modos. Podem dar uma olhada mais de perto em um erro e decidir chamar terminate. Podem disparar novamente uma exceção (Seção 13.8). Podem converter um tipo de exceção em outro, disparando uma exceção diferente. Podem executar qualquer recuperação necessária e retomar a execução depois do último tratador de exceção. Podem olhar para a situação que está causando o erro. remover a causa do erro e tentar novamente chamar a função original que provocou uma exceção (isto não criaria uma recursão infinita). Podem retornar algum valor de estado ao seu ambiente, etc.
Observação de engenharia de software 13.8
_____ É melhor incorporar sua estratégia de tratamento de exceções a um sistema desde o começo do projeto. É difícil incorporar um tratamento de exceções efetivo depois de um sistema ter sido implernentado.
Quando um bloco try não dispara exceções e o bloco try completa normalmente a execução, o controle passa para o primeiro comando depois do último catch após o try.
Não é possível se retornar ao ponto de disparo executando um comando return em um tratador catch. Tal
return simplesmente retorna para a função que chamou a função que contém o bloco catch.
Erro comum de programação comum 13.10
Assumir que, depois que uma exceção é processada, o controle retornará ao primeiro comando depois do

throw é um erro de lógica.
Observação de engenharia de software 13.9
_____ Outra razão para não usar exceções para fluxo de controle convencional é que estas “exceções adicionais” podem se confundir com exceções genuínas de tratamento de erro. Torna-se mais difícil para o programador manter o controle do número de casos de exceção. Por exemplo, quando um programa processa uma variedade excessiva de exceções, podemos realmente estar certos do que está sendo capturado por um catch (. . .) ? Situações excepcionais deveriam ser raras, e não comuns.
Quando uma exceção é capturada, é possível que recursos tenham sido alocados, mas ainda não liberados no bloco try. O tratador czatch. se possível, deveria liberar estes recursos. Por exemplo, o tratador catch deveria eliminar o espaço alocado por new e fechar quaisquer arquivos abertos no bloco try que disparou a exceção.
Um bloco catch pode processar o erro de uma maneira que possibilite ao programa continuar a executar
corretamente. Ou o bloco catch pode terminar o programa.

Um tratador catch pode ele mesmo descobrir um erro e disparar uma exceção. Tal exceção não será processada por tratadores catch associados ao mesmo bloco try que o tratador catch que está disparando a exceção. Em vez disso, a exceção disparada será capturada, se possível, por um tratador catch associado ao próximo bloco try mais externo.
Erro comum de programação 13.11
É um erro de lógica assumir que uma exceção disparada a partir de um tratador catch serci processada por aquele tratador ou por qualquer outro tratador associado ao bloco try que disparou a exceção que fez com que o tratador catch original fosse executado.
13.8 Disparando novamente uma exceção
É possível que o tratador que captura uma exceção decida que ele não pode processar a exceção ou que ele simplesmente queira liberar recursos antes de deixar algum outro tratá-la. Neste caso, o tratador pode simplesmente disparar novamente a exceção, com o comando
throw;
Tal throw sem argumentos dispara novamente a exceção. Se nenhuma exceção foi disparada, então o novo disparo provoca uma chamada para terminate.
Erro comum de programação 13.12
Colocar um comando throw vazio fora de um tratador catch; executar tal throw provoca uma chamada para terminate.
Ainda que um tratador possa processar uma exceção, e independentemente de se fazer qualquer processamento sobre aquela exceção, o tratador ainda pode disparar novamente a exceção para um processamento adicional fora do tratador.
Uma exceção disparada novamente é detectada pelo próximo bloco try externo e é tratada por um tratador de

exceção listado depois daquele bloco try externo.
Observação de engenharia de software 13. 10
_____ Use catch (. . .) para executar recuperação que não depende do tipo da exceção, tal como liberar recursos comuns. A exceção pode ser disparada novamente para alertar blocos catch externos mais
espec(ficos.
O programa da Fig. 13.2 demonstra o novo disparo de uma exceção. No bloco try de main. a função throwException é chamada na linha 31. No bloco try da função throwException, o comando throw na linha 17 dispara uma instância da classe exception da biblioteca padrão (definida no arquivo de cabeçalho <exception>). Esta exceção é imediatamente capturada no tratador catch na linha 19, que imprime uma mensagem de erro e então dispara novamente a exceção. Isto termina a função throwException e retorna o controle para o bloco try/catch em main. A exceção é novamente capturada na linha 34 e é impressa uma mensagem de erro.

```
1 // Fig. 13.2: figl3O2.cpp
2 II Demonstração de se disparar novamente uma exceção.
3 #include <iostream>
4
5 using std: :cout;
6 using std: :endl;
```

Fig. 13.2 Disparando novamente uma exceção (parte 1 de 2).


```
7
8 #include <exception>
9
10 using std::exception;
li
12 void throwException()
13
14 // Dispara urna exceção e a captura imediatamente.
15 try{
16 cout « “Função throwException\n’;
17 throw exceptionO; II gera exceção
18
19 catch( exception e
20
21 cout « “Exceção tratada na função throwException\n”;
22 throw; II dispara novamente a exceção para processamento adicional
23
24
25 cout « “Isto também não deve ser impresso\n”;
26
27
28 int main()
```

```
29
30 try{
31 throwException
32 cout « “Isto não deve ser impresso\n”;
33
34 catch ( exception e
35
36 cout « “Exceção tratada em main\n”;
37
38
39 cout « “Controle do programa continua após captura em main”
40 « endl;
41 return 0;
42
```

```
Função throwException
Exceção tratada na função throwException
Exceção tratada em main
Controle do programa continua após captura em main
```

Fig. 13.2 Disparando novamente uma exceção (parte 2 de 2).

13.9 Especificações de exceção

Uma especificação de exceção enumera uma lista de exceções que podem ser disparadas por uma função.

```
int g( double h ) throw ( a, b, c
II corpo da função
```

É possível se restringir os tipos de exceção disparadas por uma função. Os tipos de exceção são especificados na declaração da função como uma especificação de exceção (também chamada de lista de throw). A especificação de exceção lista as exceções que podem ser disparadas. Uma função pode disparar as exceções indicadas ou tipos

derivados. Apesar desta suposta garantia de que outros tipos de exceção não serão disparados, é possível se fazer isso. Se uma exceção não-listada na especificação de exceção é disparada, a função unexpected é chamada. Colocar throw ( ) (i.e., uma especificação de exceção vazia) depois da lista de parâmetros de uma função
indica que a função não disparará quaisquer exceções. Tal função poderia, de fato, disparar uma exceção; isso também geraria uma chamada à função unexpected.

Erro comum de programação 13.13

Disparar uma exceção que não está na especificação de exceções de uma função gera uma chamada para
unexpected.

Uma função sem especificação de exceção pode disparar qualquer exceção.

```
void gO; // esta função pode disparar qualquer exceção
```

O significado da função unexpected pode ser redefinido chamando a função

set_unexpected.
Um aspecto interessante do tratamento de exceções é que o compilador não considerará como um erro de
sintaxe se uma função contiver uma expressão throw para uma exceção não-listada na especificação de exceções da função. A função deve tentar disparar aquela exceção, durante a execução, antes de o erro ser capturado.
Se uma função dispara uma exceção de um tipo de classe particular, aquela função também pode disparar
exceções de todas as classes derivadas daquela classe por herança public.
13.10 Processando exceções inesperadas
A função unexpected chama a função especificada pela função set_unexpected. Se nenhuma função foi especificada desta maneira, terminate é chamada por default.
A função terminate pode ser explicitamente chamada, se urna exceção disparada não pode ser capturada, se a pilha foi corrompida durante o tratamento de exceções, como a ação default para uma chamada à função unexpected e se, durante o desempilhamento da pilha iniciada por uma exceção, uma tentativa por um destruidor de disparar uma exceção faz com que terminate seja chamada.
A função set terminate pode especificar a função que será chamada quando terminate é chamada.
Caso contrário, terminate chama abort.
Os protótipos para as funções set tertninate e set unexpected estão localizados no arquivo de
cabeçalho <exception>.
A função set terminate e a função set unexpected retornam, cada uma, um ponteiro para a última
função chamada por terminate e unexpected. Isso possibilita ao programador salvar o ponteiro da função, de modo que ele possa ser restabelecido mais tarde.
As funções set terininate e set unexpected aceitam ponteiros para funções como parâmetros.
Cada parâmetro deve apontar para uma função com tipo de retorno void e sem nenhum argumento.
Se a última ação de uma função de término definida pelo usuário não for sair de um programa, a função
abort será automaticamente chamada para terminar a execução do programa depois que os outros comandos da função de término definida pelo usuário forem executados.
13.11 Desempilhando a pilha
Quando uma exceção é disparada mas não capturada em um escopo particular, a pilha (stack) de chamadas de função é desempilhada e é feita uma tentativa para capturar a exceção no próximo bloco try/catch externo. Desempilhar a pilha de chamadas de função significa que a função em que a exceção não foi capturada termina, todas as variáveis locais naquela função são destruídas e o controle retorna para o ponto em que a função foi chamada. Se aquele ponto no

programa está em um bloco try. é feita uma tentativa para capturar a exceção. Se aquele ponto no programa não está em um bloco try ou a exceção não é capturada, o desempilhamento acontece novamente. Como mencionado na seção anterior, se a exceção não é capturada no programa, a função terminate é chamada para terminar o programa. O programa da Fig. 13.3 demonstra o desempilhamento da pilha.


```
1 II Fig. 13.3: figl3O3.cpp
2 // Demonstrando o desempilhamento da pilha.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <stdexcept>
9
10 using std: :runtime_error;
11
12 void function3() throw ( runtimeerror
13
14 throw runtimeerror( “runtimeerror em function3” );
15
16
17 void function2() throw ( runtimeerror
18
19 function3O;
20
21
22 void functionl() throw ( runtimeerror
23
24 function2O;
25
26
27 int main()
28
29 try {
30 functionlO;
31
32 catch ( runtimeerror e
33 {
34 cout « “Ocorreu exceção: « e.what() « endi;
35 1
36
37 return 0;
38 }
```

Ocorreu exceção: runtime_error em function3
Fig. 13.3 Demonstração do desempilhamento da pilha.
Em main, o bloco try na linha 30 chama functionl. Em seguida, functionl (definida na linha 22) chama function2. Então, function2 (definida na linha 17) chama function3. A linha 14 de function3 dispara um objeto exception. Como a linha 14 não está em um bloco try, acontece o desempilhamento da pilha - a funcao3 termina na linha 19 e o controle retoma para a function2. Como a linha 19 não está em um bloco try. acontece novamente o desempilhamento da pilha - a function2 termina na linha 24 e o controle retoma para a function. Como a linha 24 não está em um bloco try, o desempilhamento da pilha acontece mais um vez - a functionl termina na linha 30 e o controle retoma para main. Como a linha 30 está em um bloco try, a exceção pode ser capturada e processada no primeiro tratador catch correspondente depois do bloco try (na linha 32).

13.12 Construtores, destruidores e o tratamento de exceções

Primeiro, vamos tratar de um assunto que mencionamos, mas que ainda tem que ser satisfatoriamente resolvido. O que acontece quando um erro é descoberto em um construtor? Por exemplo, como um construtor de String

deveria responder quando new falha e indica que foi impossibilitado de obter o espaço necessário para manter a representação interna do String? O problema é que um construtor não pode retornar um valor; assim, como comunicamos ao mundo exterior que o objeto não foi corretamente construído? Um esquema é simplesmente retornar o objeto inapropriadamente construído e esperar que alguém que use o objeto faça testes apropriados para determinar que o objeto é de fato ruim. Outro esquema é inicializar alguma variável fora do construtor. O disparo de uma exceção passa para o mundo exterior as informações sobre a falha do construtor e a responsabilidade de tratá-la.

Para capturar uma exceção, o tratador de exceção deve ter acesso a um construtor de cópia para o objeto
disparado (cópia default membro a membro também é válida).

Exceções disparadas em construtores fazem com que destruidores sejam chamados para quaisquer objetos
construídos como parte do objeto que está sendo construído antes de a exceção ser disparada.

Destruidores são chamados para todo objeto automático construído em um bloco try antes de uma exceção ser disparada. Uma exceção é tratada no momento em que o tratador começa a ser executado; é garantido que o desempilhamento da pilha foi completado até aquele ponto. Se um destruidor invocado como resultado do desempilhamento da pilha dispara uma exceção, terminate é chamada.

Se um objeto tem objetos membro e se uma exceção é disparada antes de o objeto externo estar completamente construído, então destruidores serão executados para os objetos membro que foram completamente construídos antes da ocorrência da exceção.

Se um array de objetos estava parcialmente construído quando ocorreu uma

exceção, só os destruidores para
os elementos do array construídos serão chamados.
Uma exceção poderia impedir a execução de código que normalmente liberaria um recurso, causando deste
modo uma perda de recurso. Uma técnica para solucionar este problema é inicializar um objeto local quando o recurso é adquirido. Quando ocorrer uma exceção, o destruidor será invocado e pode liberar o recurso.
E possível se capturar exceções disparadas a partir de destruidores incluindo a função que chama o destruidor
em um bloco try e fornecendo um tratador catch com o tipo apropriado. O destruidor do objeto disparado é executado depois que um tratador de exceção completa a sua execução.

13.13 Exceções e herança

Várias classes de exceção podem ser derivadas de uma classe base comum. Se um catch captura um ponteiro ou referência para um objeto de exceção de um tipo de uma classe base, ele também captura um ponteiro ou referência para todos os objetos de classes derivadas daquela classe base. Isso pode permitir o processamento polimórfico de erros relacionados.

® Dica de teste e depura ção 13.2

Usar herança com exceções possibilita a um tratador de exceção capturar erros relacionados com uma notação bastante concisa. Poderíamos certamente capturar cada tipo de ponteiro ou referência para um objeto de exceção de uma classe derivada individualmente, mas é mais conciso se capturar ponteiros ou referências para objetos de exceção da classe base. Além disso, capturar ponteiros ou referências para objetos de exceção de classes derivadas individualmente é sujeito a erros se o programador se esquecer de incluir testes explícitos para um ou mais dos tipos de ponteiro ou de referência para a classe derivada.

13.14 Processando falhas de new

Existem vários métodos de se lidar com falhas de new. Até este ponto, usamos a macro assert para testar o valor retornado por new. Se aquele valor for O, a macro assert termina o programa. Isto não é um mecanismo robusto para lidar com falhas de new - ele não nos permite qualquer forma de se recuperar da falha. O padrão C++ especifica que, quando new falha, é disparada uma exceção bad alloc (definida no arquivo de cabeçalho <new>). Porém, alguns compiladores podem não estar de acordo com o padrão C++ e, portanto, usam a versão de new que retorna O quando falha. Nesta seção, apresentamos três exemplos de falhas de new. O primeiro exemplo retorna O quando new falha. O segundo e terceiro exemplos usam a versão de new que dispara uma exceção badalloc quando new falha.

A Fig. 13.4 demonstra new retornando O quando falha em alocar a quantidade solicitada de memória. A estrutura for na linha 12 deveria executar o laço 50 vezes e alocar um array de 5.000.000 de valores double (i.e., 40.000.000 bytes,

porque um double normalmente ocupa 8 bytes) a cada passagem pelo laço. A estrutura if na linha 15 testa o resultado de cada operação new. para determinar se a memória foi alocada. Se new falha e retorna 0, a mensagem “Alocação de memória falhou’ é impressa e o laço termina.

```
1 II Fig. 13.4: figl3_04.cpp
2 // Demonstrando new retornando 0
3 II quando a memória não é alocada
4 #include <iostream>
5
6 using std::cout;
7
8 int main()
9{
10 double *ptr[ 50 ];
11
12 for ( int i = 0; i < 50; i++
13 ptr[ i ] = new double[ 5000000 );
14
15 if ( ptr[ i ] == O ) ( // new falhou na alocação de memória
16 cout « “Alocação de memória falhou para ptr[
17 « i « “
18 break;
19 }
20 else
21 cout « “Alocados 5000000 doubles em ptr[
22 « i «
23 }
24
25 return 0;
26 }
```

```
Alocados 5000000 doubles em ptr[ O
Alocados 5000000 douiles em ptr[ 1
Alocados 5000000 doubles em ptr[ 2
Alocados 5000000 doubles em ptr[ 3
Alocação de memória falhou para ptr[ 4 1
```

Fig. 13.4 Demonstrando new retornando O em caso de falha.

A saída mostra que só quatro repetições do laço foram executadas antes de new falhar e as repetições do laço terminarem. Sua saída pode ser diferente, dependendo da memória física, do espaço em disco disponível para memória virtual em seu sistema e do compilador que você usar para compilar o programa.

___ A Fig. 13.5 demonstra new disparando bad alloc quando ela falhar em alocar a memória solicitada. A estrutura for na linha 18 dentro do bloco try deve executar 50 iterações e em cada uma alocar um array de 5.000.000 de valores double (i.e., 40.000.000 bytes, porque um double normalmente é 8 bytes). Se new falha e dispara uma exceção bad alloc, o laço termina e o programa

continua no fluxo de controle de tratamento de exceções na linha 24, onde a exceção é capturada e processada. A mensagem “Ocorreu uma exceção:” é impressa, seguida pelo string (contendo a mensagem específica para a exceção “Falha na alocação”) retornado por exception . what () . A saída mostra que só quatro repetições do laço foram executadas antes de new falhar e disparar a exceção bad alloc. Sua saída pode ser diferente, dependendo da memória física, do
espaço em disco disponível para memória virtual em seu sistema e do compilador que você usar para compilar o programa.

CAPÍTULO 13 - TRATAMENTO DE EXCEÇÕES 713

```
1 // Fig. 13.5: figl3O5.cpp
2 1/ Demonstrando new disparando bad_alloc
3 Il quando a memória não é alocada
4 #include <iostreain>
5
6 using std: :cout;
7 using std::endl;
8
9 #include <new>
10
11 using std: :bad_alloc;
12
13 int main()
14 {
15 double *ptr[ 50 1;
16
17 try{
18 for ( int i = 0; i < 50; i++
19 ptr[ i ] = new double[ 5000000 );
20 cout « “Alocados 5000000 doubles em ptr[
21 « i «
22 }
23 )
24 catch ( badalloc exception
25 cout « Ocorreu uma exceção:
26 « exception.what() « endl;
27 }
28
29 return 0;
30 }
Alocados 5000000 doubles em ptr[ O
Alocados 5000000 doubles em ptr[ 1
Alocados 5000000 doubles em ptr[ 2 ]
Alocados 5000000 doubles em ptr[ 3
```

Ocorreu uma exceção: falha na alocação
Fig. 13.5 Demonstrando new disparando bad_alloc em caso de falha.
Os compiladores variam em seu suporte ao tratamento de falhas de new. Muitos compiladores de C++ retomam O por default quando new falha. Alguns destes compiladores suportam new disparando uma exceção se o arquivo de cabeçalho <new> (ou <new . h>) for incluído. Outros compiladores disparam bad alloc por default, não importando se você inclui ou não o arquivo de cabeçalho <new>. Leia a documentação de seu compilador para determinar qual o suporte oferecido por seu compilador para tratamento de falhas de new.
O padrão C++ especifica que compiladores aderentes ao padrão podem ainda usar uma versão de new que retoma O quando ele falha. Para este finalidade, o arquivo de cabeçalho <new> define nothrow (do tipo nothrowt), que é usado como segue:

```
double *ptr = new( nothrow ) double[ 5000000 ];
```

O comando precedente indica que a versão de new que não dispara exceções bad alloc (i.e., nothrow) deve ser usada para alocar um array de 5.000.000 de doubles.
Observação de engenharia de software 13.11
_____ O padrão C+ + recomenda que, para tornar os programas mais robustos, os programadores usem a versão de new que dispara exceções bad alloc no caso de insucessos de new.

Existe um recurso adicional que pode ser usado para tratamento de falhas de new. A função setnewharidler (prototipada no arquivo de cabeçalho <new>) aceita como seu parâmetro um ponteiro de função para uma função que não recebe nenhum argumento e retorna void. O ponteiro de função é registrado como a função a chamar quando new falha. Isto fornece ao programador um método uniforme de processar todas as falhas de new. não importando onde a falha acontece no programa. Uma vez que um tratador de new é registrado no programa com set new handier. new não disparará bad alloc quando fracassar. O operador new é na realidade um laço que tenta adquirir memória. Se a memória é alocada, new retorna um ponteiro para aquela memória. Se new falha em alocar memória e nenhuma função tratadora de new foi registrada com set new handier, new dispara uma exceção bad alloc. Se new falha na alocação de memória e uma função tratadora de new foi registrada, a função tratadora de new é chamada. O padrão C++ especifica que a função tratadora de new deveria executar uma das tarefas seguintes:
1. Tornar disponível mais memória, apagando outra memória dinamicamente alocada, e retornar ao laço no operador new para tentar alocar a memória novamente.
2. Disparar uma exceção do tipo badalloc.
3. Chamar a função abort ou exit (ambas do arquivo de cabeçalho <csdtlib>), para terminar o programa.
O programa da Fig. 13.6 demonstra set new handier. A função

customNewHandler simplesmente imprime uma mensagem de erro e termina o programa com uma chamada para abort. A saída mostra que só três repetições do laço foram executadas antes de new ter falhado e disparado a exceção bad alloc. Sua saída pode ser diferente, dependendo da memória física, do espaço em disco disponível para memória virtual em seu sistema e do compilador que você usar para compilar o programa.

```
1 II Fig. 13.6: figl3O6.cpp
2 // Demonstrando setnewhandler
3 #include <iostream>
4
5 using std: :cout;
6 using std: :cerr;
7
8 #include <new>
9 #include <cstdlib>
10
11 using std: :set_new_handler;
12
13 void custoznNewHandler()
14
15 cerr « ‘customNewHandler foi chamada;
16 abortO;
17
18
19 int main()
20{
21 double *ptr[ 50 ];
22 setnewhandler( customNewHandler );
23
24 for O int i = o; i < 50; i++
25 ptr[ i ] = new double[ 5000000 ];
26
27 cout « ‘Alocados 5000000 doubles em ptr[
28 « i «
29 }
30
31 return 0;
32 }
```

Fig. 13.6 Demonstrando set new handier (parte 1 de 2).

```
Alocados 5000000 doubles em ptr[ O
Alocados 5000000 doubles em ptr[ 1
Alocados 5000000 doubles em ptr[ 2
Alocados 5000000 doubles em ptr[ 3 1
customNewHandler foi chamada
```

Fig. 13.6 Demonstrando set newhandler (parte 2 de 2).

13.15 A classe auto_ptr e a alocação dinâmica de memória

Uma prática comum de programação é alocar memória dinâmica (possivelmente um objeto) na memória livre, atribuir o endereço daquela memória a um ponteiro, usar o ponteiro para manipular a memória e desalocar a memória com delete quando a memória não for mais necessária. Se uma exceção acontece depois de a memória ter sido alocada e antes de o comando delete ser executado, pode ocorrer uma perda de memória. O padrão C++ fornece o gabarito de classe autoytr no arquivo de cabeçalho <memory> para lidar com esta situação.
Um objeto da classe auto_ptr mantém um ponteiro para memória dinamicamente alocada. Quando um objeto autoytr sai do escopo, ele executa a operação delete sobre seu membro de dados ponteiro. O gabarito de classe autoytr fornece operadores * e -> de modo que um objeto auto_ptr possa ser usado como uma variável ponteiro regular. A Fig. 13.7 demonstra um objeto autoytr que aponta para um objeto da classe Integer (definida nas linhas 12 a 22).

```
1 II Fig. 13.7: figl3O7.cpp
2 II Demonstrando autoytr
3 #include <iostream>

4

5 using std: :cout;
6 using std: :endl;

7

8 #include <memory>

9

using std: :auto_ptr;

class Integer {
public:
Integer( int i = O ) : value( i
cout « “Construtor para Integer “ « value « endl;
-Integer O
cout « “Destruidor
void setInteger( int i
```

```
int getInteger() const
private:
int value;

cout « “Criando um objeto autoytr que
« “aponta para um Integer\n”;

autoytr< Integer > ptrToInteger( new Integer( 7 ) );

cout « “Usando o auto2tr para manipular o Integer\n”;
ptrToInteger->setInteger( 99 );

10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32

para Integer “ « value « endl; value = i;
return value;

int main()
```

Fig. 13.7 Demonstrando autoytr (parte 1 de 2).


```
33 cout « ‘Integer depois de setInteger:
```

j 34 « (*ptrToInteger) .getInteger()
35 « “\nTerminando o programa” « endl;
36
37 return O;
38 }
Criando um objeto autoytr que aponta para um Integer
Construtor para Integer 7
Usando o autoytr para manipular o Integer
Integer depois de setInteger: 99
Terminando o programa
Destruidor para Integer 99
Fig. 13.7 Demonstrando auto_ptr (parte 2 de 2).
Alinha 29
autoytr< Integer > ptrToInteger(new Integer(7));
cria o objeto auto-ptr ptrToInteger e inicializa-o com um ponteiro para um objeto Integer dinamicamente alocado contendo o valor 7.
A linha 32
ptrToInteger->setInteger(99);
usa o operador -> sobrecarregado de auto_ptr e o operador chamada de função ( ) para chamar a função setInteger sobre o objeto Integer apontado por ptrToInteger.
A chamada
(*ptrToInteger) .getInteger()
na linha 34 usa o operador * de autoytr sobrecarregado para derreferenciar ptrToInteger e então usa o : operador ponto (.) e o operador chamada de função ( ) para chamar a função getInteger sobre o objeto
Integer apontado por ptrToInteger.
Como ptrToInteger é uma variável automática local em main, ptrToInteger é destruído quando main termina. Isto força um delete do objeto Integer apontado por ptrToInteger que, é claro, força uma chamada para o destruidor da classe Integer. Porém, mais importante, esta técnica pode prevenir perdas de memória.

13.16 Hierarquia de exceções da biblioteca padrão
A experiência mostrou que exceções se enquadram bem em várias categorias. O padrão C++ inclui uma hierarquia
de classes de exceção. Essa hierarquia é encabeçada pela classe base exception (definida no arquivo de cabeça lh <exception>) que contém a função what () que é sobrescrita em cada classe derivada, para emitir uma mensagem de erro apropriada.
Da classe base exception. algumas das classes derivadas imediatas são runtime error e logicerror
(ambas definidas no cabeçalho <stdexcept>). cada uma com várias classes derivadas.
Também derivada de exception são as exceções disparadas por recursos da linguagem C++ - por exem plo bad alloc é disparada por new (Seção 13.14), badcast é disparada por dynamiccast (Capítulo

21) e bad typeid é disparada por typeid (Capítulo 21). Incluindo std: :bad exception na lista de r
throw de uma função, se uma exceção inesperada acontece, unexpected ( ) disparará bad_exception em
vez de terminar (por default) ou em vez de chamar outra função especificada com set_unexpected.
A classe logic error é a classe base de várias classes de exceção padrão que indicam erros de lógica em
programas que podem freqüentemente ser evitados escrevendo-se código apropriado. As descrições de algumas
destas classes aparecem a seguir. A classe invalid argument indica que um parâmetro inválido foi passado

para uma função (codificação apropriada pode, é claro, evitar que parâmetros inválidos cheguem a uma função). A classe length error indica que um comprimento maior que o tamanho máximo permitido para o objeto que está sendo tratado foi usado para aquele objeto (disparamos length errors no Capítulo 19 quando lidamos com strings). A classe out of range indica que um valor tal como um subscrito para um array ou string caíram fora do intervalo válido.
A classe runtime error é a classe base de várias outras classes de exceção padrão que indicam erros em um programa que só podem ser descobertos durante a execução. A classe overflow error indica que ocorreu um erro de ovetfiow em uma operação aritmética. A classe underflow error indica que aconteceu um erro de underflow em uma operação aritmética.
Observação de engenharia de software 13.12
_____ A hierarquia padrão de exception se destina a servir como um ponto de partida. Os usuá rios podem disparar exceções padrão, disparar exceções derivadas das exceções padrão ou disparar suas próprias
exceções não-derivadas das exceções padrão.
Erro comum de programação 13.14
Classes de exceção definidas pelo usuário não precisam ser derivadas da classe exception. Assim, não é garantido que escrever catch (exception e) capture todas as exceções que um programa pode encontrar
® Dica de teste e depura ção 13.3
Para capturar todas as exceções que podem ser disparadas em um bloco try, use catch (. .
Resumo
• Alguns exemplos comuns de exceções são: subscritos fora do intervalo válido em arrays, overfiow em operações aritméticas, divisão por zero, parâmetros de função inválidos e verificação de memória insuficiente para atender a um pedido de alocação por new.
• O espírito por trás do tratamento de exceções é possibilitar aos programas capturar e tratar erros, em vez de deixá-los acontecer e simplesmente sofrer as conseqüências. Com o tratamento de exceções, se o programador não fornece

um meio de tratamento para um erro fatal, o programa terminará; erros não-fatais permitem normalmente a um programa continuar a execução, mas produzindo resultados incorretos.
• O tratamento de exceções foi projetado para lidar com erros síncronos, i.e., erros que acontecem como resultado da execução de um programa.
• O tratamento de exceções não foi projetado para tratar de situações assíncronas, tais como chegadas de mensagens pela rede, conclusões de EIS de disco, diques do mouse, etc.: estas são mais bem manipuladas através de outros meios. tal como o processamento de interrupções.
• O tratamento de exceções é tipicamente usado em situações em que o erro será tratado por uma parte diferente do programa (i.e., um escopo diferente) daquela que descobriu o erro.
• As exceções não devem ser usadas como mecanismo para especificar o fluxo de controle. O fluxo de controle com estruturas de controle convencionais é geralmente mais claro e eficiente do que com exceções.
• O tratamento de exceções deve ser usado para processar exceções de componentes do programa que não estão preparados para tratar diretamente aquelas exceções.
• O tratamento de exceções deve ser usado para processar exceções de componentes de software tais como funções, bibliotecas e classes que provavelmente serão amplamente usados e onde não faz sentido para aqueles componentes tratar suas próprias exceções.
• O tratamento de exceções deveria ser usado em grandes projetos para tratar o processamento de erros de uma maneira uniforme para o projeto inteiro.
• O tratamento de exceções em C++ é voltado para situações em que a função que descobre um erro está impossibilitada de tratá-lo. Tal função disparará uma exceção. Se a exceção corresponde ao tipo do parâmetro em um dos blocos catch. o

código para aquele bloco catch é executado. Caso contrário, é chamada a função terminate que, por default, chama a função abort.
• O programador inclui em um bloco try o código que pode gerar um erro que produzirá uma exceção. O bloco try é imediatamente seguido por um ou mais blocos catch. Cada bloco catch especifica o tipo de exceção que ele pode capturar e tratar. Cada bloco catch contém um tratador de exceção.
• Quando uma exceção é disparada, o controle do programa deixa o bloco try e procura os blocos catch em busca de um tratador apropriado. Se nenhuma exceção é disparada em um bloco try, os tratadores de exceção para aquele bloco são saltados e o programa retoma a execução depois do último bloco catch.
• Exceções são disparadas em um bloco try em uma função ou de uma função chamada direta ou indiretamente a partir do
• Uma vez que uma exceção é disparada, o controle não pode retornar diretamente ao ponto de disparo.
• É possível se passar informações para o tratador de exceção a partir do ponto em que ocorreu a exceção. Essas informações são o tipo de objeto disparado ou

informações colocadas no objeto disparado.
• Um tipo de exceção popular disparada é char*. É comum simplesmente se incluir uma mensagem de erro como o operando de throw.
• As exceções disparadas por uma função particular podem ser especificadas com uma especificação de exceção. Uma especificação de exceção vazia afirma que a função não disparará quaisquer exceções.
• As exceções são capturadas pelo tratador de exceção mais próximo (para o bloco try do qual a exceção foi disparada) especificando-se um tipo apropriado.
• Como parte do disparo de uma exceção, uma cópia temporária do operando do throw é criada e inicializada. Este objeto temporário então inicializa a variável apropriada no tratador de exceção. O objeto temporário é destruído quando se sai do tratador de exceção.
• Erros não são sempre verificados explicitamente. Um bloco try. por exemplo, pode parecer não conter nenhuma verificação de erro e não incluir nenhum comando throw. Mas o código referenciado no bloco try pode certamente fazer com que código de verificação de erro seja executado eliminar espaço.
• Uma exceção termina o bloco no qual a exceção aconteceu.
• Os tratadores de exceção estão contidos em blocos catch. Cada bloco catch começa com a palavra-chave catch seguida por parênteses contendo um tipo e um nome de parâmetro opcional. Isto é seguido por chaves delimitando o código de
tratamento de exceção. Quando uma exceção é capturada, o código no bloco catch é executado. O tratador catch define seu próprio escopo.
• O parâmetro em um tratador catch pode ter um nome ou não. Se o parâmetro tem um nome, o parâmetro pode ser referenciado n tratador. Se o parâmetro não tem nome, i.e., se somente um tipo é listado com a finalidade de procurar uma correspondência com o tipo de objeto disparado ou reticências para todos os tipos, então o tratador ignorará o objeto disparado. O tratador
pode disparar novamente o objeto para um bloco try externo.
• É possível se especificar um comportamento personalizado para substituir a função terininate. designando-se outra função a ser executada e fornecendo-se aquele nome de função como o parâmetro em uma chamada da função setterininate.
• catch ( ) significa capturar todas as exceções.
• É possível que nenhum tratador tenha uma correspondência com um objeto disparado particular. Isso faz com que a procura por uma correspondência continue em um bloco try externo.
• Os tratadores de exceção são pesquisados na ordem para encontrar uma correspondência apropriada. O primeiro tratador que tem uma correspondência é executado. Quando aquele tratador termina a execução, o controle continua com o primeiro comando depois do último bloco catch.
• A ordem dos tratadores de exceção afeta a maneira como uma exceção é tratada.
• Um objeto de uma classe derivada pode ser capturado ou por um tratador especificando o tipo da classe derivada ou por tratadores especificando os tipos de quaisquer classes base daquela classe derivada.

• Às vezes, um programa pode processar muitos tipos intimamente relacionados de exceções. Em vez de fornecer classes de exceção separadas e tratadores catch para cada uma, um programador pode fornecer uma classe de exceção única e um

bloco try.

tratador catch para um grupo de exceções. À medida que ocorre cada exceção, o objeto de exceção pode ser criado com dados private diferentes. O tratador catch pode examinar estes dados private para identificar o tipo da exceção.
• É possível que, muito embora esteja disponível uma correspondência precisa, uma correspondência exigindo conversões padrões será feita porque aquele tratador aparece antes daquele que resultaria em uma correspondência precisa.
• Por default, se nenhum tratador for encontrado para uma exceção, o programa termina.
• Um tratador de exceção não pode acessar diretamente variáveis no escopo de seu bloco try. As informações de que o tratador necessita são normalmente passadas no objeto disparado.
• Os tratadores de exceção podem examinar mais de perto um erro e decidir chamar terminate. Podem disparar novamente uma exceção. Podem converter um tipo de exceção para outro disparando uma exceção diferente. Podem executar qualquer recuperação necessária e retomar a execução depois do último tratador de exceção. Podem examinar a situação que está causando o erro, remover a causa do erro e tentar chamar novamente a função original que causou uma exceção (isto não cria uma recursão infinita). Podem simplesmente retomar algum valor de estado para seu ambiente, etc.
• Um tratador que captura um objeto de uma classe derivada deveria ser colocado antes de um tratador que captura um objeto da classe base. Se o tratador da classe base fosse colocado primeiro, capturaria tanto os objetos da classe base como os objetos de classes derivadas daquela classe base.
• Quando uma exceção é capturada, é possível que recursos possam ter sido alocados, mas ainda não liberados no bloco try. O tratador catch deveria liberar estes recursos.
• É possível que um tratador catch possa decidir que ele não pode processar a exceção. Neste caso, o tratador pode simplesmente disparar novamente a exceção. Um throw sem parâmetros dispara novamente a exceção. Se nenhuma exceção foi disparada, então o novo disparo produz uma chamada para terminate.
• Ainda que um tratador possa processar uma exceção, e não importando se ele faz qualquer processamento daquela exceção, o tratador pode disparar novamente a exceção para processamento adicional fora do tratador. Uma exceção disparada novamente é detectada pelo próximo bloco try externo e é tratada por um tratador de exceção listado depois daquele bloco try externo.

• Uma função sem especificação de exceção pode disparar qualquer exceção.
• A função unexpected chama uma função especificada com a função set unexpected. Se nenhuma função foi especificada desta maneira, terminate é chamada por default.
• A função terminate pode ser chamada de vários modos: explicitamente; se uma exceção disparada não pode ser capturada; se a pilha de chamadas de funções foi corrompido durante o tratamento de exceções; como a ação defauit em uma chamada para unexpected; ou, se durante o desempilhamento da pilha iniciada por uma exceção, uma tentativa de disparar uma exceção feita por um destruidor faz com que terminate seja chamada.
• Os protótipos para as funções set terminate e set unexpected são encontrados no arquivo de cabeçalho <exception>.
• As funções set terminate e set_unexpected retornam ponteiros para a última função chamada por terminate e unexpected. isto possibilita ao programador salvar o ponteiro da função de modo que ele possa ser restaurado mais tarde.
• As funções setterminate e set unexpected aceitam ponteiros para funções como argumentos. Cada parâmetro deve apontar para uma função com tipo de retorno void e nenhum argumento.
Se a última ação de uma função de término definida pelo usuário não é sair de um programa, a função abort será automaticamente chamada para terminar a execução do programa depois de os outros comandos da função de término definida pelo usuário terem sido executados.
• Uma exceção disparada fora de um bloco try fará com que o programa termine.

• Se um tratador não pode ser encontrado depois de um bloco try. o desempilhamento da pilha continua até um tratador apropriado ser encontrado. Se, em última instância, um tratador não é encontrado, então é chamada terminate. o que, por default, aborta o programa com abort.
• Especificações de exceção listam as exceções que podem ser disparadas a partir de uma função. Uma função pode disparar as exceções indicadas ou pode disparar tipos derivados. Se uma exceção não-listada na especificação de exceção é disparada, unexpected é chamada.
• Se uma função dispara uma exceção de um tipo de classe particular, aquela função também pode disparar exceções de todas as classes derivadas daquela classe por herança public.

• Para captural uma exceção, o tratador de exceção deve ter acesso a um construtor de cópia para o objeto disparado.
• As exceções disparadas a partir de construtores fazem com que sejam chamados destruidores para todos os objetos de classes base completados e os objetos membros do objeto que estava sendo construído antes de a exceção ter sido disparada.
• Se um array de objetos havia sido parcialmente construído quando ocorreu uma exceção, só os destruidores para os elementos completamente construídos

do array serão chamados.
• As exceções disparadas a partir de destruidores podem ser capturadas incluindo-se a função que chama o destruidor em um bloco try e fornecendo-se um tratador catch com o tipo apropriado.
• Uma razão poderosa para usar herança com exceções é criar a possibilidade de capturar facilmente uma variedade de erros relacionados, com uma notação concisa. Poderíamos certamente capturar cada tipo de objeto de exceção de uma classe
derivada individualmente, mas, se todas as exceções derivadas são tratadas da mesma forma, é muito mais conciso simplesmente se capturar o objeto de exceção da classe base.
• O padrão C++ especifica que, quando new falha, ela dispara uma exceção bad alloc (bad alloc é definido no arquivo de cabeçalho <new>).
• Alguns compiladores não estão atualizados de acordo com o padrão C++ e ainda usam a versão de new que retorna O quando falha.
• A função set new handier (prototipada no arquivo de cabeçalho <new>) aceita como seu argumento um ponteiro de função para uma função que não aceita nenhum argumento e retorna void. O ponteiro de função é registrado como a função
a ser chamada quando new falha. Uma vez que um tratador de new é registrado com set_riew_haridler, new não disparará bad alba quando ocorrer uma falha.
• Um objeto da classe autoytr mantém um ponteiro para memória alocada dinamicamente. Quando um objeto autoytr sai do escopo, ele executa automaticamente uma operação delete sobre seu membro de dados ponteiro. O gabarito de
classe autoytr fornece operadores * e -> de modo que um objeto auto_ptr pode ser usado como uma variável ponteiro regular.
• O padrão C++ inclui uma hierarquia de classes de exceção encabeçadas pela classe base exception (definida no arquivo cabeçalho <exception>), que oferece o serviço what () que é redefinido em cada classe derivada para emitir uma mensagem de erro apropriada.
• Incluindo-se std: badexception na lista de throw de uma definição de função, se uma exceção inesperada ocorrer, unexpected () disparará bad exception em vez de terminar (por default) ou em vez de chamar outra função especificada com set_unexpected.

Terminologia

abort O disparar expressão
aplicativo para missões críticas disparar novamente uma exceção
argumento de catch disparar uma exceção
arquivo de cabeçalho <exception> disparar uma exceção inesperada
arquivo de cabeçalho <memory> dynamic_cast
arquivo de cabeçalho <new> especificação de exceção
arquivo de cabeçalho <stdexcept> especificação de exceção vazia
autoptr especificação de throw vazia
bad_alloc exceção
badcast exceção não-capturada

badtypeid exit()
bloco catch função sem especificação de exceção
bloco try invalid_argument
bloco try externo lengtherror
capturar um grupo de exceções lista de disparo
capturar uma exceção lista de exceções
catch(...) bogicerror
catch(void *) macroassert
condição excepcional new_handler
declaração de exceção nothrow
desempilhamento da pilha objeto de exceção

Dicas de desempenho
13.1 Embora seja possível se usar o tratamento de exceções para propósitos diferentes do tratamento de erros, isso pode reduzir o desempenho do programa.
13.2 O tratamento de exceções é geralmente implementado em compiladores de tal maneira que quando não ocorrem exceções, pouco ou nenhum os'erhead é imposto pela presença do código de tratamento de exceções. Quando ocorrem exceções, elas incorrem em overhead durante a execução. Certamente, a presença de código de tratamento de exceções faz o programa consumir mais memória.
Observações de engenharia de software
13.1 O fluxo de controle com estruturas de controle convencionais é geralmente mais claro e eficiente do que com exceções.
13.2 O tratamento de exceções é bem adequado para sistemas com componentes desenvolvidos separadamente. O tratamento de exceções facilita a combinação dos componentes. Cada componente pode executar sua própria detecção de exceção,
separada do tratamento das exceções em outro escopo.
13.3 Ao lidar com bibliotecas, o chamador da função de biblioteca provavelmente terá em mente um processamento de erro específico para uma exceção gerada na função de biblioteca. E improvável que uma função de biblioteca execute um processamento de erro que satisfaça às necessidades particulares de todos os usuários. Portanto, exceções são meios apropriados para tratar erros produzidos por funções de bibliotecas.
13.4 Uma idéia-chave do tratamento de exceções é que a parte de um programa ou sistema que tratará a exceção pode ser bastante diferente ou distante da parte do programa que descobriu e gerou a situação excepcional.
13.5 Se é necessário passar informações sobre o erro que causou uma exceção, tal informação pode ser colocada no objeto disparado. O tratador catch então vai conter um nome de parâmetro através do qual as informações podem ser
referenciadas.
13.6 Um objeto pode ser disparado sem conter informações a serem passadas;

nesse caso, o mero conhecimento de que uma exceção desse tipo foi disparada pode fornecer informações suficientes para o tratador fazer seu trabalho corretamente.
13.7 Um ponto fraco na técnica de capturar exceções com catch . . . é que normalmente você não pode ter certeza de qual é o tipo da exceção. Outro ponto fraco é que, sem um parâmetro nomeado, não existe nenhum modo de se referir ao
objeto de exceção dentro do tratador de exceção.
13.8 E melhor incorporar sua estratégia de tratamento de exceções a um sistema desde o começo do projeto. E difícil incorporar um tratamento de exceções efetivo depois de um sistema ter sido implementado.
13.9 Outra razão para não usar exceções para fluxo de controle convencional é que estas “exceções adicionais” podem se confundir com exceções genuínas de tratamento de erro. Torna-se mais difícil para o programador manter o controle do número de casos de exceção. Por exemplo, quando um programa processa uma variedade excessiva de exceções, podemos realmente estar certos do que está sendo capturado por um catch (. . .) ? Situações excepcionais deveriam ser raras, e não comuns.
13.10 Use catch (. . .) para executar recuperação que não depende do tipo da exceção, tal como liberar recursos comuns. A exceção pode ser disparada novamente para alertar blocos catch externos mais específicos.
13.11 O padrão C++ recomenda que, para tornar os programas mais robustos, os programadores usem a versão de new que dispara exceções bad alloc no caso de insucessos de new.
13.12 A hierarquia padrão de exception se destina a servir como um ponto de partida. Os usuários podem disparar exceções padrão, disparar exceções derivadas das exceções padrão ou disparar suas próprias exceções não-derivadas das exceções
padrão.
Dicas de teste e depura ção
13.1 O programador determina a ordem em que os tratadores de exceção são listados. Esta ordem pode afetar como exceções originadas naquele bloco try são manipuladas. Se você estiver obtendo um comportamento inesperado no tratamento de exceções do seu programa, pode ser porque um bloco catch anterior está interceptando e tratando as exceções antes que possam alcançar seu tratador catch planejado.
13.2 Usar herança com exceções possibilita a um tratador de exceção capturar erros relacionados com uma notação bastante concisa. Poderíamos certamente capturar cada tipo de ponteiro ou referência para um objeto de exceção de uma classe derivada individualmente, mas é mais conciso se capturar ponteiros ou referências para objetos de exceção da classe base. Além disso, capturar ponteiros ou referências para objetos de exceção de classes derivadas individualmente é sujeito a erros se o programador se esquecer de incluir testes explícitos para um ou mais dos tipos de ponteiro ou de referência para a classe derivada.
13.3 Para capturar todas as exceções que podem ser disparadas em um bloco

try, use catch (. .).

CAPÍTULO 13 - TRATAMENTO DE EXCEÇÕES 723

Exercícios de auto-revisão
13.1 Liste cinco exemplos comuns de exceções.
13.2 Dê várias razões pelas quais as técnicas de tratamento de exceções não devem ser usadas para controle convencional do programa.
13.3 Por que exceções são apropriadas para lidar com erros produzidos por funções de bibliotecas?
13.4 O que é uma “perda de recurso?”
13.5 Se nenhuma exceção é disparada de um bloco try, para onde vai ser transferido o controle depois de o bloco try
completar a execução?
13.6 O que ocorre se uma exceção é disparada fora de um bloco try7
13.7 Dê uma vantagem importante e uma desvantagem importante de se usar catch (. .
13.8 O que ocorre se nenhum tratador catch corresponde ao tipo de um objeto disparado?
13.9 O que ocorre se vários tratadores correspondem ao tipo do objeto disparado?
13.10 Por que um programador especificaria um tipo de uma classe base como o tipo de um tratador catch e então dispararia
objetos de tipos de classes derivadas?
13.11 Como pode um tratador catch ser escrito para processar tipos de erros relacionados sem usar herança entre classes de
exceção?
13.12 Que tipo de ponteiro é usado em um tratador catch para capturar alguma exceção de qualquer tipo de ponteiro?
13.13 Suponha que um tratador catch com uma correspondência precisa ao tipo de um objeto de exceção está disponível. Em que circunstâncias pode um tratador diferente ser executado para objetos de exceção daquele tipo?
13.14 Disparar uma exceção deve fazer com que o programa termine?
13.15 O que ocorre quando um tratador catch dispara uma exceção?
13.16 O que faz o comando throw; ?
13.17 Como o programador restringe os tipos de exceções que podem ser disparados de uma função?
13.18 O que ocorre se uma função dispara uma exceção de um tipo não permitido pela especificação de exceções para a função?
13.19 O que ocorre aos objetos automáticos que foram construídos em um bloco try quando aquele bloco dispara uma exceção?
Respostas aos exercícios de auto-revisão
13.1 Memória insuficiente para satisfazer uma solicitação de new. sobrescrito de array fora dos limites, overflow em operação aritmética, divisão por zero, parâmetros de função inválidos.
13.2 (a) O tratamento de exceções foi projetado para tratar situações que

acontecem pouco freqüentemente e resultam
freqüentemente no término do programa, assim os implementadores de compiladores não são obrigados a implementar o trata _____ mento de exceções de maneira otimizada. (b) O fluxo de controle com estruturas de controle convencionais é geralmente mais claro e eficiente do que com exceções. (c) Podem ocorrer problemas porque a pilha de chamadas de funções é desempilhada quando ocorrer uma exceção e os recursos alocados antes da exceção não podem ser liberados. (d) As “exceções adicionais” podem se confundir com exceções de erro genuínas. Torna-se mais difícil para o programador manter o controle do número maior de casos de exceção. O que um catch (. . .) realmente captura? . -
13.3 É improvável que uma função de biblioteca execute processamento de erro que satisfaça às necessidades de todos os usuários.
13.4 Um programa que aborta pode deixar um recurso em um estado no qual outros programas são incapazes de adquirir o recurso.
13.5 Os tratadores de exceção (nos blocos catch) para aquele bloco try são saltados e o programa retoma a execução depois do último bloco catch.
13.6 Uma exceção disparada fora de um bloco try provoca uma chamada para terminate.

13.7 A forma catch (. . .) captura qualquer tipo de erro disparado em um bloco try. Uma vantagem é que nenhum erro disparado pode escapar. Uma desvantagem é que o catch não tem nenhum parâmetro, assim ele não pode referenciar
informações no objeto disparado e não pode saber a causa do erro.
13.8 Isto faz com que a procura por uma correspondência continue no próximo bloco try externo. À medida que o processamento continua, pode eventualmente ser determinado que não existe nenhum tratador no programa que corresponde ao tipo do objeto disparado; neste caso, é chamada terminate. o que, por default, chama abort. Uma função terminate alternativa pode ser fornecida como um parâmetro para setterminate.
13.9 É executado o primeiro tratador de exceção correspondente ao tipo da exceção depois do bloco try.
13.10 Este é um meio agradável para se capturar tipos relacionados de exceções.
13.11 Forneça uma classe de exceção e um tratador catch únicos para um grupo de exceções. À medida que ocorre cada exceção, o objeto de exceção pode ser criado com dados privados diferentes. O tratador catch pode examinar estes
dados private para identificar o tipo da exceção.
13.12 void*.
13.13 Um tratador exigindo conversões padrão pode aparecer antes de uma correspondência precisa ser encontrada.
13.14 Não, mas ele termina o bloco em que a exceção é disparada.
13.15 A exceção será processada por um tratador catch (se existir um)

associado com o bloco try (se existir um) externo ao tratador catch que causou a exceção.
13.16 Ele dispara novamente a exceção.
13.17 Forneça uma especificação de exceção listando os tipos de exceção que podem ser disparadas da função.
13.18 A função unexpected é chamada.
13.19 Através do processo de desempilhamento da pilha de chamadas, são chamados os destruidores para estes objetos.
Exercícios
13.20 Liste as várias condições excepcionais que aconteceram em programas ao longo deste texto. Liste tantas condições excepcionais quantas puder. Para cada uma delas, descreva brevemente como um programa tipicamente trataria a exceção usando as técnicas de tratamento de exceções discutidas neste capítulo. Algumas exceções típicas são divisão por zero, oveiflow aritmético, subscrito de array fora dos limites, esgotamento da memória livre, etc.
13.21 Em que circunstâncias o programador não forneceria um nome de parâmetro ao definir o tipo do objeto que será capturado por um tratador?
13.22 Um programa contém o comando
throw;
Onde você esperaria achar normalmente um tal comando? E se esse comando apareceu em uma parte diferente de um programa?
13.23 Em que circunstâncias você usaria o comando seguinte?
catch(...) { throw; }
13.24 Compare e contraste o tratamento de exceções com os vários outros esquemas de processamento de erros discutidos no texto.
13.25 Liste as vantagens do tratamento de exceções em relação aos meios convencionais de processamento de erros.
13.26 Forneça razões pelas quais exceções não deveriam ser usadas como uma forma alternativa de controle do programa.
13.27 Descreva uma técnica para tratamento de exceções relacionadas.
13.28 Até este capítulo, achávamos que lidar com erros descobertos por construtores era um pouco complicado. O tratamento de exceções nos dá um meio muito melhor de lidar com tais erros. Considere um construtor para uma classe String. O construtor usa new para obter espaço da memória livre. Suponha que new falhe. Mostre como você lidaria com isto sem o tratamento de exceções. Discuta os aspectos-chave. Mostre como você lidaria com tal esgotamento de memória com tratamento de exceções. Explique por que o método de tratamento de exceções é superior.

13.29 Suponha que um programa dispara uma exceção e o tratador de exceção apropriado começa a ser executado. Agora, suponha que o tratador de exceção apropriado dispara a mesma exceção. Isto cria uma recursão infinita? Escreva um programa para verificar sua observação.
13.3() Use herança para criar uma classe base de exceção e várias classes de

exceção derivadas. Então, mostre que um tratador
catch especificando a classe base pode capturar exceções das classe derivadas.
13.31 Mostre uma expressão condicional que retorna ou um double ou um int. Forneça um tratador catch int e um
tratador catch double. Mostre que só o tratador catch double é executado, não importando se int ou double é retornado.
13.32 Escreva um programa projetado para gerar e tratar um erro de esgotamento de memória. Seu programa deveria fazer um
laço de pedidos para alocar memória dinamicamente, através do operador new.
13.33 Escreva um programa que mostra que todos os destruidores para objetos construídos em um bloco são chamados antes de
uma exceção ser disparada daquele bloco.
13.34 Escreva um programa que mostra que os destruidores de objetos membro são chamados somente para aqueles objetos
membro que foram construídos antes de ocorrer uma exceção.
13.35 Escreva um programa que demonstra como qualquer exceção é capturada com catch (. .
13.36 Escreva um programa que mostra que a ordem dos tratadores de exceções é importante. O primeiro tratador para o qual existe uma correpondência é o que é executado. Compile e execute seu programa de duas maneiras diferentes, para mostrar que dois tratadores diferentes são executados com dois efeitos diferentes.
13.37 Escreva um programa que mostra um construtor passando informações sobre a falha do construtor para um tratador de exceção depois de um bloco try.
13.38 Escreva um programa que usa uma hierarquia de herança múltipla de classes de exceção para criar uma situação em que a ordem dos tratadores de exceções é importante.
13.39 Usando setjmp/longjmp. um programa pode transferir o controle imediatamente para uma rotina de erro de dentro de uma chamada de função profundamente aninhada. Infelizmente, como a pilha é desempilhada, os destruidores não são chamados para os objetos automáticos que foram criados durante a seqüência das chamadas de função aninhadas. Escreva um programa que demonstra que esses destruidores, de fato, não são chamados.
13.40 Escreva um programa que ilustra o novo disparo de uma exceção.
13.41 Escreva um programa que usa set unexpected para inicializar uma função definida pelo usuário para unexpected. usa set_unexpected novamente e, então. redetine unexpected de volta para sua função anterior. Escreva um programa semelhante para testar set terminate e terminate.
13.42 Escreva um programa que mostra que uma função com seu próprio bloco try não tem que capturar todos os erros possíveis gerados dentro do try. Algumas exceções podem escapar e ser tratadas em escopos externos.
13.43 Escreva um programa que dispara um erro de uma chamada de função profundamente aninhada e ainda tem o tratador catch após o bloco externo à cadeia de chamadas, capturando a exceção.

14

Processamento de arquivos

Objetivos
• Ser capaz de criar, ler, gravar e atualizar arquivos.
• Familiarizar-se com o processamento de arquivos de acesso seqüencial.
• Familiarizar-se com o processamento de arquivos de acesso aleatório.
• Ser capaz de especificar operações de EIS não-formatadas de alto desempenho.
• Compreender a diferença entre o processamento de arquivos com dados formatados e dados brutos.
• Construir um programa de processamento de transações com processamento aleatório de arquivos.

Li parte dele até ofim.
Samuel Goldwyn

Só posso supor que um documento marcado como “Não arquivar” está arquivado em um documento marcado como “Não arquivar”.
Senador Frank Church
Depoimento ao Subcomitê de Inteligência do Senado, 1975

CAPÍTULO 14 - PROCESSAMENTO DE ARQUIVOS 727
Visão geral
14.1 Introdução
14.2 A hierarquia de dados
14.3 Arquivos e streams
14.4 Criando um arquivo de acesso seqüencial
14.5 Lendo dados de um arquivo de acesso seqüencial
14.6 Atualizando arquivos de acesso seqüencial
14.7 Arquivos de acesso aleatório
14.8 Criando um arquivo de acesso aleatório
14.9 Gravando dados aleatoriamente em um arquivo de acesso aleatório
14.10 Lendo dados seqüencialmente de um arquivo de acesso aleatório
14.11 Exemplo: um programa de processamento de transações
14.12 Entrada/saída de objetos
Resumo• Terminologia. Erros comuns de programação . Boa prática de programação Dica de desempenho
Dica de portabilidade . Exercícios de auto-revisão Respostas aos exercícios de auto-revisão • Exercícios

14.1 Introdução

O armazenamento de dados em variáveis e arrays é temporário. Arquivos são usados para conservação permanente de grandes quantidades de dados. Os

computadores armazenam arquivos em dispositivos secundários de armazenamento, tais como discos magnéticos, discos ópticos e fitas. Neste capítulo, explicamos como os arquivos de dados São criados, atualizados e processados por programas em C++. Examinamos aqui tanto os arquivos de acesso seqüencial como os arquivos de acesso aleatório. Comparamos o processamento de arquivos com dados formatados e dados não-formatados. Examinamos técnicas para a entrada de dados de, e a saída de dados para, strings, em vez de arquivos, no Capítulo 19.

14.2 A hierarquia de dados

Basicamente, todos os itens de dados processados por um computador são reduzidos a combinações de zeros e uns. Isso ocorre porque é simples e econômico se construir dispositivos eletrônicos que podem assumir dois estados estáveis - um dos estados representa O e o outro representa 1. E notável que as impressionantes funções executadas pelos computadores envolvam apenas as manipulações mais elementares de Os e is.

O menor item de dados em um computador pode assumir o valor O ou o valor 1. Tal item de dados é chamado de bit (abreviação de “binary digit”, ou “dígito binário” - um dígito que pode assumir um de dois valores). Os circuitos computacionais realizam várias manipulações simples de bits, tais como determinar o valor de um bit, redefinir o valor de um bit e inverter o valor de um bit (de 1 para O ou de O para 1).

E complicado para os programadores trabalhar com dados na forma de baixo nível dos bits. Em vez disso, os programadores preferem trabalhar com dados na forma de dígitos decimais (i.e., O, 1, 2, 3, 4, 5, 6, 7, 8 e 9), letras (i.e., de A até Z e de a até z) e símbolos especiais (i.e., $, @, %, &, , (,), -, +,“, :, ?,Ie muitos outros). Dígitos, letras e símbolos especiais são chamados de caracteres. O conjunto de todos os caracteres usados para escrever programas e representar itens de dados em um determinado computador é chamado de conjunto de caracteres daquele computador. Como os computadores podem processar apenas is e Os, qualquer caractere do conjunto de caracteres de um computador é representado por uma combinação de is e Os (chamada de byte). Atualmente, os bytes são compostos normalmente de oito bits. Os programadores criam programas e itens de dados como caracteres; os computadores manipulam e processam esses caracteres como combinações de bits.

Da mesma forma que os caracteres são compostos de bits, os campos são compostos de caracteres. Um campo é um grupo de caracteres que possui um significado. Por exemplo, um campo consistindo unicamente em letras maiúsculas e minúsculas pode ser usado para representar o nome de uma pessoa.

Os itens de dados processados pelos computadores formam uma hierarquia de dados na qual os itens de dados se tornam maiores e mais complexos na estrutura à medida que evoluímos de bits para caracteres, campos e assim por

diante.
Um registro (i.e., uma struct ou class em C++) é composto de vários campos (chamados de membros
em C++). Em um sistema de folha de pagamento, por exemplo, um registro de um determinado empregado pode consistir nos seguintes campos:
1. Número de identificação de empregado
2. Nome
3. Endereço
4. Valor do salário-hora
5. Número de dispensas solicitadas
6. Total de vencimentos no ano em curso
7. Total de impostos federais retidos na fonte, etc.
Assim, um registro é um grupo de campos relacionados. No exemplo anterior, cada um dos campos pertence ao mesmo empregado. Naturalmente, uma empresa específica pode ter muitos empregados e terá um registro da folha de pagamento para cada um. Um arquivo é um grupo de registros relacionados. Um arquivo de folha de pagamento de uma empresa contém um registro para cada empregado. Assim, o arquivo de folha de pagamento de uma pequena empresa pode conter apenas 22 registros, ao passo que o arquivo da folha de pagamento de uma grande empresa pode conter 100.000 registros. Não é raro uma organização ter muitos arquivos, cada um deles possuindo milhões de caracteres de informações. A Fig. 14.1 ilustra a hierarquia de dados.
Para facilitar a recuperação de registros específicos de um arquivo, pelo menos um campo em cada registro é escolhido como uma chave de registro. A chave de registro identifica um registro como pertencendo a uma determinada pessoa ou entidade que é distinto de todos os outros registros no arquivo. No registro de folha de pagamento descrito anteriormente, o número de identificação do empregado seria normalmente escolhido como a chave dos registros.

Sally Ireto 1

ITom 1Azu1 1
Judy Iverae 1 1 Arquivo

Verde JJ Registro
t
Judy Campo
t
01001010 Byte (Caractere ASCII J)
1
Fig. 14.1 A hierarquia de dados.

Iris	ILarafia 1	1	1
Randy	It1h0I	1	1

Há muitas maneiras de se organizar registros em um arquivo. No tipo mais comum de organização, denominado arquivo seqüencial, registros são normalmente armazenados em ordem segundo o campo-chave dos registros. Em um arquivo de folha de pagamento, os registros geralmente são colocados em ordem segundo o número de identificação do empregado. O registro do primeiro empregado no arquivo contém o menor número de identificação do empregado, e os registros subseqüentes possuem números de identificação do empregado em ordem crescente.

A maioria das empresas utiliza muitos arquivos diferentes para armazenar dados. Por exemplo, as empresas podem ter arquivos de folha de pagamento, arquivos de contas a receber (listando o dinheiro devido pelos clientes), arquivos de contas a pagar (listando o dinheiro devido aos fornecedores), arquivos de estoques (listando características a respeito de todos os itens manipulados pela empresa) e muitos outros tipos de arquivos. Algumas vezes, um grupo de arquivos relacionados entre si é chamado de banco de dados. Um conjunto de programas que se destina a criar e gerenciar bancos de dados é chamado de sistema de gerenciamento de banco de dados (SGBD, ou database management system, DBMS).

14.3 Arquivos e streams

C++ vê cada arquivo simplesmente como uma seqüência de bytes (Fig. 14.2). Cada arquivo termina ou com um marcador defim de arquivo ou em um byte específico, cujo número é gravado em uma estrutura administrativa de dados mantida pelo sistema. Quando um arquivo é aberto, um objeto é criado e um stream é associado àquele objeto. No Capítulo II, vimos que quatro objetos são criados para nós - cm. cout. cerr e clog - quando <iostreani> é incluído. Os streams associados com estes objetos fornecem canais de comunicação entre um programa e um arquivo ou dispositivo particular. Por exemplo, o objeto cm (o objeto stream padrão de entrada) permite que um programa leia dados do teclado ou de outros dispositivos, o objeto cout (o objeto stream padrão de saída) permite que um programa exiba dados na tela ou em outros dispositivos e os objetos cerr e clog (os objetos stream padrão de erros) permitem a um programa enviar mensagens de erro para a tela ou outros dispositivos.

i O 1 2 3 4 5 6 8 9 marcador de fim de arquivo

Fig. 14.2 Como a linguagem C++ visualiza um arquivo de n bytes.

Para executar o processamento de arquivos em C++, devem ser incluídos os arquivos de cabeçalho <iostream> e <fstream>. O cabeçalho <fstream> inclui as definições para as classes stream ifstream (para fazer entrada de um arquivo), ofstream (para fazer saída para um arquivo) e fstream (para entrada e saída de um arquivo). Os arquivos são abertos criando-se objetos destas classes stream. Estas classes stream são derivadas das (i.é, herdam a funcionalidade de)

classes istreain, ostream e iostream. respectivamente. Assim, as funções membro, operadores e manipuladores descritos no Capítulo 11, “Entradalsaída com streams em C++”, podem todos também ser aplicados a streams de arquivos.Os relacionamentos de herança das classes de EIS discutidas até aqui são resumidas na Fig. 14.3.

14.4 Criando um arquivo de acesso seqüencial

C++ não impõe nenhuma estrutura a um arquivo. Assim, conceitos como o de “registro” não existem em arquivos em C++. Portanto, o programador deve estruturar os arquivos de modo a satisfazer as exigências das aplicações. No exemplo a seguir, vemos como o programador pode impor uma estrutura de registros simples a um arquivo. Primeiro, apresentamos o programa e, então, o analisamos em detalhe.

A Fig. 14.4 cria um arquivo simples de acesso seqüencial que pode ser usado em um sistema de contas a receber para ajudar a controlar as quantias devidas pelos clientes devedores de uma empresa. Para cada cliente, o programa obtém um número de conta, o nome do cliente e o saldo do cliente (i.e., a quantia que o cliente deve à empresa por bens e serviços recebidos no passado). Os dados obtidos para cada cliente constituem um registro para aquele cliente. O número da conta é usado como campo-chave dos registros nessa aplicação isto é, o arquivo será

jos
istream ostream
/N
ifstream iostream ofstream
fs tream

Fig. 14.3 Parte da hierarquia de classes de EIS com streams.

criado e mantido segundo a ordem dos números de contas. Esse programa assume que o usuário fornece os registros na ordem dos números das contas. Em um grande sistema de contas a receber, seria fornecido um recurso de ordenação para que o usuário pudesse entrar com os registros em qualquer ordem. Os registros seriam então ordenados e gravados no arquivo.

```
1 II Fig. 14.4: figl4_04.cpp
2 // Criando um arquivo seqüencial
3 #include <iostream>
4
5 using std::cout;
6 using std: :cin;
7 using std: :ios;
8 using std: :cerr;
9 using std::endl;
lo
11 #include <fstream>
12
13 using std::ofstreani;
```

```
14
15 #include <cstdlib>
16
17 int main()
18
19 Il construtor de ofstream abre o arquivo
20 ofstream outClientFile( “clients.dat”, ios::out );
21
22 if ( ‘outClientFile ) { Il operador ! sobrecarregado
23 cerr « “Arquivo não pode ser aberto « endi;
24 exit( 1 ); Il protótipo em cstdlib
25
26
27 cout « “Digite a conta, o nome e o saldo.\n”
28 « “Digite fim de arquivo para terminar entrada.\n? “;
29
30 int account;
31 char name[ 30 ];
32 double balance;
33
34 while ( cm » account » name » balance )
```

Fig. 14.4 Criando um arquivo seqüencial (parte 1 de 2).


```
35 outClientFile « account « « naxne
36 « « balance « ‘\n’;
37 cout « “?
38
39
40 return 0; // destruidor de ofstream fecha o arquivo
41
Digite a conta, o nome e o saldo.
Digite fim de arquivo para terminar entrada.
? 100 Jones 24.98
? 200 Doe 345.67
? 300 White 0.00
? 400 Stone -42.16
? 500 Rich 224.62
? AZ
```

Fig. 14.4 Criando um arquivo sequerluial parte ue).

Agora, examinemos esse programa. Como afirmado anteriormente, arquivos são abertos criando-se objetos das classes stream ifstream. ofstream ou fstream. Na Fig. 14.4,0 arquivo deve ser aberto para saída, assim um objeto ofstream é criado. Dois argumentos são passados para o construtor do objeto - o nome do arquivo e o modo de abertura do arquivo. Para um objeto ofstream, o modo de

abertura do arquivo pode ser tanto ios: : out para fazer saída para um arquivo como ios: app para acrescentar dados ao final de um arquivo (sem modificar qualquer dado já existente no arquivo). Arquivos existentes abertos com modo ios: out são truncados - todos os dados no arquivo são eliminados. Se o arquivo especificado ainda não existe, então é criado um arquivo com aquele nome de arquivo.
A declaração
ofstream outClientFile(“clients.dat”, ios::out);
na linha 20 cria um objeto ofstreani de nome outClientFile associado ao arquivo clients . dat que é aberto para saída. Os argumentos “clients . dat” e ios: : out são passados para o construtor de ofstreaxn. que abre o arquivo. Isto estabelece uma “linha de comunicação” com o arquivo. Por default, objetos ofstream são abertos para saída, de modo que o comando
ofstream outClientFile(“clients.dat”);
poderia ter sido usado para abrir clients . dat para saída. A Fig. 14.5 lista os modos de abertura de arquivos.

Fig. 14.5 Modos de abertura de arquivos.
Erro com um de programação 14.1
Abrir um arquivo existente para saída (ios: : out) quando, na verdade, o usuário quer preservar o arquivo; o conteúdo do arquivo é descartado sem aviso.

Modo		Descrição
ios:	: app	Grava toda a saída no fim do arquivo.
ios:	ate	Abre um arquivo para saída e avança até o fim do arquivo (normalmente usado para acrescentar dados a um arquivo). Os dados podem gravados em qualquer lugar no arquivo.
ios:	: in	Abre um arquivo para entrada.
ios:	out	Abre um arquivo para saída.
ios:	: trunc	Elimina o conteúdo do arquivo se ele existe (esta também é a ação default para ios: : out)
ios:	: binary	Abre um arquivo para entrada ou saída binária (i.e., não-texto).

Erro comum de programação 14.2
Usar um objeto ofstream incorreto para se referir a um arquivo.
Um objeto ofstream pode ser criado sem abrir um arquivo específico - um arquivo pode ser associado ao objeto posteriormente. Por exemplo, a declaração
ofstream outclientFile;
cria o objeto ofstreain de nome outClientFile. A função membro open de ofstream abre um arquivo e o associa a um objeto ofstream existente como segue:
outClientFile.open(“clients.dat”, ios: :out);

Erro comum de programação 14.3
Não abrir um arquivo antes de tentar referenciá-lo em um programa.
Após criar um objeto ofstream e tentar abri-lo, o programa testa se a operação de abertura foi bem-sucedida. A estrutura if nas linhas 22 a 25
if (!outClientFile) { II operador sobrecarregado
cerr « ‘Arquivo não pode ser aberto” « endi;
exit(1); // protótipo em cstdlib
usa a função membro operador operator! sobrecarregada de ios para determinar se a operação de abertura foi bem-sucedida. A condição retorna um valor não-zero (verdadeiro) se ou o failbit ou o badbit é ligado para o stream na operação open. Alguns erros que podem acontecer são: tentar abrir um arquivo não-existente para leitura, tentar abrir um arquivo para leitura sem permissão e abrir um arquivo para gravação quando não há espaço disponível em disco.
Quando a condição indica que a tentativa de abertura foi mal-sucedida, a mensagem de erro “Arquivo não pode ser aberto’ é exibida e a função exit termina o programa. O argumento para exit é retornado para o ambiente do qual o programa foi invocado. O argumento O indica que o programa terminou normalmente; qualquer outro valor indica que o programa terminou devido a um erro. O ambiente chamador (muito provavelmente o sistema operacional) usa o valor retornado por exit para responder adequadamente ao erro.
Uma outra função membro operador sobrecarregada de ios - operator void* - converte o stream em um ponteiro, de modo que ele possa ser testado como O (o ponteiro nulo) ou não-zero (qualquer outro valor de ponteiro). Se o failbit ou o badbit (ver Capítulo II) foram ligados para o stream, O (falso) é retornado. A condição no cabeçalho de while seguinte (linha 34) invoca implicitamente a função membro operator void*:
while (cm » account » name » balance
A condição permanece true enquanto nem o failbit nem obadbit forem ligados para cm. Digitar o indicador de fim de arquivo liga o failbit para cm. A função operator void * pode ser usada para testar um objeto de entrada quanto ao fim de arquivo em vez de chamar explicitamente a função membro eof para o objeto de entrada.
Se o arquivo é aberto com sucesso, o programa começa a processar dados. O comando seguinte (linhas 27 e 28) solicita ao usuário para digitar os diversos campos para cada registro, ou para digitar fim de arquivo quando a entrada de dados está completa:
cout « “Digite a conta, o nome e o saldo.\n”
« “Digite fim de arquivo para terminar entrada.\n? “;
A Fig. 14.6 lista as combinações de teclas para digitar fim de arquivo em diversos sistemas de computadores.

Fig. 14.6 Combinações de teclas que indicam fim de arquivo para vários sistemas computacionais populares.
A linha 34

while (cm » account » name » balance
extrai cada conjunto de dados e determina se fim de arquivo foi digitado. Quando o fim de arquivo ou dados inválidos são digitados, a operação » de extração do stream de cm retorna O (normalmente operator void*
retorna verdadeiro) e a estrutura whi le termina. O usuário digita fim de arquivo para informar ao programa que não há mais dados para serem processados. O indicador de fim de arquivo é inicializado quando a combinação de teclas que indica fim de arquivo é digitada pelo usuário. A estrutura while continua a executar o laço enquanto o indicador de fim de arquivo não for digitado.
As linhas 35 e 36
outClientFile « account « « name
« ‘ « balance « \n’;
gravam um conjunto de dados no arquivo “clients .dat” usando o operador « de inserção em stream e o objeto outClientFile associado com o arquivo no começo do programa. Os dados podem ser recuperados por um programa projetado para ler o arquivo (ver Seção 14.5). Note que o arquivo criado na Fig. 14.4 é um arquivo- texto. Ele pode ser lido por qualquer editor de texto.
Uma vez que o indicador de fim de arquivo é digitado, main termina. Isto faz com que o objeto outClientFile seja destruído, invocando desta forma sua função destruidor, a qual fecha o arquivo clients . dat. Um objeto ofstream pode ser fechado explicitamente pelo programador usando a função membro dose como segue:
outClientFile . dose O;
Dica de desempenho 14.1
_____ Feche explicitamente cada arquivo tão logo se saiba que o programa não fará referência ao arquivo novamente. isso pode reduzir o consumo de recursos no programa que vai continuar sendo executado
após não mais necessitar de um arquivo particular Essa prática também melhora a clareza do programa.
No exemplo de execução do programa da Fig. 14.4, o usuário digita informações para cinco contas e, então, sinaliza que a entrada de dados está completa digitando-se fim de arquivo (AZ aparece nas telas de compatíveis com o IBM PC). Esta janela de diálogo não mostra como os registros de dados efetivamente aparecem no arquivo. Para verificar se o arquivo foi criado com sucesso, na próxima seção criamos um programa para ler o arquivo e imprimir seu conteúdo.

14.5 Lendo dados de um arquivo de acesso seqüencial

Os dados são armazenados em arquivos de modo que possam ser recuperados para processamento quando necessário. A seção anterior demonstrou como criar um arquivo para acesso seqüencial. Nesta seção, analisaremos como ler dados seqüencialmente de um arquivo.

Sistema computacional	Combinação de teclas
Sistemas UNIX	<ctrl> d (em uma linha isolada)
IBM PC e	<ctrl> z (às vezes seguido pelo

compatíveis	pressionamento da tecla Enter)
Macintosh	<ctrl> d
VAX (VMS)	<ctrl> z

A Figura 14.7 lê registros do arquivo “clients dat” criados pelo programa da Fig. 14.4 e imprime o
conteúdo dos registros. Arquivos são abertos para entrada criando-se um objeto da classe ifstream. Dois argumentos são passados para o objeto - o nome do arquivo e o modo de abertura do arquivo. A declaração
ifstream inClientFile(“clients.dat”, ios::in);
na linha 29 cria um objeto ifstream denominado inClientFile e o associa com o arquivo clients dat que será aberto para entrada. Os argumentos entre parênteses são passados à função construtor de ifstream, a qual abre o arquivo e estabelece uma “linha de comunicação” com o arquivo.

```
1 // Fig. 14.7: figl4O7.cpp
2 // Lendo e imprimindo um arquivo seqüencial
3 #include <iostreain>
4
5 using std: :cout;
6 using std::cin;
7 using std::ios;
8 using std: :cerr;
9 using std: :endl;
10
11 #include <fstream>
12
13 using std::ifstream;
14
15 #include <iomanip>
16
17 using std::setiosflags;
18 using std::resetiosflags;
19 using std::setw;
20 using std::setprecision;
21
22 #include <cstdlib>
23
24 void outputLine( int, const char * const, double );
25
26 int main()
27 {
28 II construtor de ifstream abre o arquivo
29 ifstream inClientFile( “clients.dat”, ios::in );
30
31 if ( inC1ientFi1e ) {
```

```
32 cerr « “Arquivo não pode ser aberto\n;
33 exit( 1 );
34 1
35
36 int account;
37 char name[ 30 ];
38 double balance;
39
40 cout « setiosflags( ios::left ) « setw( 10 ) « “Conta’
41 « setw( 13 ) « “Nome” « “Saldo\n”
42 « setiosflags( ios::fixed 1 ios::showpoint );
43
44 while ( inClientFile » account » name » balance
45 outputLine( account, name, balance );
46
```

Fig. 14.7 Lendo e imprimindo um arquivo seqüencial (parte 1 de 2).


```
47
48 }
49
50 void outputLine( int acct, const char * const name, double bal
51
52 cout « setiosflags( ios::left ) « setw( 10 ) « acct
53 « setw( 13 ) « name « setw( 7 ) « setprecision( 2
54 « resetiosflags( ios: :left
55 « bal « ‘\n’;
56 }
```

Fig. 14.7 Lendo e imprimindo um arquivo seqüencial (parte 2 de 2).

Objetos da classe ifstream são abertos para entrada por default; assim, o comando

ifstream inclientFile(‘clients.dat”);

poderia ter sido usado para abrir clients . dat para entrada. Exatamente como com um objeto ofstreain, um objeto ifstream pode ser criado sem abrir um arquivo específico e um arquivo pode ser associado a ele mais tarde.

Boa prática de programação 14.1

Abrir um arquivo somente para entrada (usando ios: : in), se o conteúdo do arquivo não deve ser modificado. isso evita a modificação não-intencional do conteúdo do arquivo. Esse é um exemplo do princípio do privilégio mínimo.

O programa usa a condição ! inClientEile para determinar se o arquivo foi aberto com sucesso, antes de tentar recuperar dados do arquivo. A linha 44

while (inClientFile » account » name » balance

lê um conjunto de dados (i.e., um registro) do arquivo. Após a linha precedente

ser executada pela primeira vez, account tem o valor 100, name tem o valor “Jones” e balance tem o valor 24 . 98. Cada vez que a linha é executada, um outro registro é lido do arquivo para as variáveis account. name e balance. Os registros são exibidos usando-se a função outputLine. a qual usa manipuladores de stream parametrizados para formatar os dados para exibição. Quando o fim do arquivo é encontrado, a chamada implícita para operator void* na estrutura while retorna O (normalmente operator void* retorna verdadeiro), o arquivo é fechado pela função destruidor de ifstream e o programa termina.
Para recuperar seqüencialmente dados de um arquivo, os programas normalmente começam a ler a partir do início do arquivo e lêem todos os dados, um após o outro, até que os dados desejados sejam encontrados. Pode ser necessário processar o arquivo seqüencialmente várias vezes (desde o início do arquivo) durante a execução de um programa. Tanto a classe is treaxn como a classe os tream fornecem funções membro para reposicionar o ponteiro de posição do arquivo (o número de byte do próximo byte do arquivo a ser lido ou gravado). Estas funções membro são seekg (“seek get”) para a classe istream e seekp (“seek put”) para a classe ostream. Cada objeto istream tem um “ponteiro get” que indica o número do byte do arquivo do qual a próxima leitura deve ocorrer e cada objeto ostrearn tem um “ponteiro put” que indica o número do byte do arquivo no qual a próxima saída deverá ser colocada. O comando

return 0; // destruidor de ifstream fecha o arquivo

inclientFile.seekg(O);

Conta	Nome	Saldo	
100	Jones	24.98	
200	Doe	345.67	.
300	White	0.00	
400	Stone	-42.16	
500	Rich	224.62	

reposiciona o ponteiro de posição do arquivo no começo do arquivo (posição O) associado com inClientFile. O argumento para seekg é normalmente um inteiro long. Um segundo argumento pode ser especificado para indicar a direção de seek. A direção de seek pode ser ios: : beg (o default), para posicionamento

relativo ao início de um stream, ios: : cur para posicionamento relativo à posição corrente em um stream ou ios: : end para posicionamento relativo ao fim de um stream. O ponteiro de posição do arquivo é um valor inteiro que especifica a posição no arquivo como um número de bytes desde a posição de início do arquivo (isso é, algumas vezes, chamado de offset - ou deslocamento - desde o começo do arquivo). Alguns exemplos de posicionamento do ponteiro get de posição do arquivo são:

```
II posiciona no n-ésimo byte do fileObject
// assume ios: :beg
fileObject.seekg( n );
II posiciona n bytes para a frente no fileObject
fileObject.seekg( n, ios::cur );
II posiciona y bytes para trás a partir do fim do fileObject
fileObject.seekg( y, ios::end );
// posiciona no fim do fileObect
fileObject.seekg( O, ios::end );
```

As mesmas operações podem ser executadas com a função membro seekp de ostream. As funções membro tellg e tellp são fornecidas para retornar as posições correntes dos ponteiros gel e put, respectivamente. O comando seguinte atribui o valor do ponteiro de posição get do arquivo para a variável location do tipo long:

```
location = fileObect.tellg
```

A Fig. 14.8 possibilita a um gerente de crédito exibir as informações da conta para aqueles clientes com saldo zero _____ (i.e., clientes que não devem dinheiro à empresa), saldos credores (i.e., clientes para os quais a empresa deve dinheiro) e saldos devedores (i.e., clientes que devem dinheiro à empresa por bens e serviços recebidos no passado). O programa exibe um menu e permite ao gerente de crédito digitar uma de três opções para obter informações de crédito. A opção 1 produz uma lista de contas com saldo zero. A opção 2 produz uma lista de contas com saldos credores. A opção 3 produz uma lista de contas com saldos devedores. A opção 4 termina a execução do programa. Digitar uma opção inválida simplesmente exibe a solicitação para digitar uma outra opção. A saída do programa é mostrada na Fig. 14.9.

```
1 1/ Fig. 14.8: figl4O8.cpp
2 II Programa de consulta de crédito
3 #include <iostream>
4
5 using std::cout;
6 using std::cin;
7 using std::ios;
8 using std: :cerr;
9 using std::endl;
10
11 #include <fstream>
12
13 using std: :ifstream;
```

```
1 14
15 #include <iomanip>
16
17 using std::setiosflags;
18 using std::resetiosflags;
```

Fig. 14.8 Programa de consulta de crédito (parte 1 de 3).


```
19 using std::setw;
20 using std: :setprecision;
21
22 #include <cstdlib>
23
24 enum RequestType { ZERO BALANCE = 1, CREDIT BALANCE,
25 DEBIT BALANCE, END };
26 int getRequestO;
27 bool shouldDisplay( int, double );
28 void outputLine( int, const char * const, double );
29
30 int main()
31 {
32 Il construtor de ifstream abre o arquivo
33 ifstream inClientFile( “clients.dat”, ios: :in );
34
35 if ( !inClientFile ) {
36 cerr « ‘Arquivo não pode ser aberto” « endl;
37 exit( 1 );
38 }
39
40 int request, account;
41 char name[ 30 );
42 double balance;
43
44 cout « “Digite opção\n”
45 « “ 1 - Listar contas com saldo zero\n”
46 « “ 2 - Listar contas com saldo credor\n”
47 « “ 3 - Listar contas com saldo devedor\n”
48 « “ 4 - Fim da execução”
49 « setiosflags( ios: :fixed 1 ios: ;showpoint );
50 request = getRequestQ;
51
52 while ( request != END ) {
53
54 switch ( request
55 case ZERO BALANCE:
```

```
56 cout « “\nContas com saldo zero:\n”;
57 break;
58 case CREDIT BALANCE:
59 cout « “\nContas com saldo credor:\n”;
60 break;
61 case DEBIT BALANCE:
62 cout « “\nContas com saldo devedor:\n”;
63 break;
64
65
66 inClientFile » account » naxne » balance;
67
68 while ( !inClientFile.eof O)
69 if ( shouldDisplay( request, balance
70 outputLine( account, name, balance
71
72 inClientFile » account » name » balance;
73 }
74
75 inClientFile.clearO; II desliga fim de arquivo p/próxima entrada
76 inClientFile.seekg( O ); II vai para o início do arquivo
77 request getRequestO);
```

Fig. 14.8 Programa de consulta de crédito (parte 2 de 3).


```
78 }
79
80 cout « Fim da execução.” « endi;
81
82 return 0; // destruidor de ifstream fecha o arquivo
83 )
84
85 int getRequest()
86 {
87 int request;
88
89 do{
90 cout « “\n?
91 cm » request;
92 } while( request < ZERO_BALANCE && request > END );
93
94 return request;
95 }
96
97 bool shouldDisplay( int type, double balance
98 {
```

```
99 if ( type == CREDIT BALANCE && balance < O
100 return true;
101
102 if ( type == DEBIT BALANCE && balance > O
103 return true;
104
105 if ( type ZERO BALANCE && balance == O
106 return true;
107
108 return false;
109 }
110
111 void outputLine( int acct, const char * const name, double bal
112
113 cout « setiosflags( ios::left ) « setw( 10 ) « acct
114 « setw( 13 ) « name « setw( 7 ) « setprecision( 2
115 « resetiosflags( ios::left
116 « bal « '\n;
117 )
```

Fig. 14.8 Programa de consulta de crédito (parte 3 de 3).

```
Digite opção
1 - Listar contas com saldo zero
2 - Listar contas com saldo credor
3 - Listar contas com saldo devedor
4 - Fim da execução
Contas com saldo zero:
300 White 0.00
Contas com saldo credor:
400 Stone -42.16
```

Fig. 14.9 Exemplo de saída do programa da Fig. 14.8 (parte 1 de 2).

Fig. 14.9 Exemplo de saída do programa da Fig. 14.8 (parte 2 de 2).

14.6 Atualizando arquivos de acesso seqüencial

Dados que são formatados e gravados em um arquivo de acesso seqüencial como mostrado na Seção 14.4 não podem ser modificados sem o risco de destruir outros dados no arquivo. Por exemplo, se o nome "White" precisa ser mudado para Worthington, o nome velho não pode ser simplesmente gravado por cima, O registro para White foi gravado no arquivo como

```
300 White 0.00
```

Se esse registro fosse regravado começando na mesma posição no arquivo e usando o nome mais longo, o registro ficaria

300 Worthington 0.00

O novo registro contém seis caracteres a mais que o registro original. Portanto, os caracteres depois do segundo “o”
em “Worthington” seriam sobrepostos ao começo do próximo registro seqüencial no arquivo. O problema aqui
é que no modelo de entradaJsaída formatada usando o operador de inserção « e o operador de extração», campos
- e portanto registros - podem variar em tamanho. Por exemplo, 7, 14, -117, 2074 e 27383 são todos ints e cada um
é armazenado no mesmo número de bytes de “dados brutos” internamente, mas quando estes inteiros são enviados
como texto formatado (seqüências de caracteres) para a tela ou para um arquivo em disco, eles se tornam campos de
tamanhos diferentes. Portanto, o modelo de entradalsaída formatada não é normalmente usado para atualizar registros em sua posição original.
Tal atualização pode ser feita, mas é complicada. Por exemplo, para efetuar a mudança de nome precedente, os registros antes de 300 White 0. 00 em um arquivo de acesso seqüencial poderiam ser copiados para um novo arquivo, o registro atualizado seria então gravado no novo arquivo e os registros após 300 White O . 00 seriam copiados para o novo arquivo. Isto requer processar todos os registros no arquivo para atualizar um registro. Se muitos registros estão sendo atualizados em uma única passagem pelo arquivo, então essa técnica pode ser aceitável.

14.7 Arquivos de acesso aleatório

Até aqui, vimos como criar e acessar arquivos seqüenciais e pesquisá-los para localizar informações particulares. Arquivos de acesso seqüencial são inadequados para as chamadas aplicações de acesso instantâneo, nas quais um registro de informações particular deve ser localizado imediatamente. Aplicações comuns de acesso instantâneo são sistemas de reservas de linhas aéreas, sistemas bancários, sistemas de pontos de venda, caixas automáticos e outros tipos de sistemas de processamento de transações que exigem acesso a dados específicos. O banco no qual você tem sua conta pode ter centenas de milhares ou mesmo milhões de outros clientes e, ainda assim, quando você usa um caixa automático, é verificado se sua conta tem fundos suficientes em segundos. Este tipo de acesso instantâneo é possível com arquivos de acesso aleatório. Os registros individuais de um arquivo de acesso aleatório podem ser acessados diretamente (e rapidamente) sem pesquisar os outros registros. Como já dissemos, C++ não impõe estruturas a um arquivo. Assim, uma aplicação que queira usar arquivos de
acesso aleatório deve criá-los. Diversass técnicas podem ser usadas para criar

arquivos de acesso aleatório. Talvez a

Contas
100
200
500

com saldo devedor:
Jones
Doe
Rich

24.98
345.67
224.62

Fim da execução.

mais simples seja exigir que todos os registros em um arquivo sejam do mesmo comprimento fixo. Usar registros de comprimento fixo torna fácil para um programa calcular (como uma função do tamanho do registro e da chave do registro) a localização exata de qualquer registro em relação ao começo de um arquivo. Logo veremos como isso facilita o acesso imediato a registros específicos, mesmo em grandes arquivos.
A Fig. 14.10 ilustra a visão de C++ de um arquivo de acesso aleatório composto por registros de comprimento
fixo (cada registro tem 100 bytes de comprimento). Um arquivo de acesso aleatório é como um comboio ferroviário com muitos vagões - alguns vazios e alguns com conteúdo.

Fig. 14.10 A visão de C++ de um arquivo de acesso aleatório.
Dados podem ser inseridos em um arquivo de acesso aleatório sem destruir outros dados no arquivo. Dados armazenados previamente também podem ser atualizados ou excluídos sem regravar todo o arquivo. Nas seções seguintes, explicamos como criar um arquivo de acesso aleatório, digitar dados, ler os dados tanto seqüencial como aleatoriamente, atualizar os dados e eliminar dados não mais necessários.
14.8 Criando um arquivo de acesso aleatório
A função membro write de ostream envia um número fixo de bytes começando em uma posição específica da memória para o stream de saída especificado. Quando o stream é associado a um arquivo, os dados são gravados começando na posição do arquivo especificada pelo ponteiro put de posição do arquivo. A função membro read de istream lê um número fixo de bytes do stream de entrada especificado para uma área na memória começando em um endereço

especificado. Se o stream está associado a um arquivo, os bytes são lidos começando na posição do arquivo especificada pelo ponteiro get de posição do arquivo.
Agora, quando gravarmos um inteiro nurnber em um arquivo, em vez de usar
outFile « number;
o qual poderia imprimir tanto somente 1 dígito como até 11 dígitos (10 dígitos mais um suial, cada uni dos quais exige 1 byte de memória) para um inteiro de 4 bytes, podemos usar
outFile.write(reiriterpret_cast<const char *>(&number),
sizeof(nuniber));
a qual sempre grava 4 bytes (em uma máquina com inteiros de 4 bytes). A função write espera um dado do tipo
const char * como seu primeiro argumento, por isso usamos o operador reinterpret cast<const char*
> para converter o endereço de number para um ponteiro const char* . O segundo argumento de write é um
inteiro do tipo size_t que especifica o número de bytes a serem gravados. Como veremos, a função read de
istream pode então ser usada para ler os 4 bytes de volta para a variável inteira number.
Se um programa vai ler dados não-formatados (gravados por write), ele precisa ser compilado e executado
em um sistema que seja compatível com o programa que gravou os dados.
Programas de processamento de arquivos de acesso aleatório raramente gravam um único campo em um
arquivo. Normalmente, eles gravam um objeto struct ou class por vez, como mostraremos nos exemplos seguintes. Considere a seguinte definição de problema:

F

O 1		10 0 1		20 0 Ir		30 0 ,		40 0 ,		50 0		offsets dos bytes
1				II		II		II		II		1
	100		100		100		100		100		100	
	byt es		byt es		byt es		byt es		byt es		byt es	

Crie um programa de processamento de crédito capaz de armazenar até 100 registros de comprimento fixo para uma empresa que pode ter até 100 clientes. Cada registro deve consistirem um número de conta que será usado como a chave do registro, um sobrenome, um nome e um saldo. O programa deve ser capaz de atualizar uma conta, inserir uma nova conta, eliminar uma conta e listar

todos os registros de conta em um arquivo-texto formatado para impressão. As seções seguintes apresentam as técnicas necessárias para criarmos esse programa de processamento de crédito. A Fig. 14.11 ilustra a abertura de um arquivo de acesso aleatório, a definição do formato do registro usando uma struct (definida no arquivo de cabeçalho clntdata . h) e gravação de dados no disco em formato binário (o modo binário é especificado na linha 33). Este programa inicializa todos os 100 registros do arquivo “credit. dat” com structs vazias, usando a função write. Cada struct vazia contém O no campo de número da conta, o _____ string nulo (representado por aspas vazias) no campo de sobrenome, o string nulo no campo de nome e O . O no campo de saldo. Cada registro é inicializado com a quantidade de espaço vazio na qual os dados da conta serão armazenados e para determinar nos programas subseqüentes se um registro está vazio ou contém dados.

```
1 // clntdata.h
2 1/ Definição da struct clientData usada
3 1/ nas Figs. 14.11, 14.12, 14.14 e 14.15.
4 #ifndef CLNTDATAH
5 #define CLNTDATAH
6
7 struct clientData {
8 int accountNuniber;
9 char lastName[ 15 ];
10 char firstName[ 10 ];
11 double balance;
12 );
13
14 #endif
```

Fig. 14.11 Arquivo de cabeçalho clntdata.h.

```
15 II Fig. 14.11: figl4ll.cpp
16 /1 Criando um arquivo de acesso aleatório seqüencialmente
17 #include <iostream>
18
19 using std::cerr;
20 using std::endl;
21 using std::ios;
22
23 #include <fstream>
24
25 using std::ofstream;
26
27 #include <cstdlib>
28
29 #include “clntdata.h”
30
31 int main()
32
```

```
33 ofstream outCredit( “credit.dat”, ios::binary );
34
35 if ( outCredit ) {
36 cerr « “Arquivo não pode ser aberto.” « endi;
37 exit( 1 );
```

Fig. 14.12 Criando um arquivo de acesso aleatório seqüencialmente (parte 1 de 2).


```
38 }
39
40 clientData blankClient = { 0, “‘, ““, 0.0 };
41
42 for ( int i = 0; i < 100; i++
43 outCredit.write(
44 reinterpretcast<const char *>( &blankClient ),
45 sizeof( clientData ) );
46 return 0;
```

Fig. 14.12 Criando um arquivo de acesso aleatório seqüencialmente (parte 2 de 2).

Na Fig. 14.11,0 comando nas linhas 43 a45

```
outCredit . write
reinterpret_cast<const char *>( &blankClient ),
sizeof( clientData ) );
```

faz com que a estrutura blankClient de tamanho sizeof (clientData) seja gravada no arquivo credit. dat associado ao objeto outCredit de ofstream. Lembre-se de que o operador sizeof retoma o tamanho em bytes do objeto contido entre parênteses (ver Capítulo 5). Note que o primeiro argumento para a função write na linha 43 deve ser do tipo const char . Contudo, o tipo de dados de &blankClient é clientData
*. Para converter &blankClient para o tipo de ponteiro apropriado, a expressão

```
reinterpretcast<const char >( &blankClient ),
```

usa o operador de coerção reinterpret_cast para converter o endereço de blankClient em um const char *, de modo que a chamada para write é compilada sem provocar um erro de sintaxe.

14.9 Gravando dados aleatoriamente em um arquivo de acesso aleatório

A Fig. 14.12 grava dados no arquivo “credit.dat”. Ele usa a combinação das funções seekp e write de
ostream para armazenar dados em posições precisas no arquivo. A função seekp inicializa o ponteiro put de
posição do arquivo para uma posição específica no arquivo, então write grava os dados. Um exemplo de execução
é mostrado na Fig. 14.13. Note que o programa da Fig. 14.12 inclui (linha 16)0 arquivo de cabeçalho clntdata . h
definido na Fig. 14.11.

```
1 // Fig. 14.12: figl4l2.cpp
2 II Gravando dados aleatoriamente em um arquivo de acesso aleatório
3 #include <iostream>
4
5 using std: :cerr;
6 using std::endl;
7 using std::cout;
8 using std::cin;
9 using std::ios;
10
11 #include <fstream>
12
13 using std::ofstream;
14
15 #include <cstdlib>
16 #include “clntdata.h”
17
```

Fig. 14.12 Criando um arquivo de acesso aleatório seqüencialmente (parte 1 de 2).


```
18 int main()
19
20 ofstream outCredit( “credit.dat”, ios::binary );
21
22 if ( !outCredit
23 cerr « ‘Arquivo não pode ser aberto.” « endi;
24 exit( 1 );
25
26
27 cout « “Digite o número da conta
28 « “(1 a 100, O para terminar entrada)\n? “;
29
30 clientData client;
31 cm » client.accountNuniber;
32
33 while ( client.accountNumber > O &&
34 client.accountNuinber <= 100 ) {
35 cout « “Digite sobrenome, primeiro nome, saldo\n? “;
36 cm » client.lastName » client.firstName
37 » client.balance;
38
39 outCredit.seekp( ( client.accountNumber - 1 ) *
40 sizeof( clientData ) );
41 outCredit.write
```

```
42 reinterpret_cast<const char *>( &client ),
43 sizeof( clientData ) );
44
45 cout « “Digite número da conta\n?
46 cm » client.accountNumber;
47 }
48
49 return 0;
50
```

Fig. 14.12 Criando um arquivo de acesso aleatório seqüencialmente (parte 2 de 2).

```
Digite o número da conta (1 a 100, O para terminar entrada)
? 37
Digite sobrenome, primeiro nome, saldo
? Barker Doug 0.00
Digite o número da conta
? 29
Digite sobrenome, primeiro nome, saldo
? Brown Nancy -24.54
Digite o número da conta
? 96
Digite sobrenome, primeiro nome, saldo
? Stone San 34.98
Digite o número da conta
? 88
Digite sobrenome, primeiro nome, saldo
? Smith Dave 258.34
Digite o número da conta
? 33
Digite sobrenome, primeiro nome, saldo
? Dunn Stacey 314.33
Digite o número da conta
```

Fig. 14.13 Exemplo de execução do programa da Fig. 14.12.

As linhas 39 e 40 26
27
outCredit.seekp((client.accountNuxnber - 1) * 28
sizeof(clientData)); 29
30
posicionam o ponteiro de posição par do arquivo para o objeto outCredit na posição em bytes calculada por 31
(client . accountNumber - 1) * sizeof (clientData) Como o número da conta está entre 1 e 100,
i é subtraido do número da conta quando se calcula a posição em bytes do registro. Desta forma, para o registro 1, 34

o ponteiro de posição do arquivo é inicializado com o byte O do arquivo. Note que o objeto outCredit de ofstream 35
é aberto com o modo de abertura de arquivo ios: :binary. 36
14.10 Lendo dados seqüencialmente de um arquivo de acesso aleatório
Nas seções anteriores, criamos um arquivo de acesso aleatório e gravamos dados nesse arquivo. Nesta seção, desen- 41
volvemos um programa que lê todo o arquivo seqüencialmente e imprime somente aqueles registros que contém 42
dados. Este programa fornece um benefício adicional. Veja se você pode determinar qual é ele; nós o revelaremos no 43
fim desta seção.
A função read de istream lê um número de bytes especificado da posição corrente no stream especificado
para um um objeto. Por exemplo, as linhas 43 e 44
inCredit.read(reinterpret_cast<char *>(&client),
sizeof(clientData)); 50
51
da Fig 14.14 Icem o numero de bytes especificado por sizeof (clientData) do arquivo associado ao objeto 52
inCredit de ifstream e armazenam os dados na estrutura client. Note que a função read exige um 53
primeiro argumento do tipo char . Como &client é do tipo clientData , &client deve sofrer uma 54
coerção para char* com o operador de coerção reinterpret cast. Note que o programa da Fig. 14.14 55
inclui o arquivo de cabeçalho clntdata . h definido na Fig. 14.11. 56 57
58
1 Il Fig. 14.14: figl4_14.cpp
2 Il Lendo seqüencialmente um arquivo de acesso aleatório 60
3 #include <iostream> 61
4 62
5 using std::cout; 63
6 using std::endl; 64
7 using std: :ios; 65
8 using std::cerr; 66
10 #include <iomanip> Cor
11 29
12 using std::setprecision;
13 using std::setiosflags; 1
14 using std: :resetiosflags; 88
15 using std::setw;
16
17 #include <fstream> Fini
18
19 using std::ifstream;
20 using std::ostream; AFig

```
21 ele co
22 #include <cstdlib>
23 #include “clntdata.h”
24
25 void outputLine( ostream&, const clientData & ); usaal
```

Fig. 14.14 Lendo seqüencialmente um arquivo de acesso aleatório (parte 1 de 2). daest.


```
26
27 int main()
28
29 ifstream inCredit( “credit.dat”, ios::in );
30
31 if ( 1inCredit ) {
32 cerr « “Arquivo não pode ser aberto.” « endi;
33 exit( 1 );
34 }
35
36 cout « setiosflags( ios::left ) « setw( 10 ) « “Conta”
37 « setw( 16 ) « “Sobrenome” « setw( 11
38 « “Primeiro nome” « resetiosflags( ios::left
39 « setw( 10 ) « “Saldo” « endi;
40
41 clientData client;
42
43 inCredit.read( reinterpretcast<char*>( &client ),
44 sizeof( clientData ) );
45
46 while ( inCredit && !inCredit.eof O ) {
47
48 if ( client.accountNumber != O
49 outputLine( cout, client );
50
51 inCredit.read( reinterpret_cast<char *>( &client ),
52 sizeof( clientData ) );
53 )
54
55 return 0;
56
57
58 void outputLine( ostream &output, const clientData &c
59 {
60 output « setiosflags( ios::left ) « setw( 10
61 « c.accountNimiber « setw( 16 ) « c.lastName
62 « setw( 11 ) « c.firstName « setw( 10
```

63 « setprecision (2) « resetiosflags (ios: : left
64 « setiosflags(ios::fixed 1 ios::showpoint
65 « c.balance « '\n';
66 }

4 Fig. 14.14 Lendo seqüencialmente um arquivo de acesso aleatório (parte 2 de 2).

A Fig. 14.14 lê seqüencialmente todos os registros no arquivo "credit . dat", examina cada registro para ver se ele contém dados e exibe saídas formatadas para os registros que contêm dados. A condição na linha 46

while (inCredit && 'inCredit.eof()) {

usa a função membro eof de ios para determinar quando o fim do arquivo é encontrado e faz com que a execução da estrutura while termine. Além disso, se há um erro de leitura do arquivo, o laço terminará porque inCredit

Conta	Sobrenome	Primeiro nome Saldo		
29	Brown	Nancy	-24.54	
33	Dunn	Stacey	314.33	
37	Barker	Doug	0.00	
88	Smith	Dave	258.34	
96	Stone	Sam	34.98	

assumirá o valor false. Os dados udos do arquivo são enviados para a saída por outputLine, que aceita dois argumentos - um objeto ostream e uma estrutura clientData para ser enviada para a saída. O tipo de parâmetro ostream é interessante porque qualquer objeto ostream (tal como cout) ou qualquer objeto de uma classe derivada de ostream (tal como um objeto do tipo ofstreain) pode ser fornecido como o argumento. Isso significa que a mesma função pode ser usada, por exemplo, para executar saída pelo stream de saída padrão e para um stream de arquivo, sem necessidade de escrever funções separadas.

Mas qual o benefício adicional que prometemos? Se você examinar a janela de saída, você notará que os registros são listados em ordem (classificados pelo número de conta)! Isto é uma simples conseqüência do modo como armazenamos estes registros no arquivo, usando as técnicas de acesso direto. Comparado ao bubble sort que vimos (Capítulo 4), classificar com técnicas de acesso direto é extremamente rápido. A velocidade é obtida fazendo- se os arquivos suficientemente grandes para manter quaisquer registros que possam ser criados. Isso, naturalmente, significa que o arquivo poderia ser, na maioria das vezes, esparsamente ocupado, o que é um desperdício de memória. Assim, aqui temos um outro exemplo da solução de compromisso de trocar espaço por tempo: por usarmos grandes volumes de espaço, somos capazes de desenvolver um algoritmo de classificação muito mais rápido.

14.11 Exemplo: um programa de processamento de transações

Agora, apresentamos um programa deporte para processamento de transações (Fig. 14.15), usando um arquivo de acesso aleatório para obter processamento de acesso “instantâneo”. O programa mantém informações de contas bancárias. O programa atualiza contas existentes, acrescenta novas contas, elimina contas e armazena uma lista formatada de todas as contas correntes em um arquivo-texto para impressão. Assumimos que o programa da Fig. 14. II foi executado para criarmos o arquivo credit . dat e que o programa da Fig. 14.12 foi executado para inserir os dados iniciais.

O programa tem cinco opções (a opção é para terminar o programa). A opção 1 chama a função textFile para armazenar uma lista formatada de todas as informações das contas em um arquivo-texto chamado print. txt que pode ser impresso mais tarde. A função textFile aceita um objeto fstream como argumento para ser usado para a leitura dos dados do arquivo credit . dat. A função textFile usa a função membro read de istream e as técnicas de acesso a arquivos seqüenciais da Fig. 14.14 para ler dados de credito . dat. A função outputLine discutida na Seção 14.10 é usada para a saída dos dados para o arquivo print. txt. Note que textFile usa a função membro seekg de istream para garantir que o ponteiro de posição do arquivo está no começo do arquivo. Após a escolha da opção 1, o arquivo print. txt contém:

A opção 2 chama a função updateRecord para atualizar uma conta. A função somente atualizará um registro já existente, assim a função primeiro verifica se o registro especificado está vazio. O registro é lido para a estrutura client com a função membro read de istream, então client. accountNumber é comparado com zero para determinar se o registro contém informações. Se client . accountNuniber é zero, uma mensagem é impressa dizendo que o registro está vazio e as opções do menu são exibidas. Se o registro contém informações, a função updateRecord exibe o registro na tela usando a função outputLine. lê o montante da transação, calcula o novo saldo e regrava o registro no arquivo. A saída típica para a opção 2 é:

```
Conta Sobrenome Primeiro nome Saldo
29 Brown Nancy -24.54
33 Dunn Stacey 314.33
37 Barker Doug 0.00
88 Smith Dave 258.34
96 Stone Sam 34.98

Digite número da conta a ser atualizada (1 a 100): 37
37 Barker Doug 0.00
Digite débito (+) ou pagamento (-): +87.99
37 Barker Doug 87.99
```


using

using
using
using

using
using
using

using
using
using
using

void
void

std: :cerr; std: :cin; std: :endl; std: :ios;

std: :ofstream; std: :ostream; std: :fstream;
std: : setiosflags; std: : resetiosflags; std: :setw;
std: : setprecision;

A opção 3 chama a função newRecord para adicionar um novo registro ao arquivo. Se o usuário fornecer um número de conta já existente, newRecord exibirá uma mensagem de erro informando que a conta já existe e as opções do menu serão exibidas novamente. Essa função usa o mesmo processo que o programa da Fig. 14.12 para adicionar uma nova conta. Uma saída típica da opção 3 é

Digite o número da nova conta (1 a 100): 22
Digite sobrenome, primeiro nome, saldo
? Johnston Sarah 247.45

A opção 4 chama a função deleteRecord para excluir um registro de um arquivo. A exclusão é realizada pedindo ao usuário um número de conta. Somente um registrojá existente pode ser excluído; assim, caso a conta especificada esteja vazia, é exibida uma mensagem de erro. Uma mensagem é exibida para informar ao usuário que o registro foi excluído. Uma saída típica da opção 4 é

Digite número da conta a ser eliminada (1 a 100): 29 Conta N°29 eliminada.

O arquivo “credit. dat” é aberto criando-se um objeto fstream para ler e gravar usando-se um “ou” dos modos ios: : in e ios: : out.

```
II Fig. 14.15: figl4_15.cpp
II Este programa lê um arquivo de acesso aleatório
II seqüencialmente, atualiza dados já gravados no arquivo,
// cria novos dados a serem colocados no arquivo
// e elimina dados que já estão no arquivo.
#include <iostream>
using std::cout;

#include <fstream>

#include <iomanip>

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
```

35
Fig. 14.15 Programa de contas bancárias (parte 1 de 5).

```
#include <cstdlib>
#include ‘clntdata.h’
int enterChoiceO;
void textFile( fstream& );
void updateRecord( fstream& );
void newRecord( fstream& );
deleteRecord( fstream& );
outputLine( ostream&, const clientData & );

1

748 C++ COMO PROGRAMAR
36 int getAccount( const char * const );
37
38 enuzn Choices { TEXTFILE = 1, UPDATE, NEW, DELETE, END };
39
40 int main()
41
42 fstream inOutCredit( “credit.dat”, ios::in 1 ios::out );
43
44 if ( ‘inOutCredit
45 cerr « “Arquivo não pode ser aberto.” « endi;
46 exit ( 1 );
47 }
48
49 int choice;
50
51 while ( ( choice = enterChoice() ) != END ) {
52
53 switch ( choice
54 case TEXTFILE:
55 textFile( inOutCredit );
56 break;
57 case UPDATE:
58 updateRecord( inOutCredit );
59 break;
60 case NEW:
61 newRecord( inOutCredit );
62 break;
63 case DELETE:
64 deleteRecord( inOutCredit );
65 break;
66 default:
```

```
67 cerr « “Opção incorreta\n”;
68 break;
69 }
70
71 inOutCredit.clearO; II desliga indicador de fim de arquivo
72
73
74 return 0;
75 }
76
77 II Pede para selecionar uma opção do menu
78 int enterChoice()
79 {
80 cout « “\nDigite sua opção” « endl
81 « “1 - criar um arquivo de contas no formato de texto\n”
82 « “ chamado \“print.txt\” para impressão\n”
83 « “2 - atualizar uma conta\n”
84 « “3 - acrescentar uma nova conta\n”
85 « “4 - eliminar uma conta\n”
86 « “5 - terminar o programa\n?
87
88 int menuChoice;
89 cm » menuChoice;
90 return menuChoice;
91 }
92
93 // Cria arquivo no formato de texto para impressão
```

Fig. 14.15 Programa de contas bancárias (parte 2 de 5).


```
94 void textFile( fstream &readFromFile
95 {
96 ofstream outPrintFile( “print.txt”, ios::out );
97
98 if ( ‘outPrintFile
99 cerr « ‘Arquivo não pode ser aberto. « endi;
100 exit( 1 );
101
102
103 outPrintFile « setiosflags( ios::left ) « setw( 10
104 « “Conta” « setw( 16 ) « “Sobrenome” « setw( 11
105 « “Primeiro nome” « resetiosflags( ios::left
106 « setw( 10 ) « “Saldo” « endl;
107 readFromFile.seekg( O );
108
```

```
109 clientflata client;
110 readFromFile.read( reinterpret_cast<char *>( &client ),
111 sizeof( clientData ) );
112
113 while ( ‘readFromFile.eof() ) {
114 if ( client.accountNumber != O
115 outputLine( outPrintFile, client );
116
117 readFromFile.read( reinterpret_cast<char *>( &client ),
118 sizeof( clientData ) );
119 }
120 }
121
122 // Atualiza o saldo de uma conta
123 void updateRecord( fstream &updateFile
124
125 int account = getAccount( “Digite número da conta a ser atualizada” );
126
127 updateFile.seekg( ( account - 1 ) * sizeof( clientData ) );
128
129 clientData client;
130 updateFile.read( reinterpret_cast<char *>( &client ),
131 sizeof( clientData ) );
132
133 if ( client.accountNumber ! O ) {
134 outputLine( cout, client );
135 cout « “\nDigite débito (+) ou pagamento (-): “;
136
137 double transaction; // débito ou pagamento
138 cm » transaction; II deveria validar
139 client.balance + transaction;
140 outputLine( cout, client );
141 updateFile.seekp( ( account-1 ) * sizeof( clientData ) );
142 updateFile .write(
143 reinterpret_cast<const char *>( &client ),
144 sizeof( clientData ) );
145 }
146 else
147 cerr « “Conta N°” « account
148 « “ não tem informação.” « endl;
149
150
151 // Cria e insere um novo registro
```

Fig. 14.15 Programa de contas bancárias (parte 3 de 5).

1

750 C++ COMO PROGRAMAR
152 void newRecord(fstream &insertlnFile
153
154 int account = getAccount(“Digite o niunero da nova conta”);
155
156 insertlnFile.seekg((account-1) * sizeof(clientData));
157
158 clientData client;
159 insertlnFile.read(reinterpret_cast<char *>(&client),
160 sizeof(clientData));
161
162 if (client.accountNumber == O) {
163 cout « “Digite sobrenome, primeiro nome, saldo\n?
164 cm » client.lastName » client.firstName
165 » client.balance;
166 c1ient.accountNi.mber = account;
167 insertlnFile.seekp((account - 1) *
168 sizeof(clientData));
169 insertlnFile .write
170 reinterpret_cast<const char *>(&client),
171 sizeof(clientData));
172 }
173 else
174 cerr « “Conta N’ « account
175 « “ já contém informação.” « endi;
176
177
178 1/ Elimina um registro já existente
179 void deleteRecord(fstream &deleteFromFile
180 {
181 int account = getAccount(Digite número da conta a ser eliminada”);
182
183 deleteFromFile.seekg((account-1) * sizeof(clientData));
184
185 clientData client;
186 deleteFromFile.read(reinterpret_cast<char *>(&client),
187 sizeof(clientData));
188
189 if (client.accountNwnber != O) {
190 clientData blankClient = { 0, ‘‘, “, 0.0 };
191
192 deleteFromFile.seekp((account - 1) *
193 sizeof(clientData));
194 deleteFromFile .write

```
195 reinterpretcast<const char *>( &blankClient ),
196 sizeof( clientData ) );
197 cout « “Conta N°” « account « ‘ eliminada.” « endi;
198 }
199 else
200 cerr « “Conta N° « account « “ está vazia.” « endl;
201 }
202
203 /1 Envia uma linha com informações do cliente para a saída
204 void outputLine( ostream &output, const clientData &c
205 {
206 output « setiosflags( ios::left ) « setw( 10
207 « c.accountNumber « setw( 16 ) « c.lastName
208 « setw( 11 ) « c.firstName « setw( 10
209 « setprecision( 2 ) « resetiosflags( ios::left
```

Fig. 14.15 Programa de contas bancárias (parte 4 de 5).


```
210 « setiosflags( ios::fixed 1 ios::showpoint
211 « c.balance « ‘\n’;
212 }
213
214 // Lê um número de conta do teclado
215 int getAccount( const char * const prompt
216 {
217 int account;
218
219 do
220 cout « prompt « (1 a 100):
221 cm » account;
222 } while ( account < 1 1 1 account > 100 );
223
224 return account;
225
```

Fig. 14.15 Programa de contas bancárias (parte 5 de 5).

14.12 Entrada/saída de objetos

Neste capítulo e no Capítulo 11, discutimos o estilo de entradaJsaída orientado a objetos de C++. Mas nossos exemplos se concentraram sobre a EIS de tipos de dados tradicionais e não sobre objetos de classes definidas pelo usuário. No Capítulo 8, mostramos como fazer entrada e saída de objetos de classes usando sobrecarga de operadores. Executamos a entrada de objetos sobrecarregando o operador de extração de stream » para as classes istream apropriadas. Executamos a saída de objetos sobrecarregando o operador de inserção em stream « para as classes ostream apropriadas. Em ambos os casos, somente os membros de dados de um objeto foram udos ou transferidos para a saída e, em

cada caso, de uma forma com significado para objetos daquele tipo de dados abstrato particular. As funções membro de um objeto estão disponíveis internamente no computador e são combinadas com os valores de dados à medida que estes dados são lidos através do operador de inserção em stream sobrecarregado.
Quando membros de dados de objetos são enviados na saída para um arquivo em disco, perdemos as informações sobre o tipo do objeto. Temos somente bytes de dados, não informações sobre seu tipo, em um disco. Se o programa que vai ler estes dados sabe a que tipos de objetos eles correspondem, então os dados são simplesmente lidos para objetos daquele tipo.
Um problema interessante ocorre quando armazenamos objetos de diferentes tipos no mesmo arquivo. Como podemos distingui-los (ou suas coleções de membros de dados) quando os lemos para um programa? O problema, naturalmente, é que objetos tipicamente não têm campos de tipo (estudamos este problema cuidadosamente no Capítulo lo).
Uma abordagem seria fazer cada operador de saída sobrecarregado colocar na saída um código de tipo precedendo cada coleção de membros de dados que representa um objeto. Então, a entrada de objetos sempre começaria pela leitura do campo de código de tipo e usaria um comando switch para invocar a função sobrecarregada apropriada. Embora esta solução não tenha a elegância da programação polimórfica, ela fornece um mecanismo prático para guardar objetos em um arquivo e recuperá-los conforme necessário.
Resumo
• Todos os itens de dados processados por um computador são reduzidos a combinações de zeros e uns.
• O menor item dados em um computador pode assumir o valor O ou o valor 1. Tal item de dados é chamado de bit.
• Dígitos, letras e símbolos especiais são conhecidos como caracteres. O conjunto de todos os caracteres que podem ser usados para escrever programas e representar itens de dados em um computador em particular é chamado de conjunto de caracteres daquele computador. Cada caractere no conjunto de caracteres do computador é representado como uma combinação de oito is e Os (chamada de byte).

Um campo é um grupo de caracteres (ou bytes) que tem um significado.

• Um registro é um grupo de campos relacionados.
• Pelo menos um campo em um registro é escolhido como a chave do registro para identificar um registro como pertencente a uma pessoa ou entidade particular, que é diferente de todos os outros registros no arquivo.
• O acesso seqüencial é o método mais popular para acessar dados em um arquivo.
• Uma coleção de programas projetada para criar e gerenciar bancos de dados é chamada de sistema gerenciador de bancos de dados (SGBD).
• C÷+ vê cada arquivo como um stream seqüencial de bytes.

• Cada arquivo termina com alguma forma de marcador de fim de arquivo dependente da máquina.
• Streams fornecem canais de comunicação entre arquivos e programas.
• Os arquivos de cabeçalho <iostream> e <fstreaiu> devem ser incluídos em um programa para executar a EIS com arquivo ao estilo de C+÷. O arquivo de cabeçalho <fstreani> inclui as definições das classes para streams ifstream. ofstream efstream.
• Arquivos são abertos instanciando-se objetos das classes para streams ifstream. ofstream e fstream.
• C++ não impõe estruturas a um arquivo. Desta forma, conceitos como "registro" não existem em C++. O programador deve estruturar um arquivo para atender aos requisitos de uma aplicação particular.
• Arquivos são abertos para saída criando-se um objeto da classe ofstream. Dois argumentos são passados ao objeto - o nome do arquivo e o modo de abertura do arquivo. Para um objeto ofstream. o modo de abertura do arquivo pode ser ios: : out, para saída de dados para um arquivo, ou ios: : app, para acrescentar dados ao fim de um arquivo. Arquivos existentes abertos com modo ios: : out são truncados. Se o arquivo não existe, ele é criado.
• A função membro operador operator 1 de ios retorna um valor verdadeiro se o failbit ou o badbit foi ligado para um stream na operação open.
• A função membro operador operator void* de ios converteco stream em um ponteiro para comparação com O (o ponteiro nulo). Se o failbit ou o badbit foi ligado para o stream, é retornado O (falso).
• Programas podem não processar arquivos, processar somente um arquivo ou vários arquivos. Cada arquivo tem um nome que é único e está associado com um objeto arquivo stream apropriado. Todas as funções de processamento de arquivo devem referenciar um arquivo com o objeto apropriado.
• Um "ponteiro get" indica a posição do arquivo da qual a próxima leitura ocorrerá e um "ponteiro put" indica a posição no arquivo na qual a próxima saída será colocada. Tanto a classe istreaiu como a classe ostream fornecem funções membro para reposicionar o ponteiro de posição no arquivo. As funções são seekg ("seek get") para a classe istream e seekp ("seekput") para a classe ostream.
• As funções membro tellp e tellg retornam as posições correntes dos ponteiros put e get.
• Uma maneira conveniente de implementar arquivos de acesso aleatório é usar somente registros de comprimento fixo. Usando esta técnica, um programa pode rapidamente calcular a posição exata de um registro em relação ao começo do arquivo.
• Dados podem ser inseridos em um arquivo de acesso aleatório sem destruir outros dados no arquivo. Dados podem ser atualizados ou excluídos sem regravar todo o arquivo.
• A função membro write de ostream faz a saída para um stream especificado de algum número de bytes, começando em uma posição designada na memória. Quando o stream está associado com um arquivo, os dados são gravados na posição especificada pelo ponteiro put de posição no arquivo.

• A função membro read de isream extrai um número especificado de bytes do stream especificado para uma área na memória começando em um endereço designado. Os bytes são extraídos começando pela posição especificada pelo ponteiro ge! de posição no arquivo. A função read requer um primeiro argumento do tipo char*.
• A função write espera um primeiro argumento do tipo const char*. de modo que este argumento deve sofrer uma coerção para const char* se for de algum outro tipo de ponteiro. O segundo argumento é um inteiro que especifica o número de bytes a serem gravados.
• O operador unário sizeof retorna, durante a compilação, o tamanho em bytes do objeto contido entre parênteses; sizeof retorna um uns igned integer.

• A função membro eof de ias informa se o indicador de fim de arquivo foi ligado no stream designado. O fim de arquivo é ligado após falhar uma tentativa de leitura.

Terminologia

abrir um arquivo
arquivo
arquivo de acesso aleatório
arquivo de acesso seqüencial arquivo de cabeçalho <fstream>

bit

byte

campo

campo alfabético campo de caracteres campo numérico

cerr (saída padrão para erros, sem buifer)
chave de registro
cm (entrada padrão)

classe fstream classe ifstreain classe istream classe ofstrearn

classe ostream
clog (saída padrão para erros, com buifer) conjunto de caracteres

cout (saída padrão)
database / banco de dados

database management system (DB MS) dígito binário
dígito decimal
EIS em memória

EIS em núcleo fechar um arquivo

fim de arquivo
função membro dose função membro open
função membro operator void* função membro operator função membro seekg de istream função membro seekp de ostream função membro tellg de istreaxn

função membro tellp de ostreani hierarquia de dados
manipulator de stream ends

marcador de fim de arquivo modo de abertura de arquivo ios: : app modo de abertura de arquivo ios: : ate

modo de abertura de arquivo ios: : binary modo de abertura de arquivo ias: : in modo de abertura de arquivo ias: : out modo de abertura de arquivo ias: : trunc nome de arquivo
ponteiro de posição no arquivo ponto de início de seek ias: : beg ponto de início de seek ias: : cur ponto de início de seek ias: : end registro

símbolo especial stream
stream de entrada

stream de saída truncar um arquivo existente

Erros comuns de programação

14.1 Abrir um arquivo existente para saída(ias: : aut) quando, na verdade, o usuário quer preservar o arquivo; o conteúdo do arquivo é descartado sem aviso.

14.2 Usar um objeto afstream incorreto para se referir a um arquivo.
14.3 Não abrir um arquivo antes de tentar referenciá-lo em um programa.

Boa prática de programação

14.1 Abrir um arquivo somente para entrada (usando ias: : in), se o conteúdo do arquivo não deve ser modificado. Isso evita a modificação não-intencional do conteúdo do arquivo. Esse é um exemplo do princípio do privilégio mínimo.

Dica de desempenho

14.1 Feche explicitamente cada arquivo tão logo se saiba que o programa não fará referência ao arquivo novamente. Isso pode reduzir o consumo de recursos no programa que vai continuar sendo executado após não mais necessitar de um arquivo
particular. Essa prática também melhora a clareza do programa.

754 C++ COMO PROGRAMAR
Exercícios de auto-revisão
14.1 Preencha os espaços em branco em cada um dos seguintes itens:
a) Em última instância, todos os items de dados processados por um computador são reduzidos a combinações de
_____ e _____
b) O menor item de dados que um computador pode processar é chamado de _____
c) Um é um grupo de registros relacionados.
d) Dígitos, letras e símbolos especiais são chamados de _____
e) Um grupo de arquivos relacionados é chamado de_____
f) A função membro _____ das classes stream de arquivo fstream. ifstream e ofstreazn fecha um arquivo.
g) A função membro _____ de istreani lê um caractere do stream especificado.
h) As funções membro _____ e _____ de is tream lêem uma linha do stream especificado.
i) A função membro _____das classes streamde arquivo fstream. ifstream e ofstream abre um arquivo.
j) A função membro _____ de istreamé normalmente usada quando da leitura de dados de um arquivo em aplicações de acesso aleatório.
k) As funções membro _____ e _____ das classes istream e ostream inicializam o ponteiro de posição para uma posição específica em um stream de entrada ou saída, respectivamente.
14.2 Indique quais das seguintes afirmações são verdadeiras e quais são falsas. Se forem explique por quê:
a) A função membro read não pode ser usada para ler dados do objeto de entrada cm.
b) O programador deve criar explicitamente os objetos ci cout. cerr e clog.
c) Um programa deve chamar explicitamente a função dose para fechar um arquivo associado a um objeto ifstream. ofstream ou fstream.
d) Se o ponteiro de posição no arquivo aponta para uma posição em um arquivo seqüencial diferente do começo do arquivo, o arquivo deve ser fechado e reaberto para se ler desde o começo do arquivo.
e) A função membro write de ostream pode escrever no stream padrão de saída cout.
f) Dados em arquivos de acesso seqüencial são sempre atualizados sem gravar

por cima dos dados vizinhos.
g) Não é necessário pesquisar todos os registros em arquivos de acesso aleatório para encontrar um registro específico.
h) Registros em um arquivo de acesso aleatório devem ser de comprimento uniforme.
i) As funções membro seekp e seekg devem começar a pesquisa no início de um arquivo.
14.3 Assuma que cada uma das seguintes afiramações se aplica ao mesmo programa.
a) Escreva um comando que abre o arquivo " oldmast . dat" para entrada; use o objeto inOidMaster de ifstreaxn.
b) Escreva um comando que abre o arquivo "trans . dat para entrada; use o objeto inTransaction de ifstreain.
c) Escreva um comando que abre o arquivo " newmast. dat' para saída (e criação); use o objeto outnewMaster de ofstream.
d) Escreva um comando que lê um registro do arquivo 'oldmast . dat". O registro consiste no inteiro accountNum. do string name e do valor em ponto flutuante currentBalance: use o objeto inOidmaster de ifstrearn.
e) Escreva um comando que lê um registro do arquivo "trans. dat'. O registro consiste no inteiro accouritNum e do valor em ponto flutuante dollarAmount: use o objeto inTransaction de ifstream.
f) Escreva um comando que grava um registro no arquivo newmast. dat". O registro consiste no inteiro accountNum, do string name e do valor em ponto flutuante currentBalance: use o objeto outNewMaster
de ofstream.
14.4 Encontre o(s) erro(s) e mostre como corrigi-lo(s) em cada um dos seguintes itens.
a) O arquivo "payables . dat" referenciado pelo objeto outPayable de ofstream não foi aberto.
outPayable « account « company « amount « endi;
b) O seguinte comando deveria ler um registrn do arquivo "payables . dat". O objeto inPayable de ifstream referencia este arquivo e o objeto inReceivable de istream referencia o arquivo "receivables . dat".
inReceivable » account » company » arnount;
e) O arquivo "tools . dat" deveria ser aberto para adicionar dados ao arquivo sem descartar os dados correntes.
ofstream outTools("tools.dat", ios::out);

Respostas aos exercícios de auto-revisão
14.1 a) is, Os. b) Bit. c) Arquivo. d) Caracteres, e) Database/Banco de dados. 1) dose. g) get. h) get, getline. i) open. j) read. k) seekg. seekp.
14.2 a) Falsa. A função read pode ler de qualquer objeto de entrada stream derivado de istreain.
b) Falsa. Estes quatro streain.s são criados automaticamente para o programador. O arquivo de cabeçalho <iostreaan deve ser incluído em um

arquivo para usá-los. Este cabeçalho inclui declarações de cada um destes objetos stream.
c) Falsa. Os arquivos serão fechados quando destruidores para os objetos ifstreaxn. ofstream ou fstreain são executados, quando os objetos stream saem do escopo ou antes da execução do programa terminar, mas é uma boa prática de programação fechar explicitamente todos os arquivos com dose quando eles não são mais necessários.
d) Falsa. A função membro seekp ou seekg pode ser usada para reposicionar o ponteiro put ou get de posição no arquivo no começo do arquivo.
e) Verdadeira.
f) Falsa. Na maioria dos casos, os registros de arquivos seqüenciais não são de comprimento uniforme. Portanto, é possível que a atualização de um registro faça com que dados de outros registros sejam sobrepostos.
g) Verdadeira.
h) Falsa. Os registros em arquivos de acesso aleatório são normalmente de comprimento uniforme,
i) Falsa. E possível pesquisar do começo do arquivo, do fim do arquivo e da posição corrente no arquivo.
14.3 a) ifstream inOidMaster(“oidmast.dat’, ios: :in
b) ifstream inTransaction(“trans.dat”, ios::in);
c) ofstream outNewMaster(riewmast.dat”, ios::out);
d) iriOidmaster » accountNum » name » currentBaiance;
e) inTransaction » accountNuin » doiiarPanount;
outNewMaster « accountNum « name « currentBaiance;
14.4 a) Erro: o arquivo “payables . dat” não foi aberto antes de se tentar fazer uma saída dados para o stream.
Correção: use a função open de ostream para abrir “payabies . dat’ para saída.
b) Erro: o objeto incorreto de istream está sendo usado para ler um registro do arquivo “payabies . dat”. Correção: use o objeto inPayabie de istream para referenciar “payabies dat”.
c) Erro: o conteúdo do arquivo é descartado porque o arquivo está aberto para saída (ios: out).
Correção: para adicionar dados ao arquivo, abra o arquivo para atualização (ios: : ate) ou abra o arquivo para acréscimo (ios: : app).
Exercícios
14.5 Preencha os espaços em cada um dos seguintes itens:
a) Computadores armazenam grandes volumes de dados em dispositivos de memória secundária como_____
b) Um _____ é composto de vários campos.
c) Um campo que pode conter somente dígitos, letras e espaços é chamado de campo
d) Para facilitar a recuperação de registros específicos de um arquivo, em cada registro é escolhido um campo como a
e) A grande maioria das informações armazenadas em sistemas de computadores é armazenada em arquivos
O Um grupo de caracteres relacionados com um significado é chamado de

_____
g) Os objetos stream padrão declarados pelo arquivo de cabeçalho <iostream> são
_____ _____ e _____
h) A função membro ostreaxn _____ faz a saída de um caractere para o stream especificado.
i) A função membro ostream geralmente é usada para gravar dados em um arquivo acessado
aletoriamente.
j) A função membro istream reposiciona o ponteiro de posição do arquivo em um arquivo.
14.6 Diga quais das seguintes sentenças são verdadeiras e quais são falsas. Se forem falsas, explique porquê:
a) As impressionantes funções executadas por computadores envolvem essencialmente a manipulação de zeros e uns.
b) As pessoas preferem manipular bits em vez de caracteres e campos porque bits são mais compactos.
c) As pessoas especificam programas e items de dados como caracteres; os computadores então manipulam e processam estes caracteres como grupos de zeros e uns.
d) O CEP de 5 dígitos de uma pessoa é um exemplo de um campo numérico.

e) O endereço de uma pessoa é geralmente considerado um campo alfabético nas aplicações de computador.
f) Os itens de dados representados em computadores formam uma hierarquia de dados na qual os items de dados se tornam maiores e mais complexos à medida que progredimos de campos para caracteres, para bits, etc.
g) A chave do registro identifica um registro como pertencendo a um campo particular.
h) A maioria das organizações armazenam toda a informação em um único arquivo para facilitar o processamento por computador.
i) Cada comando que processa um arquivo em um programa C++ explicitamente faz referência àquele arquivo por nome.
j) Quando um programa cria um arquivo, o arquivo é automaticamente conservado pelo computador para referência futura.
14.7 O Exercício 14.3 pediu ao leitor para escrever uma série de comandos simples. Na verdade, esses comandos formam o núcleo de um importante tipo de programa de processamento de arquivos, ou seja, um programa de matching - comparação e correspondência de arquivos. No processamento comercial de dados, é comum ter-se vários arquivos em cada sistema de aplicação. Em um sistema de contas a receber, por exemplo, geralmente há um arquivo mestre contendo informações detalhadas sobre cada cliente, tais como o nome do cliente, o endereço, o número do telefone, o saldo pendente, o limite de crédito, as condições de desconto, aspectos contratuais e, possivelmente, um histórico condensado das compras recentes e pagamentos em dinheiro.

A medida que ocorrem as transações (por exemplo, são feitas vendas e os pagamentos chegam), elas são digitadas em um arquivo. No fim de um determinado período (um mês para algumas empresas, uma semana para outras e um dia em alguns casos) o arquivo de transações (chamado “trans . dat” no Exercício 14.3) é aplicado ao arquivo mestre (chamado “oldmast . dat” no Exercício 14.3), atualizando desta forma o registro de compras e pagamentos para cada conta. Durante um processamento de atualização, o arquivo mestre é regravado como um novo arquivo (“newmast. dat), o qual então é usado, no fim do próximo período, para começar o processo de atualização novamente. Programas de matching de arquivos devem lidar com certos problemas que não existem em programas de um único arquivo. Por exemplo, nem sempre ocorre uma correspondência. Um cliente no arquivo mestre pode não ter feito quaisquer compras ou pagamentos no exercício corrente e, portanto, nenhum registro para este cliente aparecerá no arquivo de transação. Similarmente, um cliente que efetuou algumas compras ou pagamentos pode ter acabado de entrar para o grupo de clientes e a empresa pode não ter tido oportunidade de criar um registro mestre para este cliente.
Use os comandos do Exercício 14.3 como base para escrever um programa completo de contas a receber que efetua
matching de arquivos. Use o número da conta em cada arquivo como a chave do registro para fins de matching. Assuma que cada arquivo é um arquivo seqüencial, com registros armazenados em ordem crescente por número de conta.
Quando ocorre uma correspondência (i.e., um registro com o mesmo número de conta aparece nos arquivos mestre e de
transação), some a quantia em dinheiro no arquivo de transação ao saldo corrente no arquivo mestre e grave o registro em newmast. dat”. (Assuma que compras são indicadas por quantias positivas no arquivo de transação e pagamentos são indicados por quantias negativas.) Quando há um registro mestre para uma conta particular, mas não existe o registro de transação correspondente, simplesmente grave o registro em “newmast. dat. Quando há um registro de transação, mas não um registro mestre correspondente, imprima a mensagem “Registro de transação sem correspondente para a conta número (preencha o número da conta pelo registro de transação).
14.8 Após escrever o programa do Exercício 14.7, escreva um programa simples para criar alguns dados de teste para verificar o programa. Use os seguintes exemplos de dados de contas:

100 AlanJones 348,17
300 MarySmith 27,19
500 Sam Sharp 0,00
700 Suzy Green -14,22

100 27,14
300 62,11
400 100,56

900 82,17

Arquivo Mestre
Número da Conta Nome Saldo

Arquivo de Transação
Número da conta Valor da transação

Tipo de dados Tamanho

char 1
unsigned char 1
short int 2
unsigned short int 2
int 4
unsigned int 4
long int 4
unsigned long int 4
float 4
double 8
long double 16
Nota: os tamanhos dos tipos de dados primitivos de seu computador podem diferir dos listados acima.

15

Estruturas de dados

Objetivos

• Ser capaz de formar estruturas de dados encadeadas usando ponteiros, classes auto-referentes e recursão.
• Ser capaz de criar e manipular estruturas de dados dinâmicas tais como listas encadeadas, filas, pilhas e árvores binárias.
• Entender várias aplicações importantes de estruturas de dados encadeadas.
• Entender como criar estruturas de dados reutilizáveis com gabaritos de classes, herança e composição.

Much that 1 bound, 1 could notfree;
Much that Ifreed returned to me.
Lee Wilson Dodd

‘Will you walk a littlefaster? ‘saida whiting to a snail, ‘There ‘s a porpoise dose behind us, and he ‘s treading on mv tail.’
Lewis Carroil

Há sempre espaço no topo.
Daniel Webster

Prossiga - continue andando.
Thomas Morton

1 think that 1 shall never see
A poem lovely as a tree.
Joyce Kilmer

15

4

0

Visão Geral

15.1 Introdução
15.2 Classes auto-referentes
15.3 Alocação dinâmica de memória
15.4 Listas encadeadas
15.5 Pilhas

15.6 Filas
15.7 Árvores
Resumo• Terminologia Erros comuns de programação Boas práticas de programação Dicas de desempenho Dica de portabilidade • Exercícios de auto-revisão Respostas aos exercícios de auto-revisão Exercícios Seção especial: construindo seu próprio compilador
15.1 Introdução
Estudamos estruturas de dados de tamanho fixo tais como arrays unidimensionais, arrays bidimensionais e structs. Este capítulo introduz estruturas de dados dinâmicas, que crescem e encolhem durante a execução do programa. Listas encadeadas são coleções de itens de dados “alinhadas em uma fila” - as inserções e retiradas são feitas em qualquer lugar em uma lista encadeada. Pilhas (stacks) são importantes em compiladores e sistemas operacionais - as inserções e retiradas são feitas somente em uma extremidade de uma pilha - seu topo. Filas representam filas de espera; as inserções são feitas no fim (também chamado de cauda) de uma fila e as retiradas são feitas na frente (também chamada de cabeça da fila) de uma fila. Arvores binárias facilitam a busca e a classificação de dados em alta velocidade, a eliminação eficiente de itens de dados duplicados, a representação de diretórios de sistemas de arquivos e a compilação de expressões para linguagem de máquina. Estas estruturas de dados têm muitas outras aplicações interessantes.

Discutiremos os principais tipos de estruturas de dados e implementaremos programas que criam e manipulam essas estruturas de dados. Usamos classes, gabaritos de classe, herança e composição para criar e empacotar essas estruturas de dados para reutilização e facilidade de manutenção.
Estudar este capítulo é uma preparação sólida para o Capítulo 20, “A biblioteca padrão de gabaritos (STL)”. A STL é uma parte importante da biblioteca padrão de C++. A STL fornece contêineres, iteradores para percorrer aqueles contêineres e algoritmos para processar os elementos daqueles contêineres. Você verá que a STL pegou cada uma das estruturas de dados que discutimos aqui no Capítulo 15 e as empacotou em gabaritos de classes. O código da STL foi cuidadosamente escrito para ser portável, eficiente e extensível. Uma vez que entenda os princípios e a construção de estruturas de dados conforme apresentados no Capítulo 15, você será capaz de fazer o melhor uso das estruturas de dados, iteradores e algoritmos pré empacotados da STL. A STL é de longe a melhoria isolada mais importante no padrão C++. E um conjunto de componentes de nível internacional para ajudar a pôr em prática o enfoque de “reutilize, reutilize, reutilize”.
Os exemplos do capítulo são programas práticos que você será capaz de usar em cursos mais avançados e em
aplicações em empresas. Os programas são especialmente pesados no tópico manipulação de ponteiros. Os exercícios incluem uma rica coleção de aplicações úteis.
Incentivamo-lo a tentar fazer o projeto principal descrito na seção especial intitulada “Construindo seu próprio compilador”. Você tem usado um compilador

para traduzir seus programas em C+-t- para linguagem de máquina de forma que possa executar estes programas em seu computador. Neste projeto, você construirá realmente seu próprio compilador. E lerá um arquivo de comandos escritos em uma linguagem simples, mas ainda assim poderosa, de alto nível, semelhante às primeiras versões da popular linguagem BASIC. Seu compilador traduzirá estes comandos para um arquivo de instruções em Simpletron Machine Language (SML) - a SML é a linguagem que você aprendeu no Capítulo 5, seção especial, “Construindo seu próprio computador”. Seu programa simulador do Simpletron então executará o programa em SML produzido por seu compilador! Implementar esse projeto usando uma abordagem fortemente orientada a objetos lha dará uma oportunidade maravilhosa para exercitar a maioria daquilo que você aprendeu neste curso. A seção especial o orienta cuidadosamente através da

descrição da linguagem de alto nível e descreve os algoritmos de que você necessita para converter cada tipo de comando da linguagem de alto nível em instruções de linguagem de máquina. Se você aprecia ser desafiado, pode tentar implementar as muitas melhorias tanto do compilador como do simulador do Simpletron sugeridas nos exercícios.

15.2 Classes auto-referentes

Uma classe auto-referente contém um membro ponteiro que aponta para um objeto de classe do mesmo tipo da classe. Por exemplo, a defïnição

```
class Node
public:
Node( int );
void setData( int );
int getData() const;
void setNextPtr( Node *
const Node *getNextptr() const;
private:
int data;
Node *nextptr;
```

define um tipo, Node. O tipo Node tem dois membros de dados privados - o inteiro data e o membro ponteiro nextptr. O membro nextPtr aponta para um objeto de tipo Node - um objeto do mesmo tipo que o que está sendo declarado aqui, daí o termo “classe auto-referente”. O membro nextPtr é chamado de link- i.e., nextPtr pode ser usado para “vincular” um objeto do tipo Node a outro objeto do mesmo tipo. O tipo Node também tem cinco funções membro: um construtor que recebe um inteiro para inicializar o membros data, uma função setData para inicializar o valor do membro dados, uma função getData para retornar o valor do membro data, uma função setNextPtr para inicializar o valor do membro nextPtr e uma função getNextPtr para retornar o valor do membro nextptr.

Os objetos de classes auto-referentes podem ser vinculados para formar estruturas de dados úteis, tais como listas, filas, pilhas e árvores. A Fig. 15.1 ilustra dois objetos de classes auto-referentes vinculados para formar uma lista.

Note que uma barra invertida (\), representando um ponteiro nulo (O), é colocada no membro link do segundo objeto da classe auto-referente, para indicar que o link não aponta para outro objeto. A barra é usada somente para fins de ilustração; ela não corresponde ao caractere barra invertida (\) usado em C÷+. Um ponteiro nulo normalmente indica o fim de uma estrutura de dados da mesma maneira que o caractere nulo (‘ \O T) indica o fim de um string.
Fig. 15.1 Dois objetos vinculados de uma classe auto-referente.
Erro comum de programação 15.1
Não inicializar o link no último nodo de uma lista com nulo (O).
15.3 Alocação dinâmica de memória
Criar e manter estruturas de dados dinâmicas exige alocação dinâmica de memória - a possibilidade de um programa obter mais espaço de memória durante a execução para manter novos nodos e para liberar espaço não mais

necessário. O limite para alocação dinâmica de memória pode ser tão grande quanto a quantidade de memória física disponível no computador ou a quantidade de memória virtual disponível em um sistema de memória virtual. Freqüentemente, os limites são muito pequenos porque a memória disponível deve ser compartilhada entre muitos usuários.
Os operadores new e dele te são essenciais para a alocação dinâmica de memória. O operador new recebe
como argumento o tipo do objeto que está sendo alocado dinamicamente e retorna um ponteiro para um objeto daquele tipo. Por exemplo, o comando
Node *newptr = new Node(10);
aloca sizeof (Node) bytes, executa o construtor de Node e armazena um ponteiro para esta memória em newPtr. Se não há memória disponível, new dispara uma exceção bad alloc. O 10 é o dado do objeto Node.
O operador delete executa o destruidor de Node e desaloca a memória alocada com new - a memória é retornada para o sistema de forma que a memória possa ser realocada no futuro. Para liberar a memória alocada dinamicamente pelo new precedente, use o comando
delete newPtr;
Note que newPtr em si mesmo não é deletado, mas sim o espaço para o qual newPtr aponta. Se newPtr tem o valor O (indicando um ponteiro para nada), o comando precedente não tem nenhum efeito.
As seções seguintes discutem listas, pilhas, filas e árvores. Estas estruturas de dados são criadas e mantidas
com alocação dinâmica de memória e classes auto-referentes.
Dica de portabilidade 15.1
O tamanho de um objeto de uma classe não é necessariamente a soma dos tamanhos de seus membros de dados. Isto ocorre por causa de vários requisitos de alinhamento de limites de endereços dependentes de máquina (ver o Capítulo 16) e por outras razões. Use o operador sizeof para determinar o tamanho de um objeto.
Erro comum de programação 15.2

Assumir que o tamanho de um objeto de uma classe é simplesmente a soma dos tamanhos de seus membros
de dados.
Erro comum de programação 15.3
Não liberar a memória alocado dinamicamente quando ela não é mais necessária pode fazer com que o
sistema fique prematuramente sem memória. Isto é às vezes chamado de ‘perda de memória.”
Boa prática de programação 15.1
Quando a memória que foi alocada dinamicamente com new não é mais necessária, use delete para
liberar a memória imediatamente para o sistema.
Erro comum de programação 15.4
Apagar com delete memória que não foi alocada dinamicamente com new.
Erro comum de programação 15.5
Referenciar memória que foi deletada.
Erro comum de programação 15.6
Tentar dei etar memó ria que já foi deletada pode levar a resultados impossíveis de predizer durante a
execução.

15.4 Listas encadeadas

Uma lista encadeada é um coleção linear de objetos de uma classe auto-referente, chamados de nodos, conectados por ponteiros de encadeamento - daí o termo lista “encadeada”. Uma lista encadeada é acessada através de um ponteiro para o primeiro nodo da lista. Os nodos subseqüentes são acessados através do membro ponteiro de encadeamento armazenado em cada nodo. Por convenção, o ponteiro de encadeamento do último nodo de uma lista é inicializado com nulo (zero), para marcar o fim da lista. Os dados são armazenados em uma lista encadeada dinamicamente - cada nodo é criado conforme necessário. Um nodo pode conter dados de qualquer tipo, inclusive objetos de outras classes. Se os nodos contêm ponteiros de classes base - ou referências para classes base e objetos de classes derivadas relacionadas por herança, podemos ter uma lista encadeada de tais nodos e usar chamadas de funções virtual para processar estes objetos polimorficamente. As pilhas e filas são também estruturas de dados lineares e, como veremos, são versões de listas encadeadas com restrições. As árvores são estruturas de dados não-lineares.

Listas de dados podem ser armazenadas em arrays, mas as listas encadeadas oferecem várias vantagens. Uma lista encadeada é apropriada quando é impossível se predizer o número de elementos de dados a ser representado na estrutura de dados ao mesmo tempo. As listas encadeadas são dinâmicas, assim o comprimento de uma lista pode aumentar ou diminuir conforme necessário. O tamanho de um array C++ “convencional”, porém, não pode ser

alterado, porque o tamanho de um array é fixado durante a compilação. Arrays “convencionais” ficam cheios. Listas encadeadas enchem-se somente quando o sistema não tiver memória suficiente para satisfazer a solicitações dinâmicas de alocação de memória.

1 Dica de desempenho 15.1

_____ Um array pode ser declarado para conter mais elementos que o número de itens esperados, mas isto pode desperdiçar memória. As listas encadeadas podem fornecer uma utilização de memória melhor nestas situações. As listas encadeadas permitem ao programa se adaptar durante a execução.

As listas encadeadas podem ser mantidas classificadas de acordo com uma ordem simplesmente inserindo-se cada novo elemento no ponto apropriado da lista. Os elementos existentes na lista não necessitam ser movidos.

Dica de desempenho 15.2

_____ A inserção e exclusão em um array ordenado pode consumir tempo - todos os elementos em seguida ao elemento inserido ou excluído devem ser movidos deforma apropriada.

Dica de desempenho 15.3

Os elementos de um array são armazenados contiguamente na memória. Isto permite o acesso imediato a qualquer elemento do array porque o endereço de qualquer elemento pode ser calculado diretamente com base na posição de início do array. As listas encadeadas não dispõem de tal “acesso direto” imediato a seu.s elementos.

Os nodos de uma lista encadeada normalmente não estão armazenados contiguamente na memória. Logicamente, porém, os nodos de uma lista encadeada parecem ser contíguos. A Fig. 15.2 ilustra uma lista encadeada com vários nodos.

I)ica de desempenho 15.4

_____ Usar alocação de memória dinâmica (em vez de arrays) para estruturas de dados que crescem e encolhem durante a execução pode economizar meínó ria. Tenha em mente, porém, que ponteiros ocupam espaço e que a alocação de memória dinâmica incorre no overhead das chamadas de funções.

O programa da Fig. 15.3 (cuja saída é mostrada na Fig. 15.4) usa um gabarito de classe List (‘veja Capítulo 12, “Gabaritos”) para manipular uma lista de valores inteiros e uma lista de valores de ponto flutuante. O programa de teste (fig 1503. cpp) oferece cinco opções: 1) insira um valor no início da lista (função insertAtFront), 2) insira um valor no fim da lista (função insertAtBack). 3) retire um valor do início da lista (função removeFromFront), 4) retire um valor do fim da lista (função removeFromBack) e 5) termine o processamento

da lista. Uma discussão detalhada do programa vem a seguir. O exercício 15.20 lhe pede para implementar um função recursiva que imprime uma lista encadeada de trás para diante e o Exercício 15.21 lhe pede para implementar um função recursiva que pesquisa uma lista encadeada procurando por um item de dados particular.

A Fig. 15.3 consiste em dois gabaritos de classe - ListNode e List. Encapsulado em cada objeto List está uma lista encadeada de objetos Lis tNode. O gabarito de classe Lis tNode consiste nos membros privados data e nextptr. O membro data ListNode armazena um valor do tipo NODETYPE, o parâmetro de tipo passado para o gabarito de classe. O membro nextPtr de Lis tNode armazena um ponteiro para o próximo objeto Lis tNode na lista encadeada.

firstPtr lastPtr

Fig. 15.3 Manipulando uma lista encadeada - list.h (parte 1 de 4).

Fig. 15.2 Uma representação gráfica de uma lista.
1 // Fig. 15.3: listnd.h
2 1/ Definição do gabarito ListNode
3 #ifndef LISTNDH
4 #define LISTNDH
5
6 template< class NODETYPE > class List; II declaração antecipada
7
8 template<class NODETYPE>
9 class ListNode
10 friend class List< NODETYPE >; II torna List uni friend
11 public:
12 ListNode(const NODETYPE &); II construtor
13 NODETYPE getData() const; II retorna dados do nodo
14 private:
15 NODETYPE data; // dados
16 ListNode< NODETYPE > *nextPtr; II próximo nodo na lista
17 };
18
19 // Construtor
20 template<class NODETYPE>
21 ListNode< NODETYPE >::ListNode(const NODETYPE &info
22 : data(info), nextptr(O) { }
23
24 // Retorna unia cópia dos dados no nodo
25 template< class NODETYPE >
26 NODETYPE ListNode< NODETYPE >::getData() const { return data;
27
28 #endif
Fig. 15.3 Manipulando uma lista encadeada - listnd. h.

```
29 II Fig. 15.3: list.h
30 II Definição da classe gabarito List
31 #ifndef LISTH
32 #define LISTH
```

CAPÍTULO 15 - ESTRUTURAS DE DADOS 765

```
33
34 #include <iostream>
35 #include <cassert>
36 #include “listnd.h”
37
38 using std::cout;
39
40 template< class NODETYPE >
41 class List {
42 public:
43 Listo; /1 construtor
44 List(); /1 destruidor
45 void insertAtFront( const NODETYPE & );
46 void insertAtBack( const NODETYPE & );
47 bool removeFromFront( NODETYPE & );
48 bool removeFromBack( NODETYPE & );
49 bool isEmptyO) const;
50 void print() const;
51 private:
52 ListNode< NODETYPE > *fjrstptr; II ponteiro para o primeiro nodo
53 ListNode< NODETYPE > *lastPtr; II ponteiro para o último nodo
54
55 // Função utilitária para alocar um novo nodo
56 ListNode< NODETYPE > *getNewNode( const NODETYPE & );
57 };
58
59 // Construtor default
60 template< class NODETYPE >
61 List< NODETYPE >::List() : firstPtr( O ), lastptr( O ) { }
62
63 // Destruidor
64 template< class NODETYPE >
65 List< NODETYPE >::-List()
66 {
67 if ( !isEmpty() ) { // A lista não está vazia
68 cout « “Destruindo nodos .\n”;
69
70 ListNode< NODETYPE > *currentptr = firstPtr, *tempptr;
71
72 while ( currentPtr O ) { // elimina nodos restantes
```

```
73 tempPtr currentPtr;
74 cout « tempPtr->data « '\n';
75 currentPtr = currentPtr->nextPtr;
76 delete tempPtr;
77
78
79
80 cout « "Todos os nodos destruídos\n\n";
81 }
82
83 // Insere um nodo no início da lista
84 template < class NODETYPE >
j. 85 void List< NODETYPE >::insertAtFront( const NODETYPE &value
86 {
87 ListNode< NODETYPE > *newptr = getNewNode( value );
88
89 if ( isEmptyO) ) II A lista está vazia
90 firstptr = lastPtr = newPtr;
```

Fig. 15.3 Manipulando uma lista encadeada - list.h (parte 2 de 4).


```
91 else { II A lista não está vazia
92 newPtr->nextPtr = firstPtr;
93 firstPtr = newPtr;
94
95 }
96
97 // Insere um nodo no fim da lista
98 template< class NODETYPE >
99 void List< NODETYPE >::insertAtBack( const NODETYPE &value
100
101 ListNode< NODETYPE > *Pt = getNewNode( value );
102
103 if ( isEmpty() ) II A lista está vazia
104 firstPtr = lastPtr = newPtr;
105 else { // A lista não está vazia
106 lastPtr->nextPtr = newPtr;
107 lastPtr = newPtr;
108
109 }
110
111 II Retira um nodo do início da lista
112 template< class NODETYPE >
113 bool List< NODETYPE >::removeFromFront( NODETYPE &value
114 {
115 if ( isEmpty() ) // A lista está vazia
```

```
116 return return; II retirada sem sucesso
117 else {
118 ListNode< NODETYPE > *tempPtr = firstPtr;
119
120 if ( firstPtr == lastPtr
121 firstPtr = lastPtr = 0;
122 else
123 firstPtr = firstPtr->nextPtr;
124
125 value = tempPtr->data; II dado sendo removido
126 delete tempptr;
127 return return; // retirada com sucesso
128 }
129 }
130
131 // Retira um nodo do fim da lista
132 template< class NODETYPE >
133 bool List< NODETYPE >::removeFromBack( NODETYPE &value
134 {
135 if ( isEmpty()
136 return return; II retirada sem sucesso
137 else {
138 ListNode< NODETYPE > *tempptr = lastPtr;
139
140 if ( firstPtr == lastPtr
141 firstPtr = lastPtr = 0;
142 else {
143 ListNode< NODETYPE > *currentptr = firstPtr;
144
145 while ( currentPtr->nextPtr != lastPtr
146 currentPtr = currentPtr->nextPtr;
147
148 lastPtr = currentPtr;
```

Fig. 15.3 Manipulando uma lista encadeada - list. h (parte 3 de 4).

```
149
150
151
152
153
154
155
156 }
157
158
```

160
161
162
;i163
164
165
166
167 {
168
169
170
171
172 }
173
174
175
176
177 (
178
179
180
181
182

184

185 cout « “A lista é: “;

186
187
188
189
190
191
192
193 }
194

r

while (currentPtr != O) {
cout « currentPtr->data « currentPtr = currentPtr->nextPtr;

cout « “\n\n”;

195 #endif

Fig. 15.3 Manipulando uma lista encadeada - list.h (parte 4 de 4).

CAPÍTULO 15 - ESTRUTURAS DE DADOS 767

```
currentPtr->nextPtr = O;
value = tempPtr->data;
delete tempPtr;
return return; 1/ retirada com sucesso
1/ A lista está vazia?
template< class NODETYPE >
bool List< NODETYPE >::isEmpty() const
return firstPtr 0; }
II retorna um ponteiro para uni nodo recém alocado
template< class NODETYPE >
ListNode< NODETYPE > *List< NODETYPE >: :getNewNode( const
NODETYPE &value
ListNode< NODETYPE > *ptr =
new ListNode< NODETYPE >( value );
assert( ptr != O );
return ptr;
II Exibe o conteúdo da lista
template< class NODETYPE >
void List< NODETYPE >: :print() const
if ( isEmpty() ) {
cout « “A lista está vazia\n\n”;
return;
183 ListNode< NODETYPE > *currentptr = firstPtr;

(
```

Fig. 15.3 Manipulando uma lista encadeada (parte 1 de 3).

196	II Fig. 15.3: figl5O3.cpp	
197	// Teste da classe	List
198	#include <iostream>	
199	#include “list.h”	
200		

201	using std::cin;	
202	using std::endl;	

768 C++ COMO PROGRAMAR

```
203
204 // Função para testar urna lista integer
205 template< class T >
206 void testList( List< T > &listObject, const char *type
207
208 cout « “Testando uma List de valores « type « “\n;
209
210 instructionsQ;
211 int choice;
212 T value;
213
214 do {
215 cout « “?
216 cm » choice;
217
218 switch ( choice ) {
219 case 1:
220 cout « “Digite um valor “ « type «
221 cm » value;
222 listObject.insertAtFront( value );
223 listObject.print
224 break;
225 case 2:
226 cout « “Digite uni valor « type «
227 cm » value;
228 listObject.insertAtBack( value );
229 listObject.printO;
230 break;
231 case 3:
232 if ( listObject.removeFromFront( value
233 cout « value « removido da lista\n”;
234
235 listObect.prmntO;
236 break;
237 case 4:
238 if ( listObject.removeFrornBack( value
239 cout « value « “ removido da lista\n;
240
241 listObject.printO;
242 break;
```

```
243 )
244 } while ( choice != 5 );
245
246 cout « “Fim do teste da lista\n\n’;
247
248
249 void instructions()
250 {
251 cout « “Digite um dos seguintes:\n”
252 « “ 1 para inserir no início da lista\n”
253 « 2 para inserir no fim da lista \n”
254 “ 3 para retirar do início da lista\n”
255 « “ 4 para retirar do fim da lista \n”
256 5 para terminar o processamento da lista\n”;
257 }
258
259 int main()
260
```

Fig. 15.3 Manipulando uma lista encadeada (parte 2 de 3).


```
261 List< int > integerList;
262 testList( integerList, “integer” ); II testa integerList
263
264 List< double > doubleList;
265 testList( doubleList, “double” ); II testa doubleList
266
267 return O;
268
```

Fig. 15.3 Manipulando uma lista encadeada (parte 3 de 3).

```
Testando uma List de valores integer
Digite um dos seguintes:
1 para inserir no início da lista
2 para inserir no fim da lista
3 para retirar do início da lista
4 para retirar do fim da lista
5 para terminar o processamento da lista
Digite um valor integer: 1
A lista é: 1
?I
Digite um valor integer: 2
A lista é: 2 1
?2
Digite um valor integer: 3
```

A lista é: 2 1 3
Digite um valor integer: 4
A lista é: 2 1 3 4
2 removido da lista
A lista é: 1 3 4
?3
1 removido da lista
A lista é: 3 4
?4
4 removido da lista
A lista é: 3
3 removido da lista
A lista está vazia
Fim do teste da lista
Testando uma List de valores double
Digite um dos seguintes:
1 para inserir no início da lista
Fig. 15.4 Exemplo de saída para o programa da Fig. 15.3 (parte 1 de 2).

2 para inserir no fim da lista
3 para retirar do início da lista
4 para retirar do fim da lista
5 para terminar o processamento da lista
?l
Digite um valor double: 1.1
A lista é: 1.1
?l
Digite um valor double: 2.2
A lista é: 2.2 1.1
?2
Digite um valor double: 3.3
A lista é: 2.2 1.1 3.3
Digite um valor double: 4.4
A lista é: 2.2 1.1 3.3 4.4
2.2 removido da lista
A lista é: 1.1 3.3 4.4
1.1 removido da lista
A lista é: 3.3 4.4
‘4
4.4 removido da lista
A lista é: 3.3
3.3 removido da lista
A lista está vazia
Fim do teste da lista
Todos os nodos destruidos

Todos os nodos destruídos
Fig. 15.4 Exemplo de saída para o programa da Fig. 15.3 (parte 2 de 2).
O gabarito de classe List consiste nos membros private firstPtr (um ponteiro para o primeiro ListNode em um objeto List) e lastPtr (um ponteiro para o último ListNode em um objeto List). O construtor default inicializa ambos os ponteiros com O (nulos). O destruidor assegura que todos os objetos Lis tNode em um objeto List são destruídos quando aquele objeto List é destruído. As funções primárias do gabarito de classe Lis t são insertAtFront, insertAtBack. removeFromFront e removeFromBack.
A função isEmpty é chamada de função predicado - ela não altera List. mas sim determina se List está vazia (i.e., o ponteiro para o primeiro nodo de List é nulo). Se List está vazia, retorna true: caso contrário, ela retorna false. A função print exibe o conteúdo de List.
Boa prática de programação 15.2
Atribua nulo (zero) ao membro de encadeamento (link) de um novo nodo. Os ponteiros devem ser inicializados antes de serem usados.

Durante as próximas várias páginas, discutiremos cada uma das funções membro da classe List em detalhes. A função insertAtFront (Fig. 15.5) coloca um novo nodo na frente da lista. A função consiste em vários passos:
1. Chama a função getNewNode passando para esta value. o qual é uma referência constante para o valor do nodo a ser inserido.
2. A função getNewNode usa o operador new para criar um novo nodo da lista e retorna um ponteiro para este nodo da lista. Se este ponteiro for diferente de zero, getNewNode retoma um ponteiro para este nodo recém-alocado para newPtr em insertAtFront.
3. Se a lista está vazia, então tanto o firstPtr como o lastPtr são inicializados com newptr.
4. Se a lista não está vazia, então o nodo apontado por newPtr é introduzido na lista copiando firstPtr para newPtr->nextPtr. de forma que o novo nodo aponte para o que era o primeiro nodo da lista, e newPtr é copiado para firstPtr. de forma que firstptr agora aponte para o novo primeiro nodo da lista.
A Fig.l5.5 ilustra a função insertAtFront. A parte a) da figura mostra a lista e o novo nodo antes da operação insertAtFront. As setas pontilhadas na parte b) ilustram os passos 2 e 3 da operação insertAtFront, que possibilitam que o nodo contendo 12 se torne a nova cabeça de lista.
o) firstPtr
L1 __
newPtr
I12N
b) firstPtr
newPtr
Fig.15.5 A operação insertAtFront.
A função insertAtBack (Fig. 15.6) coloca um novo nodo no fim da lista. A função consiste em vários passos:

1. Chama a função getNewNode passando para ela value. o qual é uma referência constante para o valor do nodo a ser inserido.
2. A função getNewNode usa o operador new para criar um novo nodo da lista e retornar um ponteiro para este nodo da lista. Se este ponteiro for diferente de zero. getNewNode retorna um ponteiro para este nodo
recém-alocado para newPtr em insertAtBack.
3. Se a lista está vazia, então tanto firstptr como lastPtr são inicializados com newptr.
4. Se a lista não está vazia, então o nodo apontado por newPtr é introduzido na lista copiando-se newPtr para lastPtr->nextPtr, de forma que o novo nodo seja apontado pelo que era o último nodo da lista,
e copiando-se newPtr para lastPtr, deforma que las tPtr agora aponte para o novo último nodo da

LL

lista.

1
A Fig.15.6 ilustra uma operação insertAtBack. A parte a) da figura mostra a lista e o novo nodo antes da operação. As setas pontilhadas na parte b) ilustram os passos da função insertAtBack que possibilitam que um novo nodo seja acrescentado ao fim de uma lista que não está vazia.
a) firstptr lastPtr newPtr
b) firstptr lastPtr newPtr
__ I11II---5N
Fig. 15.6 Uma representação gráfica da operaçâo insertAtBack.
A função removeFromFront (Fig. 15.7) remove o primeiro nodo da lista e copia o valor do nodo para o parâmetro de referência. A função retorna false se for feita uma tentativa para remover um nodo de uma lista vazia e retorna true se a remoção é bem-sucedida. A função consiste em vários passos:
1. Atribui a tempPtr o endereço para o qual firstPtr aponta. Em algum momento, tempPtr será usado para deletar o nodo que está sendo retirado.
2. Se firstPtr é igual a las tPtr, i.e., se a lista tem somente um elemento antes da tentativa de retirada, então inicializa firstPtr e lastPtr com zero para retirar aquele nodo da lista (deixando a lista
vazia).
3. Se a lista tem mais de um nodo antes da retirada, então deixa las tPtr como está e simplesmente inicializa firstPtr para firstptr->nextPtr, i.e., modifica firstPtr para apontar para o que era o segundo nodo antes da retirada (e é o novo primeiro nodo agora).
4. Depois de todas estas manipulações de ponteiro estarem completas, copia para o parâmetro de referência value o membro data do nodo que está sendo retirado.
5. Agora deleta o nodo apontado por tempptr.

6. Retorna true. indicando retirada bem-sucedida.
A Fig. 15.7 ilustra a função removeFromFront. A parte a) ilustra a lista antes da operação de remoção. A parte h) mostra as manipulações de ponteiro realizadas.
a)

Fig. 15.7 Uma representação gráfica da operação removeFromFront (parte 1 de 2).

CAPÍTULO 15 - ESTRUTURAS DE DADOS 773
b) firstPtr lastPtr
E1
Et
tempPtr
Fig.15.7 Uma representação gráfica da operação removeFromFront (parte 2 de 2).
A função removeFromBack (Fig. 15.8) remove o último nodo da lista e copia o valor do nodo para o parâmetro de referência. A função retorna false se for feita uma tentativa de retirar um nodo de uma lista vazia e retorna true se a retirada é bem-sucedida. A função consiste em vários passos:
1. Atribui a tempPtr o endereço para o qual lastPtr aponta. Em algum momento, tempPtr será usado para deletar o nodo que está sendo retirado.
2. Se firstPtr é igual a lastPtr. i.e., se a lista tem somente um elemento antes da tentativa de retirada, então inicializa firstPtr e lastPtr com zero para retirar aquele nodo da lista (deixando a lista vazia).
3. Se a lista tem mais de um nodo antes da retirada, então atribui a currentPtr o endereço para o qual firstptr aponta.
4. Agora, “percorre a lista” com currentPtr até que ele aponte para o nodo antes do último nodo. Isto é feito com um laço while que fica substituindo currentPtr por currentPtr->nextPtr enquanto currentPtr->nextPtr não é lastptr.
5. Atribui a lastPtr o endereço para o qual currentPtr aponta para retirar o último nodo da lista.
6. Inicializa currentPtr->nextPtr com zero no novo último nodo da lista.
7. Depois de todas as manipulações de ponteiro estarem completas, copia para o parâmetro de referência va].ue o membro data do nodo que está sendo retirado.
8. Agora, deleta o nodo apontado por tempPtr.
9. Retorna true, indicando retirada bem-sucedida.
A Fig. 15.8 ilustra a função removeFromFront. A parte a) da figura ilustra a lista antes da operação de retirada. A parte b) da figura mostra as manipulações de ponteiro realizadas.
a)
b) firstPtr
tempPtr

lastPtr

currentPtr lastPtr
El

Fig. 15.8 Uma representação gráfica da operação removeFroinBack.

A função print primeiro determina se a lista está vazia. Nesse caso, print imprime “A lista está vazia” e termina. Caso contrário, imprime os dados que estão na lista. A função inicializa currentPtr como uma cópia de firstptr e então imprime o string ‘A lista é: “. Enquanto currentPtr não é nulo, currentPtr >dado é impresso e o valor de currentPtr->nextPtr é atribuído a currentPtr. Note que se o link no 4 último nodo da lista não for nulo, o algoritmo de impressão erroneamente imprimirá dados que estão após o fim da
lista. O algoritmo de impressão é idêntico para listas encadeadas, pilhas e filas.
O tipo de lista encadeada que estivemos discutindo é uma lista simplesmente encadeada - a lista começa com um ponteiro para o primeiro nodo e cada nodo contém um ponteiro para o próximo nodo “na seqüência”. Esta lista termina com um nodo cujo membro ponteiro é 0. Uma lista simplesmente encadeada pode ser percorrida somente em uma direção.
Uma lista circular simplesmente encadeada começa com um ponteiro para o primeiro nodo e cada nodo contém um ponteiro para o próximo nodo. O “último nodo” não contém um ponteiro 0; em vez disso, o ponteiro no último nodo aponta de volta para o primeiro nodo, fechando deste modo o “círculo”.
Uma lista duplamente encadeada pode ser percorrida tanto para frente como para trás. Tal lista é freqüentemente implementada com dois “ponteiros de início” - um que aponta para o primeiro elemento da lista para permitir o percurso da lista da frente para trás e um que aponta para o último elemento da lista para permitir o percurso da lista de trás para frente. Cada nodo tem tanto um ponteiro “para a frente”, para o próximo nodo da lista na direção para a frente e um ponteiro “para trás”, para o próximo nodo na lista na direção para trás. Se sua lista contém uma lista telefônica em ordem alfabética, por exemplo, para se procurar alguém cujo nome começa com uma letra próxima ao início do alfabeto, poderíamos começar pela frente da lista. Para se procurar alguém cujo nome começa com uma letra próxima ao fim do alfabeto, poderíamos começar pelo fim da lista.
Em uma lista circular duplamente encadeada, o ponteiro “para a frente” do último nodo aponta para o primeiro nodo e o ponteiro “para trás” do primeiro nodo aponta para o último nodo, fechando deste modo o “círculo”.

15.5 Pilhas

No Capitulo 12, Gabaritos”, explicamos a noção de um gabarito de classe pilha com uma implemenrnção subjacente de array. Nesta seção, usamos uma implementação de lista encadeada baseada em ponteiros subjacentes. Também discutimos pilhas no Capítulo 20, “A biblioteca padrão de gabaritos (STL).”
Uma pilha é uma versão sujeita a restrições de uma lista encadeada - novos nodos podem ser acrescentados e removidos de uma pilha somente no topo.

Por essa razão, uma pilha é chamada de uma estrutura de dados último a entrar primeiro a sair (LIFO - last in,first out). O membro de encadeamento no último nodo da pilha é inicializado com nulo (zero) para indicar a parte inferior da pilha.
Erro comum de programação 15.7
Não inicializar o link no nodo da parte inferior de uma pilha como nulo (zero).
As funções membro primárias usadas para manipular uma pilha são push e pop. A função push insere um novo nodo no topo da pilha. A função pop retira um nodo do topo da pilha, armazena o valor desempilhado em uma variável de referência que é passada para a função que chamou e retorna true se a operação pop foi bem-sucedida (false caso contrário).
As pilhas têm muitas aplicações interessantes. Por exemplo, quando é feita uma chamada de função, a função chamada deve saber como retornar à função que chamou, assim o endereço de retorno é empurrado para o topo de uma pilha. Se acontece uma série de chamadas de funções, os valores sucessivos de retorno são empurrados para a pilha na ordem último a entrar primeiro a sair, de forma que cada função pode retornar a seu chamador. Pilhas suportam as chamadas recursivas de funções da mesma maneira que chamadas convencionais não-recursivas.
As pilhas contêm o espaço criado para as variáveis automáticas em cada invocação de uma função. Quando a função retorna a seu chamador ou dispara uma exceção, é chamado o destruidor (se houver) para cada objeto local, o espaço para aquelas variáveis automáticas da função é retirado (desempilhado) da pilha e aquelas variáveis não são mais conhecidas no programa.
As pilhas são usadas por compiladores no processo de avaliar expressões e gerar código em linguagem de máquina. Os exercícios exploram vários aplicações de pilhas, inclusive usando-as para desenvolver um compilador completo que funciona.

Aproveitar-nos-emos da relação íntima entre listas e pilhas para implementar uma classe pilha reutilizando principalmente uma classe lista. Usamos duas formas diferentes de reutilização. Primeiro, implementamos a classe pilha através de herança privada da classe lista. Então implementamos uma classe pilha que executa de forma idêntica através do uso da composição, incluindo uma classe lista como um membro privado de uma classe pilha. Naturalmente, todas as estruturas de dados neste capítulo, inclusive estas duas classes pilha, são implementadas como gabaritos (veja Capítulo 12, “Gabaritos”) para encorajar uma reutilização adicional.
O programa da Fig. 15.9 (cuja saída é mostrada na Fig. 15. 10) cria um gabarito de classe Stack principalmente através de herança private do gabarito de classe List da Fig. 15.3. Queremos que Stack tenha as funções membro push, pop. isStackEinpty e printStack. Note que estas funções são essencialmente as funções insertAtFront, removeFromFront. isEmpty e print do gabarito de classe List. Naturalmente, o gabarito de classe List contém outras funções membro (i.e., insertAtBack e removeFromBack) que não queremos tornar acessíveis através da interface public para a classe Stack. Assim, quando indicarmos que o

gabarito de classe Stack deve herdar do gabarito de classe List, especificamos herança private. Isto torna private todas as funções membro do gabarito de classe List no gabarito de classe Stack. Quando implementarmos as funções membro de Stack, então faremos cada uma destas chamar a função membro apropriada da classe List - push chama insertAtFront. pop chama removeFromFront. isStackEmpty chama isEmpty e printStack chama print.

```
1 Il Fig. 15.9: stack.h
2 Il Definição do gabarito de classe Stack
3 Il Derivado da classe List
4 #ifndef STACKH
5 #define STACKH
6
7 #include “list.h”
8
9 template< class STACKTYPE >
10 class Stack : private List< STACKTYPE >
11 public:
12 void push( const STACKTYPE &d ) { insertAtFront( d );
13 bool pop( STACKTYPE &d ) { return removeFromFront( d );
14 bool isStackEmpty() const { return isEmptyO;
15 void printStack() const { printO;
16
17
18 #endif
```

Fig. 15.9 Um programa simples de pilha- stack.h.

```
19 Il Fig. 15.9: figl5O9.cpp
20 Il Programa para testar a classe gabarito Stack
21 #include <iostream>
22 #include “stack.h”
23
24 using std::endl;
25
26 int main()
27 {
28 Stack< int > intStack;
29 int poplnteger, i;
30 cout « ‘processando um Stack integer” « endl;
31
32 for(i=Q;i<4;i+f){
```

Fig. 15.9 Um programa simples de pilha - figls 09. cpp (parte 1 de 2).


```
33 intStack.push( i );
34 intStack.printStack
35
36
```

```
37 while ( !intStack.isStackEmpty()
38 intStack.pop( poplnteger );
39 cout « poplnteger « “ retirado da pilha” « endi;
40 intStack.printStackQ;
41
42
43 Stack< double > doubleStack;
44 double vai = 1.1, popdouble;
45 cout « ‘processando um Stack double” « endi;
47 for(i0;i<4;i++){
48 doubleStack.push( vai );
49 doubleStack.printStack
50 vai += 1.1;
51
52
53 while ( !doubleStack.isStackEmpty()
54 doubleStack.pop( popdouble );
55 cout « popdouble « “ retirado da pilha” « endl;
56 doubleStack.printStack
57
58 return 0;
59
```

Fig. 15.9 Um programa simples de pilha - figl5_09. cpp (parte 2 de 2).

```
processando um Stack integer
A lista é: O
A lista é: 1 O
A lista é: 2 1 O
A lista é: 3 2 1 O
3 retirado da pilha
A lista é: 2 1 O
2 retirado da pilha
A lista é: 1 O
1 retirado da pilha
A lista é: O
o retirado da pilha
A lista está vazia
processando um Stack double
A lista é: 1.1
A lista é: 2.2 1.1
```

Fig. 15.10 Exemplo de saída do programa da Fig. 15.9 (parte 1 de 2).


```
A lista é: 3.3 2.2 1.1
A lista é: 4.4 3.3 2.2 1.1
4.4 retirado da pilha
A lista é: 3.3 2.2 1.1
```

```
3.3 retirado da pilha
A lista é: 2.2 1.1
2.2 retirado da pilha
A lista é: 1.1
1.1 retirado da pilha
A lista está vazia
Todos os nodos destruídos
Todos os nodos destruidos
```

Fig. 15.10 Exemplo de saída do programa da Fig. 15.9 (parte 2 de 2).

O gabarito da classe pilha é usado em main para instanciar a pilha de inteiros intStack do tipo Stack< int >. Inteiros de O a 3 são inseridos em intStack e então retirados de intStack. O gabarito de classe pilha é então usado para instanciar a pilha doubleStack do tipo Stack< double >. Os valores 1.1, 2.2, 3.3 e 4.4 são inseridos em doubleStack e então retirados de doubleStack.

Outro caminho para implementar um gabarito de classe Stack é reutilizar um gabarito da classe List através de composição. O programa da Fig. 15.11 usa os arquivos list . h e listnd. h do programa de List. Também usa o mesmo programa de teste que o programa de pilha anterior, exceto que o novo arquivo de cabeçalho

- stack_c . h - é incluído e substitui stack . h. A saída também é a mesma. A definição do gabarito da classe Stack agora inclui o objeto membro s, do tipo List< STACKTYPE>.

```
1 II Fig. 15.11: stack_c.h
2 II Definição da classe Stack composta com o objeto List
3 #ifndef STACKC
4 #define STACKC
5 #include “list.h’
6
7 template< class STACKTYPE >
8 class Stack
9 public:
10 // sem construtor; o construtor de List faz a inicialização
11 void push( const STACKTYPE &d ) { s.insertAtFront( d );
12 bool pop( STACKTYPE &d ) { return s.removeFromFront( d );
13 bool isStackEmpty() const { return s.isEmptyO;
14 void printStack() const ( s.print }
15 private:
16 List< STACKTYPE > s;
17 };
18
19 #endif
```

Fig. 15.11 Um programa simples com pilha usando composição - stackc.h.

15.6 Filas

Uma,fila é semelhante a uma fila no caixa de um supermercado - a primeira pessoa na fila é atendida primeiro e os outros clientes entram no fim da fila e esperam para ser atendidos. Os nodos da fila são retirados somente do início da fila e são inseridos somente no fim da fila. Por essa razão, uma fila é chamada de estrutura de dados primeiro a entrar primeiro a sair (FIFO, first in, flrst out). As operações de inserção e retirada são conhecidas como enqueue e dequeue respectivamente.
As filas têm muitas aplicações em sistemas de computadores. A maioria dos computadores têm somente um único processador, assim somente um usuário pode ser atendido de cada vez. Os pedidos dos outros usuários são colocadas em uma fila. Cada pedido avança gradualmente para a frente da fila à medida que os usuários são atendidos. O pedido no início da fila é a próximo a ser atendido.
As filas são também usadas para suportar a impressão em spool. Um ambiente multiusuário pode ter somente uma única impressora. Muitos usuários podem estar gerando saídas para serem impressas. Se a impressora estiver ocupada, outras saídas podem ainda ser geradas. Estas são postas em um spool no disco (da mesma forma que uma linha é colocada em um carretel), onde eles esperam em uma fila até a impressora se tornar disponível.
Pacotes de informações também esperam em filas em redes de computador. Toda vez que um pacote chega em um nodo de rede, ele deve ser direcionado para o próximo nodo na rede ao longo do caminho para o destino final do pacote. O nodo de direcionamento encaminha um pacote de cada vez, assim pacotes adicionais são enfileirados até que o direcionador possa encaminhá-los.

Um servidor de arquivos em uma de rede de computador trata solicitações de acesso a arquivos de muitos clientes na rede. Os servidores têm uma capacidade limitada para tratar solicitações de clientes. Quando essa capacidade é excedida, as solicitações de clientes esperam em filas.
Erro comum de programação 15.8
Não inicializar o link no último do último nodo de uma fila com nulo (zero).
A Fig. 15.12 (cuja saída é mostrada na Fig. 15.13) cria um gabarito de classe Queue principalmente através do uso de herança private do gabarito de classe List da Fig 15.3. Queremos que Queue tenha as funções membro enqueue. dequeue. isQueueEmpty e printQueue. Percebemos que estas são essencialmente as funções do gabarito de classe List: insertAtBack, removeFromFront. isEmpty e print. Naturalmente, o gabarito de classe List contém outras funções membro (i.e., insertAtFront e removeFromBack) que não queremos tornar acessíveis através da interface public da classe Queue. Assim, quando indicamos que o gabarito de classe Queue deve herdar do gabarito de classe List, especificamos herança private. Isto torna private todas as funções membro do gabarito de classe List no gabarito de classe Queue. Quando implementamos as
j funções membro de Queue, simplesmente fazemos cada uma destas chamar a função membro apropriada da classe lista - enqueue chama insertAtBack. dequeue chama rexnoveFromFront, isQueueEmpty chama isEmpty e printQueue

chama print.

Fig. 15.12 Processando uma fila - queue . h (parte 1 de 2).

1	II Fig. 15.12: queue.h		
2	// Definição do gabarito de	classe Queue	
3	// Derivado da classe List		
4	#ifndef QUEUEH		
5	#define QUEUEH		
6			
7	#include “list.h”		
8			
9	template< class QUEUETYPE >		
10	class Queue: private List<	QUEUETYPE >	{
11	public:		
12	void enqueue(const QUEU	ETYPE &d)	{ insertAtBack(d);
13	bool dequeue(QtJEUETYPE	&d	

14 { return removeFromFront(d
15 bool isQueueEmpty() const { return isEmptyO;
16 void printQueue() const { printO;
17
18
19 #endif
Fig. 15.12 Processando uma fila - queue .h (parte 2 de 2).
20 II Fig. 15.12: figl5l2.cpp
21 II Programa de teste para a classe gabarito Queue
22 #include <iostream>
23 #inciude “queue.h”
24
25 using std: :endi;
26
27 int main()
28 {
29 Queue< int > intQueue;
30 int dequeueInteger, i;
31 cout « “processando uma fila mnteger” « endl;

```
32
33 for(i=0;i<4;j++){
34 intQueue.enqueue( i );
35 intQueue.printQueueO;
36
37
38 while ( !intQueue.isQueueEmpty()
39 intQueue.dequeue( dequeueInteger );
40 cout « dequeueInteger « retirado da fila « endl;
41 intQueue.prmntQueue
42
43
44 Queue< double > doubleQueue;
45 double val = 1.1, dequeuedouble;
46
47 cout « processando uma fila double" « endl;
48
49 for ( i = 0; i < 4; i++
50 doubleQueue.enqueue( vai );
51 doubleQueue.printQueueQ;
52 vai += 1.1;
53
54
55 while ( !doubleQueue.isQueueEmpty() )
56 doubleQueue . dequeue ( dequeuedouble );
57 cout « dequeuedouble « " retirado da fila" « endl;
58 doubleQueue.printQueueO.;
59
60
61 return 0;
62 }
```

Fig. 15.12 Processando uma fila - figl5_12 . cpp.


```
processando uma fila integer
A lista é: O
A lista é: O 1
A lista é: O 1 2
A lista é: O 1 2 3
O retirado da fila
A lista é: 1 2 3
1 retirado da fila
A lista é: 2 3
2 retirado da fila
A lista é: 3
3 retirado da fila
```

A lista está vazia
processando uma fila double
A lista é: 1.1
A lista é: 1.1 2.2
A lista é: 1.1 2.2 3.3
A lista é: 1.1 2.2 3.3 4.4
1.1 retirado da fila
A lista é: 2.2 3.3 4.4
2.2 retirado da fila
A lista é: 3.3 4.4
3.3 retirado da fila
A lista é: 4.4
4.4 retirado da fila
A lista está vazia
Todos os nodos destruidos
Todos os nodos destruídos
Fig. 15.13 Exemplo de saída do programa da Fig. 15.12.

O gabarito de classe de fila é usado em main para instanciar a fila de inteiros intQueue do tipo Queue<int>.

Os inteiros O a 3 são inseridos em intQueue e então retirados de intQueue na ordem primeiro a entrar, primeiro
a sair. O gabarito de classe de fila é então usado para instanciar a fila doubleQueue do tipo Queue<double>.

Os valores 1.1, 2.2, 3.3 e 4.4 são inseridos em doubleQueue e então retirados de doubleQueue na ordem
primeiro a entrar, primeiro a sair.

15.7 Árvores

Listas encadeadas, pilhas e filas são estruturas de dados lineares. Uma árvore é uma estrutura de dados não-linear, bidimensional, com propriedades especiais. Os nodos de uma árvore contêm dois ou mais links. Esta seção discute árvores binárias (Fig. 15.14) - árvores cujos nodos contêm dois links (dos quais nenhum, um ou ambos podem ser nulos). O nodo raiz é o primeiro nodo em uma árvore. Cada link no nodo raiz referencia um filho. O filho esquerdo é o nodo raiz da subárvore esquerda e ofilho direito é o nodo raiz da subárvore direita. Os filhos de um mesmo nodo são chamados de irmãos. Um nodo sem filhos é chamado de nodo folha. Os cientistas de computação normalmente desenham árvores do nodo raiz para baixo - exatamente o oposto das árvores na natureza.

Nesta seção, é criada uma árvore binária especial chamada de árvore de pesquisa binária. Uma árvore de pesquisa binária (sem valores de nodo duplicados) tem a propriedade de que os valores em qualquer subárvore esquerda são menores do que o valor em seu nodo pai e os valores em qualquer subárvore direita são maiores do que o valor em seu nodo pai. A Fig.l5.15 ilustra uma árvore de pesquisa binária com 12 valores. Note que a forma da árvore de pesquisa binária que corresponde a um conjunto de dados

pode variar, dependendo da ordem em que os valores são inseridos na árvore.
Erro com um de programação 15.9
Não inicializar com nu/o (zero) os links dos nodos folhas de uma árvore.
Fig. 15.14 Uma representação gráfica de uma árvore binária.
(7{4N8
Fig. 15.15 Uma árvore de pesquisa binária.
O programa da Fig. 15.16 (cuja saída é mostrada na Fig. 15.17) cria uma árvore de pesquisa binária e a percorre (i.e., caminha através de todos os seus nodos) de três modos - usando percursos recursivos em ordem, em pré-ordem e em pós-ordem.


```
1 II Fig. 15.16: treenode.h
2 II Definição da classe TreeNode
3 #ifndef T1EENODEH
4 #define TREENODEH
5
6 template< class NODETYPE > class Tree; II declaração antecipada
7
8 template< class NODETYPE >
9 class TreeNode
10 friend class Tree< NODETYPE >;
11 public:
12 TreeNode( const NODETYPE &d
13 : leftPtr( O ), data( d ), rightPtr( O ) {
14 NODETYPE getData() const { return data;
15 private:
16 TreeNode< NQDETYPE > *leftptr; // ponteiro para subárvore esquerda
17 NQDETYPE data;
18 TreeNode< NODETYPE > *rightptr; II ponteiro para subárvore direita
19
20
21 #endif
```

Fig. 15.16 Criando e percorrendo uma árvore binária - treenode . h

```
22 II Fig. 15.16: tree.h
23 II Definição da classe gabarito Tree
24 #ifndef TREEH
25 #define TREEH
26
27 #include <iostream>
28 #include <cassert>
29 #include “treenode.h”
30
31 using std: :endl;
32
33 template< class NODETYPE >
```

```
34 class Tree {
35 public:
36 TreeO;
37 void insertNode( const NODETYPE & );
38 void preOrderTraversal() const;
39 void inOrderTraversal() const;
40 void postOrderTraversal() const;
41 private:
42 TreeNode< NODETYPE > *rootptr;
43
44 II funções utilitárias
45 void insertNodeHelper(
46 TreeNode< NODETYPE > const NODETYPE & );
47 void preOrderHelper( TreeNode< NODETYPE > * ) const;
48 void inOrderHelper( TreeNode< NODETYPE > * ) const;
49 void postOrderHelper( TreeNode< NODETYPE > * ) const;
50
51
52 template< class NODETYPE >
53 Tree< NODETYPE >::Tree() { rootPtr = 0; }
```

Fig. 15.16 Criando e percorrendo uma árvore binária - tree .h (parte 1 de 3).


```
54
55 template< class NODETYPE >
56 void Tree< NODETYPE >: :insertNode( const NODETYPE &value
57 { insertNodeHelper( &rootPtr, value );
58
59 II Esta função recebe um ponteiro para um ponteiro,
60 II de modo que o ponteiro pode ser modificado.
61 template< class NODETYPE >
62 void Tree< NODETYPE >: :insertNodeHelper(
63 TreeNode< NODETYPE > **ptr, const NODETYPE &value
64
65 if ( *ptr O ) { II árvore está vazia
66 *ptr = new TreeNode< NODETYPE >( value );
67 assert( *ptr != O );
68
69 else II árvore não está vazia
70 if ( value < ( *ptr )->data
71 insertNodeHelper( &( ( *ptr )->leftPtr ), value );
72 else
73 if ( value > ( *p )->data
74 insertNodeHelper( &( ( *ptr )->rightPtr ), value );
75 else
76 cout « value « duplo” « endi;
```

```
77 }
78
79 template< class NODETYPE >
80 void Tree< NODETYPE >::preOrderTraversal() const
81 { preOrderHelper( rootPtr );
82
83 template< class NODETYPE >
84 void Tree< NODETYPE >: :preOrderHelper(
85 TreeNode< NODETYPE > *ptr ) const
86
87 if ( ptr != O
88 cout « ptr->data «
89 preOrderHelper( ptr->leftPtr );
90 preOrderHelper( ptr->rightPtr );
91 )
92 }
93
94 template< class NODETYPE >
95 void Tree< NODETYPE >::inOrderTraversal() const
96 { inOrderHelper( rootPtr );
97
98 template< class NODETYPE >
99 void Tree< NODETYPE >::inOrderHelper(
100 TreeNode< NODETYPE > *ptr ) const
101 {
102 if ( ptr != O ) {
103 inOrderHelper ( ptr->leftPtr );
104 cout « ptr->data «
105 inOrderHelper( ptr->rightPtr );
106 }
107 )
108
109 template< class NODETYPE >
110 void Tree< NODETYPE >: :postOrderTraversal() const
111 { postOrderHelper( rootPtr ); )
```

Fig. 15.16 Criando e percorrendo uma árvore binária - tree . h (parte 2 de 3).


```
112
113 template< class NODETYPE >
114 void Tree< NODETYPE >: :postOrderHelper(
115 TreeNode< NODETYPE > *p ) const
116
117 if ( ptr ! O
118 postOrderHelper( ptr->leftPtr );
119 postOrderHelper( ptr->rightPtr );
```

```
120 cout « ptr->data «
121
122 }
123
124 #endif
```

Fig. 15.16 Criando e percorrendo uma árvore binária - tree .h (parte 3 de 3).

```
125 II Fig. 15.16: figl5l6.cpp
126 II Programa para testar a classe Tree
127 #include <iostreain>
128 #include <iomanip>
129 #include “tree.h”
130
131 using std::cout;
132 using std::cin;
133 using std::setiosflags;
134 using std::ios;
135 using std: :setprecision;
136
137 int main()
138 {
139 Tree< int > intTree;
140 int intVal, i;
141
142 cout « “Digite 10 valores inteiros:\n”;
143 for ( i = 0; i < 10; i++
144 cm » intVal;
145 intTree.insertNode( intVal );
146 }
147
148 cout « “\nPercorrendo na pré-ordem\n;
149 intTree.preOrderTraversalO;
150
151 cout « “\nPercorrendo em ordem\n”;
152 intTree.inOrderTraversa1
153
154 cout « “\nPercorrendo na pós-ordem\n;
155 intTree.postOrderTraversal();
156
157 Tree< double > doubleTree;
158 double doubleVal;
159
160 cout « “\n\n\nDigite 10 valores double:\n
161 « setiosflags( ios::fixed 1 ios::showpoint
162 « setprecision( 1 );
163 for ( i = O; i < 10; i++ )
164 cm » doubleVal;
```

Fig. 15.16 Criando e percorrendo uma árvore binária - figl5_16 . cpp (parte 1 de 2).


```
165 doubleTree.insertNode( doubleVal );
166
167
168 cout « "\nPercorrendo na pré-ordem\n";
169 doub1eTree.preOrderTraversa1
170
171 cout « \nPercorrendo em ordem\n";
172 doubleTree.inOrderTraversalO;
173
174 cout « "\nPercorrendo na pos-ordem\n';
175 doubleTree . postOrderTraversal O;
176
177 return 0;
178 }
```

Fig. 15.16 Criando e percorrendo uma árvore binária - figl5_16 . cpp (parte 2 de 2).

A função main começa por instanciar a árvore de inteiros intTree do tipo Tree<int>. O programa solicita ao usuário 10 inteiros, cada um dos quais é inserido na árvore binária através de uma chamada a insertnNode. O programa então percorre a árvore inttree na pré-ordem, em ordem e na pós-ordem (que serão explicadas em breve), O programa então instancia a árvore de ponto flutuante doubleTree do tipo Tree<double>. O programa solicita ao usuário 10 valores double, cada um dos quais é inserido na árvore binária através de uma chamada a insertNode. O programa então percorre a árvore doubleTree na pré-ordem, em ordem e na pós-ordem.

```
Digite 10 valores inteiros:
50 25 75 12 33 67 88 6 13 68
Percorrendo na pré-ordem
50 25 12 6 13 33 75 67 68 88
Percorrendo na ordem
6 12 13 25 33 50 67 68 75 88
Percorrendo na pós-ordem
6 13 12 33 25 68 67 88 75 50
Digite 10 valores double:
39.2 16.5 82.7 3.3 65.2 90.8 1.1 4.4 89.5 92.5
Percorrendo na pré-ordem
39.2 16.5 3.3 1.1 4.4 82.7 65.2 90.8 89.5 92.5
Percorrendo na ordem
1.1 3.3 4.4 16.5 39.2 65.2 82.7 89.5 90.8 92.5
Percorrendo na pós-ordem
1.1 4.4 3.3 16.5 65.2 89.5 92.5 90.8 82.7 39.2
```

Fig. 15.17 Exemplo de saída do programa da Fig. 15.16.
Agora, discutiremos as definições do gabarito de classe. Começamos com o gabarito de classe TreeNode que declara como friend o gabarito de classe Tree. A classe TreeNode tem como dado private o valor dados do nodo e os ponteiros leftPtr (para a subárvore esquerda do nodo) e rightPtr (para a subárvore direita do nodo). O construtor inicializa data com o valor fornecido como um parâmetro do construtor e inicializa os ponteiros leftPtr e rightPtr com zero (inicializando deste modo este nodo como um nodo folha). A função membro getData retorna o valor de data.

A classe Tree tem como dado private rootptr. um ponteiro para o nodo raiz da árvore. A classe tem as funções membro públicas insertNode (que insere um novo nodo na árvore) e preOrderTraversal, inOrderTraversal e pos tOrderTraversal, cada uma percorrendo a árvore da maneira designada. Cada uma destas funções membro chama sua própria função utilitária recursiva separada para executar as operações apropriadas sobre a representação interna da árvore, O construtor de Tree inicializa rootPtr com zero para indicar que a árvore está inicialmente vazia.
A função utilitária insertNodeHelper da classe Tree insere recursivamente um nodo na árvore. Um nodo em uma árvore de pesquisa binária somente pode ser inserido como um nodo folha. Se a árvore está vazia, um novo TreeNode é criado, inicializado e inserido na árvore.
Se a árvore não está vazia, o programa compara o valor a ser inserido com o valor de data no nodo raiz. Se o valor a inserir é menor, o programa chama recursivamente insertNodeHelper para inserir o valor na sub-
árvore esquerda. Se o valor a inserir é maior, o programa recursivamente chama insertNodeHelper para inserir o valor na sub-árvore direita. Se o valor a ser inserido é idêntico ao valor de dados no nodo raiz, o programa imprime a mensagem ‘duplo e retorna sem inserir o valor duplicado na árvore.
Cada uma das funções membro inOrderTraversal, preOrderTraversal e postOrderTraversal percorre a árvore (Fig. 15.18) e imprime os valores dos nodos.
Os passos para um inOrderTraversal são:
1. Percorra a subárvore esquerda com um inOrderTraversal.
2. Processe o valor no nodo (i.e., imprima o valor do nodo).
3. Percorra a subárvore direita com um inOrderTraversal.
O valor em um nodo não é processado até os valores em sua subárvore esquerda serem processados. O inOrderTraversal da árvore na Fig. 15.18 é:
6 13 17 27 33 42 48
‘1 Note que o inOrderTraversal de uma árvore de pesquisa binária imprime os valores dos nodos em ordem ascendente. O processo de criar uma árvore de pesquisa binária realmente classifica os dados - e deste modo este processo é chamado de classificação da árvore binária.
Os passos para um preOrderTraversal são:
1. Processe o valor no nodo.

2. Percorra a subárvore esquerda com um preOrderTraversal.
3. Percorra a subárvore direita com um preOrderTraversal.
O valor em cada nodo é processado à medida que o nodo é visitado. Depois do valor em um dado nodo ser processado, os valores na subárvore esquerda são processados e, então, os valores na subárvore direita são processados. O preørderTraversal da árvore na Fig. 15.18 é:
27 13 6 17 42 33 48
Os passos para um postOrderTraversal são:
1. Percorra a subárvore esquerda com um pos tOrderTraversal.
2. Percorra a subárvore direita com um postOrderTraversal.
3. Processe o valor no nodo.
O valor em cada nodo não é impresso até os valores de seus filhos serem impressos. O pos tOrderTraversal da árvore na Fig. 15.18 é:
6 17 13 33 48 42 27
A árvore de pesquisa binária facilita a eliminação de duplicatas. À medida que a árvore vai sendo criada, uma tentativa de inserção de um valor duplicado será reconhecida porque uma duplicata seguirá as mesmas decisões “vá

para a esquerda” ou “vá para a direita” em cada comparação que o valor original seguiu. Deste modo, em algum momento, a duplicata será comparada com um nodo contendo o mesmo valor. Neste momento, o valor duplicado pode simplesmente ser descartado.
27
13 42
6 17 33 48
Fig. 15.18 Uma árvore de pesquisa binária.
Pesquisar uma árvore binária para encontrar um valor que corresponde ao valor de uma chave também é rápido. Se a árvore está balanceada, então cada nível contém aproximadamente duas vezes o número de elementos do nível anterior. Assim, uma árvore de pesquisa binária com n elementos teria um máximo de log2n níveis e, deste modo, um máximo de log2n comparações teria que ser feito ou para achar uma correspondência ou para determinar que não existe nenhuma correspondência. Isto significa, por exemplo, que, quando pesquisarmos uma árvore de pesquisa binária (balanceada) de 1.000 elementos, não mais do que 10 comparações precisam ser feitas, porque 210> 1.000. Quando pesquisarmos uma árvore de pesquisa binária (balanceada) de 1.000.000 elementos, não mais do que 20 comparações precisam ser feitas, porque 220> 1.000.000.
Nos exercícios, são apresentados algoritmos para várias outras operações com árvores binárias, tais como eliminar um item de uma árvore binária, imprimir uma árvore binária em um formato de árvore bidimensional e percorrer uma árvore binária em ordem de nível. O percurso em ordem de nível de uma árvore binária visita os nodos da árvore linha por linha, começando no nível do nodo raiz. Em cada nível da árvore, os nodos são visitados da esquerda para a direita. Outros

exercícios de árvores binárias incluem permitir que uma árvore de pesquisa binária contenha valores duplicados, inserir valores de strings em uma árvore binária e determinar quantos níveis estão contidos em uma árvore binária.

Resumo

• Classes auto-referentes contêm membros chamados de links que apontam para objetos do mesmo tipo da classe.
• Classes auto-referentes possibilitam que muitos objetos da classe sejam encadeados em pilhas, filas, listas e árvores.
• A alocação dinâmica de memória reserva um bloco de bytes na memória para armazenar um objeto durante a execução do programa.
• Uma lista encadeada é uma coleção linear de objetos de uma classe auto-referente.
• Uma lista encadeada é uma estrutura de dados dinâmica - o comprimento da lista pode aumentar ou diminuir conforme necessário.
• As listas encadeadas podem continuar a crescer até a memória se esgotar.
• As listas encadeadas fornecem um mecanismo para inserção e exclusão de dados através da manipulação de ponteiros.
• Uma lista simplesmente encadeada começa com um ponteiro para o primeiro nodo e cada nodo contém um ponteiro para o próximo nodo “na seqtiência”. Esta lista termina com um nodo cujo membro ponteiro é 0. Uma lista simplesmente encadeada pode ser percorrida somente em uma direção.
• Uma lista circular simplesmente encadeada começa com um ponteiro para o primeiro nodo e cada nodo contém um ponteiro para o próximo nodo. O ponteiro no último nodo aponta para o primeiro nodo, fechando deste modo o círculo.
• Uma lista duplamente encadeada permite percursos tanto para a frente como para trás. Cada nodo tem um “ponteiro para a frente” para o próximo nodo na lista na direção para a frente e um “ponteiro para trás” para o próximo nodo da lista na direção para trás.

Em uma lista circular duplamente encadeada, o “ponteiro para a frente” do último nodo aponta para o primeiro nodo e o “ponteiro para trás” do primeiro nodo aponta para o último nodo, fechando deste modo o “cfrculo”.
• As pilhas e filas são versões de listas encadeadas com restrições.
• Novos nodos de uma pilha são inseridos e retirados da mesma pilha somente no topo da pilha. Por essa razão, uma pilha é chamada de uma estrutura de dados último a entrar, primeiro a sair (LIFO, last in, firsr our).
• O membro link no último nodo da pilha é inicializado com nulo (zero) para indicar a base da pilha.
• As duas operações primárias usadas para manipular uma pilha são push e pop. A operação push cria um novo nodo e o coloca sobre o topo da pilha. A operação pop remove um nodo do topo da pilha, libera a memória que foi alocada para aquele nodo e retorna o valor retirado.
• Em uma estrutura de dados de fila, nodos são removidos do início e inseridos no fim. Por essa razão, uma fila é chamada de uma estrutura de dados primeiro a entrar, primeiro a sair (FIFO, first in, first out). As operações de inserção e

retirada são conhecidas como enqueue e dequeue.
• As árvores são estruturas de dados de duas dimensões exigindo dois ou mais links por nodo.
• As árvores binárias contêm dois links por nodo.
• O nodo raiz é o primeiro nodo da árvore.
• Cada um dos ponteiros no nodo raiz se refere a um filho. O filho esquerdo é o primeiro nodo na subárvore esquerda e o filho direito é o primeiro nodo na subárvore direita. Os filhos de um mesmo nodo são chamados de irmãos. Qualquer nodo da árvore que não tem nenhum filho é chamado de nodo folha.
• Uma árvore de pesquisa binária tem a característica que o valor no filho esquerdo de um nodo é menor que o valor em seu nodo pai, e o valor no filho direito de um nodo é maior do que ou igual ao valor em seu nodo pai. Se não existirem valores de dados duplicados, o valor no filho direito é simplesmente maior que o valor em seu nodo pai.
• Um percurso em ordem de uma árvore binária percorre a subárvore esquerda em ordem, processa o valor no nodo raiz e, então, percorre a subárvore direita em ordem. O valor em um nodo não é processado até que os valores em sua subárvore esquerda sejam processados.
• Um percurso em pré-ordem processa o valor no nodo raiz, percorre a subárvore esquerda em pré-ordem e, então, percorre a subárvore direita em pré-ordem. O valor em cada nodo é processado à medida que o nodo é encontrado.
• Um percurso em pós-ordem percorre a subárvore esquerda em pós-ordem, percorre a subárvore direita em pós-ordem e, então, processa o valor no nodo raiz. O valor em cada nodo não é processado até que os valores nas duas subárvores sejam processados.

Terminologia

alocação dinâmica de memória fim de uma fila
árvore função predicado
árvore binária indireção dupla
árvore de pesquisa binária início de uma fila
classificação de árvore binária inserindo um nodo
deletar um nodo irmãos
dequeue LIFO (último a entrar, primeiro a sair)
eliminação de duplicatas lista circular duplamente encadeada
enqueue lista circular simplesmente encadeada
estrutura auto-referente lista duplamente encadeada
estrutura de dados linear lista encadeada
estrutura de dados não-linear lista simplesmente encadeada
estruturas de dados dinâmicas nodo
FIFO (primeiro a entrar, primeiro a sair) nodo filho
fila nodo folha
filho direito nodo pai
filho esquerdo nodo raiz
filhos percurso

percurso de uma árvore binária em ordem push
percurso de uma árvore binária em pós-ordem sizeof
percurso de uma árvore binária em pré-ordem subárvore
percurso em ordem de nível de uma árvore binária subárvore direita
pilha subárvore esquerda
ponteiro nulo topo
ponteiro para um ponteiro visitar um nodo
poP
Erros comuns de programação
15.1 Não inicializar o link no último nodo de uma lista com nulo (O).
15.2 Assumir que o tamanho de um objeto de uma classe é simplesmente a soma dos tamanhos de seus membros de dados.
15.3 Não liberar a memória alocada dinamicamente quando ela não é mais necessária pode fazer com que o sistema fique prematuramente sem memória. Isto é às vezes chamado de “perda de memória.”
15.4 Apagar com delete memória que não foi alocada dinamicamente com new.
15.5 Referenciar memória que foi deletada.
15.6 Tentar deletar memória que já foi deletada pode levar a resultados impossíveis de predizer, durante a execução.
15.7 Não inicializar o link no nodo da parte inferior de uma pilha como nulo (zero).
15.8 Não inicializar o link no último do último nodo de uma fila com nulo (zero).
15.9 Não inicializar com nulo (zero) os links dos nodos folha de uma árvore.
Boas práticas de programação
15.1 Quando a memória que foi alocada dinamicamente com new não é mais necessária, use delete para liberar a memória imediatamente para o sistema.
15.2 Atribua nulo (zero) ao membro de encadeamento (link) de um novo nodo. Os ponteiros devem ser inicializados antes de serem usados.
Dicas de desempenho
15.1 Um array pode ser declarado para conter mais elementos que o número de itens esperados, mas isto pode desperdiçar memória. As listas encadeadas podem fornecer uma utilização de memória melhor nestas situações. As listas encadeadas
permitem ao programa se adaptar durante a execução.
15.2 A inserção e exclusão em um array ordenado pode consumir tempo - todos os elementos em seguida ao elemento insendo ou excluído devem ser movidos de forma apropriada.
15.3 Os elementos de um array são armazenados contiguamente na memória. Isto permite o acesso imediato a qualquer elemento do array porque o endereço de qualquer elemento pode ser calculado diretamente com base na posição de início
do array. As listas encadeadas não dispõem de tal “acesso direto” imediato a seus elementos.
15.4 Usar alocação de memória dinâmica (em vez de arrays) para estruturas de dados que crescem e encolhem durante a execução pode economizar memória. Tenha em mente, porém. que ponteiros ocupam espaço e que a alocação de

memória dinâmica incorre no overhead das chamadas de funções.
Dica de portabilidade
15.1 O tamanho de um objeto de uma classe não é necessariamente a soma dos tamanhos de seus membros de dados. Isto ocorre por causa de vários requisitos de alinhamento delimites de endereços dependentes de máquina (ver o Capítulo 16)
e por outras razões. Use o operador sizeof para determinar o tamanho de um objeto.
Exercícios de auto-revisão
15.1 Preencha os espaços em branco em cada um dos seguintes itens:
a) Uma classe auto- _____ é usada para formar estruturas de dados dinâmica que podem crescer e encolher durante a execução.
b) O operador é usado para alocar memória dinamicamente e construir um objeto; este operador retorna
um ponteiro para a memória alocada.

c) Uma _____ é uma versão com restrições de uma lista encadeada, na qual os nodos podem ser inseridos e apagados somente no início da lista, e os valores dos nodos são retirados na ordem último a entrar, primeiro a sair.
d) Uma função que não altera uma lista encadeada, mas simplesmente examina a lista para determinar se ela está vazia é chamada de uma função
e) Uma fila é chamada de uma estrutura de dados porque os primeiros nodos inseridos são os primeiros
nodos removidos.
f) O ponteiro para o próximo nodo em uma lista encadeada é chamado de _____
g) O operador é usado para destruir um objeto e recuperar memória alocada dinamicamente.
h) A _____ é uma versão com restrições de uma lista encadeada na qual os nodos podem ser inseridos somente no fim da lista e retirados somente do início da lista.
i) A _____ é uma estrutura de dados não-linear, de duas dimensões, que contém nodos com dois ou mais links.
j) Uma pilha é chamada de uma estrutura de dados porque o último nodo inserido é o primeiro nodo retirado.
k) Os nodos de uma árvore _____ contém dois membros de link.
1) O primeiro nodo de uma árvore é o nodo_____
m) Cada link em um nodo de uma árvore aponta para um _____ou daquele nodo.
n) Um nodo de árvore que não tem nenhum filho é chamado de um nodo _____
o) Os quatro algoritmos de percurso dos nodos que mencionamos no texto para árvores de pesquisa binária são
_____ _____ _____ e _____
15.2 Quais são as diferenças entre uma lista encadeada e uma pilha?

15.3 Quais são as diferenças entre uma pilha e uma fila’?
15.4 Talvez um título mais apropriado para este capítulo teria sido “Estruturas de dados reutilizáveis”. Comente sobre como cada uma das entidades ou conceitos seguintes contribui para a reutilização de estruturas de dados:
a) classes
b) classes gabarito
e) herança
d) herança privada
e) composição.
15.5 Forneça manualmente os percursos em ordem, na pré-ordem e na pós-ordem da árvore de pesquisa binária da Fig. 15.19.
Respostas aos exercícios de auto-revisão
15.1 a) referente. b) new. c) pilha. d) funções predicado. e) primeiro a entrar, primeiro a sair (FIFO). 1) link. g) delete. h) fila. i) árvore. j) último a entrar, primeiro a sair (LIFO). k) binária. 1) raiz. m) filho ou subárvore. n) folha. o) em ordem,
na pré-ordem e na pós-ordem.
15.2 É possível se inserir um nodo em qualquer lugar em uma lista encadeada e remover um nodo de qualquer lugar em uma lista encadeada. Os nodos em uma pilha somente podem ser inseridos no topo da pilha e removidos do topo da pilha.
15.3 Uma fila tem ponteiros tanto para seu início como para seu fim, de forma que podemos inserir no fim e retirar do início. Uma pilha tem um ponteiro único para o topo da pilha, onde tanto a inserção como a retirada de nodos são executadas.
15.4 a) Classes nos permitem instanciar tantos objetos de estrutura de dados de um certo tipo (i.e., classe) conforme desejemos.
b) Gabaritos de classes nos possibilitam instanciar classes relacionadas - cada uma baseada em um parâmetro de tipo diferente - podemos então gerar tantos objetos de cada classe gabarito quantos quisermos.
e) A herança nos possibilita reutilizar o código de uma classe base em uma classe derivada, de forma que a estrutura de dados da classe derivada também é uma estrutura de dados da classe base (com herança pública, é claro).
d) A herança privada nos possibilita reutilizar partes do código de uma classe base para formar uma estrutura de dados da classe derivada; como a herança é private. todas as funções membro private da classe base tornam-se private na classe derivada. Isto nos possibilita evitar que clientes da estrutura de dados da classe derivada acessem funções membro da classe base que não se aplicam à classe derivada.
e) A composição nos possibilita reutilizar código tomando a estrutura de dados de um objeto de classe um membro de uma classe composta; se tornarmos o objeto de classe um membro private da classe composta, então as funções membro public do objeto de classe não estarão disponíveis através da interface do objeto composto.

los e 15.5 O percurso em ordem é:
air. 11 18 19 28 32 40 44 49 69 71 72 83 92 97 99
vazia
O percurso na pré-ordem é:
tiros
49 28 18 11 19 40 32 44 83 71 69 72 97 92 99
O percurso na pós-ordem é:
ente 11 19 18 32 44 40 28 69 72 71 92 99 97 83 49
O percurso na pré-ordem é:
mais
49 28 18 11 19 40 32 44 83 71 69 72 97 92 99
‘odo O percurso na pós-ordem é:
11 19 18 32 44 40 28 69 72 71 92 99 97 83 49
são
orno 119 32 44 62 929
Fig. 15.19 Uma árvore de pesquisa binária de 15 nodos.
5.19. Exercícios
15.6 Escreva um programa que concatena dois objetos de listas encadeadas de caracteres. O programa deve incluir a função concatenate. a qual recebe referências para ambos os objetos da lista como argumentos e concatena a segunda lista com a primeira lista.
ei 15.7 Escreva um programa que intercala (merge) dois objetos lista de inteiros ordenados em um único objeto lista de inteiros ordenada. A função merge deve receber referências para cada um dos objetos lista a serem intercalados e deve retornar uma referência para o objeto lista intercalada.
uma . . . .
15.8 Escreva um programa que insere 25 inteiros aleatorios de O ate 100 em ordem em um objeto lista encadeada. O programa deve calcular a soma dos elementos e a média dos elementos em ponto flutuante.
rdo 15.9 Escreva um programa que cria um objeto lista encadeada de 10 caracteres e, então, cria um segundo objeto lista contendo uma cópia da primeira lista, mas em ordem inversa.
[flOS 15.10 Escreva um programa que recebe como entrada uma linha de texto e usa um objeto pilha para imprimir a linha invertida.
15.11 Escreva um programa que usa um objeto pilha para determinar se um string é um palíndromo (i.e., um string que é a de soletrado da mesma maneira da frente para trás e de trás para diante). O programa deve ignorar espaços e pontuação.
ados 15.12 Pilhas são usadas por compiladores para ajudar no processo de analisar expressões e gerar código em linguagem de máquina. Neste e no próximo exercício, investigaremos como compiladores analisam expressões aritméticas consistindo somen se te em consØtes, operadores e parênteses. Seres humanos geralmente escrevem expressões como 3 + 4 e 7 / 9 em que o operador (aqui + ou i é escrito entre
o_de seus operandos - isto é chamado de notação infixa. Os computadores “preferem” a notação pós-fixa, na qual o operador é escrito çoes à direita de

seus dois operandos. As expressões infixas precedentes apareceriam em notação pós-fixa como 3 4 e 7 9 1,
respectivamente.

792 C++ COMO PROGRAMAR
Para calcular uma expressão infixa complexa, um compilador primeiro converteria a expressão para a notação pós-fixa e, então, analisaria e calcularia a versão pós-fixa da expressão. Cada um destes algoritmos exige somente uma única passagem pela expressão, da esquerda para a direita. Cada algoritmo usa um objeto pilha para suportar sua operação e, em cada passagem do algoritmo, a pilha é usada para um fim diferente.
Neste exercício, você escreverá uma versão em C++ do algoritmo de conversão de notação infixa para pós-fixa. No próximo exercício, você escreverá uma versão em C++ do algoritmo de análise da expressão pós-fixa. Mais à frente no capítulo, você descobrirá que o código que você escreveu para este exercício pode o ajudar a implementar um compilador completo que funciona.
Escreva um programa que converte uma expressão aritmética infixa comum (suponha que uma expressão válida é fornecida
como entrada) com inteiros de um único dígito, tal como
(6 + 2) * 5 - 8 / 4
para uma expressão pós-fixa. A versão pós-fixa da expressão infixa precedente é
62+5 * 84 / -
O programa deveria ler a expressão para o array de caracteres infix e usar versões modificadas das funções de pilha implementadas neste capítulo para ajudar a criar a expressão pós-fixa em um array de caracteres postfix. O algoritmo para criar uma expressão pós-fixa é o seguinte:
1) Insira um parêntese à esquerda ‘ (no topo da pilha.
2) Acrescente um parêntese à direita ‘) ao fim de infix.
3) Enquanto a pilha não estiver vazia, leia irif ix da esquerda para a direita e faça o seguinte:
Se o caractere atual em infix é um dígito, copie-o para o próximo elemento de postfix. Se o caractere atual em infix é um parêntese à esquerda, insira-o no topo da pilha.
Se o caractere atual em infix é um operador,
Remova operadores (se existirem) do topo da pilha enquanto tiverem precedência igual ou mais alta que o
operador atual e insira os operadores removidos em pos tf ix.
Insira o caractere atual em inf ix no topo da pilha.
Se o caractere atual em inf ix é um parêntese à direita
Remova operadores do topo da pilha e insira-os em postfix até um parêntese a esquerda estar ou topo da
pilha.
Remova (e descarte) da pilha o parêntese à esquerda.
As seguintes operações aritméticas são permitidas em uma expressão:
+ adição

- subtração
* multiplicação
/ divisão
exponenciação
% módulo
A pilha deve ser mantida com nodos de pilha, cada um contendo um membro de dados e um ponteiro para o próximo nodo da pilha.
Alguns dos recursos funcionais que você pode querer fornecer são:
a) Função convertTopostfix, que converte a expressão infixa para a notação pós-fixa.
b) Função isOperator. que determina se c é uni operador.
c) Função precedence. que determina se a precedência do operatori é menor que, igual a ou maior que a precedência do operator2, A função retorna -1,0 e 1, respectivamente.
d) Função push. que insere um valor na pilha.
e) Função pop, que retira um valor da pilha.
O Função stackTop, que retoma o valor do topo da pilha sem retirá-lo da pilha.
g) Função isEmpty, que determina se a pilha está vazia.
h) Função imprime Stack, que imprime a pilha.
15.13 Escreva um programa que analisa uma expressão pós-fixa (suponha que ela é válida), tal como
62+5 * 84 / -
O programa deve ler uma expressão pós-fixa que consiste em dígitos e operadores para um array de caracteres. Usando versões modificadas das funções de pilha inementadas anteriormente neste capítulo, o programa deve percorrer a expressão e calculála . O algoritmo é o seguinte:

1) Acrescente o caractere nulo (‘ \O’) ao fim da expressão pós-fixa. Quando o caractere nulo for encontrado, nenhum processamento adicional será mais necessário.
2) Enquanto \O’ não for encontrado, leia a expressão, da esquerda para a direita.
Se o caractere atual é um dígito,
Insira seu valor inteiro no topo da pilha (o valor inteiro de um caractere que é um dígito é seu valor no conjunto
de caracteres do computador menos o valor de O’ no conjunto de caracteres do computador).
Caso contrário, se o caractere atual é um operador;
Remova os dois elementos do topo da pilha para as variáveis x e y.
Calcule y operador x.
Insira o resultado do cálculo no topo da pilha.
3) Quando o caractere nulo for encontrado na expressão, remova o valor do topo da pilha. Este é o resultado da expressão pós-fixa.
Nota: na etapa 2) acima, se o operador é / ‘, o topo da pilha é 2 e o próximo elemento na pilha é 8, então remova 2 da pilha e o atribua a x. remova 8 para y,

calcule 8/2 e insira o resultado, 4, de volta no topo da pilha. Esta nota também se aplica ao operador '-'. As operações aritméticas permitidas em uma expressão são:
+ adição
- subtração
* multiplicação
/ divisão
A exponenciação
módulo
A pilha deve ser mantida com nodos de pilha que contêm um membro de dados inteiro e um ponteiro para o próximo nodo da pilha. Você pode querer fornecer os seguintes recursos funcionais:
a) Função evaluatePostfixExpression. que analisa e calcula a expressão pós-fixa.
b) Função calculate. que calcula a expressão opi operador op2.
c) Função push, que insere um valor na pilha.
d) Função pop. que retira um valor da pilha.
e) Função isEmpty, que determina se a pilha está vazia.
f) Função printStack. que imprime a pilha.
15.14 Modifique o programa de análise e cálculo de expressões pós-fixas do Exercício 15.13 de forma que ele possa processar operandos inteiros maiores do que 9.
15.15 (Simulação de supermercado) Escreva um programa que simula uma fila de caixa em um supermercado. A fila é um objeto fila. Os clientes (i.e., objetos clientes) chegam em intervalos inteiros aleatórios de 1 a 4 minutos. Além disso, cada cliente é atendido em intervalos inteiros aleatórios de 1 a 4 minutos. Obviamente, as taxas precisam ser equilibradas. Se a taxa de chegada média é maior que a taxa de atendimento médio, a fila crescerá infinitamente. Até com taxas "equilibradas", a aleatoriedade ainda pode causar longas filas. Execute a simulação do supermercado para um dia de 12 horas (720 minutos), usando o seguinte algoritmo:
1) Escolha um inteiro aleatório entre 1 e 4 para determinar o minuto no qual o primeiro cliente chega.
2) No momento de chegada do primeiro cliente:
Determine o tempo de atendimento do cliente (um inteiro aleatório de 1 a 4);
Comece a atender o cliente;
Programe o tempo de chegada do próximo cliente (um inteiro aleatório de 1 a 4 somado à hora atual). 3) Para cada minuto do dia:
Se o próximo cliente chega,
Faça o seguinte:
Insira o cliente no objeto fila;
Programe o tempo de chegada do próximo cliente;
Se foi completado o atendimento para o último cliente,
Faça o seguinte:
Retire da fila o próximo cliente a ser atendido;
Determine o tempo de conclusão do atendimento ao cliente

(um inteiro aleatório de 1 até 4 somado ao tempo atual).
Agora execute sua simulação para 720 minutos e responda a cada uma das seguintes perguntas:
a) Qual é o número máximo de clientes na fila, em qualquer momento?
b) Qual é a espera mais longa que um cliente teve?
c) O que acontece se o intervalo de chegada é mudado de 1 a 4 minutos para 1 a 3 minutos?

15.23 (Pesquisa em árvore binária) Escreva a função membro binaryTreeSearch. a qual tenta localizar um valor especificado em um objeto árvore de pesquisa binária. A função deve aceitar como argumentos um ponteiro para o nodo raiz da árvore binária e uma chave de pesquisa para ser localizada. Se o nodo contendo a chave de pesquisa é encontrado, a função deve retornar um ponteiro para aquele nodo; caso contrário, a função deve retornar um ponteiro nulo.

15.24 (Percurso de árvore binária em ordem de nível) O programa da Fig. 15.16 ilustrou três métodos recursivos de percorrer uma árvore binária - percursos em ordem, na pré-ordem e na pós-ordem. Este exercício apresenta o percurso na ordem de nível de uma árvore binária, no qual os valores dos nodos são impressos nível por nível, começando no nível do nodo raiz. Os nodos em cada nível são impressos da esquerda para a direita. O percurso em ordem de nível não é um algoritmo recursivo. Ele usa um objeto fila para controlar a impressão dos nodos. O algoritmo é o seguinte:
1) Insira o nodo raiz na fila
2) Enquanto existirem nodos na fila,
Obtenha o próximo nodo na fila
Imprima o valor do nodo
Se o ponteiro para o filho esquerdo do nodo não é nulo
Insira o nodo filho esquerdo na fila
Se o ponteiro para o filho direito do nodo não é nulo
Insira o nodo filho direito na fila.

Escreva a função membro levelOrder para executar um percurso em ordem de nível de um objeto árvore binária. Modifique o programa da Fig 15. 16 para usar esta função. (Nota: você também necessitará modificar e incorporar as funções de processamento de fila da Fig. 15.12 neste programa.)

15.25 (imprimindo árvores) Escreva uma função membro recursiva outputTree para exibir um objeto árvore binária na tela.
A função deve exibir a árvore linha por linha, com o topo da árvore à esquerda da tela e a parte inferior da árvore em direção ao
lado direito da tela. Cada linha é exibida verticalmente. Por exemplo, a árvore binária ilustrada na Fig. 15.19 é exibida como

segue:

Note que o nodo folha mais à direita aparece no topo da saída na coluna mais à direita e o nodo raiz aparece na esquerda da saída. Cada coluna de saída começa cinco espaços à direita da coluna anterior. A função outputTree deve receber um parâmetro totaiSpaces representando o número dos espaços antes do valor a ser exibido (esta variável deve começar em zero, de modo que o nodo raiz seja exibido à esquerda da tela). A função usa um percurso em ordem modificado, para exibir a árvore - ela começa no nodo mais à direita na árvore e trabalha retrocedendo para a esquerda. O algoritmo é o seguinte:
Enquanto o ponteiro para o nodo atual não for nulo
Recursivamente chama outputTree com a subárvore direita do nodo atual e totaiSpaces + 5
Usa uma estrutura for para contar de 1 até totaiSpaces e imprimir espaços
Exibe o valor no nodo atual
Inicializa o ponteiro para o nodo atual para apontar para a subárvore esquerda do nodo atual
Incrementa totaiSpaces em 5.

99

97

83

92

72

71

49

69

40

44
32

28

19

18

11

Seção especial: construindo seu próprio compilador
Nos Exercícios 5.18 e 5.19, introduzimos a Simpletron Machine Language (SML) e você implementou um simulador do computador Simpletron para executar programas escritos em SML. Nesta seção, construímos um compilador que converte programas escritos em uma linguagem de programação de alto nível para a SML. Esta seção “unifica” todo o processo de programação. Você escreverá programas nesta nova linguagem de alto nível, compilará os programas com o compilador que você construirá e executará os programas no simulador que você construiu no Exercício 7.19. Você deve fazer todo esforço para implementar seu compilador de uma maneira orientada a objetos.
15.26 (A linguagem Simple) Antes de começarmos a construir o compilador, discutiremos uma linguagem simples, mas ainda assim poderosa, de alto nível e semelhante às primeiras versões da popular linguagem BASIC. Chamamos esta linguagem de Simple. Todo comando em Simple consiste em um número de linha e uma instrução de Simple. Os números de linha devem aparecerem ordem ascendente. Cada instrução começa com um dos seguintes comandos de Simple: rem. input. let. print. goto, if/goto ou end (ver Fig. 15.20). Todos os comandos, exceto end, podem ser usados repetidamente. Simple avalia somente expressões inteiras usando os operadores +, -‘ *. e 1. Estes operadores têm a mesma precedência que em C. Parênteses podem ser usados para mudar a ordem de avaliação de uma expressão.
Nosso compilador de Simple reconhece somente letras mínusculas. Todos os caracteres em um arquivo de Simple devem ser minúsculas (letras maiúsculas resultam em um erro de sintaxe, a menos que apareçam em um comando rem. caso em que são ignoradas). Um nome de variável é uma única letra. Simple não permite nomes de variáveis descritivos, assim as variáveis deveriam ser explicadas em observações, para indicar seu uso em um programa. Simple usa somente variáveis do tipo inteiro. Simple não tem declarações de variáveis simplesmente mencionar um nome de variável em um programa faz com que a variável seja declarada e automaticamente inicializada com zero. A sintaxe de Simple não permite a manipulação de strings (ler um string, escrever um string, comparar srrings, etc.). Se um string é encontrado em um programa de Simple (após um outro comando que não rem), o compilador gera um erro de sintaxe. A primeira versão de nosso compilador assumirá que os programas em Simple são escritos e digitados corretamente. O Exercício 15.29 pede ao estudante para modificar o compilador para executar a verificação de erros de sintaxe.
Simple usa o comando condicional if/goto e o comando incondicional goto para alterara fluxo de controle durante a execução do programa. Se a condição no

comando if/goto for verdadeira, o controle é transferido para uma linha específica do programa. Os seguintes operadores relacionais e de igualdade são válidos em um comando if/goto: <,>,<. >. == ou
A precedência destes operadores é a mesma que em C++.
Iremos agora considerar vários programas que demonstram as características de Simple. O primeiro programa (Fig. 15.21) lê dois inteiros do teclado, armazena os valores nas variáveis a e b e calcula e imprime sua soma (armazenada na variável c).

II
:4

Fig. 15.20 Comandos de Simple.

Comando	Ex	emplo de comando		Descrição
rem	50	rem isto	é um comentário	Texto que segue o comando rem se destina a documentação e é ignorado pelo compilador.
input	30	input x		Exibe um ponto de interrogando solicitando uma entrada do usuário para entrar um inteiro. Lê aquele inteiro do teclado e armazena o inteiro em x.
let	80	let u = 4	* (j - 56)	Atribui au o valor de 4 * - 56). Note que uma expressão arbitrariamente complexa pode aparecer à direita do sinal de igual.
print	10	print w		Exibeovalordew.
goto	70	goto 45		Transfere o controle do programa para a linha 45.
if/goto	35	if i == z goto 80		Compara i e z quanto à igualdade e transfere o controle para a linha 80 se a condição é verdadeira; caso contrário, continua a execução com o próximo comando.
end	99	end		Termina a execução do programa.

Fig. 15.21 Programa em Simple que determina a soma de dois inteiros.

O programa da Fig. 15.22 determina e imprime o maior de dois inteiros. Os inteiros são fornecidos pelo teclado e armazenados

em s e t. O comando if/goto testa a condição s >= t. Se a condição é verdadeira, o controle é transferido para linha 90 e s
é impresso; caso contrário, t é impresso e o controle é transferido para o comando end na linha 99, onde o programa termina.
Simple não oferece uma estrutura de repetição (tal como for, while ou do/while de C++). Porém, Simple pode simular cada uma das estruturas de repetição de C++ usando os comandos if/goto e goto. A Fig.15.23 usa um laço controlado por sentinela para calcular os quadrados de vários inteiros. Cada inteiro é fornecido pelo teclado e armazenado na variável j. Se o valor lido é a sentinela -9999, o controle é transferido para a linha 99. onde o programa termina. Caso contrário, o quadrado de j é atribuído a k, k é exibido na tela e o controle é passado para a linha 20, onde o próximo inteiro é lido.

npuimas Você irá e seu

inda ï de vem Lnt, valia Leses

vem
sao
íveis
eiro.

ue a (ler)utro)grapara

ante

ífica

ou

(Pig.
la na

irada
o do

Fig. 15.22 Programa em Simple que encontra o maior de dois inteiros.

Fig. 15.23 Calcula o quadrado de vários inteiros.

1	10	rem determina e imprime	a soma	de dois inteiros		
2	1	rem				

	5					
3	20	rem lê os dois inteiros				
4	30	input a				
5	40	input b				
6	45	rem				
7	50	rem soma os inteiros e	guarda o resultado		em	e
8	60	letc=a+b				
9	65	rem				
10	70	rem imprime o resultado				
11	80	print c				
12	90	rem termina a execução	do programa			
13	99	end				

1	10	rem	determina	o maior		de	dois inteiros			
2	20	input	s							
3	30	input	t							
4	32	rem								
5	35	rem	testa se	s >=	t					
6	40	if s	>= tgoto	90						
7	45	rem								
8	50	rem	t é maior	do	que	s,	portanto	imprime t		
9	60	print	t							
10	70	goto	99							

li	75	rem								
12	80	rem	s é maior	do	que	ou	igual a	t, portanto	imprime	s
13	90	print	s							
14	99	end								

1	10	rem	calcula	o quadrado de diversos inteiros			
2	20	input	j				
3	23	rem					
4	25	rem	testa o	valor sentinela			
5	30	if j	= -9999	goto 99			
6	33	rem					
7	35	rem	calcula	o quadrado de j e atribui o resultado a k			
8	40	letk	=j * j				
9	50	print	k				
10	53	rem					
11	55	rem	repete	o laço para obter	próximo j		
12	60	goto	20				
13	99	end					

Usando os programas de exemplo da Fig. 15.21, Fig. 15.22 e Fig. 15.23 como orientação, escreva um programa em Simple para executar cada um dos seguintes itens:

a) Receba como entrada três inteiros, determine sua média e imprima o resultado.

b) Use um laço controlado por sentinela para receber como entrada 10 inteiros e calcule e imprima sua soma.

c) Use um laço controlado por contador para receber como entrada 7 inteiros, algum positivos e alguns negativos, e calcule e imprima sua média.
d) Receba como entrada uma série de inteiros e determine e imprima o maior. O primeiro inteiro lido indica quantos números devem ser processados.
e) Receba como entrada lO inteiros e imprima o menor.
f) Calcule e imprima a soma dos inteiros pares de 2 até 30.
g) Calcule e imprima o produto dos inteiros ímpares de 1 a 9.
15.27 (Construindo um compilador; pré-requisito: complete os Exercícios 5.18, 5.19, 15.12, 15.13 e 15.26) Agora que a linguagem Simple foi apresentada (Exercício 15.26), discutiremos como construir um compilador de Simple. Primeiro, consideramos o processo para que um programa em Simple seja convertido em SML e executado pelo simulador do Simpletron (veja Fig. 15.24). Um arquivo contendo um programa em Simple é lido pelo compilador e convertido em código em SML. O código em SML é colocado em um arquivo em disco, no qual as instruções de SML aparecem uma por linha. O arquivo em SML é então carregado para o simulador do Simpletron e os resultados são enviados para um arquivo em disco e para a tela. Note que o programa em Simpletron desenvolvido no Exercício 5.19 recebe sua entrada pelo teclado. Ele deve ser modificado para ler um arquivo, de forma que ele possa executar os programas produzidos pelo nosso compilador.
saída para
saida para ateia
o disco
Fig. 15.24 Escrevendo, compilando e executando um programa na linguagem Simple.
O compilador de Simple executa duas passa gens pelo programa em Simple para convertê-lo para SML. A primeira passagem constrói uma tabela de símbolos (objeto) em que todo número de linha (objeto), nome de variável (objeto) e constante (objeto) do programa em Simple é armazenado com seu tipo e localização correspondente no código SML final (a tabela de símbolos é discutida em detalhes abaixo). A primeira passagem também produz o(s) objeto(s) instrução(ões) correspondente(s) de SML para cada um dos comandos em Simple (objeto, etc.). Como veremos, se o programa em Simple contém comandos que transferem o controle para uma linha mais à frente no programa, a primeira passagem resulta em um programa de SML contendo algumas instruções “inacabadas”. A segunda passagem do compilador localiza e completa as instruções inacabadas e grava o programa SML em um arquivo.
Primeiro passo
O compilador começa lendo um comando do programa em Simple para a memória. A linha deve ser separada em suas “unidades léxicas” (tokens) individuais (i.e., “pedaços” de um comando) para processamento e compilação (a função strtok da biblioteca padrão pode er usada para facilitar esta tarefa). Lembre que toda instrução começa com um número de linha seguido por um comando. A medida que o compilador quebra um comando em “unidades léxicas”, se a “unidade léxica” é um número de linha, uma variável ou uma constante, ela é colocada na tabela de símbolos. Um número de linha é

colocado na tabela de símbolos somente se ele é a primeira “unidade léxica” em uma instrução. O objeto symbolTable é um array de objetos tableEntry, cada um representando um símbolo no programa. Não existe nenhuma restrição quanto ao número de símbolos que podem aparecer no programa. Então, a symbolTable para um programa particular pode ser grande. Por ora, use para a syinbolTable um array de 100 elementos. Você pode aumentar ou diminuir seu tamanho depois que o programa estiver funcionando.

Cada objeto tableEntry contém três membros. O membro symbol é um inteiro contendo a representação em ASCII de uma variável (lembre-se que nomes de variáveis são caracteres simples), um número de linha ou uma constante. O membro type é um dos seguintes caracteres, que indicam o tipo do símbolo: ‘C’ para constante, ‘L’ para número de linha, ou ‘V’ para variável. O membro location contém a posição de memória no Simpletron (00 a 99) à qual o símbolo se refere. A memória do Simpletron é um array de 100 inteiros no qual as instruções de SML e dados são armazenados. Para um número de linha, a localização é o elemento no array de memória do Simpletron no qual as instruções de SML para o comando de Simple começam. Para uma variável ou constante, a localização é o elemento no array de memória do Simpletron em que a variável ou constante é armazenada. Variáveis e constantes são alocadas do fim da memória do Simpletron para o início. A primeira variável ou constante é armazenada na localização 99, a próxima na localização 98, etc.
A tabela de símbolos desempenha um papel importante na conversão de programas em Simple para SML. Aprendemos no Capítulo 5 que uma instrução de SML é um inteiro de quatro dígitos composto de duas partes - o código de operação e o operando. O código de operação é determinado pelos comandos em Simple. Por exemplo, ao comando Simple input corresponde o código de operação de SML 10 (ler) e ao comando em Simple print corresponde o código de operação de SML 11 (escrever). O operando é uma posição de memória contendo os dados sobre os quais o código de operação executa sua tarefa (por exemplo, o código de operação 10 lê um valor do teclado e armazena-o na posição de memória especificada pelo operando). O compilador pesquisa symbolTable para determinar a posição de memória do Simpletron para cada símbolo e, assim, a localização correspondente pode ser usada para completar as instruções de SML.
A compilação de cada comando de Simple é baseada em seu comando. Por exemplo, depois que o número de linha em um

comando rem é inserido na tabela de símbolos, o restante do comando é ignorado pelo compilador porque um comentário serve somente para fins de documentação. Os comandos input, print, goto e end correspondem às instruções de SML ler escrever desviar (para uma posição específica) e parar. Os comandos contendo esses comandos de Simple são diretamente convertidos para SML (note que um comando goto pode conter uma referência não-resolvida

se o número de linha especificado se refere a um comando mais à frente no arquivo do programa em Simple; isto é às vezes chamado de referência antecipada).
Quando um comando goto é compilado com uma referência não-resolvida, a instrução de SML deve ser marcada, para indicar que na segunda passagem o compilador deve completar a instrução. As marcas são armazenadas em um array flags de 100 elementos do tipo int, no qual cada elemento é inicializado com -1. Se a posição de memória a que um número de linha no programa em Simple se refere ainda não é conhecida (i.e., não está na tabela de símbolos), o número da linha é armazenada no array flags. no elemento com o mesmo índice que a instrução incompleta. O operando da instrução incompleta é temporariamente inicializado com 00. Por exemplo, uma instrução de desvio incondicional (que está fazendo uma referência antecipada) é deixada como +4 000 até a segunda passagem do compilador. A segunda passagem do compilador será descrita em breve.
) A compilação de comandos if/goto e let é mais complicada que a de outros comandos - eles são os únicos comandos que produzem mais de uma instrução de SML. Para um if/goto. o compilador produz código para testar a condição e para desviar para outra linha, se necessário. O resultado do desvio pode ser uma referência não-resolvida. Cada um dos operadores relacionais e de igualdade podem ser simulados usando-se as instruções desvie quando zero e desvie quando negativo de SML (ou, possivelmente, uma combinação de ambas).
Para um comando let, o compilador produz código para avaliar uma expressão aritmética arbitrariamente complexa consistindo em variáveis inteiras e/ou constantes. As expressões devem separar cada operando e operador com espaços. Os Exercícios 15.12 e 15.13 apresentaram o algoritmo de conversão infixa para pós-fixa e o algoritmo de avaliação pós-fixa usado por compiladores para avaliar expressões. Antes de continuar seu compilador, você deve completar cada um destes exercícios. Quando um compilador encontra uma expressão, ele converte a expressão da notação infixa para a notação pós-fixa e, então, avalia a expressão pós-fixa.
Como o compilador produz a linguagem de máquina para avaliar uma expressão que contém variáveis’? O algoritmo de avaliação pós-fixa contém um “gancho onde o compilador pode gerar instruções de SML em vez de avaliar realmente a expressão. Para posssibilitar este “gancho” no compilador, o algoritmo de avaliação pós-fixa deve ser modificado para pesquisar a tabela de simbolos para cada símbolo que ele encontra (e possivelmente inseri-lo), determinar a posição de memória correspondente ao símbolo e inserir a posição de memória no topo da pilha (em vez do símbolo). Quando um operador é encontrado na expressão pós-fixa, as duas posições de memória no topo da pilha são recuperadas e é produzida a linguagem de máquina para executar a operação usando as posições de memória como operandos. O resultado de cada subexpressão é armazenado em uma posição temporária na memória e empurrado de volta para o topo da pilha, de modo que a avaliação da expressão pós-fixa possa continuar. Quando a avaliação pós-fixa está completa, a posição de memória contendo o

resultado é a única posição deixada no topo da pilha. Esta é retirada da pilha e instruções de SML são geradas para atribuir o resultado à variável à esquerda no comando

let.

Segundo passo

O segundo passo do compilador executa duas tarefas: soluciona quaisquer referências não-resolvidas e grava o código de SML em um arquivo. A resolução de referências acontece como segue:
a) Percorre o array flags procurando uma referência não-resolvida (i.e., um elemento com um valor diferente de -1).
b) Localiza o objeto no array symbolTable que contém o símbolo armazenado no array flags (assegure-se de que o tipo do símbolo é L' para um número de linha).
. -

80() C++ COMO PROGRAMAR
c) Insira a posição de memória do membro location na instrução com a referência não-resolvida (lembre que uma instrução que contém uma referência não-resolvida tem operando 00).
d) Repita os passos 1, 2 e 3 até atingir o fim do array flags.
Depois do processo de resolução estar completo, o array inteiro que contém o código SML é gravado em um arquivo em disco, com uma instrução de SML por linha. Este arquivo pode ser lido pelo Simpletron para execução (depois de o simulador ser modificado para ler sua entrada de um arquivo). Compilar seu primeiro programa em Simple, gravá-lo em um arquivo de SML e então executar esse arquivo, deve lhe dar uma verdadeira sensação de realização pessoal.
Um exemplo completo
O exemplo seguinte ilustra uma conversão completa de um programa em Simple para SML, tal como eia será executada pelo
compilador de Simple. Considere um programa em Simple que recebe como entrada um inteiro e soma os valores de 1 até aquele
inteiro, O programa e as instruções de SML produzidos pelo primeiro passo do compilador Simple são ilustrados na Fig. 15.25.
A tabela de símbolos construída pelo primeiro passo é mostrada na Fig. 15.26.

Fig. 15.26 Tabela de símbolos para o programa da Fig. 15.25 (parte 1 de 2).

Localização e Programa em Simple instrução SML Descrição	
5 rem soma 1 a x nenhuma comentário ignorado	
10 input x 09 +2095 lêxparaaposição 99	
15 rem testa se y == x nenhuma comentário ignorado	

20 if y == x goto 60 01 +2098 carregay (98> no acumulador	
02 +3199 subtraix (99) do acumulador	
03 +4200 desvia se zero para endereço não-calculado	
25 rem incrementa y nenhuma comentário ignorado	
30 let y = y + 1 04 +2098 carrega y no acumulador	
05 +3097 somai (97) ao acumulador	
06 +2196 armazena na posição temporária 96	
07 +2096 carrega da posição temporária 96	
08 +2 198 armazena o acumulador em y	
35 rem soma y a total nenhuma comentário ignorado	
40 let t = t + y 09 +2095 carrega t (95) no acumulador	
10 +30 98 soma y ao acumulador	
ii +2194 armazena na posição temporária 94	
12 +20 94 carrega da posição temporária 94	
13 +2 195 armazena o acumulador em t	
45 rem repete y nenhuma comentário ignorado	
50 goto 20 14 +4001 desvia para o endereço 01	
55 rem escreve resultado nenhuma comentário ignorado	
60 print t 15 +1195 enviatparaatela	
99 end 16 +4300 termina execução	
Fig. 15.25 Instruções SML produzidas após o primeiro passo do compilador.	
Símbolo Tipo Endereço	

30 L 04
1 C 97
35 L 09
40 L 09
‘t’ V 95
45 L 14
50 L 14
55 L 15
60 L 15
99 L 16
Fig. 15.26 Tabela de simbolos para o programa da Fig. 15.25 (parte 2 de 2).
A maioria dos comandos de Simple são convertidos diretamente em uma única instrução de SML. As exceções neste programa são os comentários (rem). o comando if/goto na linha 20 e os comandos let. As observações não se traduzem para linguagem de máquina. Porém, o número da linha de um comentário é colocado na tabela de símbolos no caso do número da linha ser referenciado em um comando goto ou um comando if/goto. A linha 20 do programa especifica que, se a condição y == x
‘ é verdadeira, o controle do programa é transferido para a linha 60. Como a

linha 60 aparece mais à frente no programa, a primeira passagem do compilador ainda não colocou 60 na tabela de símbolos (os números de linha dos comandos são colocados na tabela de símbolos somente quando aparecem como a primeira “unidade léxica” em um comando). Então, não é possível neste momento se determinar o operando da instrução de SML desvia se zero na posição 03 no array de instruções de SML. O compilador coloca 60 na localização 03 do array flags para indicar que a segunda passagem completará esta instrução.

Devemos manter o controle da próxima posição de instrução no array de SML porque não existe uma correspondência um para um entre comandos de Simple e instruções de SML. Por exemplo, o comando if/goto da linha 20 é compilado para três instruções de SML. Toda vez que uma instrução é produzida, devemos incrementar o contador de instruções para a próxima localização no array de SML. Note que o tamanho da memória do Simpletron poderia ser um problema para programas de Simple com muitos comandos, variáveis e constantes. E possível que o compilador fique sem memória. Para testar este caso, seu programa deveria conter um contador de dados para manter a localização em que a próxima variável ou constante será armazenada no array de SML. Se o valor do contador de instruções for maior que o valor do contador de dados, o array de SML está cheio. Neste caso, o processo de compilação deveria terminar e o compilador deveria imprimir uma mensagem de erro indicando que ele ficou sem memória durante a compilação. Isto serve para enfatizar que, embora o programador esteja liberado dos problemas de administrar a memória pelo compilador, o próprio compilador deve determinar cuidadosamente a colocação de instruções e dados na memória e deve testar a ocorrência de erros tais como o esgotamento da memória durante o processo de compilação.

Uma visão passo a passo do processo de compilação

Iremos agora fazer um walkthrough do processo de compilação para o programa em Simple na Fig. 15.25. O compilador lê a primeira linha do programa

```
5 rem soma 1 a x
```

para a memória. A primeira “unidade léxica” no comando (o número de linha) é determinada usando-se strtok (veja Capítulos 5 e 16 para uma discussão das funções de manipulação de strings de C++). A “unidade léxica” retomada por strtok é convertida em um inteiro usando atoi, de modo que o símbolo 5 possa ser localizado na tabela de símbolos. Se o símbolo não é encontrado, é inserido na tabela de símbolos. Uma vez que estamos no começo do programa e esta é a primeira linha, nenhum símbolo ainda está na tabela. Assim, 5 é inserido na tabela de símbolos como tipo L (número de linha) e atribuído à primeira posição no array de SML (00). Embora esta linha seja um comentário, um espaço na tabela de símbolos é ainda alocado para o número de linha (no caso de ele ser referenciado por um goto ou um if/goto). Nenhuma instrução de SML é gerada para um comando rem, assim o contador de instruções não é incrementado.

O comando

```
10 input x
```

é o próximo a ser separado em “unidades léxicas”. O número de linha 10 é colocado na tabela de símbolos como tipo L e atribuído à primeira posição no

array de SML (00, porque um comentário começou o programa e assim o contador de instruções é atualmente 00). O comando input indica que a próxima "unidade léxica" é uma variável (somente uma variável pode aparecer em um comando input). Como input corresponde diretamente a um código de operação de SML, o compilador tem que determinar a posição dcx no array de SML. O símbolo x não é encontrado na tabela de símbolos. Assim, ele é inserido na tabela de símbolos como a representação em ASCII de x, tipo de dados V, e atribuído à posição 99 no array de SML (a memória de dados começa em 99 e é alocada para trás). O código de SML pode agora ser gerado para este comando. O código de operação 10 (o código da operação SML read) é multiplicado por 100 e a localização de x (conforme determinada na tabela de símbolos)

é separado em "unidades léxicas", o número de linha 35 é inserido na tabela de símbolos como tipo L e atribuído à posição 09. O comando
40 let t = t + y
é semelhante à linha 30. A variável t é inserida na tabela de símbolos como tipo V e atribuída à posição de SML 95. As
instruções seguem a mesma lógica e formato que a linha 30 e as instruções 09+2095, 10+3098. 11+2194. 12+2094 e
13+2 195 são geradas. Note que o resultado de t + y é atribuído à posição temporária 94 antes de ser atribuído a t (95). Uma
vez mais, o leitor notará que as instruções nas posições de memória 11 e 12 parecem ser redundantes. Repetindo, discutiremos
isto em breve.
O comando
45 rein repete y
é um comentário; assim, a linha 45 é acrescentada à tabela de símbolos como tipo L e atribuída à posição de SML 14.
O comando
50 goto 20
transfere o controle para a linha 20. O número da linha 50 é inserido na tabela de símbolos como tipo L e atribuído à posição de
SML 14. O equivalente do goto em SML é a instrução desvio incondicional (40) que transfere o controle para uma posição de
SML específica. O compilador procura na tabela de símbolos a linha 20 e descobre que ela corresponde à posição de SML 01.
O código de operação (40) é multiplicado por 100 e a posição 01 é somada a ele para produzir a instrução 14 +4001.
O comando
55 rexu exibe resultado
e um comentário; assim, a linha 55 e inserida na tabela de símbolos como tipo L e atribuída à posição de SML 15.
O comando
60 print t
é um comando de saída. O número de linha 60 é inserido na tabela de símbolos

como tipo L e atribuído à posição de SML 15. O
equivalente de print em SML é o código de operação 11 (write). A posição de t é obtida da tabela de símbolos e somada ao
resultado do código de operação multiplicado por 100.
O comando
99 end
é a linha final do programa. O número de linha 99 é armazenado na tabela de símbolos como tipo L e atribuído à posição de SML
16. O comando end produz a instrução de SML +4300 (43 é halt em SML) que é armazenada como a instrução final no array
de memória de SML.
Isso completa a primeira passagem do compilador. Agora, consideraremos a segunda passagem. O array flags é percorrido, procurando valores diferentes de -1. A posição 03 contém 60; assim, o compilador sabe que a instrução 03 está incompleta. O compilador completa a instrução procurando 60 na tabela de símbolos, determinando sua posição na memória e somando a posição à instrução incompleta. Neste caso, a busca determina que a linha 60 corresponde à posição de SML 15; assim, a instrução completa 03 +4215 é produzida, substituindo 03 +4200. O programa em Simple agora foi compilado com sucesso.
Para construir o compilador, você terá que executar cada uma das tarefas seguintes:
a) Modifique o programa simulador do Simpletron que você escreveu no Exercício 5.19, para aceitar como entrada um arquivo especificado pelo usuário (veja Capítulo 14). O simulador deve gravar seus resultados em um arquivo em disco, no mesmo formato que o da tela de saída. Converta o simulador para ser um programa orientado a objetos. Em particular, tome cada parte do hardware um objeto. Organize os tipos de instrução em uma hierarquia de classes usando herança. Então, execute o programa polimorficamente, simplesmente dizendo a cada instrução para executar a si própria com uma mensagem executeInstruction.
b) Modifique o algoritmo de conversão de notação infjxa para pós-fixa do Exercício 15.12, para processar operandos inteiros de vários dígitos e nomes de variávejs de uma única letra. Sugestão: a função strtok da biblioteca padrão pode ser usada para localizar cada constante e variável em uma expressão e as constantes podem ser convertidas de strings para inteiros usando a função atoi da biblioteca padrão. (Nota: a representação de dados da expressão pósfixa deve ser alterada para suportar nomes de variáveis e constantes inteiras).
c) Modifique o algoritmo de avaliação de expressões pós-fixas para processar operandos inteiros de vários dígitos e nomes de variáveis. Também agora, o algoritmo deve implementar o “gancho” discutido acima de forma que instruções de SML sejam produzidas em lugar de avaliar diretamente a expressão. Sugestão: a função da biblioteca padrão strtok pode ser usada para localizar cada constante e variável em uma expressão, e constantes podem ser conver

tidas de strings para inteiros usando a função da biblioteca padrão atoi. (Nota: a representação de dados da expressão pós-fixa deve ser alterada para suportar constantes inteiras e nomes de variáveis.)
d) Construa o compilador. Incorpore as partes (b) e (c) para avaliar expressões em comandos let. Seu programa deve conter uma função que executa a primeira passagem do compilador e uma função que executa a segunda passagem do compilador. Ambas as funções podem chamar outras funções para realizar suas tarefas. Torne seu compilador tão orientado a objetos quanto possível.
15.28 (Otimizando o compilador de Simple) Quando um programa é compilado e convertido para SML, é gerado um conjunto de instruções. Freqüentemente, certas combinações de instruções repetem a si mesmas, normalmente em triplas chamadas de produções. Uma produção consiste normalmente em três instruções tais como carregar somar e armazenar Por exemplo, a Fig. 15.27 ilustra cinco das instruções de SML que foram produzidas na compilação do programa da Fig. 15.25 As três primeiras instruções compõe a produção que somai a y. Note que as instruções 06 e 07 armazenam o valor do acumulador na localização temporária 96 e, então, carregam o valor de volta para o acumulador, de modo que a instrução 08 possa armazenar o valor na localização 98. Freqüentemente, uma produção é seguida por uma instrução de carga para a mesma localização que acabou de ser armazenada. Este código pode ser otimizado eliminando-se a instrução armazenar e a instrução de carga subseqüente, que opera sobre a mesma localização de memória, possibilitando deste modo ao Simpletron executar o programa mais rapidamente. A Fig 15.28 ilustra a SML otimizada para o programa da Fig. 15.25. Note que existem quatro instruções a menos no código otimizado - uma economia de espaço e memória de 25%.
Modifique o compilador para fornecer uma opção para otimizar o código em Simpletron Machine Language que ele
produz. Compare manualmente o código não-otimizado com o código otimizado e calcule a redução percentual.

Fig. 15.28 Código otimizado para o programa da Fig. 15.25.
15.29 (Modficações do compilador de Simple) Execute as seguintes modificações no compilador de Simple. Algumas dessas modificações podem também exigir modificações no programa simulador do Simpletron escrito no Exercício 5.19.

1 04 +2098 (load)	
2 05 +3097 (add)	
3 06 +2196 (store)	
4 07 +2096 (load)	
5 08 +2198 (store)	
Fig. 15.27 Código não-otimizado do programa da Fig. 15.25.	

Localização e	
Programa em Simple Instrução SML	Descrição
5 rem soma 1 a x nenhuma	comentário ignorado
10 input x 00 +1099	lêxparaaposição99
15 rem testa se y == x nenhuma	comentário ignorado
20 if y == x goto 60 01 +2098	carregay (98) no acumulador
02 +3199	subtraix (99) do acumulador
03 +4211	desvia para o endereço 11 se zero
25 rem incrementa y nenhuma	comentário ignorado
30 let y = y + 1 04 +2098	carregayno acumulador
05 +3097	somai (97) ao acumulador
06 +2198	armazena acumulador em y (98)
35 rem soma y a total nenhuma	comentário ignorado
40 let t = t + y 07 +2096	carrega tdo endereço (96)
08 +3098	soma y (98) ao acumulador
09 +2196	armazena acumulador em t (96)
45 rem repete y nenhuma	comentário ignorado
50 goto 20 10 +4001	desvia para o endereço 01
55 rem exibe resultado nenhuma	comentário ignorado
60 print t 11 +1196	envia (96) paraatela
99 end 12 +4300	termina execução

CAPÍTULO 15 - ESTRUTURAS DE DADOS 805

a) Permita que o operador módulo (%) seja usado em comandos let. A Simpletron Machine Language deve ser modificada para incluir uma instrução módulo.

b) Possibilite a exponenciação em um comando let usando A como o operador de exponenciação. A Simpletron Machine Language deve ser modificada para incluir uma instrução de exponenciação.

c) Possibilite que o compilador reconheça letras maiúsculas e minúsculas em comandos de Simple (por exemplo, ‘A’ é equivalente a ‘a’). Nenhuma modificação do simulador é necessária.

d) Possibilite que comandos input leiam valores para diversas variáveis, tal como input x , y. Não são necessárias modificações no simulador do Simpletron.

e) Possibilite que o compilador exiba diversos valores em um único comando de

impressão, tal como print a ,b , c. Não são necessárias modificações no simulador do Simpletron.
O Acrescente recursos de verificação de sintaxe ao compilador, de forma que sejam exibidas mensagens de erro quando forem encontrados erros de sintaxe em um programa em Simple. Nenhuma modificação no simulador do Simpletron é necessária.
g) Possibilite arrays de inteiros. Nenhuma modificação no simulador do Simpletron é necessária.
h) Possibilite sub-rotinas especificadas pelos comandos de Simple gosub e return. O comando gosub passa o controle do programa para uma sub-rotina e o comando return devolve o controle para o comando depois do gosub. Isto é semelhante a uma chamada de função em C++. A mesma sub-rotina pode ser chamada por muitos comandos gosub distribuídos ao longo de um programa. Nenhuma modificação no simulador do Simpletron é necessária.
i) Permita estruturas de repetição da forma
for x = 2 to 10 step 2
comandos simples
next
Este comando for itera de 2 até 10. com um incremento de 2. A linha next marca o fim do corpo da linha for.
Nenhuma modificação no simulador do Simpletron é necessária.
j) Permita estruturas de repetição da forma
for x = 2 to 10
comandos simples
next
Este comando for itera de 2 até 10 com um incremento default de 1. Nenhuma modificação no simulador do
Simpletron é necessária.
k) Permita ao compilador processar a entrada e saída de strings. Isto exige que o simulador do Simpletron seja modificado para processar e armazenar valores de .strings. (Dica: cada palavra do Simpletron pode ser dividida em dois grupos, cada um guardando um inteiro de dois dígitos. Cada inteiro de dois dígitos representa o equivalente decimal em ASCII de um caractere. Acrescente uma instrução de linguagem de máquina que imprimirá um string começando em uma determinada posição de memória do Simpletron. A primeira metade da palavra naquela posição é o número de caracteres no ,string (i.e., o comprimento do string). Cada meia palavra subseqüente contém um caractere em ASCII expresso como dois dígitos decimais. A instrução de linguagem de máquina confere o comprimento e imprime o string traduzindo cada número de dois dígitos para o seu caractere equivalente).
1) Permita que o compilador processe valores de ponto flutuante além de inteiros. O simulador do Simpletron deve também ser modificado para processar valores de ponto flutuante.
15.30 (Um interpretador de Simple) Um interpretador é um programa que lê um comando de um programa em uma linguagem de alto nível, determina a operação a ser executada pelo comando e executa a operação imediatamente,

O programa em linguagem de alto nível não é convertido primeiro para linguagem de máquina. Os interpretadores executam lentamente, porque cada comando encontrado no programa deve primeiro ser decifrado. Se os comandos estão contidos em um laço, os comandos são decifrados toda vez em que são encontrados no laço. As primeiras versões da linguagem de programação BASIC foram implementadas com interpretadores.
Escreva um interpretador para a linguagem Simple discutida no Exercício 15.26. O programa deve usar o conversor de notação infixa para pós-fixa desenvolvido no Exercício 15.12 e o avaliador de expressões em notação pós-fixa desenvolvido no Exercício 15.13 para avaliar expressões em um comando let. As mesmas restrições impostas à linguagem Simple no Exercício 15.26 deveriam ser obedecidas neste programa. Teste o interpretador com os programas em Simple escritos no Exercício 15.26. Compare os resultados da execução desses programas pelo interpretador com os resultados de programas em Simple compilados, executando-os no simulador do Simpletron construído no Exercício 5.19.
15.31 (inserção/exclusão em qualquer lugar de uma lista encadeada) Nosso gabarito de classe para listas encadeadas permitia inserções e exclusões somente no início e no fim da lista encadeada. Esses recursos nos foram convenientes quando usamos

herança privada e composição para produzir um gabarito de classe pilha e um gabarito de classe fila com uma quantidade mínima de código, reutilizando o gabarito da classe lista. Na realidade, as listas encadeadas são mais gerais que aquelas que fornecemos. Modifique o gabarito da classe lista encadeada que desenvolvemos neste capítulo para tratar inserções e exclusões em qualquer lugar da lista.
15.32 (Lista efilas sem ponteiros de fim) Nossa implementação de uma lista encadeada (Fig. 15.3) usou tanto um firstPtr como um lastPtr. O lastPtr era útil para as funções membro insertAtBack e removeFromBack da classe List. A função insertAtBack corresponde à função membro enqueue da classe Queue. Rescreva a classe List de forma que ela não use um lastPtr. Deste modo, quaisquer operações sobre o fim de uma lista devem começar a pesquisar a lista pela frente (início da lista). Isso afeta nossa implementação da classe Queue (Fig. 15.12)?
15.33 Use a versão de composição do programa de pilha (Fig. 15.11) para formar um programa completo de pilha que funcione. Modifique esse programa para tomar mime as funções membro. Compare as duas abordagens. Resuma as vantagens e desvantagens de colocar em linha as funções membro.
15.34 (Desempenho da classificação e pesquisa em árvores binárias) Um problema com a classificação de árvores binárias é que a ordem em que os dados são inseridos afeta a forma da árvore - para a mesma coleção de dados, diferentes ordens podem produzir árvores binárias de formas drasticamente diferentes. O desempenho de classificação da árvore binária e dos algoritmos de

busca é sensível à forma da árvore binária. Qual forma teria uma árvore binária se seus dados fossem inseridos em ordem crescente? E em ordem decrescente? Qual forma a árvore deveria ter para atingir o máximo desempenho de pesquisa?

15.35 (Listas indexadas) Conforme apresentado no texto, listas encadeadas devem ser pesquisadas seqüencialmente. Para listas grandes, isso pode resultar em um desempenho fraco. Uma técnica comum para melhorar o desempenho da busca em listas é criar e manter um índice para alista. Um índice é um conjunto de ponteiros para vários lugares-chave na lista. Por exemplo, uma aplicação que pesquisa uma lista grande de nomes poderia melhorar seu desempenho criando um índice com 26 entradas - uma para cada letra do alfabeto. Uma operação de busca de um sobrenome começando com 'Y' iria então primeiro pesquisar o índice para determinar onde começam as entradas com 'Y' e, então, "saltaria" para a lista naquele ponto, pesquisando seqüencialmente a partir daí até encontrar o nome desejado. Isso seria muito mais rápido do que pesquisar a lista encadeada desde o início. Use a classe List da Fig. 15.3 como a base para uma classe indexedList. Escreva um programa que demonstre o funcionamento de listas indexadas. Não deixe de incluir as funções membro insertlnlndexedList, searchlndexedList e deieteFromlndexedList.

Bits	,	Caracteres, Strings e Estruturas
Objetivos		
• Ser capaz de criar e usar estruturas. • Ser capaz de passar estruturas para funções através de chamadas por valor e chamadas por referência. • Manipular dados com os operadores sobre bits e criar campos de bits para armazenar dados de forma compacta. • Ser capaz de usar as funções da biblioteca de manipulação de caracteres (cctype). • Ser capaz de usar as funções de conversão de strings da biblioteca de utilitários		

genéricos (cstdlib). • Ser capaz de usar as funções de processamento de strings da biblioteca de manipulação de strings (cstring). • Apreciar o poder das bibliotecas de funções como meio de conseguir a reutilização de software.		

The sarne old charitable lie
Repeated as the years scoot by
Perpetually makes a hit -
“You really haven ‘t changed a bit!”
Margaret Fishback
The chi ef defect of Henry King
Was chewing little bits ofstring.
Hilaire Belloc
A escrita vigorosa é concisa. Urna sentença não deve conter palavras desnecessárias; um parágrafo, sentenças desnecessárias.
William Strunk,

Visão Geral
16.1 Introdução

16.2 Definições de estrutura
16.3 Inicializando estruturas
16.4 Usando estruturas com funções
16.5 typedef
16.6 Exemplo: uma simulação de alto desempenho do embaralhamento e distribuição de cartas
16.7 Operadores sobre bits
16.8 Campos de bits
16.9 A biblioteca de manipulação de caracteres
16.10 Funções de conversão de strings
16.11 Funções de pesquisa da biblioteca de manipulação de strings
16.12 Funções de memória da biblioteca de manipulação de strings
16.13 Uma outra função da biblioteca de manipulação de strings
Resumo• Terminologia Erros comuns de programação Boas práticas de programação • Dicas de desempenho . Dicas de portabilidade Exercícios de auto-revisão • Respostas aos exercícios de auto-revisão Exercícios
16.1 Introdução
Neste capítulo, falamos mais sobre estruturas e, então, discutimos a manipulação de bits, caracteres e strings. Muitas das técnicas que apresentamos seguem o estilo empregado em C e são incluídas para ajudar o programador de C++ que tenha que trabalhar com código legado em C.
As estruturas podem conter variáveis de muitos tipos de dados diferentes - em contraste com arrays, que contêm somente elementos do mesmo tipo de dados. Este fato e a maior parte do que falamos sobre estruturas nas próximaspáginas aplicam-se também a classes. Repetindo: a principal diferença entre estruturas e classes em
é que membros de estrutura tem acesso default public e membros de classes tem acesso default private. As estruturas são usadas comumente para definir registros de dados para serem armazenados em arquivos (veja o Capítulo 14, “Processamento de arquivos”). Ponteiros e estruturas facilitam a formação de estruturas de dados mais complexas, tais como listas encadeadas, filas, pilhas e árvores (ver Capítulo 15, “Estruturas de dados”). Discutiremos como declarar estruturas, inicializá-las e passá-las para funções. Então, apresentaremos uma simulação de alto desempenho de embaralhamento e distribuição de cartas.
16.2 Definições de estrutura
Considere a seguinte definição de estrutura:
```
struct Card {
char *face;
char *sujt;
```
A palavra-chave struct inicia a definição da estrutura Card. O identificador Card é o nome da estrutura e é usado em C++ para declarar variáveis do tipo estrutura (em C, o nome de tipo da estrutura precedente é struct Card). Neste exemplo, o tipo de estrutura é Card. Dados (e possivelmente funções - da mesma maneira que com classes) declarados dentro das chaves da definição da estrutura são os membros da estrutura. Membros da mesma

estrutura devem ter nomes únicos, mas duas estruturas diferentes podem conter membros com o mesmo nome sem risco de conflito. Cada definição de estrutura deve terminar com um ponto-e-vírgula.

Erro comum de programação 16.1

Esquecer o ponto-e-vírgula que termina uma definição de estrutura.

A definição de Card contém dois membros do tipo char* - face e suit. Os membros da estrutura podem ser variáveis dos tipos de dados básicos ou primitivos (por exemplo: int, float. etc.), ou agregados, tais como arrays e outras estruturas. Como vimos no Capítulo 4, cada elemento de um array deve ser do mesmo tipo. Membros de dados de uma estrutura, porém, podem ser de tipos de dados diferentes. Por exemplo, uma estrutura Employee poderia conter membros do tipo string de caracteres para o nome e sobrenome, um membro int para a idade do empregado, um membro char contendo M' ou 'F' para o sexo do empregado, um membro double para o salário-hora do empregado, e assim por diante.

Uma estrutura não pode conter uma instância de si mesma. Por exemplo, uma variável estrutura Card não pode ser declarada na definição da estrutura Card. Um ponteiro para uma estrutura Card. porém, pode ser incluído. Uma estrutura que contém um membro que é um ponteiro para o mesmo tipo de estrutura é chamada de uma estrutura auto-referente. As estruturas auto-referentes são usadas no Capítulo 15 para construir vários tipos de estruturas de dados encadeadas.

A definição de estrutura precedente não reserva qualquer espaço na memória; em vez disso, a definição cria
um novo tipo de dados que é usado para declarar variáveis do tipo estrutura. As variáveis estrutura são declaradas como as variáveis de outros tipos. A declaração

```
Card oneCard, deck[ 52 ], *cptr;
```

declara oneCard como uma variável estrutura do tipo Card, deck como sendo um array com 52 elementos do tipo Card e cPtr como um ponteiro para uma estrutura Card. As variáveis de um dado tipo de estrutura também podem ser declaradas colocando-se uma lista de nomes de variáveis separados por vírgulas entre a chave de término da definição da estrutura e o ponto-e-vírgula que termina a definição da estrutura. Por exemplo, a declaração precedente poderia ter sido incorporada à definição da estrutura Card como segue:

```
struct Card {
char *face;
char *suit;
} oneCard, deck[ 52 ], *cptr;
```

O nome de estrutura é opcional. Se uma definição de estrutura não contém um nome de estrutura, variáveis do tipo da estrutura somente podem ser declaradas na definição da estrutura - não em uma declaração separada.

Boa prática de programação 16.1

Forneça um nome de estrutura quando criar um tipo de estrutura. O nome de estrutura é conveniente para declarar novas variáveis do tipo da estrutura mais

àfrente no programa e é obrigatório se a estrutura for passada como um parâmetro para uma função.
As únicas operações primitivas válidas que podem ser executadas sobre estruturas são atribuir uma estrutura a uma estrutura do mesmo tipo, pegar o endereço (&) de uma estrutura, acessar os membros de uma estrutura (ver Capítulo 6, “Classes e abstração de dados”) e usar o operador sizeof para determinar o tamanho de uma estrutura. Como acontece com classes, a maioria dos operadores podem ser sobrecarregados para trabalhar com objetos de um tipo estrutura.
Os membros de estruturas não estão necessariamente armazenados em bytes sucessivos de memória. As vezes, existem “buracos” em uma estrutura, porque os computadores podem armazenar tipos de dados específicos somente em certos limites de memória, tais como início de uma meia palavra, palavra ou palavra dupla. Uma palavra é uma unidade padrão de memória usada para armazenar dados em um computador - normalmente 2 bytes ou 4

bytes. Considere a seguinte definição de estrutura em que são declaradas as variáveis estrutura (na verdade, objetos) samplel e sample2 do tipo Example:

```
structExample {
char c;
int i;
} samplel, sample2;
```

Um computador com palavras de 2 bytes pode exigir que cada um dos membros de Example seja alinhado em um limite de palavra, i.e., no princípio de uma palavra (isto é dependente da máquina). A Fig. 16.1 mostra um exemplo de alinhamento de memória para um objeto de tipo Example ao qual foram atribuídos o caractere ‘a’ e o inteiro 97 (são mostradas as representações dos valores em binário). Se os membros são armazenados no início de limites de palavras, existe um buraco de 1 byte (byte 1 na figura) na memória para objetos do tipo Example. O valor do buraco de 1 byte é indefinido. Se os valores dos membros samplel e sample2 são de fato iguais, a comparação das estruturas não é necessariamente igual, porque os buracos indefinidos de 1 byte provavelmente não conterão valores idênticos.
Erro comum de programação 16.2
Comparar estruturas é um erro de sintaxe por causa dos diferentes requisitos de alinhamento nos diversos
sistemas.
Dica de portabilidade 16.1
_____ Como o tamanho dos itens de dados de um tipo particular é dependente de máquina e como considerações de alinhamento de memória são dependentes de máquina, assim também o é a representação de uma estrutura.

Fig. 16.1 Um alinhamento de memória possível para uma variável do tipo Example mostrando uma área indefinida na memória.

16.3 Inicializando estruturas

As estruturas podem ser inicializadas usando-se listas de inicializadores, como é feito com arrays. Para inicializar uma estrutura, coloque em seguida ao nome da variável na declaração de estrutura um sinal de igual e uma lista de inicializadores entre chaves, separados por vírgulas. Por exemplo, a declaração

Card oneCard = { “Três”, “Copas”);

cria uma variável oneCard do tipo estrutura Card (como definida anteriormente) e inicializa o membro face com “Três” e o membro suit com “Copas”. Se existem menos inicializadores na lista do que membros na estrutura, os membros restantes são automaticamente inicializados com 0. Variáveis estrutura declaradas fora de uma definição de função (i.e., externamente) são inicializadas com 0, se não forem explicitamente inicializadas na declaração externa. As variáveis estrutura podem também ser inicializadas em comandos de atribuição, atribuindo- se a elas uma variável estrutura do mesmo tipo ou atribuindo valores aos membros de dados individuais da estrutura.

16.4 Usando estruturas com funções

Existem dois modos de passar as informações contidas em estruturas para funções. Você pode passar a estrutura inteira ou pode passar membros isolados da estrutura. Por default, os dados (exceto membros individuais de um

Byte O 1 2 3
01100001 00000000
01100001

array) são passados através de chamadas por valor. As estruturas e seus membros podem também ser passados através de chamadas por referência, passando ou referências ou ponteiros.

Para passar uma estrutura através de uma chamada por referência, passe o endereço da variável estrutura ou

uma referência para a variável estrutura. Arrays de estruturas - como todos os outros arrays - são passados através de chamadas por referência.

No Capítulo 4, afirmamos que um array poderia ser passado através de uma chamada por valor usando uma estrutura. Para passar um array por uma chamada por valor, crie uma estrutura (ou uma classe) com o array como um membro. Como estruturas são passadas através de chamadas por valor, o array é passado através de uma chamada por valor.

IErro com um de programação 16.3

Assumir que estruturas, como arrays, são passadas através de chamadas por referência e tentar modificar os valores da estrutura do chamador na função chamada.

Dica de desempenho 16.1

_____ Passar estruturas (especialmente estruturas grandes) através de chamada por refrréncia é mais eficiente do que passar estruturas através de chamadas por valor (o que exige a cópia da estrutura inteira).

16.5 typedef
A palavra-chave typedef fornece um mecanismo para criar sinônimos (ou nomes alternativos) para tipos de dados previamente definidos. Os nomes para tipos estrutura são freqüentemente definidos com typedef para criar nomes de tipo menores ou mais legíveis. Por exemplo, o comando
typedef Card *CardPtr;
define o novo nome de tipo CardPtr como um sinônimo para o tipo Card .
Boa prática de programação 16.2
Comece nomes typedef com maiúsculas, para enfatizar que estes nomes são sinônimos de outros nomes
de tipo.
Criar um novo nome com typedef não cria um novo tipo de dados; typedef simplesmente cria um novo nome de tipo que pode ser então usado no programa como um nome alternativo para um nome de tipo existente.
Sinônimos para tipos de dados primitivos podem ser criados com typedef. Por exemplo, um programa que necessita de inteiros de 4 bytes pode usar o tipo int em um sistema e o tipo long int em outro sistema que tem inteiros de 2 bytes. Os programas projetados pensando em portabilidade podem usar typedef para criar um nome alternativo, tal como Integer, para inteiros de 4 bytes. Integer pode então ser tornado sinônimo de int em sistemas com inteiros de 4 bytes e pode ser tornado sinônimo de long int em sistemas com inteiros de 2 bytes onde valores long int ocupam 4 bytes. Então, para escrever programas portáveis, o programador simplesmente declara todas as variáveis inteiras de 4 bytes como sendo do tipo Integer.
Dica de portabilidade 16.2
_____ Usar typedef pode ajudar a tornar um programa mais portável.
16.6 Exemplo: uma simulação de alto desempenho do
embaralhamento e distribuição de cartas
O programa na Fig. 16.2 se baseia na simulação do embaralhamento e distribuição de cartas discutido no Capítulo 5.
O programa representa o baralho de cartas como um array de estruturas e usa algoritmos de alto desempenho para
embaralhamento e distribuição. A saída é mostrada na Fig. 16.3.

60
1 II Fig. 16.2: figl6O2.cpp 61
2 II Programa para embaralhar e distribuir cartas usando estruturas 62
3 #include <iostream> 63
64
5 using std::cout; 65
6 using std::cin; 66
7 using std: :endl; 67
8 using std: :ios; 68
69
10 #include <iomanip> 70

```
11 71
12 using std::setiosflags; -
13 using std::setw; Fig.
14
15 #include <cstdlib>
16 #include <ctime> Nop
17 Ásal
18 struct Card { deal
19 char *face; ment
20 char *suit;
21 entre
22 namt
23 void fillDeck( Card * const, char *E], char *[] ); eoa
24 void shuffle( Card * const ); com
25 void deal ( Card * const ); própi
26 array
27 int main()
28 {
29 Card deck[ 52 ];
30 char *face [] = ( Ás", "Dois", "Três", "Quatro",
31 "Cinco", "Seis", "Sete", "Oito",
32 "Nove", "Dez", "Valete", "Dama",
33 "Rei" };
34 char *suit[] = { "Copas", "Ouros",
35 "Paus", "Espadas" };
36
37 srand( time( O ) ); // randomiza
38 fillDeck( deck, face, suit );
39 shuffle( deck );
40 deal( deck );
41 return 0;
42 }
43
44 void fillDeck( Card * const wDeck, char *wFace[),
45 char *wsuit[]
46 {
47 for ( int i = 0; i < 52; i++
48 wDeck[ i ) .face = wFace[ i % 13 ];
49 wDeck[ i ) .suit = wSuit[ i / 13
50
51 }
52
53 void shuffle( Card * const wDeck
54
55 for ( int i = 0; i < 52; i++
56 int j = rand() % 52;
```

```
57 Card temp = wDeck[ i ];
58 wDeck[ i ] = wDeck[ j ];
59 wDeck[ j ) = temp;
```
Fig. 16.2 Simulação de alto desempenho de embaralhamento e distribuição de cartas (parte 1 de 2). Fig.


```
60 }
61 }
62
63 void deal( Card * const wDeck
64
65 for ( int i 0; i < 52; i++
66 cout « setiosflags( ios::right
67 « setw( 5 ) « wDeck[ i ] .face « de
68 « setiosflags ( ios: : left
69 « setw( 8 ) « wDeck[ i ].suit
70 «((i+1)%2?\t' :
71
```

Fig. 16.2 Simulação de alto desempenho de embaralhamento e distribuição de cartas (parte 2 de 2).
No programa, a função fillDeck inicializa o array Card segundo a ordem de strings de caracteres representando As a Rei de cada naipe (“suit”). O array Cardé passado para a função shuffle. onde o algoritmo de embaralhamento de alto desempenho está implementado. A função shuffle recebe um array de 52 estruturas Card como argumento. A função executa um laço através de todas as 52 cartas (índices de array O a 51). Para cada carta, um número entre O e SI é escolhido aleatoriamente. Em seguida, a estrutura Card atual e a estrutura Card selecionada aleatoriamente são trocadas de posição no array. Um total de 52 trocas são feitas em uma única passagem de todo o array e o array de estruturas Card está embaralhado! Este algoritmo não sofre do problema de adiamento indefinido como o algoritmo de embaralhamento apresentado no Capítulo 5. Como as estruturas Card foram trocadas no próprio array, o algoritmo de alto desempenho implementado na função deal exige somente uma passagem pelo array para distribuir as cartas embaralhadas.

Erro comum de programação 16.4

ras.

Esquecer de incluir o índice de array quando refrrenciar estruturas individuais em um array de estrutu Oito

Fig. 16.3 Saída da simulação de alto desempenho de embaralhamento e distribuição de cartas.

	de	Ouros
Oito	de	Paus
Sete	de	Copas
As	de	Paus
Dois	de	Espadas
Sete	de	Espadas
Valete	de	Paus
Rei	de	Copas
Três	de	Copas
Três	de	Paus
Dez	de	Copas
Dez	de	Paus
Seis	de	Paus
Seis	de	Copas
Nove	de	Ouros
Valete	de	Espadas
Rei	de	Ouros
Nove	de	Espadas
Seis	de	Espadas
Dama	de	Ouros
As	de	Espadas
Rei	d	Paus

	e	
Rei	de	Espadas
Dama	de	Copas
Quatro	de	Espadas
Quatro	de	Paus

Ás	de	Copas
Cinco	de	Espadas
Dois	de	Ouros
Dez	de	Ouros
Seis	de	Ouros
Dois	de	Paus
Dez	de	Espadas
Valete	de	Ouros
Três	de	Ouros
Nove	de	Paus
Dois	de	Copas
Sete	de	Ouros
Dama	de	Espadas
Tres	de	Espadas
As	de	Ouros
Cinco	de	Paus
Sete	de	Paus
Quat	d	Copa

ro	e	s
Oito	de	Espadas
Cinco	de	Ouros
Nove	de	Copas
Cinco	de	Copas
Quatro	de	Ouros
Oito	de	Copas
Valete	de	Copas
Dama	de	Paus

16.7 Operadores sobre bits

C++ fornece recursos de manipulação de bits abrangentes para os programadores que necessitam descer ao chamado nível de “bits e bytes”. Sistemas operacionais, software de equipamentos de teste, software de rede e muitos outros tipos de software exigem que o programador se comunique “diretamente com o hardware”. Nestas e nas próximas seções, discutimos os recursos de manipulação de bits de C++. Introduzimos cada um dos muitos operadores sobre bits de C++ e discutimos como economizar memória usando campos de bits.

Todos os dados são representados internamente pelos computadores como seqüências de bits. Cada bit pode assumir o valor O ou o valor 1. Na maioria dos sistemas, uma seqüência de 8 bits forma um byte - a unidade padrão de armazenamento para uma variável do tipo char. Outros tipos de dados são armazenados em quantidades maiores de byres. Os operadores sobre bits são usados para manipular os bits de operandos integrais (char, short, int e long; tanto signed como uns igned). Inteiros uns igned são normalmente usados com os operadores sobre bits.

Dica de portabilidade 16.3

_____ As manipulações de dados sobre bits são dependentes de máquina.

Note que as discussões de operadores sobre bits nesta seção mostram as representações binárias dos operandos inteiros. Para uma explicação detalhada da notação binária (também chamada de sistema de numeração de base 2), veja o apêndice C, “Sistemas de numeração”. Por causa da natureza dependente de máquina das manipulações sobre bits, estes programas podem não funcionar em seu sistema.

Os operadores sobre bits são: AND (&) sobre bits, OR inclusivo (1) sobre bits, OR exclusivo (A) sobre bits, deslocamento à esquerda («), deslocamento à

direita (») e complemento (-). (Note que temos usado &. « e» para outras finalidades. Este é um exemplo clássico de sobrecarga de operador). Os operadores sobre bits AND, OR inclusivo e OR exclusivo comparam seus dois operandos bit a bit. O operador sobre bits AND define cada bit no resultado como 1 se o bit correspondente em ambos os operandos é 1. O operador sobre bits OR inclusivo define cada bit no resultado como 1 se o bit correspondente em qualquer um dos (ou ambos os) operandos é 1. O operador sobre bits OR exclusivo define cada bit no resultado como 1 se o bit correspondente em exatamente um operando é 1. O operador de deslocamento à esquerda move os bits de seu operando esquerdo para a esquerda pelo número de bits especificados em seu operando direito. O operador de deslocamento à direita move os bits em seu operando esquerdo para a direita pelo número de bits especificados em seu operando direito. O operador sobre bits complemento transforma todos os bits O em seu operando em 1 no resultado e todos os bits 1 em O no resultado. As discussões detalhadas de cada operador sobre bits aparecem nos exemplos seguintes. Os operadores sobre bits estão resumidos na Fig. 16.4.

Ao se usar os operadores sobre bits, é útil se imprimir valores em sua representação binária para ilustrar os efeitos precisos destes operadores. A Fig. 16.5 imprime um inteiro unsigned em sua representação binária, em grupos de oito bits. A função displayBits usa o operador AND sobre bits para combinar a variável value com a constante MASK. Freqüentemente, o operador AND sobre bits é usado com um operando chamado de máscara - um valor inteiro com bits específicos colocados em 1. Máscaras são usadas para esconder alguns bits em um valor ao mesmo tempo que selecionam outros bits. Em displayBits, a máscara constante MASK recebe o valor

1 « SHIFT

A constante SHIFT recebe o resultado de

8 * sizeof(unsigned) - 1

Operador Nome Descrição

& AND sobre bits Os bits no resultado são inicializados com 1 se os bits corres pondente nos dois operandos são ambos 1.

Fig. 16.4 Os operadores sobre bits (parte 1 de 2).


```
1 II Fig. 16.5: figl6O5.cpp
2 // Imprimindo um inteiro sem sinal em binário
3 #include <iostream>
4
5 using std::cout;
6 using std::cin;
7
8 #include <iomanip>
```

```
9
10 using std::setw;
li using std: :endl;
12
13 void displayBits ( unsigned );
14
15 int main()
16 {
17 unsigned x;
18
19 cout « “Digite um inteiro sem sinal:
20 cm » x;
21 displayBits ( x
22 return 0;
23
24
25 void displayBits( unsigned value
26 {
27 const int SHIFT = 8 * sizeof( unsigned ) - 1;
28 const unsigned MASK = 1 « SHIFT;
29
30 cout « setw( 7 ) « value « “ =
31
32 for ( unsigned e 1; c <= SHIFT + 1; c++
33 cout « ( value & MASK ? 1’ : ‘0
```

Fig. 16.5 Imprimindo um inteiro sem sinal em binário (parte 1 de 2).

Operador		Nome	Descrição
	1	OR inclusivo sobre bits	Os bits no resultado são inicializados com 1 se pelo menos um
			dos bits correspondentes no dois operandos é 1.
	A	OR exclusivo sobre bits	Os bits no resultado são inicializados com 1 se exatamente um dos bits correspondentes nos dois operandos é 1.
	«	deslocamento para a esquerda	Desloca para a esquerda os bits do primeiro operando pelo número de bits especificado pelo segundo operando; preenche a partir da direita com bits O.
	»	deslocamento para a direita com extensão de sinal	Desloca para a direita os bits do primeiro operando pelo número de bits especificado pelo segundo operando; a forma de preencher a partir da esquerda depende da máquina.
	-	complemento unário	Todos os bits O são inicializados com is e todos os bits is são
			inicializados com 0.

Fig.	16.4	s operadores sobre bits (parte 2 d	e 2).

34 value «= 1;
35
36 if(c%8==0)
37 cout«
38
39
40 cout « endl;
41
Digite um inteiro sem sinal: 65000
65000 = 00000000 00000000 11111101 11101000
Fig. 16.5 Imprimindo um inteiro sem sinal em binário (parte 2 de 2).
Esta expressão multiplica a quantidade de bytes necessários para armazenar um valor uns igned por 8 para obter o número total de bits e. então, subtrai 1. A representação de 1 « SHIFT é
10000000 00000000 00000000 00000000
O operador de deslocamento à esquerda desloca o valor 1 do bit menos significativo (o mais à direita) para o bit mais significativo (o mais à esquerda) em MASK e preenche com bits O a partir da direita, O comando
cout « (value & MASK ? 1 : 0');
determina se um 1 ou um O deve ser impresso para o bit atualmente mais à esquerda da variável value. Suponha queavariávelvaluecontém 65000 (00000000 00000000 11111101 1ii0i000).QuandovalueeMASK são combinados usando &, todos os bits, exceto o bit de mais alta ordem da variável value. são "mascarados" (escondidos), porque qualquer bit combinado por AND comO fica 0. Se o bit mais à esquerda é 1. value & MASK é avaliada como
00000000 00000000 11111101 11101000 (value)
10000000 00000000 00000000 00000000 (MASK)
00000000 00000000 00000000 00000000 (value & MASK)
que é interpretado como false. e O é impresso. A variável value é então deslocada para a esquerda em um bit, pela expressão value «= 1 (isto é equivalente à atribuição value = value « 1). Estes passos são repetidos para cada bit da variável value. Em algum momento, um bit com o valor 1 é deslocado para a posição mais à esquerda, e a manipulação dos bits é como segue:
11111101 11101000 00000000 00000000 (value)
10000000 00000000 00000000 00000000 (MASK)
10000000 00000000 00000000 00000000 (value & MASK)
Como os dois bits da esquerda são is, é impresso o valor 1. A Fig. 16.6 resume os resultados de combinar dois bits com o operador AND sobre bits.
Erro comum de programação 16.5
Usar o operadorAND lógico (&&) como o operadorAND sobre bits (&), e vice-

versa.

Fig. 16.6 Resultados de combinar dois bits com o operador AND sobre bits (&).

O programa da Fig. 16.7 demonstra o uso do operador AND sobre bits, o operador OR inclusivo sobre bits, o operador OR exclusivo sobre bits e o operador complemento sobre bits. A função displayBits imprime os valores inteiros unsigned. A saída é mostrada na Fig. 16.8.

```
1 II Fig. 16.7: figl6_07.cpp
2 // Usando os operadores sobre bits AND, OR inclusivo,
3 II OR exclusivo e complemento.
4 #include <iostream>
5
6 using std::cout;
7 using std::cin;
8
9 #include <iomanip>
10
11 using std: :endl;
12 using std::setw;
13
14 void displayBits ( unsigned );
15
16 int main()
17
18 unsigned numberl, number2, mask, setBits;
19
20 nuxnberl = 2179876355;
21 mask = 1;
22 cout « “O resultado de combinar os seguintes\n’;
23 displayBits( nuitiberl );
24 displayBits ( mask );
25 cout « ‘usando o operador AND sobre bits, & é\n”;
26 displayBits( numberl & mask );
27
28 nuniberi = 15;
29 setBits = 241;
30 cout « “\nO resultado de combinar os seguintes\n”;
31 displayBits ( nuinberl );
32 displayBits ( setBits );
33 cout « “usando o operador OR inclusivo sobre bits, I é\n”;
34 displayBits( numberl setBits );
35
36 nuinberl = 139;
37 number2 = 199;
```

```
38 cout « “\nO resultado de combinar os seguintes\n”;
39 displayBits( numberl );
40 displayBits ( number2 );
```

Fig. 16.7 Usando os operadores sobre bits AND, OR inclusivo, OR exclusivo e complemento (parte 1 de 2).

Bil 1	Bit 2	Bit 1 & Bit 2
o	o	o
1	O	O
O	1	1
1	1	1


```
41 cout « “usando o operador OR exclusivo sobre bits, “ é\n’;
42 displayBits ( numberl A nuniber2 );
43
44 numberl = 21845;
45 cout « “\nO complemento de um de\n’;
46 displayBits( numberi );
47 cout « “é « endl;
48 displayBits( -number1 );
49
50 return 0;
51
52
53 void displayBits( unsigned value
54 {
55 const int SHIFT = 8 * sizeof( unsigned ) - 1;
56 const unsigned MASK = 1 « SHIFT;
57
58 cout « setw( 10 ) « value « =
59
60 for ( unsigned c = 1; c <= SHIFT + 1; c++ ) {
61 cout « ( value & MASK ? 1 : ‘0’ );
62 value «= 1;
63
64 if(c%8==0)
65 cout«
66 )
67
68 cout « endl;
69 }
```

Fig. 16.7 Usando os operadores sobre bits AND, OR inclusivo, OR exclusivo e complemento (parte 2 de 2).

O resultado de combinar os seguintes
2179876355 = 10000001 11101110 01000110 00000011
1 = 00000000 00000000 00000000 00000001
usando o operador AND sobre bits, & é
1 = 00000000 00000000 00000000 00000001
O resultado de combinar os seguintes
15 = 00000000 00000000 00000000 00001111
241 = 00000000 00000000 00000000 11110001
usando o operador OR inclusivo sobre bits, 1 é
255 = 00000000 00000000 00000000 11111111
O resultado de combinar os seguintes
139 = 00000000 00000000 00000000 10001011
199 = 00000000 00000000 00000000 11000111
usando o operador OR exclusivo sobre bits, A
76 = 00000000 00000000 00000000 01001100
O complemento de um de
21845 = 00000000 00000000 01010101 01010101
e
4294945450 = 11111111 11111111 10101010 10101010
Fig. 16.8 Saída do programa da Fig. 16.7.

CAPÍTULO 16- BITs, CARACTERES, STRINGS E ESTRUTURAS 819
Na Fig. 16.7, o valor 1 (00000000 00000000 00000000 00000001) é atribuído à variável niask e o valor
2179876355(10000001 11101110 01000110 00000001)é atribuído àvariável nuniberi. Quando mask
e nuxnberl são combinados usando-se o operador AND sobre bits (&), na expressão nuxnberl & mask. o resultadoé 00000000 00000000 00000000 00000001.Todososbits,excetoobitdeordemmaisbaixanavariável
numberi, são “mascarados” (escondidos) pela operação de AND com a constante MASK.
O operador sobre bits OR inclusivo é usado para colocar bits específicos de um operando em 1. Na Fig. 16.7, é atribuído 15 (00000000 00000000 00000000 00001111) à variável nuniberleé atribuído 241 (00000000 00000000 00000000 11110001)àvariávelsetBits.QuandonumberlesetBitssãocombinadosusando-seooperadorsobrebits ORnaexpressãonuxnberl 1 setBits, oresultadoé255 (00000000 00000000 00000000 11111111). A Fig. 16.9 resume os resultados de combinar dois bits com o operador sobre bits OR inclusivo.

Bit 1 Bit 2 Bit 1 1 Bit 2

Fig. 16.9 Resultados de combinar dois bits com o operador sobre bits OR inclusivo (1).
Erro comum de programação 16.6
Usar o operador OR lógico (1 1) em vez do operador OR sobre bits (1) e vice-versa.

O operador sobre bits OR exclusivo (“) coloca cada bit no resultado em 1 se exatamente um dos bits correspondentes em seus dois operandos for 1. NaFig. 16.7, são atribuídos os valores 139 (00000000 00000000 00000000 10001011) e199 (00000000 00000000 00000000 11000111) àsvariáveisnuniberlenumber2. Quando estas variáveis forem combinadas com o operador OR exclusivo na expressão nuniberi number2, o resultado é 00000000 00000000 00000000 01001100. A Fig. 16.10 resume os resultados de combinar dois bits com o operador sobre bits OR exclusivo.

Bit 1 Bit 2 Bit 1 A Bit 2

Fig. 16.10 Resultados de combinar dois bits com o operador sobre bits OR exclusivo (A).

O operador sobre bits complemento (-) muda todos os bits 1 em seu operando para O no resultado e muda todos os bits O no resultado para 1- também chamado de “complementar para um o valor”. Na Fig. 16.7, à variável nuniberi éatribuídoovalor2l845 (00000000 00000000 01010101 01010101).Quandoaexpressão -numberi éavaliada,oresultadoé(11111111 11111111 10101010 10101010).
O programa da Fig. 16.11 demonstra o operador de deslocamento à esquerda («) e o operador de deslocamento à direita (»).A função displayBits é usada para imprimir os valores inteiros unsigned.

```
1 // Fig. 16.11: figl6_11.cpp
2 II Usando os operadores de deslocamento sobre bits
3 #irzclude <iostream>
```

Fig. 16.11 Usando os operadores de deslocamento sobre bits (parte 1 de 2).


```
4
5 using std::cout;
6 using std::cin;
7 using std::endl;
8
9 #include <iomanip>
10
11 using std::setw;
12
13 void displayBits( unsigned );
14
15 int main()
16
17 unsigned numberi = 960;
18
```

```
19 cout « ‘O resultado de deslocar para a esquerda\n”;
20 displayBits( numberl );
21 cout « “8 bits usando o operador de
22 « deslocamento para a esquerda é\n;
23 displayBits ( nuinberl « 8 );
24 cout « “\nO resultado de deslocar para a direita\n;
25 displayBits( numberi );
26 cout « “8 bits usando o operador de
27 « “deslocamento para a esquerda é\n”;
28 displayBits ( numberi » 8 );
29 return 0;
30 }
31
32 void displayBits( unsigned value
33
34 const int SHIFT = 8 * sizeof( unsigned ) - 1;
35 const unsigned MASK = 1 « SHIFT;
36
37 cout « setw( 7 ) « value « “ =
38
39 for ( unsigned c = 1; c <= SHIFT + 1; c++
40 cout « ( value & MASK ? ‘1’ : ‘0’ );
41 value «= 1;
42
43 if(c%8==0)
44 cout« ‘
45 }
46
47 cout « endl;
48
O resultado de deslocar para a esquerda
960 = 00000000 00000000 00000011 11000000
8 bits usando o operador de deslocamento para a esquerda é
245760 = 00000000 00000011 11000000 00000000
O resultado de deslocar para a direita
960 = 00000000 00000000 00000011 11000000
8 bits usando o operador de deslocamento para a direita é
3 = 00000000 00000000 00000000 00000011
```

Fig. 16.11 Usando os operadores de deslocamento sobre bits (parte 2 de 2).

O operador de deslocamento à esquerda («) desloca os bits de seu operando esquerdo para a esquerda pelo número
de bits especificados em seu operando direito. Os bits liberados à direita são substituídos com Os; os bits à esquerda

deslocados parafora são perdidos. No programadafig. 16.11,0 valor 960 (00000000 00000000 00000011
11000000) é atribuído à variável nuiTiberi. O resultado de deslocar a variável numberi à esquerda 8 bits na
expressãonumberl « 8é245760 (00000000 00000011 11000000 00000000).
O operador de deslocamento à direita (») desloca os bits de seu operando esquerdo para a direita pelo número de bits especificados em seu operando direito. Executar um deslocamento à direita em um inteiro unsigned faz com que os bits desocupados na esquerda sejam substituídos por Os; os is deslocados para fora à direita são perdidos. No programa da Fig. 16.11, o resultado de fazer um deslocamento à direita de nuiriberi na expressão nuniberl » 8é3 (00000000 00000000 00000000 00000011).
Erro comum de programação 16.7
O resultado de fazer um deslocamento de um valor é indefinido se o operando direito for negativo ou se o operando direito for maior que o número de bits em que o operando esquerdo está armazenado.
Dica de portabilidade 16.4
_____ O resultado de fazer um deslocamento à direita de um valor com sinal é dependente de máquina. Algumas máquinas preenchem com zero e outras usam o bit de sinal.
Cada operador sobre bits (exceto o operador de complemento) tem um operador de atribuição correspondente. Estes operadores de atribuição sobre bits são mostrados na Fig. 16.12 e são usados de maneira semelhante aos operadores de atribuição aritméticos introduzidos no Capítulo 2.

Fig. 16.12 Os operadores de atribuição sobre bits.

A Fig. 16.13 mostra a precedência e associatividade dos vários operadores introduzidos até aqui no texto. Eles são mostrados, de cima para baixo, em ordem decrescente de precedência.

Fig. 16.13 Precedência e associatividade de operadores (parte 1 de 2).

Operadores	de atribuição sobre bits
&	Operador de atribuição AND sobre bits.
	Operador de atribuição OR inclusivo sobre bits.
A	Operador de atribuição OR exclusivo sobre bits.
«	Operador de atribuição de deslocamento à esquerda.
»=	Operador de atribuição de deslocamento à direita com extensão de sinal.

Operadores		Associatividad	Tipo

		e	
(unário; direita para esquerda)	: (binário; esquerda para direita)	esquerda para a direita	mais alta
O [1 . -> ++ --		esquerda para a direita	expressão pós-fixa
static_cast<!ype> O			
++ -- + - ! delete	sizeof	direita para a esquerda	unário
* & new			
* / %		esquerda para a direita	multiplicativo
+ -		esquerda para a direita	aditivo
« »		esquerda para a direita	deslocamento
< <= > >=		esquerda para a direita	relacional
== =		esquerda para a direita	igualdade

Fig. 16.13 Precedência e associatividade de operadores (parte 2 de 2).

16.8 Campos de bits

C÷+ fornece a possibilidade de especificar o número de bits no qual um membro unsigned ou int de uma classe ou uma estrutura (ou uma união - ver Capítulo 18) é armazenado. Tal membro é chamado de campo de bits. Os campos de bits possibitam uma melhor utilização de memória, armazenando dados no número mínimo de bits exigidos. Os membros de um campo de bits devem ser declarados como int ou unsigned.

1 Dica de desempenho 16.2

f Campos de bits ajudam a economizar memória.

Considere a seguinte definição de estrutura:

```
struct BitCard {
unsigned face 4;
unsigned suit : 2;
unsigned color : 1;
```

A definição contém três campos de bits uns igned - face. suit e color - usados para representar uma carta de um baralho de 52 cartas. Um campo de bits é declarado seguindo-se um membro unsigned ou int por um dois-pontos (:) e uma constante inteira representando o comprimento do campo (i.e., o número de bits em que o membro é armazenado). O comprimento deve ser uma constante do tipo inteiro entre O e o número total de bits usados para armazenar um int em seu sistema.

A definição de estrutura precedente indica que o membro face é armazenado em 4 bits, o membro suit é

armazenado em 2 bits e o membro color é armazenado em 1 bit. O número de bits se baseia no intervalo desejado
de valores para cada membro da estrutura. O membro face armazena valores entre O (As) e 12 (Rei) - 4 bits
podem armazenar um valor entre De 15. O membro suit armazena valores entre O e 3 (O = Ouros, 1 = Copas, 2 =
Paus, 3 = Espadas) -2 bits podem armazenar um valor entre O e 3. Finalmente, o membro color armazena ou O
(vermelho) ou 1 (preto) - 1 bit pode armazenar ou O ou 1.
O programa na Fig. 16.14 (saída mostrada na Fig. 16.15) cria o array deck, que contém 52 estruturas bitCard. A função fillDeck insere as 52 cartas no array deck e a função deal imprime as 52 cartas. Note que membros campos de bits de estruturas são acessados exatamente como qualquer outro membro de uma estrutura. O membro color é incluído como um meio de indicar a cor da carta em um sistema que permite exibir cores.

Operadores						Associatividade	Tipo
&						esquerda para a direita	AND sobre bits
A						esquerda para a direita	XOR sobre bits
1						esquerda para a direita	OR sobre bits
&&						esquerda para a direita	AND lógico
1						esquerda para a direita	OR lógico
?:						direita para a esquerda	condicional
	+=	-=	=	1=	%=	direita para a esquerda	atribuição
&=	=	A...	«.	».			
,						esquerda para a direita	vírgula


```
1 II Fig. 16.14: figl6_14.cpp
2 II Exemplo usando um campo de bits
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
```

```
7
8 #include <iomanip>
9
10 using std: :setw;
11
12 struct BitCard {
13 unsigned face : 4; II 4 bits; 0-15
14 unsigned suit : 2; II 2 bits; 0-3
15 unsigned color : 1; II 1 bit; 0-1
16
17
18 void fillDeck( BitCard * const );
19 void deal( const BitCard * const );
20
21 int main()
22 {
23 BitCard deck[ 52 ];
24
25 fillDeck( deck );
26 deal( deck );
27 return 0;
28 }
29
30 void filloeck( BitCard * const wDeck
31 {
32 for ( inti=0; i<51; i++) {
33 wDeck[ i ].face = i % 13;
34 wDeck[ i ] .suit = i / 13;
35 wDeck[ i ].color = i / 26;
36
37 )
38
39 // Exibe as cartas em formato de duas colunas. Cartas O a 25 com subscrito
40 // kl (coluna 1) . Cartas 26 a 51 com subscrito k2 (coluna 2)
41 void deal( const BitCard * const wDeck
42 {
43 for ( mntkl = 0, k2 =kl +26; kl <= 25; kl++, k2++
44 cout « “Carta:” « setw( 3 ) « wDeck[ kl ).face
45 « Naipe:” « setw( 2 ) « wDeck[ kl ].suit
46 « “ Cor:” « setw( 2 ) « wDeck[ kl ].color
47 « “ “ « “Carta:” « setw( 3 ) « wDeck[ k2 ].face
48 « “ Naipe:” « setw( 2 ) « woeck[ k2 ] .suit
49 « “ Cor:” « setw( 2 ) « woeck[ k2 ].color
50 « endl;
51 }
52
```

Fig. 16.14 Usando campos de bits para armazenar um baralho.

Fig. 16.15 Saída do programa da Fig. 16.14 (parte 1 de 2).

Carta:	O	Naipe:	O	Cor:	O	Carta:	O	Naipe:	2	Cor:	1
Carta:	1	Naipe:	O	Cor:	O	Carta:	1	Naipe:	2	Cor:	1
Carta:	2	Naipe:	O	Cor:	O	Carta:	2	Naipe:	2	Cor:	1
Carta:	3	Naipe:	O	Cor:	O	Carta:	3	Naipe:	2	Cor:	1

Carta: 4 Naipe: O Cor: O Carta: 4 Naipe: 2 Cor: 1
Carta: 5 Naipe: O Cor: O Carta: 5 Naipe: 2 Cor: 1
Carta: 6 Naipe: O Cor: O Carta: 6 Naipe: 2 Cor: 1
Carta: 7 Naipe: O Cor: O Carta: 7 Naipe: 2 Cor: 1
Carta: 8 Naipe: O Cor: O Carta: 8 Naipe: 2 Cor: 1
Carta: 9 Naipe: O Cor: O Carta: 9 Naipe: 2 Cor: 1
Carta: 10 Naipe: O Cor: O Carta: 10 Naipe: 2 Cor: 1
Carta: 11 Naipe: O Cor: O Carta: 11 Naipe: 2 Cor: 1
Carta: 12 Naipe: O Cor: O Carta: 12 Naipe: 2 Cor: 1
Carta: O Naipe: 1 Cor: O Carta: O Naipe: 3 Cor: 1
Carta: 1 Naipe: 1 Cor: O Carta: 1 Naipe: 3 Cor: 1
Carta: 2 Naipe: 1 Cor: O Carta: 2 Naipe: 3 Cor: 1
Carta: 3 Naipe: 1 Cor: O Carta: 3 Naipe: 3 Cor: 1
Carta: 4 Naipe: 1 Cor: O Carta: 4 Naipe: 3 Cor: 1
Carta: 5 Naipe: 1 Cor: O Carta: 5 Naipe: 3 Cor: 1
Carta: 6 Naipe: 1 Cor: O Carta: 6 Naipe: 3 Cor: 1
Carta: 7 Naipe: 1 Cor: O Carta: 7 Naipe: 3 Cor: 1
Carta: 8 Naipe: 1 Cor: O Carta: 8 Naipe: 3 Cor: 1
Carta: 9 Naipe: 1 Cor: O Carta: 9 Naipe: 3 Cor: 1
Carta: 10 Naipe: 1 Cor: O Carta: 10 Naipe: 3 Cor: 1
Carta: 11 Naipe: 1 Cor: O Carta: 11 Naipe: 3 Cor: 1
Carta: 12 Naipe: 1 Cor: O Carta: 12 Naipe: 3 Cor: 1

Fig. 16.15 Saída do programa da Fig. 16.14 (parte 2 de 2).

É possível se especificar um campo de bits sem nome, caso em que o campo é usado como enchimento na estrutura. Por exemplo, a definição de estrutura usa um campo de 3 bits sem nome como enchimento - nada pode ser armazenado naqueles três bits. O membro b é armazenado em outra unidade de memória.

```
struct Example {
unsigned a : 13;
unsigned : 3;
```

unsigned b : 4;
Um campo de bits sem nome com um comprimento zero é usado para alinhar o próximo campo de bits de uma unidade memória em um novo limite. Por exemplo, a definição de estrutura
struct Exaxnple {
unsigned a : 13;
unsigned : 0;
unsigned b : 4;
usa um campo de bits O sem nome - para saltar os bits restantes (tantos quantos existirem) da unidade de memória em que a é armazenado e alinhar b no próximo limite de unidade de memória.
Dica de portabilidade 16.5
_____ As manipulações de campos de bit são dependentes de máquina. Por exemplo, alguns computadores permitem que campos de bits cruzem limites de palavra, enquanto outros não.
Erro comum de programação 16.8
Tentar acessar bits individuais de um campo de bits como se fossem elementos de um array. Campos de bits não são “array de bits”.

Conforme isdigit:
8 é um dígito
não é um dígito

A Fig. 16.17 demonstra as funções isdigit, isalpha, isainume isxdigit. A função isdigit determina
se seu argumento é um dígito (0-9). A função isalpha determina se seu argumento é uma letra maiúscula (A-Z)
ou uma letra minúscula (a-z). A função isalnum determina se seu argumento é uma letra maiúscula, uma letra
minúscula ou um dígito. A função isxdigit determina se seu argumento é um dígito hexadecimal (A-F. a-f. 0-
9).

1
2
3
4

1

II Fig. 16.17: figl6l7.cpp
II Usando as funções isdigit, isalpha, isainum e isxdigit #include <iostream>
using std::cout;

```
using std::endl;
#include <cctype>
int main()

um” : “8 não é um” um” : “# não é um”

1

uma” : “A não é uma” uma” : “b não é uma” uma” : “& não é uma” uma” : “4 não é
uma”

um” : “A não é um”
“8 não é um”
‘ não é um”
um” : “F não é um”
um” : “J não é um” um” : “7 não é um” um” : “$ não é um”

“f não é um”
```

Fig. 16.17 Usando isdigit, isalpha, isainum e isxdigit (parte 1 de 2).

5				
6				
7				
8				
9				
10				
11	{			
12		cout	«	“Conforme isdigit:\n’
13			«	(isdigit(‘8’) ? “8 é
14			«	“ dígito\n”
15			«	(isdigit(‘#‘) ? “# é
16			«	“ dígito\n”;
17		cout	«	“\nConforme isalpha:\n”
18			«	(isalpha(‘A’) ? “A é
1			«	“ letra\n”

9				
20			«	(isalpha(‘b’) ? “b é
21			«	“ letra\n”
22			«	(isalpha(‘&‘) ? “& é
23			«	“ letra\n”
24			«	(isalpha(‘4’) ? “4 é
25			«	“ letra\n”;
26		cout	«	“\nConforme isalnum:\n”
27			«	(isalnum(‘A’) ? “A é
28			«	“ dígito ou uma letra\n”
29			«	(isalnum(‘8’) ? “8 é um”
30			«	“ dígito ou uma letra\n”
31			«	(isalnum(‘#‘) ? “# é um”
32			«	“ dígito ou uma letra\n”;
33		cout	«	“\nConforme isxdigit:\n”
34			«	(isxdigit(‘F’) ? “F é
35			«	“ dígito hexadecimal\n”
36			«	(isxdigit(‘J’) ? “J é
37			«	“ dígito hexadecimal\n”
38			«	(isxdigit(‘7’) ? “7 é
39			«	“ dígito hexadeciinal\n”
40			«	(isxdigit(‘$‘) ? “$ é
41			«	“ dígito hexadecimal\n”
4			«	(isxdigit(‘f’) ? “f é um”

2				
43			«	“ dígito hexadecimal” « endi;
44		return	0;	
45	}			

Conforme isalpha:
A é uma letra
b é uma letra
& não é uma letra
4 não é uma letra
Conforme isalnum:
A é um dígito ou uma letra
8 é um dígito ou uma letra
não é um dígito ou uma letra
Conforme isxdigit:
F é um dígito hexadecimal
J não é um dígito hexadecimal
7 é um dígito hexadecimal
$ não é um dígito hexadecimal
f é um dígito hexadecimal
Fig. 16.17 Usando isdigit, isalpha, isalnume isxdigit (parte 2 de 2).
O programa da Fig. 16.17 usa o operador condicional (?:)com cada função para determinar se o string é um” ou o string “ não é um “ deve ser impresso na saída para cada caractere testado. Por exemplo, a expressão
isdigit(‘8’) ? “8 é um “ : “8 não é um
indica que se ‘8’ é um dígito (i.e., isdigit retorna um valor verdadeiro (não-zero)), o string “8 é um “ é impresso e se ‘8’ não é um dígito (i.e., isdigit retorna O), o string “8 não é um “ é impresso.
O programa da Fig. 16.18 demonstra as funções islower, isupper, tolower e toupper. A função islower determina se seu argumento é uma letra minúscula (a-z). A função isupper determina se seu parâmetro é uma letra maiúscula (A-Z). A função tolower converte uma letra maiúscula em uma letra minúscula e retorna a letra minúscula. Se o argumento não é uma letra maiúscula, tolower retorna o argumento inalterado. A função toupper converte uma letra minúscula em uma letra maiúscula e retorna a letra maiúscula. Se o argumento não é uma letra minúscula, toupper retorna o argumento inalterado.
```
1 // Fig. 16.18: figl6l8.cpp
2 // Usando as funções islower, isupper, tolower e toupper
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
```

```
7
8 #include <cctype>
9
10 int main()
11
12 cout « “Conforme islower:n
13 « ( islower( ‘p’ ) ? “p é uma” : “p não é uma”
14 « “ letra minúscula\n”
15 « ( islower( ‘P’ ) ? “p é uma” : “P não é uma”
16 « “ letra minúscula\n”
17 « ( islower( ‘5’ ) ? “5 é uma” : “5 não é uma”
18 « “ letra minúscula\n”
19 « ( islower( ‘ ! ‘ ) ? “! é uma” : “ não é uma”
20 « “ letra minúscula\n”;
21 cout « “\nConforme isupper:\n”
```

Fig. 16.18 Usando islower, isupper, tolower e toupper (parte 1 de 2).


```
Conforme islower:
p é uma letra minúscula
P não é urna letra minúscula
5 não é urna letra minúscula
não é uma letra minúscula
Conforme isupper:
D é urna letra maiúscula
d não é urna letra maiúscula
8 não é uma letra maiúscula
$ não é urna letra maiúscula
u convertido para maiúscula é U
7 convertido para maiúscula é 7
$ convertido para maiúscula é $
L convertido para minúscula é 1
```

Fig. 16.18 Usando islower, isupper, tolower e toupper (parte 2 de 2).

A Fig. 16.19 demonstra as funções isspace, iscntrl, ispunct, isprint e isgraph. A função isspace detennina se seu argumento é um caractere de espaço em branco - tal como espaço (1 1), avanço de formulário .‘ \f’, nova hnha (‘\n’), retorno de carro (T \r’), tabulação horizontal (‘ \t’) ou marca de tabulação vertical (‘ \v’). A função iscntrl determina se seu argumento é um caractere de controle, tal como uma marca de tabulação horizontal, uma marca de tabulação vertical, um avanço de formulário, alarme (‘ \a T), retrocesso (‘ \b T), retorno do carro ou nova linha. A função ispunct determina se seu parâmetro é um caractere de impressão diferente de um espaço, dígito ou letra, talcomo$, #. , [, J, {, }, ;, :, %,etc.Afunçãoisprintdeterminaseseuparâmetroéumcaracterequepode ser exibido na tela (inclusive o caractere espaço). A função isgraph testa os mesmos caracteres que isprint: porém, o caractere espaço não está incluído.

39
40

return 0;

Fig. 16.19 Usando isspace, iscntrl, ispunct, isprint e isgraph (parte 1 de 3).

1	// Fig. 16.19: figl6	19.cpp
2	// Usando as funções	isspace, iscntrl, ispunct, isprint, isgraph
3	#include <iostream>	
4		
5	using std: :cout;	

22		«	(isupper(‘D’) ? “D é uma’ : “D não	é uma ”	)
23		«	“ letra maiúscula\n”		
24		«	(isupper(‘d’) ? “d é urna” : “d não	é uma ”	)
25		«	“ letra maiúscula\n”		
26		«	(isupper(‘8’) ? ‘8 é urna” : “8 não	é uma ”	)
27		«	“ letra maiúscula\n”		
28		«	(isupper(’$’) ? “$ é urna” : “$ não é	urna ”	
29		«	“ letra maiúscula\n”;		
30	cout	«	“\nu convertido para maiúscula é		
31		«	static_cast< char >(toupper(‘u’		
32		«	“\n7 convertido para maiúscula é		
3		«	static_cast< char >(toupper(‘7’		

3					
34		«	“\n$ convertido para maiúscula é		
35		«	static_cast< char >(toupper(‘$‘		
36		«	‘\nL convertido para minúscula é		
37		«	static_cast< char >(tolower(‘L’))	« endl;	
38					


```
6 using std: :endl;
7
8 #include <cctype>
9
10 int main()
11
12 cout « “Conforme isspace:\nNova linha
13 « ( isspace( ‘\n’ ) ? “é um” : “não é um”
14 « “ caractere de espaçamento \nTabulação horizontal
15 « ( isspace( ‘\t’ ) ? “é um” “não é um”
16 « “ caractere de espaçamento \n”
17 « ( isspace( ‘%‘ ) ? “% é um” : ‘% não é um”
18 « “ caractere de espaçamento \n”;
19
20 cout « “\nConforme iscntrl:\nNova linha
21 « ( iscntrl( ‘\n’ ) ? “é um” “não é um”
22 « “ caractere de controle\n”
23 « ( iscntrl ( $‘ ) ? “$ é um” : “$ não é um”
24 « “ caractere de controle\n”;
25
26 cout « “\nConforme ispunct:\n”
27 « ( ispunct ( ‘ ;‘ ) ? “; é um” : “; não é um”
28 « “ caractere de pontuação\n”
29 « ( ispunct( ‘Y’ ) ? “Y é um” “Y não é um”
30 « “ caractere de pontuação\n”
31 « ( ispunct(’#’) ? ‘1 é um” : “# não é um”
32 “ caractere de pontuação\n”;
33
34 cout « “\nConforme isprint:\n”
35 « ( isprint( ‘$‘ ) ? “$ é um” : “$ não é um”
36 « “ caractere de impressão\nAlert
37 « ( isprmnt( ‘\a’ ) ? “é um” “não é um”
38 « “ caractere de impressão\n”;
```

```
39
40 cout « “\nConforme isgraph:\n”
41 « ( isgraph( ‘Q’ ) ? “Q é um” “ não é um”
42 « “ caractere de impressão diferente de espaço\nEspaço
43 « ( isgraph(’ ‘) ? “é um” : “não é um”
44 « ‘ caractere de impressão diferente de espaço” « endl;
45
46 return 0;
47
```

Fíg. 16.19 Usando isspace, iscntrl, ispunct, isprint e isgraph (parte 2 de 3).

Conforme isspace:	
Nova linha é um caractere de espaçamento	
Tabulação horizontal é um caractere de	espaçamento
% não é um caractere de espaçamento	
Conforme iscntrl:	
Nova linha é um caractere de controle	
$ não é um caractere de controle	
Conforme ispunct:	
é um caractere de pontuação	
Y não é um caractere de pontuação	
# é um caractere de pontuação	


```
Conforme isprint:
$ é um caractere de impressão
Alert não é um caractere de impressão
Conforme isgraph:
é um caractere de impressão diferente de espaço
Espaço não é um caractere de impressão diferente de espaço
```

Fig. 16.19 Usando isspace, iscntrl, ispunct, isprint e isgraph (parte 3 de 3).

16.10 Funções de conversão de strings

No Capítulo 5, discutimos várias funções de manipulação de strings de caracteres, a maioria popular em C++. Nas próximas seções, cobriremos as funções restantes, inclusive funções para converter strings em valores numéricos, funções para pesquisar strings e funções para manipular. comparar e pesquisar blocos de memória.

Esta seção apresenta as funções de conversão de strings da biblioteca de utilitários gerais <cstdlib>. Estas funções convertem strings de dígitos para

valores inteiros e em ponto flutuante. A Fig. 16.20 resume as funções de conversão de strings. Note o uso de const para declarar a variável nPtr nos cabeçalhos de função (lido da direita para a esquerda como “nPtr é um ponteiro para uma constante do tipo caractere’); const declara que o valor do argumento não será modificado. Quando usar funções da biblioteca de utilitários gerais, não deixe de incluir o arquivo de cabeçalho <cstdlib>.

Fig. 16.20 Resumo das funções de conversão de strings da biblioteca de utilitários gerais.
A função atof (Fig. 16.21) converte seu argumento - um string que representa um número em ponto flutuante - em
um valor double. A função retorna o valor double. Se o string não pode ser convertido - por exemplo, se o
primeiro caractere do string não é um dígito - a função atof retoma 0.

```
1 // Fig. 16.21: figl6_21.cpp
2 // Usando atof
3 #include <iostream>
4
```

Fig. 16.21 Usando atof (parte 1 de 2).

Protótipo Descrição
double atof (const char *nptr) Converte o string nPtr para double.
int atoi (const char *nPtr) Converte o string nPtr para int.
long atol (const char *nptr) Converte o string nPtr para long int.
double strtod(const char *nptr, char **endptr) Converte o string nPtr para double.
long strtol (const char *nptr, char **endptr, int base) Converte o string nPtr para long.
unsigned long strtoul (const char *nptr, char **endptr, int base) Converte o string nPtr para unsigned long.


```
5 using std::cout;
6 using std: :endl;
7
8 #include <cstdlib>
9
10 int main()
11
12 double d = atof( ‘99.0” );
13
14 cout « “O string \“99.0\” convertido para double é
15 « d « “\nO valor convertido dividido por 2 é
16 «d/2.0«endl;
```

```
17 return 0;
18
O string “99.0” convertido para double é 99
O valor convertido dividido por 2 é 49.5
```
Fig. 16.21 Usando atof (parte 2 de 2).

A função atoi (Fig. 16.22) converte seu argumento - um string de dígitos que representa um inteiro - em um valor int. A função retorna o valor int. Se o string não pode ser convertido, a função atoi retoma zero.

```
1 II Fig. 16.22: figl6_22.cpp
2 // Usando atoi
3 #include <iostream>
4
5 using std: :cout;
6 using std: :endl;
7
8 #include <cstdlib>
9
int main()
int i = atoi( “2593” );
cout « “O string \“2593\” convertido para int é “ « i
« “\nO valor convertido menos 593 é “ « i - 593
« endl;
return 0;
```

A função atol (Fig. 16.23) converte seu argumento - um string de dígitos representando um inteiro longo - em um
valor long. A função retoma o valor long. Se o string não pode ser convertido, a função atol retoma zero. Se int
e long são ambos armazenados em 4 bytes, a função atoi e a função atol funcionam de forma idêntica.

```
1 II Fig. 16.23: fig1623.cpp
2 // Usando atol
3 #include <iostream>
4

1

Nas
[C05,

.b>.
e as

lido que e de

10
```

11
12
13
14
15
16
17
18

O
O

string “2593” convertido para int é 2593 valor convertido menos 593 é 2000

em

Fig. 16.22 Usando atoi.

Fig. 16.23 Usando atol (parte 1 de 2).

5 using std: :cout;
6 using std::endl;
7
8 #include <cstdlib>
9
10 int main()
11
12 long x = atol(“1000000”);
13
14 cout « “o string \“lOOOOOO\” convertido para long é “ « x
15 « “\nO valor convertido dividido por 2 é “ « x / 2
16 « endl;
17 return 0;
18
O string “1000000” convertido para long é 1000000
O valor convertido dividido por 2 é 500000
Fig. 16.23 Usando atol (parte 2 de 2).
A função strtod (Fig. 16.24) converte uma seqüência de caracteres representando um valor em ponto flutuante para double. A função strtod recebe dois argumentos - um string (char *) e um ponteiro para um string (i.e., um char **). O string contém a seqüência de caracteres a ser convertida em double. O segundo argumento é atribuído à posição do primeiro caractere depois da parte convertida do string. O comando
d = strtod(string, &stringPtr);

do programa da Fig. 16.24 indica que ad é atribuído o valor double convertido de string e a & stringPtr é atribuída a posição do primeiro caractere depois do valor convertido (51 . 2) em string.

```
1 // Fig. 16.24: fig1624.cpp
2 // Usando strtod
3 #include <iostreaxn>
4
5 using std::cout;
6 using std::endl;
7
8 #include <cstdlib>
9
10 int main()
11 {
12 double d;
13 const char *string = “51.2% são admitidos”;
14 char *stringptr;
15
16 d = strtod( string, &stringPtr );
17 cout « “O string \“ « string
18 « ““ é convertido para o\nvalor double “ « d
19 « “ e o string \“ « stringPtr « “\“ « endl;
20 return 0;
21
O string “51.2% são admitidos” é convertido para o
valor double 51.2 e o string “% são admitidos”
```

Fig. 16.24 Usando strtod.

A função strtol (Fig. 16.25) converte para long uma seqüência de caracteres representando um inteiro. A função recebe três argumentos - um string (char*), um ponteiro para um string e um inteiro, O string contém a seqüência de caracteres a ser convertida. Ao segundo argumento é atribuída a localização do primeiro caractere depois da parte convertida do string. O inteiro especifica a base do valor a ser convertido. O comando

```
x = strtol( string, &remainderPtr, O );
```

indica que o valor long convertido do string é atribuído a x. Ao segundo argumento, & reniainderptr. é atribuído
o resto do string após a conversão. Usar NULL como o segundo argumento faz com que o resto do string seja
ignorado. O terceiro argumento, O, indica que o valor a ser convertido poder estar em octal (base 8), decimal (base
10) ou hexadecimal (base 16).

Em uma chamada para a função strtol, a base pode ser especificada como zero ou qualquer valor entre 2 e
36 (ver Apêndice C, “Sistemas de numeração”, para uma explicação detalhada

dos sistemas de numeração octal,
decimal, hexadecimal e binário). As representações numéricas de inteiros nas bases 11 a 36 usam os caracteres de A
a Z para representar os valores 10 a 35. Por exemplo, valores hexadecimais podem consistir nos dígitos 0-9 e os
caracteres A-F. Um inteiro na base II pode consistir nos dígitos 0-9 e o caractere A. Um inteiro na base 24 pode
consistir nos dígitos 0-9 e os caracteres A-N. Um inteiro na base 36 pode consistir nos dígitos 0-9 e os caracteres A z. lante 1 II Fig. 16.25: figl6_25.cpp
(i.e., 2 II Usando strtol
ntoé 3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <cstdlib>
re
10 int main()
11 {
- 12 longx;
13 const char *string = “-1234567abc”;
14 char *remajnderptr;
15
16 x = strtol(string, &remainderptr, O);
17 cout « “O string original é \“ « string
18 « “\“\nO valor convertido é « x
19 « “\nO resto do string original é \“
20 « remainderPtr
21 « “\“\nO valor convertido mais 567 é
22 « x + 567 « endl;
23 return 0;
24 }
O string original é “-l234567abc”
O valor convertido é -1234567
O resto do string original é “abc”
O valor convertido mais 567 é -1234000
Fig. 16.25 Usando strtol.
A função strtoul (Fig. 16.26) converte para uns igned long uma seqüência de caracteres representando um inteiro unsigned long. A função funciona da mesma maneira que a função strtol. O comando
x = strtoul(string, &remainderPtr. O);

no programa da Fig. 16.26 indica que a x é atribuído o valor unsigned long convertido do string. Ao segundo argumento, & remainderPtr, é atribuído o restante do string depois da conversão, O terceiro argumento, 0, indica que o

valor a ser convertido pode estar em formato octal, decimal ou hexadecimal.

```
1 II Fig. 16.26: fig1626.cpp
2 // Usando strtoul
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <cstdlib>
9
10 int main()
11
12 unsigned long x;
13 const char *string = “1234567abc’;
14 char *remajnderptr;
15
16 x = strtoul( string, &remainderptr, O );
17 cout « “O string original é \““ « string
18 « \“\nO valor convertido é “ « x
19 « ‘\nO resto do string original é \“
20 « remainderptr
21 « “\“\nO valor convertido menos 567 é
22 «x-567«endl;
23 return 0;
24 }
```

```
O string original é “1234567abc”
O valor convertido é 1234567
O resto do string original é “abc”
O valor convertido menos 567 é 1234000
```

Fig. 16.26 Usando strtoul.

16.11 Funções de pesquisa da biblioteca de manipulação de strings

Esta seção apresenta as funções da biblioteca de manipulação de strings usadas para procurar caracteres e outros strings em um string. As funções estão resumidas na Fig. 16.27. Note que as funções strcspn e strspn especificam o tipo de retorno size-t. O tipo size-t é um tipo definido pelo padrão como o tipo integral do valor retornado pelo operador sizeof.

Dica de portabilidade 16.6

O tipo si zet é um sinônimo dependente de sistema ou para um tipo unsigned long ou para um tipo unsigned int.

Fig. 16.27 Funções de pesquisa da biblioteca de manipulação de strings (parte 1 de 2).

Protótipo		Descrição	
ch	*strc	const char *s, int c)	

ar	hr(		
		Localiza a primeira ocorrência do caractere c no string s. Se c é encontrado, um para c em s é retornado. Caso contrário, um ponteiro NULL é retornado.	pont eiro

char *Strrchr(const char *, int c
Localiza a última ocorrência de c no string s. Se c é encontrado, um ponteiro para c no
string s é retornado. Caso contrário, é retornado um ponteiro NULL.
size_t strspn(const char *si, const char *s2
Determina e retorna o comprimento do segmento inicial do string si consistindo somente
em caracteres contidos no string s2.
char *strpbrk(const char *si, const char *s2
Localiza a primeira ocorrência no string si de qualquer caractere no string s2. Se um caractere do string s2 é encontrado, é retornado um ponteiro para o caractere no string si. Caso contrário, é retornado um ponteiro NULL.
size_t strcspn(const char *si, const char *s2
Determina e retorna o comprimento do segmento inicial do string si que consiste em
caracteres não-contidos no string s2.
char *strstr(const char *sl, const char *s2)
Localiza a primeira ocorrência no string si do string s2. Se o string é encontrado, um
ponteiro para o string em si é retornado. Caso contrário, um ponteiro NULL é retornado.
Fig. 16.27 Funções de pesquisa da biblioteca de manipulação de strings (parte 2 de 2).
A função strchr procura a primeira ocorrência de um caractere em um string. Se o caractere é encontrado,
strchr retorna um ponteiro para o caractere no string; caso contrário, strchr retorna NULL. O programa da Fig.
16.28 usa strchr para procurar as primeiras ocorrências de ‘u’ e ‘z no string “Isto é um teste”.

```
i II Fig. i6.28: figi6_28.cpp
2 1/ Usando strchr
3 #inciude <iostream>
4
5 using std: :cout;
6 using std::endi;
7
8 #inciude <cstring>
9
iO int main()
ii {
```

```
i2 const char *string = “Isto é um teste”;
i3 char characteri = ‘u’, character2 =
14
i5 if ( strchr( string, characteri ) != NULL
i6 cout « ‘\‘ ‘ « characteri « “‘ foi encontrado em \“
i7 « string « “\“.\n”;
i8 eise
19 cout « ‘\‘‘ « characteri « “‘ não foi encontrado em \“
20 « string « “\“.\n”;
21
22 if ( strchr( string, character2 ) ! NULL
23 cout « ‘\‘‘ « character2 « “‘ foi encontrado em \“
24 « string « “\“.\n”;
25 eise
26 cout « ‘\‘ ‘ « character2 « “‘ não foi encontrado em \“
27 « string « “\“.“ « endi;
28 return 0;
29 }
```

Fig. 16.28 Usando strchr (parte 1 de 2).


```
‘u’ foi encontrado em “Isto é uni teste”.
Lz não foi encontrado em “Isto é um teste”.
```

Fig. 16.28 Usando strchr (parte 2 de 2).

A função strcspn (Fig. 16.29) determina o comprimento da parte inicial do string em seu primeiro argumento que não contém quaisquer caracteres do string em seu segundo argumento. A função retorna o comprimento do segmento.

```
1 II Fig. 16.29: fig1629.cpp
2 // Usando strcspn
3 #include <iostream>
4
5 using std::cout;
6 using std: :endl;
7
8 #include <cstring>
9
10 int main()
11 {
12 const char *stringl = “O valor é 3.14159”;
13 const char *string2 = “1234567890”;
14
15 cout « “stringl = “ « stringl « “\nstring2 = “ « string2
16 « ‘\n\nO tamanho do segmento inicial de stringl’
17 « “\nque não contém nenhum caractere de string2 =
18 « strcspn( stringl, string2 ) « endl;
19 return O;
```

```
20 } t
stringl = O valor é 3.14159
string2 = 1234567890
O tamanho do segmento inicial de stringl
que não contém nenhum caractere de string2 = 10
```
Fig. 16.29 Usando strcspn.

A função strpbrk procura a primeira ocorrência em seu primeiro argumento string de qualquer caractere em seu segundo argumento srring. Se um caractere do segundo argumento é encontrado, strpbrk retorna um ponteiro para o caractere no primeiro argumento; caso contrário, strpbrk retorna NULL. O programa da Fig. 16.30 localiza a primeira ocorrência em stringl de qualquer caractere de string2.

```
1 // Fig. 16.30: figl63O.cpp
2 // Usando strpbrk
3 #include <iostream>
5 using std::cout;
6 using std::endl;
7
8 #include <cstring>
```
Fig. 16.30 Usando strpbrk (parte 1 de 2). 4


```
9
10 int main()
11
12 const char *stringl = “Isto é um teste”;
13 const char *string2 = “cuidado”;
14
15 cout « “Dos caracteres em \“ « string2 « “\“\n’”
16 « *trpbk( stringl, string2 ) « ‘\‘
17 « “ é o primeiro caractere que aparece em\n\”
18 « stringl « ‘\“‘ « endl;
19 return 0;
20
Dos caracteres em “cuidado”
‘o’ é o primeiro caractere que aparece em
“Isto é um teste”
```
Fig. 16.30 Usando strpbrk (parte 2 de 2).

A função strrchr procura a última ocorrência do caractere especificado em um string. Se o caractere é encontrado, strrchr retorna um ponteiro para o caractere no string; caso contrário, strrchr retorna 0. O programa da Fig. 16.3 1 procura pela última ocorrência do caractere ‘z’ no string “Um zoológico tem muitos animais inclusive zebras”.

```
1 II Fig. 16.31: figl63l.cpp
2 // Usando strrchr
```

```
3 #include <iostream>
4
5 using std: :cout;
6 using std::endl;
7
8 #include <cstring>
9
10 int main()
11 {
12 const char *stringl = “Um zoológico tem muitos animais”
13 “inclusive zebras”;
14 int c =
15
16 cout « “O resto do stringl começando com a\n”
17 « “última ocorrência do caractere
18 « static_cast< char >( c
19 « “‘ is: \“ « strrchr( stringl, c ) « ‘\“‘ « endl;
20 return 0;
21 }
```

```
O resto do stringl começando com a
última ocorrência do caractere ‘z’ é: “zebras”
```

Fig. 16.31 Usando strrchr.

A função strspn (Fig. 16.32) determina o comprimento da parte inicial do string em seu primeiro argumento que contém somente caracteres do string em seu segundo argumento. A função retorna o comprimento do segmento.


```
1 II Fig. 16.32: figl6_32.cpp
2 II Usando strspn
3 #include <iostream>
4
5 using std: :cout;
6 using std::endl;
7
8 #include <cstring>
9
10 int main()
11
12 const char *stringl = “0 valor é 3.14159;
13 const char *string2 = “arél 00v”;
14
15 cout « “stringl = “ « stringl
16 « “\nstring2 = “ « string2
17 « “\n\nO tamanho do segmento inicial de stringl\n”
18 « “que contém somente caracteres de string2 =
19 « strspn( stringl, string2 ) « endi;
```

```
20 return 0;
21 }
stringl = O valor é 3.14159
string2 = arél 00v
O tamanho do segmento inicial de stringl
que contém somente caracteres de string2 = 10
```
Fig. 16.32 Usando strspn.

A função strstr procura a primeira ocorrência do seu segundo argumento string no seu primeiro argumento string. Se o segundo string é encontrado no primeiro string, um ponteiro para a localização do string no primeiro argumento é retornado, O programa da Fig. 16.33 usa strstr para encontrar o string ‘def” no string “abcdefabcdef’.

```
1 II Fig. 16.33: figl6_33.cpp
2 // Usando strstr
3 #include <iostream>
4
5 using std::cout;
6 using std: :endl;
7
8 #include <cstring>
9
10 int main()
11
12 const char *stringl = “abcdefabcdef”;
13 const char *string2
14
15 cout « “stringl = “ « stringl « ‘\nstring2 = “ « string2
16 « “\n\nO resto de stringl começando com a\n”
17 « “primeira ocorrência de string2 é:
18 « strstr( stringl, string2 ) « endi;
19 return 0;
20 )
```
Fig. 16.33 Usando strstr (parte 1 de 2).


```
stringl = abcdefabcdef
string2 = def
O resto de stringl começando com a
primeira ocorrência de string2 é: defabcdef
```
Fig. 16.33 Usando strstr (parte 2 de 2).

16.12 Funções de memória da biblioteca de manipulação de strings

As funções da biblioteca de manipulação de strings apresentadas nesta seção facilitam a manipulação, a comparação e a pesquisa de blocos de memória. As funções tratam blocos de memória como arrays de caracteres. Estas funções podem manipular qualquer bloco de dados. A Fig. 16.34 resume as funções de memória da biblioteca de manipulação de strings. Nas discussões das funções,

“objeto” refere-se a um bloco de dados.

Protótipo Descrição

void *memcpy(void *si, const void*s2, size_t n
Copia n caracteres do objeto apontado por s2 para o objeto apontado por si. Retorna um
ponteiro para o objeto resultante.
void *memmove(void *si, const void *s2, sizet n
Copia n caracteres do objeto apontado por s2 para o objeto apontado por si. A cópia é executada como se os caracteres fossem primeiro copiados do objeto apontado por s2 para um array temporário e, então, do array temporário para o objeto apontado por si. E retomado um ponteiro para o objeto resultante.
int memcmp(const void *si, const void *s2, size_t n
Compara os primeiros n caracteres dos objetos apontados por si e s2. A função retorna O,
menor que O ou maior que O se si é igual, menor do que ou maior do que s2.
void *memchr(const void *5 int c, sizet n
Localiza a primeira ocorrência de c (convertido para unsigned char) nos primeiros n
caracteres do objeto apontado por s. Se c é encontrado, é retornado um ponteiro para o
objeto c. Caso contrário, é retomado O.
void *memset(void *, int c, sizet n
Copia c (convertido para uns igned char) para os primeiros n caracteres do objeto apontado por s. Retoma um ponteiro para o resultado.
Fig. 16.34 Funções de memória da biblioteca de manipulação de strings.
Os parâmetros ponteiros destas funções são declarados void*. No Capítulo 5, vimos que um ponteiro para qualquer tipo de dados pode ser diretamente atribuído a um ponteiro do tipo void*. Por essa razão, estas funções podem receber ponteiros para qualquer tipo de dados. Lembre-se de que um ponteiro de tipo void* não pode ser diretamente atribuído a um ponteiro para qualquer tipo de dados. Como um ponteiro void* não pode ser derreferenciado, cada função recebe um parâmetro de tamanho que especifica o número de caracteres (bytes) que a função processará. Para simplificar, os exemplos nesta seção manipulam arrays de caracteres (blocos de caracteres).
A função memcpy copia um número especificado de caracteres (bytes) do objeto apontado por seu segundo parâmetro para o objeto apontado por seu primeiro parâmetro. A função pode receber um ponteiro para qualquer tipo de objeto. O resultado desta função é indefinido se os dois objetos se sobrepõem na memória, i.e., eles são partes do mesmo objeto. O programa da Fig. 16.35 usa memcpy para copiar o string no array s2 para o array si.


```
1 II Fig. 16.35: figl6_35.cpp
2 II Usando memcpy
```

```
3 #include <iostream>
4
5 using std: :cout;
6 using std: :endl;
7
8 #inciude <cstring>
9
10 int main()
11
12 char sl[ 17 ), s2[] = “Copie este string”;
13
14 memcpy( si, s2, 17
15 cout « ‘Depois que s2 é copiado para si com memcpy,\n”
16 « si contém \““ « si « \“‘ « endi;
17 return O;
18
```

Fig. 16.35 Usando memcpy.

A função memmove, como memcpy. copia um número especificado de bytes do objeto apontado por seu segundo
argumento para o objeto apontado por seu primeiro argumento. A cópia é executada como se os bytes fossem
copiados do segundo argumento para um array temporário de caracteres e, então, copiados do array temporário para
o primeiro argumento. Isto permite que caracteres de uma parte de um string possam ser copiados para outra parte
do mesmo string.

Erro de programação comum 16.10

As funções de manipulação de strings que copiam caracteres, exceto memmove, têm resultados indefinidos quando a cópia ocorre entre partes do mesmo string.

O programa na Fig. 16.36 usa memmove para copiar os últimos 8 bytes do array x para os primeiros 8 bytes do array x.

```
1 II Fig. 16.36: fig1636.cpp
2 // Usando memmove
3 #include <iostream>
4
5 using std: :cout;
6 using std: :endi;
7
8 #inciude <cstring>
9
10 int main()
ii
12 char x[] = “Lar Doce Lar”;
13
```

```
14 cout « “0 string no array x antes de memmove é: “ « x;
15 cout « “\nO string no array x depois de memmove é:
```

Fig. 16.36 Usando meinmove (parte 1 de 2).

Depois que	s 2	é	copiado para si com memcpy,
si contém	“Copi e		este string’


```
16 « static cast<char *>( memmove( x, &x[ 4 ], 8
17 « endi;
18 return O;
19
O string no array x antes de memmove é: Lar Doce Lar
O string no array x depois de mexnmove é: Doce Lar Lar
```

Fig. 16.36 Usando mexnmove (parte 2 de 2).

A função mexncmp (F’ig. 16.37) compara o número de caracteres especificado em seu primeiro argumento com os caracteres correspondentes de seu segundo argumento. A função retorna um valor maior do que zero se o primeiro parâmetro é maior do que o segundo argumento, retorna zero se os argumentos são iguais e retorna um valor menor do que zero se o primeiro argumento é menor do que o segundo argumento.

```
1 II Fig. 16.37: figi6_37.cpp
2 II Usando memcinp
3 #inciude <iostreani>
4
5 using std::cout;
6 using std::endl;
7
8 #inciude <iomanip>
9
10 using std::setw;
ii
12 flinclude <cstring>
13
14 int main()
15 {
16 char si[) = °ABCDEFG”, s2[) = “ABCDXYZ”;
17
18 cout « “si = “ « si « “\ns2 = “ « s2 « endi
19 « “\nmemcmp(sl, s2, 4) = “ « setw( 3
20 « memcmp( si, s2, 4 ) « “\nmemcmp(sl, s2, 7)
21 « setw( 3 ) « memcmp ( si, s2, 7
```

```
22 « “\nmemcmp(s2, si, 7) “ « setw( 3
23 « memcmp( s2, si, 7 ) « endi;
24 return 0;
25
si = ABCDEFG
s2 = ABCDXYZ
memcmp (si, s2, 4) = O
niemcmp(sl, s2, 7) = -19
memcmp(s2, si, 7) = 19
```

Fig. 16.37 Usando memcmp.

A função memchr procura a primeira ocorrência de um byte, representado como um unsigned char, no número especificado de bytes de um objeto. Se o byte é encontrado, um ponteiro para o byte no objeto é retornado; caso contrário, é retornado NULL. O programa da Fig. 16.38 procura o caractere (byte) ‘S no string “Leila, Gal e Simorie”.


```
1 II Fig. 16.38: figl6_38.cpp
2 // Usando memchr
3 #include <iostream>
4
5 using std: :cout;
6 using std: :endl;
7
8 #include <cstring>
9
10 int main()
11 {
12 char s[] = Leila, Gal e Simone”;
13
14 cout « “O resto de s depois que o caractere ‘S’
15 « é encontrado é \““
16 « static_cast<char *>( memchr( s, ‘S’, 19
17 « \“ « endi;
18 return O;
19 }
```

```
O resto de s depois que o caractere ‘s’ é encontrado é Simone”
```

Fig. 16.38 Usando memchr.

A função memset copia o valor do byte em seu segundo argumento para um número especificado de bytes do objeto apontado por seu primeiro argumento. O programa na Fig. 16.39 usa memset para copiar ‘b para os primeiros 7 bytes de stringl.

```
1 II Fig. 16.39: fig1639.cpp
2 // Usando memset
```

```
3 #include <iostream>
4
5 using std: :cout;
6 using std: :endi;
7
8 #include <cstring>
9
10 int main()
11 {
12 char stringl[ 15 ] = BBBBBBBBBBBBB";
13
14 cout « stringl = « stringl « endi;
15 cout « "stringl depois de memset =
16 « static cast<char *>( memset( stringl, b, 7 )
17 « endi;
18 return 0;
19 }
stringl = BBBBBBBBBBBBB
stringl depois de memset = bbbbbbbBBBBBB
```

Fig. 16.39 Usando memset.

16.13 Uma outra função da biblioteca de manipulação de strings

A função restante da biblioteca de manipulação de strings é strerror. A Fig. 16.40 resume a função strerror.
A função strerror recebe um número de erro e cria um string com uma mensagem de erro. Um ponteiro para o
string é retornado, O programa da Fig. 16.41 demonstra strerror.

Dica de portabilidade 16.7
_____ A mensagem gerada por strerror é dependente do sistema.

j Protótipo Descrição

Fig. 16.40 Uma outra função de manipulação de strings da biblioteca de manipulação de strings.

```
1 II Fig. 16.41: figl64l.cpp
2 // Usando strerror
3 #include <iostream>
4
5 using std: :cout;
6 using std::endl;
7
```

```
8 #include <cstring>
9
10 int main()
11 {
12 cout « strerror( 2 ) « endi;
13 return O;
14 }
No such file or directory
```

Fig. 16.41 Usando strerror.

Resumo

• Estruturas são coleções de variáveis relacionadas, às vezes chamadas de agregados, com um nome.
• Estruturas podem conter variáveis de tipos de dados diferentes.
• Toda definição de estrutura começa com a palavra-chave struct. Dentro das chaves da definição de estrutura estão as declarações de membros da estrutura.

• Membros da mesma estrutura devem ter nomes únicos.
• Uma definição de estrutura cria um novo tipo de dados que pode ser usado para declarar variáveis.
• Uma estrutura pode ser inicializada com uma lista de inicializadores, colocando-se em seguida à variável na declaração um sinal de igual e uma lista de inicializadores separados por vírgulas, entre chaves. Se existem menos inicializadores na lista do que membros na estrutura, os membros restantes são automaticamente inicializados com zero (ou NULL para membros do tipo ponteiro).

char*strerror(interrornum
Mapeia errornum para um string de texto de uma maneira dependente do sistema. Retorna
um ponteiro para o string.

• Variáveis estruturas inteiras podem ser atribuídas a variáveis estrutura do mesmo tipo.
• Uma variável estrutura pode ser inicializada com uma variável estrutura do mesmo tipo.
• Variáveis estruturas e membros individuais de estruturas são passados para funções por chamada por valor. Os membros array são, naturalmente, passados por referência.
• Para passar uma estrutura através de uma chamada por referência, passe o endereço da variável estrutura. Um array de estruturas é passado automaticamente através de chamada por referência. Para passar um array através de chamada por valor, crie uma estrutura com o array como um membro.
• Criar um novo nome com typedef não cria um novo tipo; cria um nome que é

sinônimo de um tipo definido previamente.
• O operador AND sobre bits (&) aceita dois operandos integrais. Um bit no resultado é mudado para um se os bits correspondentes em cada um do operandos são um.
• As máscaras são usadas para esconder alguns bits enquanto preservam outros.
• O operador sobre bits OR inclusivo (1) aceita dois operandos. Um bit no resultado é colocado em um se o bit correspondente em um ou outro dos operandos for um.
• Cada um dos operadores sobre bits (exceto complemento) tem um operador de atribuição correspondente.
• O operador sobre bits OR exclusivo (A) aceita dois operandos. Um bit no resultado é colocado em um se exatamente um dos bits correspondentes nos dois operandos tem valor um.
• O operador deslocamento à esquerda («) desloca os bits de seu operando esquerdo para a esquerda pelo número de bits especificados por seu operando direito. Os bits desocupados à direita são substituídos por Os.
• O operador de deslocamento à direita (») desloca os bits de seu operando esquerdo para a direita pelo número de bits especificados em seu operando direito. Executar um deslocamento à direita sobre um inteiro unsigned faz com que os bits desocupados na esquerda sejam substituídos por zeros. Os bits desocupados em inteiros signed podem ser substituídos por zeros ou uns - isso é dependente de máquina.
• O operador sobre bits complemento () recebe um operando e inverte seus bits - isso produz o complemento de um do operando.
• Campos de bits reduzem o uso de memória armazenando dados no número mínimo de bits necessários. Os membros de campos de bits devem ser declarados como int ou unsigned.
• Um campo de bits é declarado seguindo-se um nome de membro unsigned ou int com um dois-pontos e o comprimento do campo de bits.
• O comprimento do campo de bits deve ser uma constante inteira entre zero e o número total de bits usados para armazenar um variável int em seu sistema.
• Se um campo de bits é especificado sem um nome, o campo é usado como preenchimento na estrutura.
• Um campo de bits sem nome com comprimento O alinha o próximo campo de bits em um novo limite de palavra de máquina.
• A função islower determina se seu argumento é uma letra minúscula (a-z). A função isupper determina se seu argumento é uma letra maiúscula (A-Z).
• A função isdigit determina se seu argumento é um dígito (0-9).
• A função isalpha determina se seu argumento é uma letra maiúscula (A-Z) ou uma letra minúscula (a-z).
• A função isainum determina se seu argumento é uma letra maiúscula (A-Z), uma letra minúscula (a-z), ou um dígito (O-
9).
• A função isxdigit determina se seu argumento é um dígito hexadecimal (A-F. a-f, 0-9).

• A função toupper converte uma letra minúscula em uma letra maiúscula. A função tolower converte uma letra maiúscula em uma letra minúscula.
• A função isspace determina se seu argumento é um dos seguintes caracteres impressos como caracteres de espaço em branco: (espaço), ‘\f’, ‘\n, ‘\r’. ‘\t’ ou
• A função iscntrl determina se seu argumento é um dos seguintes caracteres de controle: ‘\t’, \v , \f , ‘\a’. ‘\b’, ‘\r’ ou

CAPÍTULO 16 - BIT5, CARACTERES, STRINGS E ESTRUTURAS 845
• A função ispunct determina se seu argumento é um caractere de impressão diferente de um espaço, um dígito ou uma letra.
• A função isprint determina se seu argumento é qualquer caractere imprimível, inclusive o caractere espaço.
• A função isgraph determina se seu argumento é um caractere de impressão diferente do caractere espaço.
• A função atof converte seu argumento - um string que começa com uma série de dígitos que representa um número de ponto flutuante - em um valor double.
• A função atoi converte seu argumento - um string que começa com uma série de dígitos que representa um inteiro - em um valor int.
• A função atol converte seu argumento - um string que começa uma série de dígitos que representa um inteiro long - em um valor long.
• A função strtod converte uma seqüência de caracteres representando um valor de ponto flutuante para double. A função recebe dois argumentos - um string (char *) e um ponteiro para char*. O string contém a seqüência de caracteres a serem convertidos e ao ponteiro para char* é atribuído o restante do string depois da conversão.
• A função strtol converte uma seqüência de caracteres representando um inteiro long. A função recebe três argumentos - um string (char*). um ponteiro para char* e um inteiro, O .string contém a seqüência de caracteres a ser convertida, ao ponteiro para char* é atribuído o restante do string depois da conversão e o inteiro especifica a base do valor que está sendo convertido.
• A função strtoul converte uma seqüência de caracteres representando um inteiro para unsigned long. A função recebe três argumentos - um string (char *), um ponteiro para char* e um inteiro, O string contém a seqüência de caracteres a ser convertida, ao ponteiro para char* é atribuído o restante do string depois da conversão e o inteiro especifica a base do valor que está sendo convertido.
• A função strchr procura a primeira ocorrência de um caractere em um string. Se o caractere é encontrado, strchr retorna um ponteiro para o caractere no string; caso contrário, strchr retoma NULL.
• A função strcspn determina o comprimento da parte inicial do srring em seu primeiro argumento que não contém quaisquer caracteres do string em seu segundo argumento. A função retoma o comprimento do segmento.
• A função strpbrk procura a primeira ocorrência em seu primeiro argumento de qualquer caractere que aparece em seu segundo argumento. Se um caractere do segundo argumento é encontrado, strpbrk retoma um ponteiro para o caractere; caso contrário, strpbrk retorna NULL.
• A função strrchr procura a última ocorrência de um caractere em um string. Se

o caractere é encontrado, strrchr retorna um ponteiro para o caractere no string; caso contrário, ela retoma NULL.
• A função strspn determina o comprimento da parte inicial do string em seu primeiro argumento que contém somente caracteres do string em seu segundo argumento. A função retoma o comprimento do segmento.
• A função strstr procura a primeira ocorrência de seu segundo argumento string em seu primeiro argumento string. Se o segundo string é encontrado no primeiro string, é retornado um ponteiro para a localização do string no primeiro argumento.
• A função memcpy copia um número especificado de caracteres do objeto para o qual seu segundo argumento aponta para o objeto que seu primeiro argumento aponta. A função pode receber um ponteiro para qualquer tipo de objeto. Os ponteiros são recebidos por memcpy como ponteiros void e convertidos em ponteiros char para uso na função. A função memcpy manipula os bytes do argumento como caracteres.
• A função memmove copia um número especificado de bytes do objeto apontado por seu segundo argumento para o objeto apontado por seu primeiro argumento. A cópia é realizada como se os bytes fossem copiados do segundo argumento para um array de caracteres temporário e, então, copiados do array temporário para o primeiro argumento.
• A função memcmp compara o número especificado de caracteres de seu primeiro e segundo argumentos.
• A função memchr procura a primeira ocorrência de um byte, representado como urisigned char, no número especificado de bytes de um objeto. Se o byte é encontrado, um ponteiro para o byte é retomado; caso contrário, um ponteiro NULL é retornado.
• A função memset copia seu segundo argumento, tratado como um unsigned char, para um número especificado de bytes do objeto apontado pelo primeiro argumento.
• A função strerror mapeia um número de erro de tipo inteiro para um string de texto, de uma maneira dependente de sistema. E retornado um ponteiro para o string.

Terminologia
& operador AND sobre bits isalpha
&= operador de atribuição AND sobre bits iscntrl
operador OR exclusivo sobre bits isdigit
A= operador de atribuição OR exclusivo sobre bits isgraph
1 operador OR inclusivo sobre bits islower
= operador de atribuição OR inclusivo sobre bits isprint
operador complemento de 1 ispunct
«operador de atribuição de deslocamento à esquerda isspace
«operador de deslocamento à esquerda isupper
<cctype> isxdigit
<cstdlib> literal

<string> máscara
» operador de atribuição de deslocamento à direita mascarando bits
» operador de deslocamento à direita memchr
array de estruturas memcmp
ASCII memcpy
atof memmove
atoi memset
atol operadores sobre bits
atribuição de estrutura ponteiro para uma estrutura
bibliotecade utilitários genéricos preenchimento
campo de bits processamento de strings
campo de bits de comprimento zero processamento de texto
campo de bits sem nome registro
caractere de controle strchr
caractere imprimível strcspn
caracteres de espaço em branco strerror
código de caractere string
complementando string de procura
complemento de 1 string literal
comprimento de um campo de bits strpbrk
conjunto de caracteres strrchr
constante de caractere strspn
constante do tipo string strstr
delimitador strtod
deslocamento à direita strtol
deslocamento à esquerda strtoul
dígitos hexadecimais struct
estrutura auto-referente tipo de uma estrutura
fazer um deslocamento tolower
funções de conversão de strings toupper
inicialização de estrutura troca espaço versus tempo de execução
inicialização de estruturas typedef
isainum
Erros comuns de programação
16.1 Esquecer o ponto-e-vírgula que termina uma definição de estrutura.
16.2 Comparar estruturas é um erro de sintaxe por causa dos diferentes requisitos de alinhamento nos diversos sistemas.
16.3 Assumir que estruturas, como arrays, são passados através de chamadas por referência e tentar modificar os valores da estrutura do chamador na função chamada.
16.4 Esquecer de incluir o índice de array quando referenciar estruturas individuais em um array de estruturas.
16.5 Usar o operador AND lógico (&&) como o operador AND sobre bits (&), e vice-versa.
16.6 Usar o operador OR lógico (1 1) em vez do operador OR sobre bits (1), e vice-versa.

16.7 O resultado de fazer um deslocamento de um valor é indefinido se o operando direito for negativo ou se o operando direito for maior que o número de bits em que o operando esquerdo está armazenado.
16.K Tentar acessar bits individuais de um campo de bits como se fossem elementos de um array. Campos de bits não são “array de bits”.

CAPITULO 16 - Brrs, CARACTERES, STRÍNGS E ESTRUTURAS 847

16.9 Tentar pegar o endereço de um campo de bits (o operador& não pode ser usado com campos de bits porque eles não têm endereços).
16.10 As funções de manipulação de strings que copiam caracteres, exceto memmove. têm resultados indefinidos quando a cópia ocorre entre partes do mesmo string.
Boas práticas de programação
16.1 Forneça um nome de estrutura quando criar um tipo de estrutura. O nome de estrutura é conveniente para declarar novas variáveis do tipo da estrutura mais à frente no programa e é obrigatório se a estrutura for passada como um parâmetro para uma função.
16.2 Comece nomes typedef com maiúsculas, para enfatizar que estes nomes são sinônimos de outros nomes de tipo.
Dicas de desempenho
16.1 Passar estruturas (especialmente estruturas grandes) através de chamada por referência é mais eficiente do que passar estruturas através de chamadas por valor (o que exige a cópia da estrutura inteira).
16.2 Campos de bits ajudam a economizar memória.
16.3 Embora campos de bits economizem espaço, usá-los pode fazer com que o compilador gere código de linguagem de máquina de execução mais lenta. Isto acontece porque são necessárias operações adicionais em linguagem de máquina para acessar somente partes de uma unidade de memória endereçável. Este é um dos muitos exemplos dos tipos de trocas de espaço versus tempo de execução que acontecem em Ciência da Computação.
Dicas de portabilídade
16.1 Como o tamanho dos itens de dados de um tipo particular é dependente de máquina e como considerações de alinhamento de memória são dependentes de máquina, assim também o é a representação de uma estrutura.
16.2 Usar typedef pode ajudar a tomar um programa mais portável.
16.3 As manipulações de dados sobre bits são dependentes de máquina.
16.4 O resultado de fazer um deslocamento à direita de um valor com sinal é dependente de máquina. Algumas máquinas preenchem com zero e outras usam o bit de sinal.
16.5 As manipulações de campos de bit são dependentes de máquina. Por exemplo, alguns computadores permitem que campos de bits cruzem limites de palavra, enquanto outros não.
16.6 O tipo size_t é um sinônimo dependente de sistema ou para um tipo unsigned long ou para um tipo unsigned int.
16.7 A mensagem gerada por strerror é dependente do sistema.

Exercícios de auto-revisão

16.1 Preencha os espaços em branco em cada um dos seguintes itens:

a) Uma _____ é uma coleção de variáveis relacionadas com um nome.

b) Os bits no resultado de uma expressão usando o operador são colocados em um se os bits correspondentes em cada operando são iguais a um. Caso contrário, os bits são colocados em zero.

c) As variáveis declaradas em uma definição de estrutura são chamadas de seus _____

d) Os bits no resultado de uma expressão que usa o operador são colocados em um se pelo menos um dos bits correspondentes em um ou outro operando tem o valor um. Caso contrário, os bits são colocados em zero.

e) A palavra-chave inicia uma declaração de estrutura.

f) A palavra-chave é usada para criar um sinônimo para um tipo de dados previamente definido.

g) Os bits no resultado de uma expressão usando o operador são colocados em um se exatamente um dos bits correspondentes em um ou outro operando tem o valor um. Caso contrário, os bits são colocados em zero.

h) O operador AND sobre bits & é usado freqüentemente para bits (i.e., para selecionar certos bits de um string de bits enquanto zera outros).

i) O nome da estrutura é chamado de _____ da estrutura.

j) Um membro de estrutura é acessado tanto com o operador como com o operador

k) Os operadores e _____ são usados para deslocar os bits de um valor para a esquerda ou para a direita, respectivamente.

16.2 Diga se cada um dos seguintes itens é verdadeiro ou falso. Se falso, explique por quê.

a) Estruturas podem conter somente um tipo de dados.

b) Os membros de estruturas diferentes devem ter nomes únicos.

e) A palavra-chave typedef é usada para definir novos tipos de dados.

d) Estruturas são sempre passadas para funções por chamadas por referência.

16.3 Escreva um comando único ou um conjunto de comandos para realizar cada um dos seguintes:

a) Defina uma estrutura chamada Part que contém a variável int partNumber e o array char partNa!ne cujos valores podem ser tão longos quanto 25 caracteres.

b) Defina PartPtr como um sinônimo para o tipo Part .

c) Declare a variável a como sendo do tipo Part. o array b [10] como sendo do tipo Part e a variável ptr como sendo um ponteiro para o tipo Part.

d) Leia um número de peça (partNumber) e um nome de peça (partName) do teclado para os membros da variável

a.

e) Atribua os valores dos membros da variável a ao elemento três do array b.

f) Atribua o endereço do array b à variável ponteiro ptr.

g) Imprima os valores dos membros do elemento três do array b usando a variável ptr e o operador de ponteiro de estrutura para se referir aos membros.

16.4 Encontre o erro em cada um dos seguintes itens:
a) Assuma que a struct Card foi definida contendo dois ponteiros para o tipo char, isto é, face e suit. Além disso, a variável c foi declarada como sendo do tipo Card e a variável cPtr foi declarada como sendo um ponteiro para o tipo Card. À variável cPtr foi atribuído o endereço de c.
cout « *cptr face « endi;
b) Assuma que a struct Card foi definida contendo dois ponteiros para o tipo char. isto é, face e suit. Além disso, o array hearts [13] foi declarado como sendo do tipo Card. O comando seguinte deveria imprimir o membro face do elemento lOdo array.
cout « hearts.face « endi;
e) struct Pessoa
char sobreNome [15];
char primeiroNome[15];
int idade;
d) Assuma que a variável p foi declarada como do tipo Pessoa e a variável c foi declarada como do tipo Card. p = c;
16.5 Escreva um único comando para realizar cada um dos seguintes. Assuma que as variáveis c (que armazena um caractere),
x, y e z são do tipo int, as variáveis d, e e f são do tipo fioat, a variável ptr é do tipo char * e os arrays si [100] e
s2 [100] são do tipo char.
a) Converta o caractere armazenado na variável c para uma letra maiúscula. Atribua o resultado à variável c.
b) Determine se o valor da variável c é um dígito. Use o operador condicional como mostrado nas Figs. 16.17, 16.18 e 16.19 para imprimir “é um’ ou “não é um” quando o resultado é exibido.
e) Converta o string “1234567” para long e imprima o valor.
d) Determine se o valor da variável c é um caractere de controle. Use o operador condicional para imprimir “é um” ou “não é um” quando o resultado for exibido.
e) Atribua a ptr a posição da última ocorrência de cem si.
f) Converta o string “8. 63582” para doubie e imprima o valor.
g) Determine se o valor de c é uma letra. Use o operador condicional para imprimir “é uma “ ou “não é uma quando o resultado for exibido.
h) Atribua a ptr a posição da primeira ocorrência de s2 em si.
i) Determine se o valor da variável c é um caractere imprímivel. Use o operador condicional para imprimir “é um ou “não é um” quando o resultado for exibido.
j) Atribua a ptr a posição da primeira ocorrência em si de qualquer caractere de s2.
k) Atribua a ptr a posição da primeira ocorrência de c em si.
1) Converta o string “- 21” para int e imprima o valor.

Respostas aos exercícios de auto-revisão
16.1 a) estrutura. b) AND(&) sobre bits. c) membros. d) OR inclusivo (1) sobre bits. e) struct. f) typedef. g) OR exclusivo (A) sobre bits. h) máscara. i)

identificador. j) membro de estrutura (.), ponteiro de estrutura (->). k) operador de deslocamento à esquerda («), operador de deslocamento à direita (»).

16.2 a) Falso. Uma estrutura pode conter muitos tipos de dados.

b) Falso. Os membros de estruturas separadas podem ter os mesmos nomes, mas os membros da mesma estrutura deve ter nomes únicos.

e) Falso. typedef é usado para definir nomes alternativos para tipos de dados previamente definidos.

d) Falso. As estruturas são sempre passadas para funções por chamadas por valor.

16.3 a) struct Part
int partNumber;
char partName[26];

b) typedef Part * PartPtr;

e) Part a, b[10], *ptr;

d) cm » a.partNu!nber » a.partNaxne;

e) b[3] = a;

f) ptr =

g) cout « (ptr + 3)->partNuriber «
« (ptr + 3)->partName « endi;

16.4 a) Erro: os parênteses que deveriam incluir *cPtr foram omitidos, fazendo com que a ordem Ue avaliaçao Ga expressao fique incorreta.

b) Erro: o índice do array foi omitido. A expressão deveria ser hearts [10] . face.

e) Erro: é necessário um ponto-e-vírgula para terminar uma definição de estrutura.

d) Erro: variáveis de diferentes tipos de estrutura não podem ser atribuídas umas às outras.

16.5 a) c = toupper(c);

b) cout « '\'' « c « "\'
« (isdigit(c) ? "é um " "não é um
« "dígito" « endi;

e) cout « atol("1234567") « endi;

d) cout « '\'' « c « "\'
« (iscntrl (e) ? "é um " : "não é um
« "caractere de controle" « endl;

e) ptr = strrchr(si, c);

f) out « atof("8.63582") « endi;

g) cout « '\'' « c « "\'
« (isalpha (e) ? "é uma " : "não é uma
« "letra" « endi;

h) ptr = strstr(si, s2);

i) cout « '\' ' « c « "\'
« (isprint(c) ? "é um ": 'não é um
« "caractere imprimível" « endi;

j) ptr = strpbrk(si, s2);

k) ptr = strchr(si, e
1) cout « atoi(“-21”) « endi;
Exercícios
16.6 Forneça a definição para cada uma das estruturas e uniões seguintes:
a) Estrutura Inventory que contém o array de caracteres partName [30]. o inteiro partNumber, o ponto flutuante price. o inteiro stock e o inteiro reorder.
b) Uma estrutura chamada Endereco que contém os arrays de caracteres enderecoNaRua [25], cidade [20]. estado[3] ecep[6].

e) Estrutura Estudante que contém os arrays primeiroNome [15 e sobreNome [15], e a variável enderecoResidencial do tipo Endereco da parte (b).
d) Estrutura Teste que contém 16 campos de bits com comprimentos de 1 bit. Os nomes dos campos de bits são as letras de a a p.
16.7 Considere as seguintes definições de estrutura e declarnções de variáveis:
struct
Customer { \\ Cliente
char lastName[15 1; \\ sobrenome
char firstName[15]; \\ primeiro nome
int customerNumber; \\ número do cliente
struct
char phoneNumber[11]; \\ número do telefone
char address[50]; \\ endereço
char city[15]; \\ cidade
char state[3]; \\ estado
char zipCode[6 1; \\ CEP
personal; \\ dados pessoais
} customerRecord, *customerptr;
customerPtr = &customerRecord;
Escreva uma expressão separada que acesse cada uma das seguintes partes dos membros da estrutura:
a) Membro lastName da estrutura customerRecord.
b) Membro las tName da estrutura apontada por customerPtr.
c) Membro firstName da estrutura customerRecord.
d) Membro firstName da estrutura apontada por customerPtr.
e) Membro customerNumber da estrutura customerRecord.
f) Membro customerNumber da estrutura apontada por customerPtr.
g) Membro phoneNumber do membro personal da estrutura customerRecord.
h) Membro phoneNuniber do membro personal da estrutura apontada por customerPtr.
i) Membro address do membro personal da estrutura customerRecord.
j) Membro adress do membro personal da estrutura apontada por customerPtr.
k) Membro city de membro personal da estrutura customerRecord.
1) Membro city do membro personal da estrutura apontada por customerPtr.
m) Membro state do membro personal da estrutura customerRecord.

n) Membro state do membro personal da estrutura apontada por customerPtr.
o) Membro zipcode do membro personal da estrutura customerRecord.
p) Membro zipCode do membro personal da estrutura apontada por customerPtr.

16.8 Modifique o programa para embaralhamento de cartas da Fig. 16.14 usando uma função shuffle de alto desempenho
comomostrado na Fig. 16.2. Imprima o baralho resultante em um formato de duas colunas, como na Fig. 16.3. Preceda cada carta com sua cor.
16.9 Escreva um programa que desloca para a direita uma variável inteira de 4 bits. O programa deve imprimir o inteiro em binário antes e depois da operação de deslocamento. O seu sistema coloca zeros ou uns nos bits desocupados?
16.10 Se seu computador usa inteiros de 4 bytes, modifique o programa da Fig. 16.5 de forma que ele trabalhe com inteiros de
4 bytes.
16.11 Fazer um deslocamento à esquerda de um inteiro unsigned por 1 bit é equivalente a multiplicar o valor por 2. Escreva a função power2 que recebe dois argumentos inteiros number e pow e calcula
nuniber * 2P0
Use um operador de deslocamento para calcular o resultado. O programa deve imprimir os valores como inteiros em decimal e

em binário.

16.12 O operador de deslocamento à esquerda pode ser usado para agrupar dois valores do tipo caractere em uma variável inteira uns igned de 2 bytes. Escreva um programa que recebe como entrada dois caracteres do teclado e passa-os para a função packCharacters. Para empacotar dois caracteres em uma variável inteira unsigned. atribua o primeiro caractere à variável unsigned, desloque a variável unsigned para a esquerda 8 posições de bit e combine a variável unsigned com o segundo caractere usando o operador sobre bits 0k inclusivo. O programa deve exibir os caracteres em seu formato binário antes e depois de serem agrupados no inteiro unsigned, para provar que os caracteres foram de fato agrupados corretamente na variável uns igned.
16.13 Usando o operador de deslocamento à direita, o operador AND sobre bits e uma máscara, escreva a função unpackCharacters que recebe o inteiro unsigned do Exercício 16.12 e desempacota o mesmo em dois caracteres. Para desempacotar dois caracteres de um inteiro de 2 bytes unsigned, combine o inteiro unsigned com a máscara 65280 (1111111100000000) e desloque o resultado 8 bits à direita. Atribua o valor resultante a uma variável char. Então, combine o inteiro unsigned com a máscara 255 (0000000011111111). Atribua o resultado a uma outra variável char. O programa deve imprimir o inteiro unsigned em binário antes de ser desempacotado e, então, imprimir os caracteres em binário para confirmar que eles foram corretamente desempacotados.
16.14 Se seu sistema usa inteiros de 4 bytes, rescreva o programa do Exercício

16.12 para empacotar 4 caracteres.
16.15 Se seu sistema usa inteiros de 4 bytes, rescreva a função unpackCharacters do Exercício 16.13 para desempacotar
4 caracteres. Crie as máscaras de que você necessita para desempacotar os 4 caracteres fazendo um deslocamento à esquerda do
valor 255 na variável máscara por 8 bits O, 1, 2, ou 3 vezes (dependendo do byte que você está desempacotando).
16.16 Escreva um programa que inverte a ordem dos bits em um valor inteiro unsigned. O programa deve receber como entrada o valor do usuário e chamar a função reverseBits para imprimir os bits em ordem inversa. Imprima o valor em binário tanto antes como depois dos bits serem invertidos, para confirmar que os bits são invertidos corretamente.
16.17 Escreva um programa que demonstre a passagem de um array por valor (dica: use uma struct). Prove que uma cópia foi passada modificando-se a cópia do array na função chamada.
16.18 Escreva um programa que lê um caractere do teclado e testa o caractere com cada uma das funções da biblioteca de manipulação de caracteres. O programa deve imprimir o valor retomado por cada função.
16.19 O programa seguinte usa a função multiple para determinar se o inteiro lido do teclado é um múltiplo de algum inteiro X. Examine a função multiple e, então, determine o valor dcx.

```
1 /1 ex1619.cpp
2 // Este programa determina se um valor é um múltiplo de X
3 #include <iostream>
4
5 using std::cout;
6 using std::cin;
7 using std::endl;
8
9 bool multiple( int );
10
11 int main()
12
13 int y;
14
15 cout « ‘Digite um inteiro entre 1 e 32000: ;
16 cm » y;
17
18 if ( multiple( y )
19 cout « y « “ é um múltiplo de X” « endl;
20 else
21 cout « y « não é um múltiplo de X” « endl;
22
23 return 0;
```

```
24 }
25
26 bool multiple( int num
27 {
28 bool mult = true;
29
30 for ( int i = 0, mask = 1; i < 10; i++, mask «= 1
31 if ( ( num & mask ) != O ) {
32 mult = false;
33 break;
34 )
35
36 return mult;
37 )
```

16.20 O que faz o programa a seguir?

```
1 II ex1620.cpp
2 #include <iostream>
3
4 using std: :cout;
5 using std::cin;
6 using std: :endl;
7
8 int mystery( unsigned );
9
10 int main()
11 {
12 unsigned x;
13
14 cout « “Digite um inteiro:
15 cm » x;
16 cout « “O resultado é “ « mystery( x ) « endl;
17 return 0;
18
19
20 int mystery( unsigned bits
21 {
22 const int SHIFT = 8 * sizeof( unsigned ) - 1;
23 const unsigned MASK = 1 « SHIFT;
24 unsigned total = 0;
25
26 for ( int i = 0; i < SHIFT + 1; i++, bits «= 1
27 if ( ( bits & MASK ) == MASK
28 ++total;
29
30 return ! ( total % 2 );
31
```

16.21 Escreva um programa que lê uma linha de texto com a função membro getline de istream (ver Capítulo 11) para o array de caracteres s [100 J. Imprima a linha em letras maiúsculas e letras minúsculas.
16.22 Escreva um programa que lê quatro strings que representam inteiros, converte os strings em inteiros, soma os valores e imprime o total dos quatro valores.

CAPÍTULO 16 - BITs, CARACTERES, STRINGS E ESTRUTURAS 853
16.23 Escreva um programa que lê quatro strings que representam valores em ponto flutuante, converte os strings para valores double. soma os valores e imprime o total dos quatro valores.
16.24 Escreva um programa que lê uma linha de texto e um string para busca pelo teclado. Usando a função strstr, localize a primeira ocorrência do string para busca na linha de texto e atribua esta posição à variável searchPtr do tipo char*. Se o string para busca for encontrado, imprima o restante da linha de texto, começando com o string para busca. Então, use strstr novamente para localizar a próxima ocorrência do string para busca na linha de texto. Se uma segunda ocorrência for encontrada. imprima o restante da linha de texto começando com a segunda ocorrência. (Dica: a segunda chamada de strstr deve conter searchPtr+1 como seu primeiro argumento).
16.25 Escreva um programa baseado no programa do Exercício 16.24 que lê várias linhas de texto e um string para busca e usa a função strstr para determinar o número total de ocorrências do string nas linhas de texto. Imprima o resultado.
16.26 Escreva um programa que lê várias linhas de texto e um caractere de busca e usa a função strchr para determinar o número total de ocorrências do caractere nas linhas de texto.
16.27 Escreva um programa baseado no Exercício 16.26 que lê várias linhas de texto e usa a função strchr para determinar o número total de ocorrências de cada letra do alfabeto no texto. Letras maiúsculas e minúsculas devem ser contadas juntas. Armazene os totais para cada letra em um array e imprima os valores em formato de tabela após terem sido determinados os totais.
16.28 A tabela no Apêndice B mostra as representações em códigos numéricos dos caracteres do conjunto ASCII de caracteres. Estude essa tabela e, então, diga qual das seguintes afirmações é verdadeira ou falsa.
a) A letra ‘A’ vem antes da letra “B”.
b) O dígito “9” vem antes do dígito “O”.
e) Os símbolos comumente usados para adição, subtração, multiplicação e divisão vêm todos antes dos dígitos.
d) Os dígitos vêm antes das letras.
e) Se um programa de classificação classifica strings em ordem ascendente, então o programa colocará o símbolo para um parêntese à direita antes do símbolo para um parêntese à esquerda.
16.29 Escreva um programa que lê uma série de strings e imprime somente aqueles strings começando com a letra “b”.
16.30 Escreva um programa que lê uma série de strings e imprime somente

aqueles strings que terminam com as letras “ADO”.
16.31 Escreva um programa que lê um código ASCII e imprime o caractere correspondente. Modifique este programa de maneira que ele gere todos os códigos de três dígitos possíveis, no intervalo de 000 a 255, e tente imprimir os caracteres correspondentes. O que acontece quando este programa é executado?
16.32 Usando a tabela de caracteres ASCH do Apêndice B como guia, escreva suas próprias versões das funções de manipulação de caracteres da Fig. 16.16.
16.33 Escreva suas próprias versões das funções da Fig. 16.20 para conversão de strings em números.
16.24 Escreva suas versões das funções da Fig. 16.27 para pesquisa em strings.
16.35 Escreva suas próprias versões das funções da Fig. 16.34 para manipulação de blocos de memória.
16.36 (Projeto:um verificador ortográfico) Muitos pacotes de software para processamento de texto populares têm verificadores ortográficos incorporados. Usamos recursos de verificação ortográfica na preparação deste livro e descobrimos que, não importando o cuidado que pensávamos ter usado na preparação de um capítulo, o software sempre era capaz de encontrar mais alguns erros de ortografia além dos que havíamos descoberto manualmente. Neste projeto, é-lhe pedido que você desenvolva o seu próprio utilitário de verificação ortográfica. Fazemos sugestões
para ajudá-lo a começar. Você deve então pensar em adicionar mais recursos. Você pode achar útil usar um dicionário computadorizado como fonte de palavras.
Por que digitamos tantas palavras com erros de ortografia? Em alguns casos, simplesmente não conhecemos a grafia correta e, assim, “damos o chute que parace mais razoável”. Em alguns casos, trocamos duas letras (p. ex., “pardão” em vez de “padrão”). Algumas vezes, digitamos duas vezes a mesma letra acidentalmente (p. ex., “dooce” em vez de “doce”). Algumas vezes, tocamos uma tecla próxima daquela que queríamos (p. ex., “anibersário” em vez de “aniversário”). E assim por diante.
Projete e implemente um programa de verificação ortográfica. O seu programa mantém um array wordList de strings
de caracteres. Você pode digitar estes strings ou obtê-los de um dicionário computadorizado.
O seu programa solicita ao usuário que forneça uma palavra. O programa então procura aquela palavra no array wordList. Se a palavra consta no array, o seu programa deve imprimir “Palavra está grafada corretamente”.

Se a palavra não consta no array, seu programa deve imprimir “Palavra não está grafada corretamente”. Então, seu programa deveria procurar outras palavras em wordList, uma das quais pode ser a palavra que o usuário pretendia digitar. Por exemplo, você pode tentar todas as trocas simples de letras adjacentes no teclado para descobrir que a palavra “padrão” corresponde exatamente a uma

das palavras em wordList. Naturalmente, isso implicaria que seu programa testasse todas as trocas simples, como “apdrão”, “pdarão”, “pardão”, “padãro” e “padrão”. Quando você encontra uma nova palavra igual a uma em wordList, imprima aquela palavra em uma mensagem tal como “Você quis dizer “padrão” ?“.
Implemente outros testes, tais como substituir cada letra duplicada por uma única letra e quaisquer outros testes que você
possa desenvolver para aumentar a utilidade do seu verificador ortográfico.

17
O pré-processador
Objetivos
Ser capaz de usar #include para desenvolver programas
de usar #define para criar macros e macros
com argumentos
,(7 Compreender a compilação condicional
> • Ser capaz de exibir mensagens de erro durante a compila çã condicional
• Ser capaz de usar asserções para testar se os valores de ex pressõe estão corretos
Mantenha o bom; defina-o bem.
Alfred, Lord Tennyson
Encontrei-lhe um argumento; mas não
sou obrigado afazê-lo entender
Samuel Johnson
Um bom símbolo é o melhor argumento
e é capaz de persuadir milhares.
Raph Waldo Emerson
Condições são fundamentalmente saudáveis.
Herbert Hoover (dezembro de 1929)
O partidário, ao se engajar em uma disputa, não se preocupa em nada com os direitos da questão, tnas fica ansioso somente para convencer seus ouvintes de suas próprias afirmações. Platão

Visão Geral
17.1 Introdução
17.2 A diretiva #include do pré-processador
17.3 A diretiva #define do pré-processador: constantes simbólicas
17.4 A diretiva #define do pré-processador: macros
17.5 Compilação condicional
17.6 As diretivas #error e #pragma do pré-processador
17.7 Os operadores # e ##
17.8 Números de linhas
17.9 Constantes simbólicas predefinidas
17.10 Asserções
Resumo . Terminologia Erros comuns de programação Boas práticas de programação Dica de desempenho• Observação de engenharia de software Exercícios de auto-revisão Respostas aos exercícios de auto-revisão • Exercícios
17.1 Introdução

Este capítulo apresenta o pré-processador O pré-processamento ocorre antes de um programa ser compilado. Algumas das ações possíveis são: inclusão de outros arquivos no arquivo que está sendo compilado, definição de constantes simbólicas e macros, compilação condicional do código do programa e execução condicional das diretivas do pré-processador. Todas as diretivas do pré-processador começam com e somente caracteres de espaço em branco podem aparecer antes de uma diretiva para o pré-processador em uma linha. As diretivas para o pré-processador não são comandos de C++, de modo que não terminam com um ponto-e-vírgula (;). As diretivas para o pré-processador são completamente processadas antes de a compilação começar.
Erro comum de programação 17.1
Colocar um ponto-e-vfrgula no fim de uma diretiva para o pré-processador pode levar a diversos erros,
dependendo da diretiva para o pré-processador
Observação de engenharia de software 17.1
_____ Muitos dos recursos do pré-processador (especialmente macros) são mais apropriados para pro gramadores de C do que para programadores de C+ +. Os programadores de C+ + devem se familiarizar com o pré-processador porque eles podem necessitar trabalhar com código legado em C.
17.2 A diretiva #include do pré-processador
A diretiva do pré-processador #include tem sido usada ao longo de todo este texto. A diretiva #include faz com que uma cópia de um arquivo especificado seja incluída no lugar da diretiva. As duas formas da diretiva #include são:
#include <filename>
#include filename”
A diferença entre estas duas é a localização em que o pré-processador procura o arquivo a ser incluído. Se o nome do arquivo (“filename”) está incluso entre os sinais de menor e maior (< e >) - usado para arquivos de cabeçalho da

biblioteca padrão - o pré-processador procura o arquivo especificado de uma maneira dependente da imp1ementaço, normalmente em diretórios pré-designados. Se o nome do arquivo está incluso entre aspas, o pré-processador primeiro pesquisa o diretório em que está o arquivo que está sendo compilado e, então, da mesma maneira dependente de implementação que usa para um nome de arquivo incluso entre menor e maior. Este método é normalmente usado para incluir arquivos de cabeçalho definidos pelo programador.
A diretiva #include é normalmente usada para incluir arquivos de cabeçalho padrão como <iostreaxn> e <iomanip>. A diretiva #include também é usada com programas que consistem em diversos arquivos-fonte que devem ser compilados juntos. Um arquivo de cabeçalho contendo declarações e definições comuns aos arquivos de programas separados freqüentemente é criado e incluído no arquivo. Exemplos de tais declarações são dasses, estruturas, uniões, enumerações e protótipos de funções.
17.3 A diretiva #define do pré-processador: constantes simbólicas
A diretiva para o pré-processador #define cria constantes simbólicas -

constantes representadas como símbolos
- e macros - operações definidas como símbolos. O formato da diretiva #define para o pré-processador é #define identificador texto de substituição
Quando esta linha aparece em um arquivo, todas as ocorrências subseqüentes de identificador (exceto aquelas dentro de um string) serão substituídas pelo texto de substituição antes que o programa seja compilado. Por exemplo,
#define P1 3.14159
substitui todas as ocorrências subseqüentes da constante simbólica P1 pela constante numérica 3. 14159. Constantes simbólicas permitem ao programador criar um nome para uma constante e usar esse nome ao longo de todo o programa. Se a constante precisar ser modificada em todo o programa, ela pode ser modificada apenas uma vez na diretiva #define - e quando o programa é recompilado, todas as ocorrências da constante no programa serão modificadas.
Nota: tudo o que está à direita do nome da constante simbólica substitui a constante simbólica. Por exemplo, #define P1 = 3.14159 faz com que o compilador substitua cada ocorrência de P1 por = 3.14159. Isto é a causa de muitos erros sutis de lógica e de sintaxe. Redefinir uma constante simbólica com um novo valor também é um erro. Note que em C÷+ é preferível usar variáveis const em vez de constantes simbólicas. Variáveis constantes têm um tipo de dado específico e são visíveis por nome para um depurador. Uma vez que uma constante simbólica é substituída pelo seu texto de substituição, somente o texto de substituição permanece visível para o depurador. Uma desvantagem das variáveis const é que elas podem necessitar de uma posição de memória do tamanho do seu tipo de dado - constantes simbólicas não necessitam de qualquer quantidade de memória adicional.
Erro comum de programação 17.2
Usar constantes simbólicas em um arquivo diferente daquele no qual as constantes são definidas é um
erro de sintaxe.
Boa prática de programação 17.1
Usar nomes que têm um significado para constantes simbólicas ajuda a tornar os programas mais auto-
documentados.
17.4 A diretiva #define do pré-processador: macros
[Nota: esta seção foi incluída para auxiliar os programadores de C++ que necessitarão trabalhar com código legado em C. Em C++, as macros foram substituídas por gabaritos e funções mimei. Uma macro é uma operação definida em uma diretiva #define para o pré-processador. Da mesma maneira que ocorre com as constantes simbólicas, o identificador da macro é substituído pelo texto de substituição antes de o programa ser compilado. Macros podem ser definidas com ou sem argumentos. Uma macro sem argumentos é processada como uma constante simbólica. Em uma macro com argumentos, os argumentos são substituídos no texto de substituição e, então, a

macro é expandida - ou seja, o texto de substituição substitui o identificador da

macro e a lista de argumentos no programa. (Nota: não há verificação de tipos de dados para os argumentos de uma macro. Uma macro é usada simplesmente para substituição de texto).
Considere a seguinte definição de uma macro com um argumento, para o cálculo da área de um círculo:
#define CIRCLEAREA(x) (P1 * (x) * (x
Onde quer que CIRCLE_AREA (x) apareça no arquivo, o valor de x substitui x no texto de substituição, a constante simbólica P1 é substituída pelo seu valor (definido anteriormente) e a macro é expandida no programa. Por exemplo, o comando
area = CIRCLEAREA(4);
é expandido para
area = (3.14159 * (4) * (4))
Como a expressão consiste apenas em constantes, durante a compilação o valor da expressao e calculado e o resultado é atribuído a area durante a execução. Os parênteses em torno de cada x, no texto de substituição, e em torno de toda a expressão forçam a ordem correta de cálculo quando o argumento da macro é uma expressão. Por exemplo, o comando
= CIRCLEAREA(c = 2);
é expandido para
area= (3.14159 * (c+2) * (c+2));
a qual é avaliada corretamente porque os parênteses forçam a ordem certa de cálculo. Se os parênteses são omitidos. a expansão da macro é
area=3.14159 * c+2 * c+ 2;
que é calculada, incorretamente, como
area= (3.14159 * c) * (2 * C) + 2;
por causa das regras de precedência de operadores.
Erro comum de programação 17.3
Esquecer de colocar entre parênteses argumentos de uma macro, no texto de substituição.
A macro CIRCLE AREA poderia ser definida como uma função. A função circleArea
double circleArea (double x) { returri 3.14159 * x * x;
executa o mesmo cáculo que CIRCLE AREA. mas o overhead de uma chamada de função está associado com a função circleArea. As vantagens do uso de CIRCLE AP.EA estão no fato de que macros inserem o código diretamente no programa - evitando overhead das funções - e o programa permanece legível porque CIRCLE_AREA é definida separadamente e recebe um nome com significado. Uma desvantagem é que o argumento é avaliado duas vezes. Além disso, cada vez que uma macro aparece no programa, a macro é expandida. Se a macro for grande, isso produz um aumento do tamanho do programa. Assim, existe um compromisso entre velocidade de execução e tamanho de programa (o espaço em disco pode ser pequeno). Note que funções in].ine (ver Capítulo 3) são preferíveis para obter o desempenho das macros e o benefício de engenharia de software das funções.

I-tDica de desempenho 17.1
-F Às vezes, as macros podem ser usadas para substituir uma chamada defunção por código mime, antes da execução do programa. Isso elimina o overhead de uma chamada de função. Funções mime são
preferíveis a macros, porque oferecem os serviços de vercação de tipos das funções.
A seguinte definição é uma definição de macro com 2 argumentos para o cálculo da área de um retângulo:
#define RECTANGLEAREA(x, y) ((x) (y)
Onde quer que RECTANGLE AREA (x, y) apareça no programa, os valores de x e y são substituídos no texto
de substituição da macro e a macro é expandida no lugar do nome da macro. Por exemplo, o comando
rectArea = RETANGLEAREA (a + 4, b + 7);
é expandido para:
rectàrea ((a+4) * (b+7));
O valor da expressão é avaliado e atribuído à variável rectArea.
O texto de substituição para uma macro ou constante simbólica normalmente é qualquer texto existente na linha após o identificador na diretiva #define. Se o texto de substituição para uma macro ou constante simbólica é mais longo que o restante da linha, uma barra invertida (\ ) deve ser colocada no fim da linha para indicar que o texto de substituição continua na próxima linha.
Constantes simbólicas e macros podem ser descartadas usando-se a diretiva para o pré-processador #undef. A diretiva #undef “anula a definição” do nome de uma constante simbólica ou macro. O escopo de uma constante simbólica ou macro vai da sua definição até o ponto em que sua definição é anulada com #undef ou até o final do arquivo.
As funções da biblioteca padrão algumas vezes são definidas como macros baseadas em outras funções de
biblioteca. Uma macro comumente definida no arquivo de cabeçalho <cstdio> é
#defirie getchar() getc(stdin
A definição de macro de getchar usa a função getc para obter um caractere do stream padrão de entrada. A função putchar do arquivo de cabeçalho <cstdio> e as funções de manipulação de caracteres do arquivo de cabeçalho <cctype> também são freqüentemente implementadas como macros. Note que expressões com efeitos colaterais (i.e, os valores de variáveis são modificados) não devem ser passadas para uma macro, porque os argumentos de uma macro podem ser avaliados mais de uma vez.
17.5 Compilação condicional
A compilação condicional permite ao programador controlar a execução das diretivas do pré-processador e a compilação do código do programa. Cada uma das diretivas condicionais para o pré-processador avalia uma expressão constante inteira que irá determinar se o código será compilado. Expressões de coerção, expressões com sizeof e constantes de enumerações não podem ser avaliadas em diretivas para o pré-processador.

A instrução condicional para o pré-processador é bastante semelhante à estrutura de seleção if. Considere o
seguinte código para o pré-processador:
#if !defined(NULL
#define NULL O
#endif
Essas diretivas determinam se a constante simbólica NULL já está definida. A expressão defined (NULL) é calculada, produzindo o valor 1 se NULL está definido; O caso contrário. Se o resultado é O, ! defined (NULL
produz 1 e NULL está definido. Caso contrário, a diretiva #define é pulada. Cada instrução #if termina com

endif. As diretivas #ifdef (nome) e #ifndef (nome) são abreviações de #ifdefined (nome) e #if ! defined (nome) . Uma instrução para o pré-processador de múltiplas partes pode ser testada usando-se as diretivas #elif (o equivalente de else if em uma estrutura if) e #else (o equivalente de else em uma estrutura if).
Durante o desenvolvimento do programa, os programadores freqüentemente acham útil se eliminar partes significativas do código, transformando-o em comentários e evitando desta maneira que este código seja compilado. Se o código contém comentários no estilo de C, / e / não podem ser usados para realizar esta tarefa. Em vez disso, o programador pode usar a seguinte instrução para o pré-processador:
#if O
código que não será compilado
#endif
Para habilitar o código para compilação, simplesmente substitua o valor O na instrução precedente pelo valor 1.
A compilação condicional freqüentemente é usada como um auxflio para a depuração do programa. Comandos de saída são freqüentemente usados para imprimir os valores de variáveis e para confirmar o fluxo de controle. Esses comandos de saída podem ser incluídos em diretivas condicionais do pré-processador, de maneira que os comandos são compilados somente até terminar o processo de depuração. Por exemplo,
#ifdef DEBUG
cerr « Variável x = “ « x « endi;
#endif
faz com que o comando cerr seja compilado no programa se a constante simbólica DEBUG foi definida (#defi-
ne DEBUG) antes da diretiva #ifdef DEBUG. Quando a depuração está completa, a diretiva #define é removi da do arquivo-fonte e os comandos de saída inseridos para fins de depuração são ignorados durante a compilação.
Em programas maiores, pode ser desejável definir várias constantes simbólicas diferentes que controlam a compila çã condicional em seções separadas do arquivo-fonte.

Erro comum de programação 17.4
Inserir comandos de saída compilados condicionalmente, para fins de depura ção, em lugares onde C++ na verdade espera um comando simples, pode levar a erros de sintaxe e lógica. Nesse caso, o comando compilado condicionalmente deve ser incluído em um comando composto. Assim, quando programa é compilado com comandos para depura ção, o fluxo de controle do programa não é alterado.
17.6 As diretivas #error e #pragma do pré-processador
A diretiva #error
#error unidades léxicas
imprime uma mensagem dependente de implementação que inclui as “unidades léxicas” especificadas na diretiva.
As “unidades léxicas” são seqüências de caracteres separadas por espaços. Por exemplo,
#error 1 - Erro de subscrito fora do intervalo válido
contém nove “unidades léxicas”. Em um compilador C++ popular, por exemplo, quando uma diretiva #error é processada, as “unidades léxicas” na diretiva são exibidas como uma mensagem de erro, o pré-processamento para o programa não é compilado.
A diretiva #pragma
#pragma unidades léxicas

I
CAPÍTULO 17 - O PRÉ-PROCESSADOR 861
f provoca uma ação definida pela implementação. Um pragma não-reconhecido pela implementação é ignorado. Por LS exemplo, um compilador C++ particular pode reconhecer pra gmas que possibilitam ao programador tirar pleno a proveito dos recursos daquele compilador específico. Para mais informações sobre #error e #pragma, verifique
a documentação de sua implementação de C++÷.
17.7 Os operadores # e ##
Os operadores # e ## do pré-processador estão disponíveis em C++ e em ANSI C. O operador # faz com que uma “unidade léxica” de um texto de substituição seja convertida em um string entre aspas. Considere a seguinte definição de macro:
#define ALO(x) cout « “Alô, #x « endl;
Quando ALÔ (Pedro) aparece em um arquivo de programa, ele é expandido para
cout « “Alô, “ “Pedro” « endi;
O string “Pedro” substitui #x no texto de substituição. Strings separados por espaços em branco são concatenados durante o pré-processamento, de forma que o comando acima seria equivalente a
cout « “Alô, Pedro” « endi;
Note que o operador # deve ser usado em uma macro com argumentos, porque o operando de * se refere a um argumento da macro.
O operador ## concatena duas “unidades léxicas”. Considere a seguinte

definição de macro:
#define TOKENCONCAT(x, y) x ## y
Quando TOKENCONCAT aparece em um programa, seus argumentos são concatenados e usados para substituir a macro. Por exemplo, TOKENCONCAT (0, K é substituído por 0K no programa. O operador *# deve ter dois operandos.
17.8 Números de linhas
A diretiva para opré -processador *line faz com que as linhas subseqüentes do código-fonte sejam renumeradas, começando com o valor inteiro constante especificado. A diretiva
#line 100
começa a numerar as linhas a partir de 100, iniciando com a próxima linha do código-fonte. Um nome de arquivo pode ser incluído na diretiva *line. A diretiva : #line 100
indica que as linhas são numeradas a partir de 100, começando com a próxima linha de código-fonte, e que o nome do arquivo para fins de qualquer mensagem do compilador é “file 1 . cpp”. A diretiva normalmente é usada para ajudar a tornar as mensagens produzidas por erros de sintaxe e advertências do compilador mais significativas. Os números de linhas não aparecem no arquivo-fonte.

17.9 Constantes simbólicas predefinidas
Existem quatro constantes simbólicas predefinidas (Fig. 17.1). Os identificadores para cada uma das constantes simbólicas predefinidas começam e terminam com dois caracteres sublinhado (_). Estes identificadores e o identificador defined (usado na Seção 17.5) não podem ser usados nas diretivas #define e #undef.
17.10 Asserções
A macro assert - definida no arquivo de cabeçalho <cassert> - testa o valor de uma expressão. Se o valor da expressão é O (falso), então assert imprime uma mensagem de erro e chama a função abort (da biblioteca de utilitários gerais - <cstdlib>) para terminar a execução do programa. Esta é uma ferramenta útil para depuração, para testar se uma variável tem um valor correto. Por exemplo, suponha que a variável x nunca deva ser maior que 10 em um programa. Pode ser usada uma asserção para testar o valor de x e imprimir uma mensagem de erro se o valor de x estiver incorreto. O comando seria:
assert(x <=10);
Se x for maior que 10 quando o comando precedente é encontrado em um programa, é impressa uma mensagem de erro contendo o número da linha e o nome do arquivo, e o programa termina. O programador pode, então, se concentrar nesta área do código para encontrar o erro. Se a constante simbólica NDEBUG está definida, as asserções subseqüentes serão ignoradas. Assim, quando as asserções não são mais necessárias (i.e, quando a depuração está completa), a linha
#defirie NDEBUG
é inserida no programa, em vez de deletar manualmente cada asserção.

Fig. 17.1 As constantes simbólicas predefinidas.
Muitos compiladores C++ agora incluem o tratamento de exceções. Os programadores de C++ preferem usar exceções em lugar de asserções. Porém, as asserções ainda têm seu valor para os programadores de C++ que trabalham com código legado em C.
Resumo
• Todas as diretivas para o pré-processador começam com e são processadas antes de o programa ser compilado.
• Somente caracteres em branco podem aparecer em uma linha antes de uma diretiva para o pré-processador.
• A diretiva #include inclui uma cópia do arquivo especificado. Se o nome do arquivo está entre aspas, o pré-processador começa a busca do arquivo a ser incluído no mesmo diretório que o arquivo que está sendo compilado. Se o nome do arquivo está entre menor (<) e maior (>), a busca é feita segundo uma forma definida pela implementação.
• A diretiva #define para o pré-processador é usada para criar constantes simbólicas e macros.

Constante simbólica	Descrição
	O número de linha da linha de código-fonte atual (uma constante inteira).
	O nome presumido de um arquivo-fonte (um string).
	A data em que o arquivo fonte é compilado (um string da forma “Mmm dd yyyy” tal como “Jan 19 2001”).
	A hora em que o arquivo fonte é compilado (um string literal da forma “hh:mm:ss”).

• Uma constante simbólica é um nome para uma constante.
• Uma macro é uma operação definida em uma diretiva #define. Macros podem ser definidas com ou sem argumentos.
• O texto de substituição para uma macro ou constante simbólica é qualquer texto no restante da linha, após o identificador da diretiva #define. Se o texto de substituição para uma macro ou constante simbólica for muito longo para caber de forma clara em uma linha. uma barra invertida (\) é colocada no fim da linha, indicando que o texto de substituição continua na linha seguinte.
• Constantes simbólicas e macros podem ser descartadas usando-se a diretiva #undef. A diretiva #undef anula a definição do nome da constante simbólica ou macro.
• O escopo de uma constante simbólica ou macro vai de sua definição até que seja anulada por #undef ou até o final do arquivo.
• A compilação condicional possibilita ao programador controlar a execução das

diretivas para o pré-processador e a compilação do código do programa.
• As diretivas condiconais para o pré-processador avaliam expressões inteiras constantes. Expressões de coerção, expressões com sizeof e constantes de enumeração não podem ser avaliadas em diretivas para o pré-processador.
• Cada instrução #if termina com uma instrução #endif.
• As diretivas #ifdef e #ifndef são fornecidas como abreviações para #if defined (nome) e #if ! defined (nome).
• Uma instrução condicional de múltiplas partes para o pré-processador pode ser testada usando-se as diretivas #elif e #else.
• A diretiva #error imprime uma mensagem dependente da implementação que inclui as “unidades léxicas” especificadas na diretiva e termina o pré-processamento e a compilação.
• A diretiva #pragma provoca uma ação dependente da implementação. Se opragma não é reconhecido pela implementação, o pra gmo é ignorado.
• O operador # faz com que uma “unidade léxica” do texto de substituição seja convertida em um string entre aspas. O operador # deve ser usado com uma macro com argumentos, porque o operando de # deve ser um argumento da macro.
• O operador ## concatena duas “unidades léxicas”. O operador ## deve ter dois operandos.
• A diretiva #line faz com que as linhas de código-fonte subseqüentes sejam renumeradas, começando com o valor constante inteiro especificado.
• Existem quatro constantes simbólicas pre-definidas. A constante LINE_ é o número da linha atual do código-fonte (um inteiro). A constante FILE é o nome presumido de um arquivo (um string). A constante _DATE_ é a data em que o arquivo-fonte é compilado (um string). A constante TIME é a hora em que o arquivo fonte é compilado (um string). Note que cada uma das constantes predefinidas começam e terminam com dois caracteres sublinhado (_).
• A macro assert - definida no arquivo de cabeçalho <cassert> - testa o valor de uma expressão. Se o valor da expressão é O (falso), então assert imprime uma mensagem de erro e chama a função abort para terminar a execução do programa.
Terminologia
operador de concatenação do pré-processador DATE
#define FILE
#elif LIME
#else TIME
#endif <cstdio>
#error <cstdlib>
#if abort
#ifdef argumento
#ifndef arquivo de cabeçalho
#include “filenarne” arquivos de cabeçalho da biblioteca padrão
#include <filename> assert
#line caractere de continuação barra invertida (\)
#pragina cassert

#undef compilação condicional

constante simbólica expandir uma macro
constantes simbólicas predefinidas macro
depurador macro com argumentos
diretiva para o pré-processador operador #
diretivas pré-processador
escopo de uma constate simbólica ou de uma macro pré-processador de conversão para string
execução condicional do pré-processador texto de substituição
17.1 Colocar um ponto-e-vírgula no fim de uma diretiva para o pré-processador pode levar a diversos erros, dcpcndcndu ua diretiva para o pré-processador.
17.2 Usar constantes simbólicas em um arquivo diferente daquele no qual as constantes são definidas é um erro de sintaxe.
17.3 Esquecer de colocar entre parênteses argumentos de uma macro, no texto de substituição.
17.4 Inserir comandos de saída compilados condicionalmente, para fins de depuração, em lugares onde C++ na verdade espera um comando simples, pode levar a erros de sintaxe e lógica. Nesse caso, o comando compilado condicionalmente deve ser incluído em um comando composto. Assim, quando programa é compilado com comandos para depuração, o fluxo de controle do programa não é alterado.
Boa prática de programação
17.1 Usar nomes que têm uni signiticado para constantes simbólicas ajuda a tornar os programas mais autodocuinentados.
Dica de desempenho
17.1 Às vezes, as macros podem ser usadas para substituir uma chamada de função por código mime, antes da execução do programa. Isso elimina o overhead de uma chamada de função. Funções mime são preferíveis a macros, porque oferecem os serviços de verificação de tipos das funções.
Observação de engenharla de software
17.1 Muitos dos recursos do pré-processador (especialmente macros) são mais apropriados para programadores de C do que para programadores de C++. Os programadores de C++ devem se familiarizar com o pré-processador porque eles podem necessitar trabalhar com código legado em C.
Exercícios de auto-revisão
17.1 Preencha os espaços em branco em cada um dos seguintes itens:
a) Cada diretiva para o pré-processador deve começar com _____
b) A instrução de compilação condicional pode ser estendida para se testar múltiplos casos usando-se a diretiva _____ e a diretiva _____
c) A diretiva _____ cria macros e constantes simbólicas.

d) Em uma linha, somente caracteres podem aparecer antes de uma diretiva para o pré-processador.
e) A diretiva _____ descarta nomes de constantes simbólicas e de macros.
f) A diretiva e a diretiva _____ são fornecidas como abreviações para #if defined (nome) e #if ! defined (nome).
g) possibilita ao programador controlar as diretivas para o pré-processador e a compilação do código do programa.
h) A macro imprime uma mensagem e termina a execução do programa se o valor da expressão da macro é avaliado como O.
i) A diretiva _____ insere um arquivo em outro arquivo.
j) O operador concatena seus dois argumentos.
k) O operador converte seu operando em um string.

Erros comuns de programação

CAPÍTULO 17 - O PRÉ-PROCESSADOR 865
1) O caractere _____ indica que o texto de substituição para uma constante simbólica ou uma macro continua na próxima linha.
m) A diretiva _____ faz com que as linhas de código-fonte sejam numeradas a partir do valor indicado, começando na linha seguinte do código-fonte.
17.2 Escreva um programa que imprime os valores das constantes simbólicas predefinidas listadas na Fig. 17.1.
17.3 Escreva uma diretiva para o pré-processador para realizar cada um dos seguintes itens:
a) Defina a constante simbólica YES como tendo o valor 1.
b) Defina a constante simbólica NO como tendo o valor 0.
c) Inclua o arquivo de cabeçalho cornmom h. O de cabeçalho se encontra no mesmo diretório do arquivo que está sendo compilado.
d) Renumere as linhas remanescentes no arquivo começando com o número de linha 3000.
e) Se a constante simbólica TRUE está definida, “anule-a” e redefina-a como 1. Não use #ifdef.
O Se a constante simbólica TRUE está definida, “anule-a” e redefina-a como 1. Use a diretiva do pré-processador #ifdef.
g) Se a constante simbólica ACTIVE não é igual a 0, defina a constante simbólica INACTIVE como 0. Caso contrário, defina INACTIVE como 1.
h) Defina a macro CUBE VOLUME que calcula o volume de um cubo (recebe um argumento).
Respostas aos exercícios de auto-revisão
17.1 a) #. b) #elif, #else. c) #define. d) espaço em branco, e) #undef. #ifdef, #ifndef. g) Compilação condicional. h) assert. i) #include.j) ##. k) #. 1) \. m) #line.
17.2 (Ver abaixo)
1 #include <iostreajn>
2 using std::cout;

```
3 usirig std::endl;
4 int main()
5 {
6 cout « “LINE = “ « LINE « endi;
7 cout « “FILE = “ « FILE « endl;
8 cout « “DATE = ‘ « _DATE_ « endi;
9 cout « “TIME = “ « TIME « endi;
10 return 0;
11
LINE =6
ZFILEZ = C:\exl7_02.cpp
_DATE_ = Apr 28 2000
TIME = 13:48:58
17.3 a) #define YES 1
b) #define NO O
e) #include “common.h”
d) #line 3000
e) #if defmned(TRUE)
#undef TRUE
#define TRUE 1
#endif
1) #ifdef TRUE
#undef TRUE
#define TRUE 1
#endif
```



```
g) #if ACTIVE
#define INACTIVE O
#else
#define INACTIVE 1
#endif
h) #define CUBE VOLUME( x ) ( ( x ) * ( x ) * ( x
```

Exercícios

17.4 Escreva um programa que define uma macro com um argumento para calcular o volume de uma esfera. O programa deve
calculas o volume para esferas de raios de 1 a 10 e imprimir os resultados em formato de tabela. A fórmula do volume da esfera é:

```
( 4.0 / 3 ) * it * r3
```

onde it é 3.14159.

17.5 Escreva um programa que produza a seguinte saída:

```
1 Asomadexeyél3
```

O programa deve definir a macro SUM com dois argumentos, x e y, e usar SUM para produzir a saída.

17.6 Escreva um programa que usa a macro MINIMO2 para determinar o menor de dois valores numéricos. Fomeça os valores pelo teclado.

17.7 Escreva um programa que usa a macro MINIMO3 para determinar o menor de três valores numéricos. A macro MINIMO3 deve usar a macro MINIMO2 definida no Exercício 17.6 para determinar o menor número. Forneça os valores pelo teclado.
17.8 Escreva um programa que usa a macro IMPRIMIR para imprimir o valor de um string.
17.9 Escreva um programa que usa a macro IMPRIMIRARRAY para imprimir um array de inteiros. A macro deve receber o array e o número de elementos no array como argumentos.
17.10 Escreva um programa que usa a macro SOM ARARRAY para somar os valores em um array numérico. A macro deve receber o array e o número de elementos no array como argumentos.
17.11 Rescreva as soluções para 17.4 a 17.10 como funções mune.
17.12 Para cada uma das seguintes macros, identifique os possíveis problemas (caso existam) quando o pré-processador expan dir as macros:
a) #define SQR(x) x * x
b) #define SQR(x) (x * x
e) #define SQR(x) (x) * (x)
d) #define SQR(x) ((x) * (x))

18
Tópicos sobre código legado em C

.

Objetivos

Ser capaz de redirecionar a entrada via teclado para vir de um arquivo e redirecionar a saída no vídeo para um arquivo. Ser capaz de escrever funções que usam listas de argumentos de comprimento variável.
Ser capaz de processar argumentos da linha de comando. Ser capaz de processar eventos inesperados, dentro de um programa.
Ser capaz de alocar memória dinamicamente para arrays usando o estilo C de alocação dinâmica de memória. Ser capaz de redimensionar a memória alocada dinamicamente usando o estilo C de alocação dinâmica de memória.
Usaremos um sinal que testei e considerei de longo alcance, e fácil de gritar Uaa-huu!
Zane Grey
Use it up, wear it out;
rnake it do, or do without.
Anônimo
It is quite a three-pipe problem*.
Sir Arthur Conan Doyle
Mas ainda assim urna união na divisão.
William Shakespeare
Jamais consegui entender o que aqueles pontos miseráveis queriam dizer
Winston Churchili
* N. de R. T.: Tradução literal: “É um problema que dá para resolver em três caximbadas”, aludindo aos pires que são abordados no texw.

Visão Geral
18.1 Introdução
18.2 Redirecionando entradalsaída nos sistemas UNIX e DOS
18.3 Lista de argumentos com tamanho variável
18.4 Usando argumentos na linha de comando
18.5 Notas sobre compilação de programas de múltiplos arquivos-fonte
18.6 Terminando um programa com exit e atexit
18.7 O qualificador de tipo volatile
18.8 Sufixos para constantes inteiras e de ponto flutuante

18.9 Tratamento de sinais
18.10 Alocação dinâmica de memória com calloc e realloc
18.11 Desvio incondicional: goto
18.12 Uniões
18.13 Especificações de ligação
Sumário• Terminologia • Erros comuns de programação Dicas de desempenho Dicas de portabilidade. Observações de engenharia de sofrtvare Exercícios de auto-revisão • Respostas dos exercícios de auto- revisão • Exercícios
18.1 Introdução
Este capítulo apresenta vários tópicos avançados, não abordados em cursos introdutórios. Muitos dos recursos aqui discutidos são específicos para sistemas operacionais particulares, especialmente UNIX e/ou DOS. Muito deste material serve para beneficiar os programadores C++ que necessitam trabalhar com código mais antigo, legado de C.
18.2 Redirecionando entrada/saída nos sistemas UNIX e Dos
Normalmente, a entrada de dados para um programa é feita através do teclado (entrada padrão) e a saída de dados de um programa é exibida no vídeo (saída padrão). Na maioria dos sistemas operacionais de computadores - em particular nos sistemas UNIX e DOS - é possível se redirecionar a entrada de dados para que sejam lidos de arquivos e redirecionar as saídas para que sejam armazenadas em arquivos. Ambas as formas de redirecionamento podem ser completadas sem o uso dos recursos de processamento das bibliotecas padrão. Existem várias maneiras de se redirecionar a entrada e a saída a partir da linha de comando do UNIX. Considere o arquivo executável sum que lê números inteiros, um por vez, e acumula o total dos valores lidos até que um indicador de final de arquivo seja encontrado e qle, então, imprime o resultado. Normalmente, o usuário digita números inteiros no teclado, bem como o indicador de final de arquivo, sendo este uma combinação de teclas que indica que não há mais valores para serem lidos. Com o redirecionamento de entrada, estes dados podem ser armazenados em um arquivo. Por exemplo, se os dados são armazenados no arquivo input, a linha de comando
$ sum < input
faz com que o programa sum seja executado; o símbolo de redirecionamento de entrada (<) indica que os dados do arquivo input (em vez do teclado) devem ser usados como os dados de entrada para o programa. O redirecionamento da entrada no sistema DOS é executado de forma idêntica.
Note que $ é o prompt da linha de comando do UNIX (alguns sistemas UNIX usam como prompt o %).
Estudantes, muitas vezes, têm dificuldade de entender que redirecionamento é uma função do sistema operacional e não outra característica de C++.

O segundo método de redirecionamento de entrada é o uso de pipes. Um pipe (1) faz com que a saída de um programa seja redirecionada como entrada para outro programa. Suponha que o programa random tenha como saída uma série de números inteiros randômicos; a saída do programa randoin pode ser

“canalizada” diretamente para o programa sum usando a linha de comando UNIX
$ randoin 1 sum
Isto faz com que seja calculada a soma dos números inteiros produzidos por random. Ouso de pipes pode ser feito em UNIX e em DOS.
A saída de um programa pode ser redirecionada para um arquivo usando-se o símbolo de redirecionamento
(>)(o mesmo símbolo é utilizado em UNIX e DOS). Por exemplo, para redirecionar a saída do programa randorn para o arquivo out, use
$ random > out
Finalmente, a saída de um programa pode ser acrescentada no final de um arquivo que já exista utilizando-se o
símbolo de acrescentar (») (o mesmo símbolo é utilizado em UNIX e em DOS). Por exemplo, para acrescentar a
saída do programa random no final do arquivo out, criado na linha de comando acima, use a linha de comando
$ random » out
18.3 Lista de argumentos com tamanho variável
[Nota: este material foi incluído para beneficiar os programadores de C++ que têm que trabalhar com codigo legado escrito em C. Em C++, os programadores utilizam a sobrecarga de funções para conseguir muito do que os programadores de C conseguem com uma lista de argumentos de tamanho variávelj. E possível se criar funções que recebam uma quantidade não especificada de argumentos. Reticências (. . .) em um protótipo de função indicam que a função recebe um número variável de argumentos de qualquer tipo. Note que as reticências devem ser colocadas no final da lista de argumentos. ou seja. como o último nome de argumento da lista. Macros e definições do cabeçalho de argumentos variáveis <cstdarg> (Fig. 18.1) oferecem os recursos necessários para construir funções com listas de argumentos de tamanho variável.

Fig. 18.1 O tipo e as macros definidas no cabeçalho cstdarg.
A Fig. 18.2 demonstra a função average que recebe um número variável de argumentos. O primeiro argumento de average é sempre a quantidade de valores cuja média deve ser calculada.

Identificador	Descrição
va_list	Um tipo adequado para armazenar as informações necessárias para as macros va_arg e va_end. Para acessar os argumentos em uma lista de argumentos de tamanho variável, deve ser declarado um objeto do tipo va_list.
vastart	Uma macro que é chamada antes que se possa acessar os argumentos de uma lista de argumentos de tamanho variável. A macro inicializa o objeto declarado com va list para ser utilizado pelas

	macros va_arg e vaend.
vaarg	Uma macro que é expandida para uma expressão de valor e tipo do próximo argumento da lista de argumentos de tamanho variável. Cada invocação de va_arg modifica o objeto declarado com va_list de modo que passe a apontar para o próximo argumento na lista.
va_end	Uma macro que facilita o retomo normal de uma função cuja lista de argumentos de tamanho variável foi referenciada pela macro vastart.


```
1 // Fig. 18.2: figlBO2.cpp
2 II Usando listas de argumentos de tamanho variável
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7 using std::ios;
8
9 #include <iomanip>
10
11 using std::setw;
12 using std::setprecision;
13 using std::setiosflags;
14
15 #include <cstdarg>
16
17 double average( int, ...
18
19 int main()
20 {
21 doublew=37.5, x=22.5, y=l.7, z=10.2;
22
23 cout « setiosflags( ios::fixed 1 ios::showpoint
24 « setprecision( 1 ) « w = « w « “\nx = “ « x
25 « “\ny = « y « ‘\nz = “ « z « endl;
26 cout « setprecision( 3 ) « “\nA média de w e x é
27 « average( 2, w, x
28 « “\nA média de w, x e y é
29 « average( 3, w, x, y
30 « “\nA média de w, x, y e z é
31 « average( 4, w, x, y, z ) « endl;
32 return 0;
33
34
35 double average( int i,
36 {
37 double total = 0;
```

```
38 va_list ap;
39
40 va_start( ap, i );
41
42 for ( int j = 1; j <= i; ++
43 total += va_arg( ap, double );
44
45 va_end( ap );
46
47 return total / i;
48 )
w = 37.5
x = 22.5
y = 1.7
z = 10.2
Amédia de w ex é 30.000
Amédiadew, xeyé 20.567
A média de w, x, y e z é 17.975
```
Fig. 18.2 Usando listas de argumentos de tamanho variável.

A função average usa todas as definições e macros do cabeçalho <cstdarg>. O objeto ap, do tipo valist, é usado pelas macros va_start, va arg e va_end para processar a lista de argumentos de tamanho variável da função average. A função invoca va_start para inicializar o objeto ap que será utilizado em va_arg e va_end. A macro recebe dois argumentos - o objeto ap e o identificador do argumento mais à direita da lista de argumentos antes das reticências - i neste caso (va start usa i aqui para determinar onde inicia a lista de argumentos). Em seguida, a função average, repetidamente, adiciona argumentos da lista de argumentos de tamanho variável à variável total. O valor a ser adicionado em total é recuperado através da lista de argumentos chamando-se a macro va arg. A macro va arg recebe dois argumentos - o objeto ap e o tipo de valor esperado na lista de argumentos (double neste caso) - e retorna o valor do argumento. A função average chama a macro vaend com o objeto ap como argumento para facilitar o retomo normal de average para main. Finalmente, a média é calculada e retornada para main. Note que usamos apenas argumentos double para a parte da lista com número variável de argumentos. Na verdade, qualquer tipo de dados ou a mistura de tipos de dados podem ser usados, visto que o tipo exato é especificado cada vez que va arg é usado.

Erro comum de programação 18.1

Colocur as reticências no meio da lista de parâmetros da função. As reticências só podem ser colocadas nofim da lista de parâmetros.

18.4 Usando argumentos na linha de comando

Em muitos sistemas -. DOS e UNIX em particular - é possível se passar argumentos para main através da linha de comandos, incluindo-se os

parâmetros int argce char *argv[] na listade parâmetros de main. O parâmetro argc recebe o número de argumentos da linha de comando. O parâmetro argv é um array de strings onde foram armazenados os argumentos da linha de comando atual. O uso comum de argumentos da linha de comando inclui imprimir esses argumentos, a passagem de opções para o programa e a passagem de nomes de arquivos para o programa.
A Fig. 18.3 copia um arquivo para outro, um caractere por vez. O nome do arquivo que contém o programa executável chama-se copy. Uma linha de comando típica para o programa copy no sistema UNIX é
$ copy input output
Esta linha de comando indica que o arquivo input é copiado para o arquivo output. Quando o programa é executado, se argc não é 3 (copy é contado como um dos argumentos), o programa imprime uma mensagem de erro e termina. Caso contrário, o array argv contém os strings “copy”, “input” e “output”. O segundo e terceiro argumentos, na linha de comando, são utilizados como nomes de arquivos pelo programa. Os arquivos são abertos criando-se um objeto ifstream inFile e um objeto ofstream outFile. Se os dois arquivos são abertos com sucesso, os caracteres são lidos do arquivo input através da função membro get e gravados no arquivo output através da função membro put até que o indicador de final do arquivo input seja encontrado. Então, o programa termina. O resultado é a cópia exata do arquivo input. Note que não são todos os sistemas operacionais de computadores que suportam argumentos na linha de comando de maneira tão fácil como o UNIX e o DOS. Os sistemas Macintosh e VMS, por exemplo, requerem uma inicialização especial para processar argumentos da linha de comando. Veja o manual do seu sistema para maiores informações sobre o uso de argumentos na linha de comando.

```
1 II Fig. 18.3: figl8_03.cpp
2 // Usando argumentos de linha de comando
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7 using std::ios;
8
```

Fig. 18.3 Usando argumentos na linha de comando (parte 1 de 2).


```
9 #include <fstream>
10
11 using std::ifstream;
12 using std::ofstream;
13
14 int main( int argc, char *argv[]
15 {
16 if ( argc != 3
```

```
17 cout « “Uso: copiar entrada saida” « endi;
18 else {
19 ifstream inFile( argv[ 1 1, ios::in );
20
21 if ( !inFile ) {
22 cout « argv[ 1 ] « “ não pode ser aberto” « endi;
23 return -1;
24 )
25
26 ofstream outFile( argv[ 2 ], ios::out );
27
28 if ( !outFile ) {
29 cout « argv[ 2 ] « ‘ não pode ser aberto” « endl;
30 inFile.closeO;
31 return -2;
32
33
34 while ( ‘inFile.eof()
35 outFile.put( static_cast< char >( inFile.get() ) );
36
37 inFile.close()
38 outFile.close();
39 }
40
41 return 0;
42 }
```

Fig. 18.3 Usando argumentos na linha de comando (parte 2 de 2).

1

18.5 Notas sobre compilaçao de programas de multiplos arquivos-fonte

Como citado anteriormente neste texto, é possível se criar programas que consistam em multiplos arqulvos-tonte (ver Capítulo 6). Existem várias considerações quando se criam programas a partir de múltiplos arquivos. Por exempio, a definição de uma função deve estar contida em um único arquivo - não pode estar espalhada em dois ou mais arquivos.

No Capítulo 3, introduzimos os conceitos de classes de memória e escopo. Aprendemos que variáveis declaradas fora das definições de função são da classe de memória static, por default, e são referências às variáveis globais. Variáveis globais são acessíveis por qualquer função definida no mesmo arquivo, após a declaração da variável. Variáveis globais também são acessíveis por funções que estão em outros arquivos, porém, variáveis globais devem ser declaradas em cada arquivo que forem utilizadas. Por exemplo, se definirmos uma variável global inteira flag em um arquivo e nos referirmos a ela em um segundo arquivo, o segundo arquivo deve ter a declaração

```
extern int flag;
```

antes das variáveis usadas neste arquivo. Na declaração acima, o especificador

de classe de memória extern indica para o compilador que a variável flag é definida ou mais adiante, no mesmo arquivo, ou em outro arquivo. O compilador informa ao editor de ligação que uma referência não resolvida para a variável flag aparece no arquivo (o compilador não sabe onde flag foi definida, então deixa o editor de ligação tentar encontrar flag). Se

o editor de ligação não localizar a definição de flag, um erro de edição de ligação é reportado e o arquivo executável não é gerado. Se a definição global apropriada é encontrada, o editor de ligação resolve a referência indicando onde flag foi localizado.

Dica de desempenho 18.1

_____ Variáveis globais melhoram o desempenho, porque podem ser acessadas diretamente por qualquer função - o overhead de passagem de dados para funções é eliminado.

Observação de engenharia de software 18.1

_____ Variáveis globais devem ser evitadas, a menos que o desempenho da aplicação seja importante, porque elas violam o princípio do mínimo privilégio e tornam o software de difícil manutenção.

Da mesma forma que as declarações extern podem ser usadas para declarar variáveis globais para outros arquivos de programa, protótipos de função podem estender o escopo de uma função além do arquivo em que foi definida (o especificador extern não é requerido no protótipo de função). Isto se consegue incluindo-se o protótipo da função em cada arquivo em que a função for chamada e compilando os arquivos todos juntos (ver Seção 17.2). Protótipos de função indicam para o compilador que a função especificada foi definida ou mais adiante, no mesmo arquivo, ou em outro arquivo. O compilador não tenta resolver referências a tal função - esta é uma tarefa para o editor de ligação. Se o editor de ligação não localizar uma definição da função, é gerado um erro. Como um exemplo do uso de protótipos de função para estender o escopo de uma função, considere qualquer programa contendo a diretiva do pré-processador #include <cstring>. Esta diretiva inclui no arquivo protótipos de funções, tais como strcmp e strcat. Outras funções no arquivo podem usar strciup e strcat para completar as suas tarefas. As funções strcmp e strcat foram definidas para nós separadamente. Não precisamos saber onde elas foram definidas. Simplesmente reutilizamos o código em nossos programas. O editor de ligação resolve nossas referências para essas funções. Esse processo nos permite usar as funções da biblioteca padrão.

Observação de engenharia de software 18.2

_____ Criar programas em múltiplos arquivos-fonte facilita a reutilização e a boa engenharia de soflware. Funções podem ser comuns a muitas aplicações. Em tais casos, essas funções devem ser armazenadas em seu próprio arquivo-fonte e cada arquivo-fonte deve ter um arquivo de cabeçalho correspondente, contendo os protótipos das funções. Isto permite que os programadores de diferentes aplicações reutilizem o mesmo código, incluindo o arquivo de cabeçalho apropriado e compilando a sua aplicação com o arquivo-fonte

correspondente.
Dica de portabilidade 18.1
_____ Alguns sistemas não suportam nomes de variáveis globais ou nomes de funções com mais de 6 caracteres. Isto deve ser considerado quando escrevemos programas que serão portados para múltiplas plataformas.
É possível se restringir o escopo de uma variável global ou função ao arquivo onde é definida. O especificador da classe de armazenamento static, quando aplicado a uma variável global ou a uma função, evita que ela seja usada por qualquer função que não tenha sido definida no mesmo arquivo. Isto é chamado de ligação interna. Variáveis globais e funções que não são precedidas por static em suas definições têm ligação externa - elas podem ser acessadas em outros arquivos se estes arquivos contiverem declarações e/ou protótipos de funções apropriados.
A declaração da variável global
static double pi = 3.14159;
cria a variável pi do tipo double, inicializa-a com 3. 14159 e indica que pi somente é conhecida por funções no arquivo onde é definida.
Ouso do especificador static é comum em funções utilitárias que são chamadas apenas por funções em um arquivo particular. Se uma função não é requerida fora de um arquivo particular, o princípio de mínimo privilégio deve ser reforçado com o uso de static. Se uma função é definida antes de ser usada em um arquivo, static deve ser aplicado à definição da função. Caso contrário, static deve ser aplicado ao protótipo da função.

Quando construímos programas grandes, distribuídos em múltiplos arquivos-fonte, compilar o programa se torna tedioso se pequenas alterações devem ser feitas em um dos arquivos e é preciso recompilar todo o programa. Muitos sistemas oferecem utilitários para recompilar apenas o arquivo do programa modificado. No sistema UNIX,
o utilitário é chamado make. O utilitário make lê um arquivo chamado makefile, que contém instruções para compilar e ligar o programa. Sistemas como o Borland C++ e o Microsoft Visual C++ para PCs oferecem utilitários make e “projetos”. Para maiores informações sobre os utilitários make, veja o manual do seu sistema em particular.
18.6 Terminando um programa com exit e atexit
A biblioteca de utilitários genéricos (cstdlib) oferece métodos para terminar a execução de um programa, diferentes do retorno convencional da função main. A função exit força o programa a terminar como se tivesse sido executado normalmente. Esta função é utilizada para terminar um programa quando é detectado um erro na entrada ou se o arquivo a ser processado pelo programa não pode ser aberto. A função atexit registra uma função para ser chamada no programa chamador quando esse termina com sucesso - i.e., quando o programa termina encontrando o fim de main ou quando exit é chamada.
A função atexit recebe um ponteiro para uma função (i.e., o nome da função) como um argumento. Funções chamadas para o término de programas não têm

argumentos e não podem retornar um valor. Até 32 funções podem ser registradas para execução no término de um programa.
A função exit recebe um argumento. O argumento é normalmente a constante simbólica EXIT_SUCCESS ou EXIT FAILURE. Se exit é chamado com EXIT SUCCESS, o valor definido na implementação para término com sucesso é retornado para o ambiente no qual foi chamado o programa. Se exit é chamado com EXIT_FAILURE. o valor definido na implementação para término sem sucesso é retornado. Quando a função exit é chamada, todas as funções previamente registradas por atexit são chamadas, em ordem inversa à do seu registro, iodos os streams associados ao programa são liberados e fechados e o controle retorna ao ambiente hospedeiro. A Fig. 18.4 testa as funções exit e atexit. O programa está preparado para que o usuário determine de que forma o programa deve terminar, ou por exit ou encontrando o fim de main. Note que a função print é executada ao término do programa em cada caso.

```
1 // Fig. 18.4: figl8O4.cpp
2 II Usando as funções exit e atexit
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7 using std::cin;
8
9 #include <cstdlib>
10
11 void print( void );
12
13 int main()
14
15 atexit( print ); II registra a função print
16 cout « ‘Digite 1 para terminar o programa com a função exit
17 « “\nDigite 2 para terminar o programa normalmente\n”;
18
19 int answer;
20 cm » answer;
21
22 if ( answer 1
23 cout « ‘\nTerminando o programa com a função exit\n;
24 exit( EXITSUCCESS );
```

Fig. 18.4 Usando as funções exit e atexit (parte 1 de 2).


```
25 }
26
27 cout « ‘ \nTerminando o programa atingindo o fim de main”
28 « endi;
29
```

```
30 return 0;
31
32
33 void print( void
34
35 cout « “Executando a função print ao terminar o programa\n”
36 « “Programa terminado” « endi;
37 1
Digite 1 para terminar o programa com a função exit
Digite 2 para terminar o programa normalmente
:1
Terminando o programa com a função exit
Executando a função print ao terminar o programa
Programa terminado
Digite 1 para terminar o programa com a função exit
Digite 2 para terminar o programa normalmente
:2
Terminando o programa atingindo o fim de main
Executando a função print ao terminar o programa
Programa terminado
```

Fig. 18.4 Usando as funções exit e atexit (parte 2 de 2).

18.7 O qualificador de tipo volatile

O qualificador de tipo volatile é aplicado a uma definição de uma variável que pode vir a ser alterada de fora do programa (i.e., a variável não está completamente sob controle do programa). Assim, o compilador não pode executar otimizações (como acelerar a execução do programa ou reduzir o consumo de memória, por exemplo) que dependem de “saber que o comportamento de uma variável é influenciado somente pelas atividades do programa que o compilador pode observar”.

18.8 Sufixos para constantes inteiras e de ponto flutuante

C++ oferece sufixos inteiros e de ponto flutuante para especificar os tipos de constantes inteiras e de ponto flutuante. Os sufixos de inteiros são: u ou U para um inteiro unsigned, 1 ou L para um inteiro long e ul ou UL para um inteiro uns igned long. As seguintes constantes são do tipo unsigned. long e uns igned long, respectivamente:

```
174u
8358L
28373u1
```

Se uma constante inteira não é seguida por um sufixo, seu tipo é determinado pelo primeiro tipo capaz de armazenar um valor daquele tamanho (primeiro int, depois long int e, então, uns igned long int).

Os sufixos de ponto flutuante são: f ou F para float e 1 ou L para um long double. As constantes a
seguir são do tipo long double e float, respectivamente:

3. 14159L
1. 28f
Uma constante de ponto flutuante sem sufixo é do tipo double. Uma constante sem um sufixo apropriado resulta ou em um aviso ou um erro do compilador.
18.9 Tratamento de sinais
Um evento inesperado, ou sinal, pode terminar um programa prematuramente. Alguns eventos inesperados incluem interrupções (pressionar Ctrl-c em um sistema UNIX ou DOS), instruções ilegais, violações de segmentação, ordens de término vindas do sistema operacional e exceções de ponto flutuante (divisão por zero ou multiplicação de valores muito grandes em ponto flutuante). A biblioteca de tratamento de sinais oferece a função signal para capturar eventos inesperados. A função signal recebe dois argumentos - um número de sinal inteiro e um ponteiro de acesso à função de tratamento de sinal. Sinais podem ser gerados pela função raise. que recebe um número de sinal inteiro como um argumento. A Fig. 18.5 resume os sinais padrão definidos no arquivo de cabeçalho <csignal>. A Fig. 18.6 demonstra as funções signal e raise.

Fig. 18.5 Os sinais definidos no cabeçalho csignal.
A Fig. 18.6 captura um sinal interativo (SIGINT) com a função signal. O programa chama signal com SIGINT e um ponteiro para a função signal handler (lembre que o nome de uma função é um ponteiro para a função). Agora. quando ocorre um sinal do tipo SIGINT. a função signal handier é chamada. uma mensagem é impressa e o usuário recebe a opção de continuar a execução do programa normalmente. Se o usuário optar por continuar a execução, o acesso ao sinal é reinicializado chamando-se signal novamente (alguns sistemas exigem que o tratador de sinal seja reinicializado) e o controle retorna ao ponto do programa em que foi detectado o sinal. Neste programa, a função raise é utilizada para simular um sinal interativo. Um número randômico entre 1 e 50 é escolhido. Se o número é 25, então raise é chamada para gerar um sinal. Normalmente, sinais interativos são inicial izados fora do programa. Por exemplo, pressionar Ctrl-c durante a execução de um programa nos sistemas UNIX e DOS gera um sinal que termina a execução do programa. O tratamento de sinais pode ser usado para capturar o sinal interativo e evitar que o programa seja terminado.
1 II Fig. 18.6: figl8_06.cpp
2 // Utilizando tratamento de sinais
3 #include <iostreani>
4
Fig. 18.6 Utilizando tratamento de sinais (parte 1 de 3).

Sinal	Explicação
SIGABRT	Término anormal de um programa (como uma chamada para abort).
5 IGFPE	Uma operação aritmética errada, como divisão por zero uma operação resultando em estouro.

SIGILL	Detecção de uma instrução ilegal.
SIGINT	Recebimento de um sinal de atenção interativo.
SIGSEGV	Acesso à memória inválido.
SIGTERM	Requisição de término enviada a um programa.


```
5 using std: :cout;
6 using std::cin;
7 using std::endl;
8
9 #include <iomanip>
10
11 using std::setw;
12
13 #include <csignal>
14 #include <cstdlib>
15 #include <ctime>
16
17 void signal_handler( int );
18
19 int main()
20
21 signal( SIGINT, signalhandier );
22 srand( time( O ) );
23
24 for ( int i = 1; i < 101; i++
25 int x = 1 ÷ randO % 50;
26
27 if (x==25)
28 raise( SIGINT );
29
30 cout « setw( 4 ) « i;
31
32 if(i%10=0)
33 cout « endi;
34 }
35
36 return 0;
37 }
38
39 void signal_handler( int signalValue
40
41 cout « “\nSinal de interrupção (“ « signalValue
```

```
42 « “) recebido.\ri’
43 « Deseja continuar (1 = sim ou 2 = não)?
44
45 int response;
46 cm » response;
47
48 while ( response != 1 && response = 2 ) {
49 cout « “(1 = sim ou 2 = não)?
50 cm » response;
51
52
53 if ( response == 1
54 signal( SIGINT, signalhandier );
55 else
56 exit( EXITSUCCESS );
57 }
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
```
Fig. 18.6 Usando tratamento de sinais (parte 2 de 3).


```
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88
Sinal de interrupção (4) recebido.
Deseja continuar (1 = sim ou 2 = não)? 1
89 90
91 92 93 94 95 96 97 98 99 100
```
Fig. 18.6 Utilizando tratamento de sinais (parte 3 de 3).

1810 Alocação dinâmica de memória com calloc e realloc

No Capítulo 7, quando discutimos o estilo de alocação dinâmica de memória de C÷+ com new e delete. comparamos new e delete com as funções de C malloc e free. Os programadores de C++ devem usar new e delete e não malloc e free. Entretanto, a maioria dos programadores de C+’i- vai ter necessidade de ler uma grande quantidade de código legado escrito em C e, por isso, incluímos essa discussão adicional sobre o estilo C de alocação dinâmica de memória.

A biblioteca de utilitários genéricos (<cstdlib>) oferece duas outras funções para alocação dinâmica de memória - calloc e realloc. Estas funções podem ser utilizadas para criar e modificar arrays dinâmicos. Como mostrado no Capítulo 5, o ponteiro para um array pode ser subscrito como um array. Assim, um ponteiro para uma porção contígua de memória, criada por cal].oc, pode ser manipulado como um array. A função calloc aloca dinamicamente memória para um array e

inicializa automaticamente a memória com zeros, O protótipo de calloc é
void *cajloc (size_t ninemb, size_t size);
Ela recebe dois argumentos o número de elementos (nmexnb) co tamanho de cada elemento (size) - e inicializa os elementos do array com zero. A função retorna um ponteiro para a memória alocada, ou um ponteiro nuil (0) se a memória não foi alocada.
A função realloc altera o tamanho de um objeto alocado anteriormente por uma chamada a malloc, calloc ou realloc. O conteúdo original dos objetos não é modificado, desde que a memória alocada seja maior do que a quantidade alocada anteriormente. Caso contrário, o conteúdo se mantém inalterado até o tamanho do novo objeto. O protótipo para realloc é
void *realloc (void *ptr, size_t size);
A função realloc recebe dois argumentos - um ponteiro para o objeto original (ptr) e o novo tamanho do objeto (size). Se ptr é 0, realloc age da mesma forma que malloc. Se size é O e ptr é diferente de 0, a memória para o objeto é liberada. Por outro lado, se ptr é diferente de O e size é maior do que zero, realJ.oc tenta alocar um novo bloco de memória. Se o novo espaço não pode ser alocado, o objeto apontado por ptr não é alterado. A função realloc retorna ou um ponteiro para a memória realocada ou um ponteiro nulo.
18.11 Desvio incondicional: goto
Ao longo de todo o texto, temos salientado a importância do uso das técnicas de programação estruturada para se construir software consistente de fácil depuração, manutenção e modificação. Em alguns casos, o desempenho é mais importante do que uma restrita observância às técnicas de programação estruturada. Nestes casos, podem ser utilizadas algumas técnicas de programação não-estruturada. Por exemplo, podemos usar break para terminar a execução de uma estrutura de repetição antes que a condição do laço de continuação seja falsa. Isto evita repetições desnecessárias do laço se a tarefa foi completada antes do término do laço.
Outra instância da programação não-estruturada é o comando goto - um desvio incondicional. O resultado
do comando goto é o desvio do fluxo de controle do programa para o primeiro comando após o rótulo especificado

no comando goto. Um rótulo é um identificador seguido por dois-pontos (:). Um rótulo deve aparecer na mesma função que o comando goto que se refere a esse rótulo . A Fig. 18.7 usa o comando goto para repetir um laço dez vezes e imprime o valor do contador a cada iteração. Depois de inicializar count com 1, o programa testa se count é maior do que 10 (o rótulo start é saltado porque rótulos não executam nenhuma ação). Se for, o controle é transferido por goto para o primeiro comando depois do rótulo end. Caso contrário, count é impresso e incrementado e o controle é transferido por goto para o primeiro comando depois do rótulo start.
No Capítulo 2, estabelecemos que eram necessárias apenas três estruturas de

controle para se escrever qualquer programa - seqüência, seleção e repetição. Quando são seguidas as regras da programação estruturada, é possível se criar estruturas de controle profundamente aninhadas, das quais é difícil sair de uma maneira eficiente. Alguns programadores usam o comando goto em tais situações como uma saída rápida de uma estrutura profundamente aninhada. Isso elimina a necessidade de testar diversas condições para sair de uma estrutura de controle.

```
II Fig. 18.7: figl8_07.cpp
II Usando goto
#include <iostream>
using std::cout;
using std::endl;
int main()
int count = 1;
start: II rótulo
if ( count > 10
goto end;
cout « count «
++count;
goto start;
end: // rótulo
cout « endl;
return 0;
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```

23

2 3 4 5 6 7 8 9 10

Fig. 18.7 Usando goto.
Dica de desempenho 18.2
_____ O comando goto pode ser usado para sair de maneira eficiente de uma estrutura de controle profundamente aninhada.
Observação de engenharia de software 18.3
_____ O comando goto deve ser utilizado apenas em aplicações orientadas para desempenho. O comando goto é não-estruturado e pode levar a programas que são mais difíceis de se depurar manter e modi ficar

18.12 Uniões
Uma união (definida com a palavra-chave union) é uma região de memória que, a cada momento, pode conter objetos de tipos diferentes. Contudo, em um determinado momento, uma uniori pode conter no máximo um objeto, porque os membros de uma união compartilham o mesmo espaço de armazenamento. E de responsabilidade do programador assegurar que o dado em uma union seja referenciado com um nome de membro do tipo de dado apropriado.
Erro comum de programação 18.2
O resultado da referência a um membro de uma união diferente do último armazenado é indefinido. Ela trata os dados armazenados como um tipo diferente.
Dica de portabilidade 18.2
_____ Se dados são armazenados em uma união como um tipo e são referenciados como outro tipo, os resultados são dependentes da implementação.
Em momentos diferentes durante a execução de um programa, alguns objetos podem não ser relevantes, enquanto outro objeto é - assim, uma union compartilha o espaço em vez de desperdiçar memória com objetos que não estão sendo usados, O número de bytes usados para armazenar uma união deve ser pelo menos suficiente para conter o maior membro.
tDica de desempenho 18.3
• Usar union economiza memória.
Dica de portabilidade 18.3
_____ A quantidade de memória requerida para armazenar uma union é dependente da implementação.
Dica deportabilidade 18.4
_____ A/guinas union podem não ser fáceis de se portar para outros sistemas de computador Se uma união é portável ou não freqüentemente depende dos requisitos de alinhamento de memória para os tipos de dados
dos membros da união em um dado sistema.
Uma union é declarada com o mesmo formato de uma estrutura ou de uma classe. Por exemplo,

union Number {
int x;
double y;
indica que Number é um tipo uriion com os membros int x e double y. A definição de union normalmente precede main em um programa, de modo que a definição pode ser utilizada para declarar variáveis em todas as funções do programa.
Observação de engenharia de software 18.4
_____ Assim como uma declaração de uma struct ou de uma class, uma declaração de uma union simplesmente cria um novo tipo. Colocar uma declaração de union ou structfora de qualquer função
não cria uma variável global.
As únicas operações primitivas válidas que podem ser executadas sobre uma união são: atribuí-la a outra união do mesmo tipo, obter o endereço (&) de uma união e acessar membros da união utilizando o operador de membro de estrutura (.) e o operador de ponteiro de estrutura (->). Uniões não podem ser comparadas, pelo mesmo motivo que estruturas não podem ser comparadas.

Erro comum de programação 18.3
Comparar unions é um erro de sintaxe, porque o compilador não sabe qual membro de cada união está
ativo e, portanto, qual membro de uma deve ser comparado a qual membro da outra.
Uma union é similar a uma classe, no que diz respeito a poder ter um construtor para inicializar qualquer um dos seus membros. Uma union que não tenha um construtor pode ser inicializada com uma outra union do mesmo tipo, com uma expressão do tipo do primeiro membro da union ou com um inicializador (colocado entre chaves (}) do tipo do primeiro membro da union. Union podem ter outras funções membro, tais como destruidores, mas as funções membro de uma union não podem ser declaradas como virtual. Os membros de uma union são public por default.
Erro comum de programação 18.4
Inicializar uma union em uma declaração com um valor ou uma expressão cujo tipo é diferente do tipo
do primeiro membro da union.
Uma union não pode ser usada como uma classe básica em herança, i.e., classes não podem ser derivadas de unions. Unions podem ter objetos como membros se esses objetos não tiverem um construtor, um destruidor ou um operador de atribuição sobrecarregado. Nenhum dos membros de dados de uma union pode ser declarado como static.
O programa na Fig. 18.8 usa a variável value do tipo union nuniber para exibir o valor armazenado na union tanto como um int quanto como um double. A saída do programa é dependente da implementação. A saída do programa mostra que a representação interna de um valor double pode ser bem diferente da representação de um int.
1 II Fig. 18.8: figl8O8.cpp

```
2 1/ Um exemplo de urna união
3 #include <iostrearn>
4
5 using std::cout;
6 using std: :endl;
7
8 union Nuinber {
9 intx;
10 double y;
11 };
12
13 int main()
14
15 Nuniber value;
16
17 value.x = 100;
18 cout « “Coloca um valor no membro int\n”
19 « “e imprime os dois membros.\nint:
20 « value.x « “\ndouble: “ « value.y « “\n\n”;
21
22 value.y = 100.0;
23 cout « “Coloca um valor no membro double\n”
24 « “e imprime os dois mernbros.\nint:
25 « value.x « “\ndouble: “ « value.y « endl;
26 return 0;
27
```

Fig. 18.8 Imprimindo o valor de uma union com os tipos de dados dos dois membros (parte 1 de 2).


```
Coloca um valor no membro int
e imprime os dois membros.
int: 100
double: -9.25596e+06l
Coloca um valor no membro double
e imprime os dois membros.
int: O
double: 100
```

Fig. 18.8 Imprimindo o valor de uma union com os tipos de dados dos dois membros (parte 2 de 2).

Uma union anônima é uma união sem um nome de tipo, que não tenta definir objetos ou ponteiros antes de seu ponto-e-vírgula final. Uma union como esta não cria um tipo, mas cria um objeto sem nome. Os membros de uma union anônima podem ser acessados diretamente no escopo em que a union anônima foi declarada, como se fosse outra variável local - não há necessidade do uso dos operadores ponto (.) e seta (->).

Union anônimas têm algumas restrições. Uniões anônimas podem conter somente membros de dados. Todos
os membros de uma union anônima são public. E uma union anônima declarada globalmente (i.e., em escopo de arquivo) deve ser explicitamente declarada static. A Fig. 18.9 ilustra o uso de uma union anônima.

```
1 II Fig. 18.9: figl809.cpp
2 // Usando uma união anônima
3 #include <iostream>
4
5 using std: :cout;
6 using std: :endl;
7
8 int main()
9{
10 // Declara urna união anônima.
11 // Note que os membros b, d e fPtr compartilham o mesmo espaço.
12 union {
13 int b;
14 double d;
15 char *fptr;
16 };
17
18 // Declara variáveis locais convencionais
19 intal;
20 double c = 3.3;
21 char *eptr = União;
22
23 II Atribui um valor para cada membro da união
24 II sucessivamente e imprime cada um deles.
25 cout«a«
26 b=2;
27 cout « b « endl;
28
29 cout « c «
30 d=4.4;
31 cout « d « endl;
32
```

Fig. 18.9 Usando uma union anônima (parte 1 de 2).


```
33 cout « ePtr «
34 fPtr = anônima";
35 cout « fPtr « endl;
36
37 return O;
38 }
```

12
3.3 4.4
União anônima
Fig. 18.9 Usando uma union anônima (parte 2 de 2).
18.13 Especificações de ligação
É possível, a partir de um programa em C++, se chamar funções escritas e compiladas por um compilador C. Como afirmado na Seção 3.20, C++ codifica de maneira especial nomes de funções para garantir a edição de ligação segura quanto ao tipo. C por outro lado, não codifica esses nomes de funções. Assim, uma função compilada em C não pode ser reconhecida quando é feita uma tentativa de se ligar o código C com o código C++, porque o código C++ espera uma codificação especial do nome da função. C++ permite que o programador forneça especificações de ligação para informar ao compilador que aquela função foi compilada por um compilador C e preveni-lo de que o nome da função foi inicialmente codificado por um compilador C++. As especificações de ligação são de grande ajuda quando foram desenvolvidas grandes bibliotecas de funções especializadas e o usuário ou não tem acesso ao código- fonte para uma recompilação em C++ ou não tem tempo para converter a biblioteca de funções de C para C++.
Para se informar ao compilador que uma ou várias funções foram compiladas em C, escreva os protótipos de
funções da seguinte maneira:
extern ‘C” protótipo da função II uma única função
extern “C” II múltiplas funções
protótipos das funções
Essas declarações informam ao compilador que as funções especificadas não foram compiladas em C++, de modo que a codificação de nomes não deve ser executada sobre as funções listadas na especificação de edição. Essas funções podem então ser adequadamentre ligadas com o programa. Os ambientes de C ++ normalmente incluem as bibliotecas padrão de C e não exigem que o programador use especificações de ligação para essas funções.
Resumo
Em muitos sistemas - em particular nos sistemas UNIX e DOS - é possível se redirecionar entrada para um programa e saída de um programa. A entrada é redirecionada das linhas de comando do UNIX e do DOS usando o símbolo de redirecionamento de entrada (<) ou usando pipe (1). A saída é redirecionada a partir da linha de comando do UNIX e do DOS usando o símbolo de redirecionamento de saída (>) ou o símbolo de acrescentar à saída (»). O símbolo de redirecionamento de saída armazena a saída do programa em um arquivo e o símbolo de acrescentar à saída coloca a saída do programa no final de um arquivo.
• As macros e as definições do cabeçalho de argumentos variáveis cstdarg fornecem os recursos necessários para se criar
funções com listas de argumentos de tamanho variável.
• Reticências (. . .) em um protótipo de função indicam que a função recebe um número variável de argumentos.

• O tipo va list é usado para reter informações necessárias às macros va start, va arg e va end. Para se acessar os argumentos em uma lista de argumentos de tamanho variável, um objeto do tipo va list deve ser declarado.

S84 C++ COMO PROGRAMAR
A macro va start é invocada antes que os argumentos de uma lista de argumentos de tamanho variável possam ser acessados. A macro inicializa o objeto declarado com va list para ser usado pelas macros va arg e vaend.
• A macro va arg expande para uma expressão com o valor e o tipo do próximo argumento da lista de argumentos de tamanho variável. Cada chamada de va arg modifica o objeto declarado por va list, cujo objeto agora aponta para o próximo argumento da lista.
• A macro va end facilita o retomo normal da função cuja lista de argumentos de tamanho variável foi referenciada pela macro vastart.
• Em muitos sistemas - DOS e UNIX em particular - é possível se passar argumentos para main através da linha de comando, incluindo na lista de parâmetros de main os parâmetros int argc e char *argv (]. O parâmetro argc é o número de argumentos na linha de comando. O parâmetro argv é um array de strings contendo os argumentos da linha de comando.
• A definição de uma função deve estar contida em um arquivo - não pode estar espalhada em dois ou mais arquivos.
• Variáveis globais devem ser declaradas em cada arquivo em que são utilizadas.
• Protótipos de funções podem estender o escopo da função para fora do arquivo no qual elas foram definidas (o especificador extern não é requerido no protótipo da função). tsto é conseguido com a inclusão do protótipo da função em cada arquivo em que a função for chamada e a compilação em conjunto desses arquivos.
• O especificador de classe de armazenamento static, quando aplicado a uma variável global ou a uma função. evita o uso desta por outras funções que não foram definidas no mesmo arquivo. Isto é chamado de ligação interna. Variáveis globais e funções que não são precedidas por static em suas definições têm uma ligação externa - elas podem ser acessadas em outros arquivos se esses arquivos contiverem as declarações e/ou protótipos de funções apropriados.
• O especificador static é comumente usado em funções utilitárias que são chamadas por funções em um arquivo particular. Se a função não é requerida fora desse arquivo, o princípio de mínimo privilégio deve ser garantido pelo uso de static.
• Quando construímos programas grandes em múltiplos arquivos-fonte, a compilação pode vir a ser tediosa se pequenas alterações são feitas em um arquivo e se recompila todos os programas. Muitos sistemas fornecem utilitários especiais para recompilar apenas do programa alterado. No sistema UNIX, o utilitário é chamado de make. O utilitário make lê o arquivo chamado niakefile que contém instruções para compilar e ligar o programa.
• A função exit força um programa a terminar como se tivesse executado normalmente.

• A função atexit registra uma função do programa para ser chamada ao término normal do programa - i.e., ou quando um programa termina encontrando o final de mairi ou quando exit é chamada.
• A função atexit recebe um ponteiro para uma função (i.e., um nome de função) como um argumento. Funções chamadas para terminar um programa não podem ter argumentos e não retornam valores. Até 32 funções podem ser registradas para a execução no término de um programa.
• A função ezit recebe um argumento. O argumento normalmente é a constante simbólica EXIT SUCCESS ou a constante simbólica EXIT FAILURE. Se exit é chamada com EXIT SUCCESS, o valor definido na implementação para o término com sucesso é retornado para o ambiente chamador. Se exit é chamada com EXIT FAILURE, o valor definido na implementação para término sem sucesso é retornado.
• Quando exit é chamada, todas as funções registradas por atexit são chamadas, na ordem inversa do seu registro, todas as áreas associadas ao programa são liberadas e fechadas e o controle retorna ao ambiente principal.
• O qualificador volatile é usado para evitar otimizações de uma variável, porque ela pode ser modificada fora do escopo do programa.
• C++ oferece sufixos de inteiros e ponto flutuante para especificar constantes do tipo inteiro e ponto flutuante. Os sufixos de inteiros são: u ou U para um inteiro unsigned. 1 ou L para um inteiro long e uJ. ou uL para um inteiro unsigned long. Se uma constante inteira não tiver sufixo, o seu tipo é determinado pelo primeiro tipo capaz de armazenar o tamanho do seu valor (primeiro int, depois long int e por fim unsigned long int). Os sufixos de ponto flutuante são f ou F para float e 1 ou L para long double. Uma constante de ponto flutuante sem sufixo é do tipo double.
• A biblioteca de acesso a sinais provê a capacidade de registrar uma função para tratar eventos inesperados com a função signal. A função signal recebe dois argumentos - um número de sinal inteiro e um ponteiro de acesso à função signal.
• Sinais podem ser gerados com a função raise e um argumento inteiro.
• A biblioteca de utilitários genéricos cstdlib provê as funções calloc e realloc para alocação dinâmica de memória. Estas funções podem ser utilizadas para se criar arrays dinâmicos.

• A função calloc recebe dois argumentos - o número de elementos (nmemb) e o tamanho de cada elemento (size) - e inicializa os elementos do array com zeros. A função retoma ou um ponteiro para a memória alocada ou um ponteiro nulo se a memória não foi alocada.
• A função realloc altera o tamanho de um objeto alocado previamente por uma chamada a malloc, calloc ou realloc. O conteúdo original dos objetos não é modificado desde que o tamanho da memória seja maior do que a que foi alocada anteriormente.
• A função realloc recebe dois argumentos - um ponteiro para o objeto original (ptr) e o novo tamanho do objeto (size). Se ptr é NULL, realloc age exatamente

como malloc. Se size é O e o ponteiro recebido não é NULL, a memória para o objeto é liberada. Por outro lado, se ptr não é NULL e size é maior do que zero, realloc tenta alocar o novo bloco de memória para o objeto. Se o novo espaço não pode ser alocado o ponteiro ptr do objeto não é alterado. A função realloc pode retornar ou um ponteiro para a memória realocada ou um ponteiro NULL.
• O resultado do comando goto é a alteração do fluxo de controle de um programa. A execução do programa continua a partir do primeiro comando depois do rótulo do comando goto.
• Um rótulo é um identificador seguido de dois pontos (:). Um rótulo deve aparecer na mesma função que o comando goto que se refere a ele.
• Uma uriion é um tipo de dado derivado, cujos membros compartilham o mesmo espaço na memória. Os membros podem ser de qualquer tipo. A área reservada para uma union é tal que possa conter o seu maior membro. Em muitos casos, union podem conter dois ou mais tipos de dados. Somente um membro e, conseqüentemente, um tipo de dado pode ser referenciado a cada momento.
• Uma union é declarada da mesma forma que uma estrutura.
• Uma union pode ser inicializada somente com um valor do tipo do seu primeiro membro.
• C++ habilita o programador a prover especificações de ligação para informar ao compilador que certas funções foram compiladas por um compilador C e para evitar que os nomes dessas funções sejam codificados pelo compilador C++.
• Para inÍhrmar para o compilador que uma ou mais funções foram compiladas pelo compilador C, escreva os protótipos de função como segue:
extern “C” protótipo da função // um única função
extern “C° II múltiplas funções
protótipos das funções
• Essas declarações informam ao compilador que as funções especificadas não são compiladas em C++, de modo que a codificação de nomes não deve ser feita para as funções listadas na especificação de ligação. Essas funções podem então ser ligadas corretamente com o programa.
• Os ambientes de C++ normalmente incluem bibliotecas padrão C e não requerem que o programador use especificações de ligação para utilizá-las.
Terminologia
argumentos de linha de comando especificador de classe armazenamento extern
argv evento
arrays dinâmicos exceção de ponto flutuante
atexit exit
biblioteca de tratamento de sinais EXITFAILURE
calloc EXITSUCCESS
capturar extern “C”
comando goto instrução ilegal
cons t interrupção
csigna]. ligação externa
cstdarg ligação interna
especificador da classe armazenamento static lista de argumentos de tamanho

variável

CAPÍTULO 18 - Tópicos SOBRE CÓDIGO LEGADO EM C 887
Exercícios de auto-revisão
18.1 Preencha os espaços em branco em cada um dos seguintes:
a) O símbolo _____ redireciona dados de entrada do teclado para obtê-los de um arquivo.
b) O símbolo _____ é usado para redirecionar a saída para a tela a ser colocada em um arquivo.
c) O símbolo_____ é usado para acrescentar a saida de um programa ao fim de um arquivo.
d) Um _____ é usado para dirigir a saída de um programa como a entrada para outro programa.
e) na lista de argumentos de uma função indicam que a função recebe um número variável de argumentos.
f) A macro _____ deve ser invocada antes que os parâmetros em uma lista variável de argumentos possam ser acessados.
g) A macro _____ é usada para acessar os parâmetros individuais de uma lista variável de parâmetros.
h) A macro _____ facilita um retorno normal de uma função cuja lista variável de parâmetros foi referida pela macro vastart.
i) O parâmetro de main recebe o número de parâmetros em uma linha de comando.
j) O parâmetro de main armazena parâmetros de linha de comando como strings de caracteres.
k) O utilitário do UNIX _____ lê um arquivo chamado que contém instruções para compilar e ligar
um programa que consiste em múltiplos arquivos-fonte. O utilitário somente recompila um arquivo se o mesmo foi
modificado desde que foi compilado pela última vez.
1) A função força um programa a terminar a execução.
m) A função registra uma função para ser chamada no término normal do programa.
n) O qualificador de tipo indica que um objeto não deve ser modificado depois de é inicializado.
o) Um _____ de inteiro ou ponto flutuante pode ser anexado a uma constante inteira ou de ponto flutuante para especificar o tipo exato da constante.
p) A função pode ser usada para registrar uma função para capturar eventos inesperados.
q) A função gera um sinal de dentro de um programa.
r) A função aloca memória dinamicamente para um array e inicializa os elementos com zeros.
s) A função muda o tamanho de um bloco de memória alocada dinamicamente.
t) Uma _____ é uma classe contendo uma coleção de variáveis que ocupam a mesma memória, mas em tempos

diferentes.
u) A palavra-chave é usada para introduzir uma definição de uma união.
Respostas aos exercícios de auto revisão
18.1 a) redireciona a entrada (<). b) redireciona a saída (>). c) anexa à saída (>>). d) pipe(1). e) reticências (. . .). f) va_start.g)vaarg. h)va end. i)argc. j)argv. k)make, makefile. l)exit. m)atexit. n)const. o)
sufixo, p) signal. q) raise. r) calloc. s) realloc. t)união. u) union.
Exercícios
18.2 Escreva um programa que calcula o produto de uma série de inteiros que são passados para a função product usando uma lista de argumentos de tamanho variável. Teste sua função com várias chamadas, cada uma com um número diferente de argumentos.
18.3 Escreva um programa que imprime os argumentos da linha de comando do programa.
18.4 Escreva um programa que classifica um array de inteiros em ordem ascendente ou descendente. O programa deve usar argumentos da linha de comando para passar o parâmetro -a para ordem ascendente ou o parâmetro -d para ordem descendente. (Nota: este é o formato padrão para passar opções a um programa em UNIX.)
18.5 Leia os manuais do seu sistema para determinar que sinais são suportados pela biblioteca de tratamento de sinais (csignal). Escreva um programa com tratadores de sinal para os sinais SIGABRT e SIGINT. O programa deve testar a captura destes sinais chamando a função abort para gerar um sinal do tipo SIG.ABRT e pela digitação de Ctrl-c para gerar um sinal do tipo SIGINT.
18.6 Escreva um programa que aloca dinamicamente um array de inteiros, O tamanho do array deve ser fornecido pelo teclado. Os elementos do array devem ser valores atribuídos a partir da entrada pelo teclado. Imprima os valores do array. Em seguida,

realoque a memória do array para metade do número atual de elementos. Imprima os valores restantes no array para confirmar se eles correspondem aos valores da primeira metade do array original.
18.7 Escreva que um programa que recebe dois nomes de arquivos como argumentos da linha de comando, lê os caracteres do primeiro arquivo um de cada vez e escreve os caracteres em ordem contrária no segundo arquivo.
18.8 Escreva um programa que usa comandos goto para simular uma estrutura de laços aninhados que imprime um quadrado de asteriscos, como segue:
**** *
* *
* *
* *
**** *
O programa deve usar só os três comandos de saída seguintes:
cout « *•
cout«

cout « endi;
18.9 Forneça a definição para union Data contendo char c, short s, long 1, f].oat f e double d.
18.10 Crie union Integer com membros char c, short s, int i e long 1. Escreva um programa que recebe como entrada valores dos tipos char, short. int e long e armazena os valores em variáveis união do tipo union Integer. Cada variável união deve ser impressa como um char. um short, um int e um long. Os valores são sempre impressos corretamente?
18.11 Crie union FloatingPoint com membros fJ.oat f, double de long doubJ.e 1. Escreva um programa que recebe como entrada valores dos tipo float, double e long double e armazena os valores em variáveis união do tipo union FloatingPoint. Cada variável união deve ser impressa como um float. um double e um long double. Os valores são sempre impressos corretamente?
18.12 Dada aunion
union A {
double y;
char *Z;
qual dos seguintes itens são comandos corretos para inicializar esta union?
a) A p = B; // B é do mesmo tipo que A
b)A q = x; //x é uns double
c)A r = 3.14159;
d)A $ = { 79.63 };
e) A t = { “Oi,você!’ };
fA u = { 3.14159, “Pi” };

ar se

19

A classe string e o processamento em stream de strings

Objetivos

• Usar a classe string da biblioteca padrão de C++ para que possamos tratar strings como objetos no pleno sentido
• Ser capaz de atribuir, concatenar, comparar, pesquisar e intercambiar (swap) strings
• Ser capaz de determinar as características de um string
• Ser capaz de encontrar, substituir e inserir caracteres em um string
• Ser capaz de converter strings ao estilo de strings de C
• Ser capaz de usar iteradores de strings
• Ser capaz de executar entrada e saída de strings na memória

A dftrença entre a palavra certa e a quase certa é realmente uma questão importante - é a diferença entre um vaga-lume e um relâmpago.
Mark Twain, Carta a George Bainton (15 de outubro de 1888)

Escrevi uma carta longa demais, pois me faltou tempo para escrevê-la mais curta.
Biaise Pascal

Silenciosa é a palavra.
Miguel de Cervantes, Don Quixote de la Mancha

Que a ação sirva à palavra, e a palavra, à ação; com essa observação particulai que você não transgrida a modéstia da natureza.
William Shakespeare, Hamlet

Visão Geral

19.1 Introdução
19.2 Atribuição e concatenação de strings
19.3 Comparando stríngs
19.4 Substrings
19.5 Intercambiando strings

19.6 Características de stríng
19.7 Encontrando caracteres em um string
19.8 Substituindo caracteres em um string
19.9 Inserindo caracteres em um string
19.10 Conversão para strings char* no estilo da linguagem C
19.11 Iteradores
19.12 Processamento de strings em streams
Resumo• Terminologia Erros comuns de programação Boa prática de programação • Dica de desempenho • Dica de teste e depura ção • Exercícios de auto-revisão Respostas aos exercícios de auto-revisão• Exercícios
19.1 Introdução
A classe gabarito basic string de C++ fornece operações típicas de manipulação de strings, tais como cópia, pesquisa, etc. A definição do gabarito e de todos os recursos de apoio estão no namespace std estas incluem o comando typedef
typedef basic_string< char > string;
que cria o tipo alias (nome alternativo) string para basicstrirzg< char >. Também é fornecido um typedef para o tipo wchar_t. O tipo wchar_t armazena caracteres (por exemplo, caracteres de 2 bytes, caracteres de 4 bytes, etc.) para suportar outros conjuntos de caracteres. Em todo este capítulo, usamos exclusivamente string. Para usar strings. inclua o arquivo de cabeçalho <string> da biblioteca padrão de C++. [Nota: o tipo wchart é comumente usado para representar Unicode, que tem caracteres de 16 bits, mas o tamanho da wchat_t não é fixado pelo padrão.j
Um objeto string pode ser inicializado com um argumento para o construtor tal como
string sl(“Bom dia!”); 1/ cria um string a partir de um const char*
que cria um string contendo os caracteres em “Bom dia! “ exceto, talvez, o caractere de terminação ‘\0 , ou com dois argumentos como em
string s2(8, ‘x’); 1/ string de 8 caracteres ‘x’
que cria um string contendo oito caracteres ‘x’ . A classe strng também fornece um construtor default e um construtor de cópia.
Um string também pode ser inicializado através da sintaxe alternativa de construção na definição de um string, como em
string mês = “março”; // o mesmo que: string mês(“março”);
Lembre-se de que o operador = na declaração precedente não é uma atribuição, mas sim uma chamada para o construtor de cópia da classe string, o qual faz a conversão implicitamente.

Note que a classe string não fornece conversões de int ou char para string em uma definição de
string. Por exemplo, as definições

string errorl= ‘c’;
string error2(u’);
string error3 = 22;
string error4(8);
resultam em erros de sintaxe. Note que atribuir um único caractere a um objeto string é permitido em um comando de atribuição, como em
s =
Erro comum de programação 19.1
Tentar converter um int ou um char para um string através de uma atribuição em uma declaração
ou através de um argumento para um construtor é um erro de sintaxe.
Erro comum de programação 19.2
Construir um string que é longo demais para ser representado dispara uma exceção lengtherror.
Diferentemente dos strings char* ao estilo de C, strings não são necessariamente terminados por um caractere nulo. O comprimento de um string é armazenado no objeto string e pode ser acessado com a função membro lenght. O operador subscrito, [ ] , pode ser usado com strings para acessar caracteres individuais. Como strings ao estilo de C, strings têm um primeiro subscrito O e um último subscrito de length-1. Note que um string não é um ponteiro - a expressão &s [O 1 não é equivalente a s quando s é um string.
A maioria das funções membro de string aceitam como argumentos uma posição inicial dada por um
subscrito e o número de caracteres sobre os quais operar.
Tentar passar para uma função membro de string um valor maior do que o comprimento do string (como o número de caracteres a serem processados) resulta no valor sendo feito igual à diferença entre o valor passado e o comprimento do string. Por exemplo, passar 2 (subscrito inicial) e 100 (número de caracteres) para uma função que opera sobre um string de tamanho 50 resulta em 48 (50-2) sendo usado para o número de caracteres.
O operador de extração de stream (»)é sobrecarregado para suportar strings. O comando
string stringObject;
cm » stringObject;
lê um string do dispositivo padrão de entrada. A entrada é delimitada por caracteres em branco. A função getline (do arquivo de cabeçalho <string>) também é sobrecarregada para strings. O comando
string s;
getline(cm, s);
lê um string a partir do teclado para s. A entrada é delimitada por um caractere de nova linha (‘ \n’).
19.2 Atribuição e concatenação de strings
O programa da Fig. 19.1 demonstra a atribuição e concatenação de strings.

Fig. 19.1 Demonstrando a atribuição e concatenação de string (parte 1 de 2).

1	// Fig. 19.1: figl9Ol.cpp
2	II Demonstrando a atribuição e concatenação de strings
3	#include <iostream>
4	
5	using std::cout;


```
6 using std: :endl;
7
8 #include <string>
9
10 using std::string;
11
12 int main()
13 {
14 string sl( “cat” ) , s2, s3;
15
16 s2 = si; /1 atribui si a s2 com =
17 s3.assign( si ) ; II atribui si a s3 com assigri()
i8 cout « “si: ‘ « si « “\ns2: “ « s2 « “\ns3:
19 « s3 « “\n\n”;
20
21 II modifica s2 e s3
22 s2[0]=s3[2)=’r’;
23
24 cout « “Após a modificação de s2 e s3:\n’
25 « “si: “ « si « ‘\ns2: “ « s2 « “\ns3: “;
26
27 II demonstrando a função membro atO
28 int ien = s3.1ength
29 for ( int x = 0; x < len; ++x
30 cout « s3.at( x );
31
32 II concatenação
33 string s4( si + ‘apuit” ), s5; II declara s4 e s5
34
35 II + sobrecarregado
36 s3 += “pet”; II cria “carpet”
37 si.append( “acomb” ); II cria “catacomb”
38
```

```
39 II acrescenta as posições com subscritos 4 até o fim de si
40 II para criar o string “conib” (inicialmente, s5 estava vazio)
41 s5.append( si, 4, si.size() );
42
43 cout « \n\nApós a concatenação:\n « “si: “ « si
44 « “\ns2: “ « s2 « ‘\ns3: “ « s3 « ‘\ns4: “ « s4
45 « “\ns5: “ « s5 « endi;
46
47 return 0;
48 }
si: cat
s2: cat
s3: cat
Após a modificação de s2 e s3:
si: cat
s2: rat
s3: car
Após a concatenação:
si: catacomb
s2: rat
s3: carpet
s4: catapult
s5: comb
```

Fig. 19.1 Demonstrando a atribuição e concatenação de string (parte 2 de 2).

A linha 8 inclui o cabeçalho string à classe string. Três strings si, s2 e s3 são criados na linha 14. A linha

```
16
s2 = si; II atribui si a s2 com =
```

atribui o string si para s2. Após ocorrer a atribuição, s2 é uma cópia de si, mas s2 não está vinculado a si de nenhuma maneira. A linha 17

```
s3.assign( si ) ; II atribui si a s3 com assign()
```

usa a função membro assign para copiar si a s3. É feita uma cópia separada (i. e., si e s3 são objetos independentes). A classe string também fornece uma versão sobrecarregada da função assign, que copia um número especificado de caracteres, como em

```
meuString.assign( s, inicio, numeroDeCaracteres);
```

onde s é o string a ser copiado, inicio é o subscrito inicial e numeroDeCaracteres é o número de caracteres a serem copiados.
A linha 22

```
s2[ O 1 = s3[ 2 1 =
```

usa o operador subscrito para atribuir r’ para s 3 [2) (formando car ‘) e para atribuir T r’ a s2 [O 1 (formando “rat”). Os strings são então exibidos.

As linhas 28 a 30
int len = s3.iengthO;
for (int x = O; x < len; ++x
cout « s3.at(x
usam um laço for para exibir o conteúdo de s3 um caractere por vez, usando a função at. A função at fornece um acesso com verificação de intervalo, i.e., ultrapassar o fim do string dispara uma exceção out of range (ver o Capítulo 13 para uma discussão detalhada do tratamento de exceções). Note que o operador subscrito, [J, não fornece um acesso com verificação de intervalo. Isto é consistente com o uso de arrays.
Erro comum de programação 19.3
Acessar um subscrito de string fora dos limites do string usando afunção at dispara uma exceção out_of_range.
Erro comum de programação 19.4
Acessar um elemento além do tamanho de um string usando o operador subscrito é um erro de lógica.
O string s4 é declarado (linha 33) e inicializado com o resultado da concatenação de si e “apuit” usando o operador de adição, +, sobrecarregado, o qual para a classe string indica concatenação. A linha 36
s3 += “pet”; II cria carpet”
usa o operador adição de atribuição, +=, para concatenar s3 e “pet’.
A linha 37
si.append(“acoznb”); // cria “catacomb”
usa a função append para concatenar si e “acomb”. A linha 41
s5.append(si, 4, si.size());
anexa caracteres de si a s5. Os caracteres do quarto ao último elemento de si são concatenados a s5. A função size retorna o número de caracteres no string si.

19.3 Comparando strings
A classe string fornece funções para comparar strings. O programa da Fig. 19.2 demonstra os recursos para comparação de strings.

```
1 II Fig. 19.2: figi9_02.cpp
2 II Demonstrando os recursos para comparação de strings
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <string>
9
10 using std::string;
11
12 int main()
13 {
```

```
14 string si( “Testando as funções de comparação.”),
15 s2(Hello” ), s3( “stinger’ ), zi( s2 );
16
17 cout « “si: “ « si « “\ns2: « s2
18 « “\ns3: “ « s3 « \nzi: ‘ « zi « “\n\n”;
19
20 Il comparing si and zi
21 if(si==zi)
22 cout « “si == zi\n”;
23 else { // si != zi
24 if(sl>zl)
25 cout « “si > zi\n’;
26 else // si < zi
27 cout « “si < zi\n”;
28 )
29
30 Il comparing si and s2
31 int f = si.compare( s2 );
32
33 if (f==0)
34 cout « “si.compare( s2 ) == 0\n”;
35 elseif ( f>0
36 cout « “sl.compare( s2 ) > 0\n”;
37 else // f< O
38 cout « “si.compare( s2 ) < 0\n”;
39
40 Il cornparing si (elements 2 - 5) and s3 (elements O - 5)
41 f = si.compare( 2, 5, s3, 0, 5 );
42
43 if(f==0)
44 cout « “si.compare( 2, 5, s3, 0, 5 ) == 0\n”;
45 elseif ( f>0
46 cout « ‘sl.compare( 2, 5, s3, 0, 5 ) > 0\n”;
47 else // f<0
48 cout « “sl.compare( 2, 5, s3, 0, 5 ) < 0\n”;
49
50 Il comparing s2 and zi
51 f = zi.compare( 0, s2.size(), s2 );
```

Fig. 19.2 Comparando strings (parte 1 de 2).


```
52
53 if (f==0)
54 cout « zi.compare( 0, s2.sizeO, s2 ) 0 « endi;
```

```
55 elseif ( f>0
56 cout « ‘zi.compare( 0, s2.size(), s2 ) > 0 « endi;
57 else // f< O
58 cout « “zl.compare( 0, s2.size(), s2 ) < 0” « endi;
59
60 return 0;
61 }
si: Testando as funções de comparação.
s2: Helio
s3: stinger
zi: Heilo
si > zi
si.compare( s2 ) > O
sl.compare( 2, 5, s3, 0, 5 ) == O
zi.compare( 0, s2.size, s2 ) == O
```

Fig. 19.2 Comparando strings (parte 2 de 2).

O programa compara quatro strings com as linhas 14 e 15

```
string sl( “Testando as funções de comparação.”
s2(”Hello” ), s3( stinger’ ), zi( s2 );
```

e envia para a saída cada string (linhas 17 e 18). A condição

```
si == zi
```

na linha 21 testa se si é igual a zi. Se a condição é true, “si == zi” é exibido. Se a condição é faise, a condição

```
si > zi
```

na linha 24 é testada. Todas as funções de operador sobrecarregadas demonstradas aqui, bem como as não demonstradas aqui (!=, <, >= e <=) retornam valores booi.

A linha 31

```
int f = sl.compare( s2 );
```

usa a função compare de string para comparar si com s2. A variável fé declarada e recebe O se os strings são equivalentes, um número positivo se si é lexicograficamente maior que s2 ou um número negativo se si é lexicograficamente menor que s2.

A linha 41

```
f = sl.compare( 2, 5, s3, 0, 5 );
```

usa uma versão sobrecarregada da função compare. para comparar partes de si e s3. Os dois primeiros argumentos (2 e 5) especificam o subscrito inicial e o comprimento da parte de si a ser comparada com s3. O terceiro argumento é o string de comparação. Os últimos dois argumentos (0 e 5) são o subscrito inicial e o comprimento da parte do string de comparação a ser comparada. O valor atribuído a f é O para igualdade, um número positivo se si é lexicograficamente maior que s3 ou um número negativo se si é lexicograficamente menor que s3. Um string é então impresso com base no valor de f.

A linha 5 1

```
f = zi.compare( 0, s2.sizeO, s2 );
```

usa outra versão sobrecarregada da função compare para comparar zl e s2. O primeiro argumento especifica o subscrito inicial de zi na comparação. O segundo argumento especifica o comprimento da parte de zi usada na comparação. A função size retorna o número de caracteres no string especificado. O último argumento é o string de comparação. O valor atribuído a f é ou O para igualdade, um número positivo se zi é lexicograticamente maior que s2 ou um número negativo se zi é lexicograficamente menor que s2. Um string é então impresso com base no valor de f.

19.4 Substrings

A classe string fornece a função substr para recuperar um substring de um string. O programa da Fig. 19.3 demonstra substr.

```
1 II Fig. 19.3: figl9_03.cpp
2 // Demonstrando a função substr
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <string>
9
10 using std::string;
11
12 int main()
13
14 string s ( “O aeroplano desapareceu no horizonte.”
15
16 // procura o substring “plano”, que começa
17 // no subscrito 6 e consiste em 5 elementos
18 cout « s.substr( 6, 5 ) « endi;
19
20 return 0;
21}
```

```
plano
```

Fig. 19.3 Demonstrando a função substr.

O programa declara e inicializa um string na linha 18. A linha

```
cout « s.substr( 6, 5 ) « endi;
```

usa a função substr para recuperar um substring de s. O primeiro argumento especifica o subscrito onde começa o substring. O último argumento especifica o número de caracteres a extrair.

19.5 Intercambiando strings

A classe string fornece a função swap para intercambiar strings. O programa da Fig. 19.4 intercambia dois strings.

```
1 II Fig. 19.4: figl9_04.cpp
2 // Usando a função swap para intercambiar dois strings
3 #include <iostream>
```

Fig. 19.4 Usando a função swap para intercambiar dois strings (parte 1 de 2).

CAPÍTULO 19- A CLASSE String E o PROCESSAMENTO EM STREAM DE STRINGS 897

```
4
5 using std::cout;
6 using std: :endl;
7
8 #include <string>
9
10 using std::string;
11
12 int main()
13
14 string first( “um” ), second( “dois” );
15
16 cout « “Antes de swap:\nprimeiro: “ « first
17 « “\n segundo: “ « second;
18 first.swap( second );
19 cout « “\n\nDepois de swap:\nprimeiro: “ « first
20 « “\n segundo: “ « second « endi;
21
22 return 0;
23 }
```

```
Antes de swap:
primeiro: um
segundo: dois
Depois de swap:
primeiro: dois
segundo: um
```

Fig. 19.4 Usando a função swap para intercambiar dois strings (parte 2 de 2).

A linha 14 declara e inicializa os strings first e second. Cada string é então exibido. A linha 18

```
first.swap( second );
```

usa a função swap para permutar os valores de first e second. Os dois strings são novamente impressos para confirmar que eles foram de fato trocados.

19.6 Características de string

A classe string fornece funções para coletar informações sobre o tamanho de um string, seu comprimento, capacidade, comprimento máximo e outras características. O tamanho ou comprimento de um string é o número de caracteres comumente armazenados no string. A capacidade de um string é o número total de elementos que podem ser armazenados no string sem aumentar a capacidade de memória do string. O tamanho máximo do string é o tamanho do maior string que pode ser armazenado em um objeto string. O programa da Fig. 19.5 demonstra as funções da classe string para encontrar o tamanho, o comprimento e outras características de um string.

Fig. 19.5 Imprimindo características de um string (parte 1 de 3).

1	II Fig. 19.5: figl9_05.cpp	
2	II Demonstrando funções	relacionadas a tamanho e capacidade
3	#include <iostream>	
4		
5	using std: :cout;	
6	using std::endl;	


```
7 using std::cin;
8
9 #include <string>
10
11 using std::string;
12
13 void printStats ( const string & );
14
15 int main()
16
17 string s;
18
19 cout « “Características antes da leitura:\n;
20 printStats( s
21
22 cout « “\n\nDigite um string:
23 cm » s; /1 delimitado por espaço em branco
24 cout « “O string digitado era: “ « s;
25
26 cout « “\nCaracterísticas depois da leitura:\n’;
27 printStats( s
28
29 s.resize( s.length() + 10 );
30 cout « “\n\nCaracterísticas após redimensionar com (length+l0):\n”;
31 printStats ( s );
32
33 cout « endl;
34 return 0;
35 }
```

```
36
37 void printStats( const string &str
38
39 cout « “capacity: “ « str.capacity()
40 « \nmaxsize: « str.maxsize()
41 « “\nsize: « str.size()
42 « “\nlength: “ « str.length()
43 « “\nempty: « ( str.empty() ? true: “false” );
44 }
```

```
Características antes da leitura:
capacity: O
max size: 4294967293
size: O
length: O
empty: true
Digite uni strirxg: tomates secos
O string digitado era: tomates
Características depois da leitura:
capacity: 31
maz size: 4294967293
size: 6
length: 6
empty: false
Características após redimensionar com (length + 10)
capacity: 31
```

Fig. 19.5 Imprimindo características de um string (parte 2 de 3).


```
max size: 4294967293
size: 16
length: 16
empty: false
```

Fig. 19.5 Imprimindo características de um string (parte 2 de 2).

O programa declara o string vazio s (linha 17) e passa o mesmo para a função print_Stats (linha 20). Um string vazio é um string que não contém nenhum caractere, O string “tomates” é lido do teclado. Note que strings são delimitados por brancos, o que evita que o string remanescente seja lido.

A função printStats recebe como argumento uma referência para um string const e exibe a sua capacidade (usando a função capacity), seu tamanho máximo (usando a função max size), seu tamanho (usando a função size), seu comprimento (usando a função lenght) e indica se o string está ou não vazio (usando a função empty). A chamada inicial para printStats indica que os valores iniciais para a capacidade, o tamanho e o comprimento de s são 0. Como a capacidade inicial é 0, quando forem colocados caracteres em s será necessário alocar memória para acomodar os novos caracteres. O tamanho e o

comprimento O indicam que no momento não há caracteres armazenados em s. O tamanho e o comprimento são sempre idênticos. O tamanho máximo para esta implementação é 4294967293. O string s é um string vazio, assim a função empty retorna true.
A linha 23 lê um string para s. Note que o operador de extração de stream, », é usado. A linha 29
s.resize(s.length() + 10
usa a função resize para aumentar o comprimento de s em 10 caracteres.
19.7 Encontrando caracteres em um string
A classe string fornece funções para encontrar strings e caracteres em um string. O programa da Fig. 19.6 demonstra as funções de procurafind. Todas as funções de procura são const.

```
1 II Fig. 19.6: figl9O6.cpp
2 // Demonstrando as funções de procura em strings
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <string>
9
10 using std::string;
11
12 int main()
13
14 // o compilador concatena todas as partes em uma só constante string
15 string s( “The values in any left subtree”
16 “\nare less than the value in the’
17 “\nparent node and the values in”
18 ‘\nany right subtree are greater
19 “\nthan the value in the parent node” );
20
```

Fig. 19.6 Demonstrando as funções de procura em um string f ind (parte 1 de 2).


```
21 // encontra “subtree” nas posições 23 e 102
22 cout « “String original:\n” « s
23 « “\n\n(find) \“subtree\” foi encontrado na posição:
24 « s.find( “subtree”
25 « “\n(rfind) \“subtree\” foi encontrado na posição:
26 « s.rfind( “subtre&’ );
27
28 II encontra ‘p’ de “parent’ nas posições 62 e 144
29 cout « “\n(findfirstof) primeiro caractere de \‘qpxz\” na posição:
30 « s.find_first_of( “qpxz”
31 « “\n(findlastof) último caractere de \“qpxz\” na posição:
```

```
32 « s.findlastof( “qpxz” );
33
34 Il encontra ‘b na posição 25
35 cout « “\n(find_first_not_of) primeiro caractere não\n”
36 contido em \‘heTv lusinodrpayft\:
37 « s.findfirstnotof( ‘heTv lusinodrpayft” );
38
39 Il encontra \n na posição 121
40 cout « ‘\n(findlastnotof) último caractere não\n’
41 contido em \‘heTv lusinodrpayft\”:
42 « s.findlastnotof( “heTv lusinodrpayft” ) « endi;
43
44 return 0;
45
String original:
The values in any left subtree
are less than the value in the
parent node and the values in
any right subtree are greater
than the value in the parent node
(find) “subtree” foi encontrado na posição: 23
(rfind) “subtree” foi encontrado na posição: 102
(findfirst_of) primeiro caractere de “qpxz na posição: 62
(findlastof) ultimo caractere de ‘qpxz’ na posição: 144
(find_first_not_of) primeiro caractere não
contido em “heTv lusinodrpayft’: 25
(findlastnotof) último caractere não
contido em “heTv lusinodrpayft”: 121
```

Fig. 19.6 Demonstrando as funções de procura em um string find (parte 2 de 2).

O string s é declarado e inicializado na linha 15. O compilador concatena todos os cinco literais strings em um único literal. Para evitar erros de sintaxe, o final de cada string deve ser fechado com aspas antes de passar para a linha seguinte e começar outro string.

Erro comum de programa ção 19.5

Não terminar um string com aspas é um erro de sintaxe.

A linha 24, que faz parte da operação de inserção

```
« s . find ( “subtree”
```


tenta encontrar o string “subtree” no string s usando a função f ind. Se o string é encontrado, o subscrito da posição inicial daquele string é retornado. Se o string não é encontrado, é retornado o valor string: : npos (uma constante public static definida na classe string). Este valor é retornado pelas funções de procura em string relacionadas, para indicar que um substring ou caractere não foi encontrado no string.

O último item exibido com a inserção no siream na linha 26
s.rfind(“subtree”) II procura reversa
usa a função rfind para pesquisar o string s de trás para diante. Se o string procurado é encontrado, a posição do subscrito é retornada. Se o string não é encontrado, string: : npos é retornado. (Nota: as demais funções apresentadas nesta seção retornam o mesmo tipo de valor, salvo observação em contrário). Note que a constante string: : npos também é usada em um contexto diferente - para indicar todos os elementos de um string.
A chamada
« s.find_first_of(“xz”
na linha 30 usa a função find_first_of para encontrar a primeira ocorrência no string s de qualquer caractere de “qpxz ‘. A busca é feita a partir do início de s. O caractere ‘p’ é encontrado na posição 62.
A chamada
« s.find_last_of(“apxz);
na linha 32 usa a função find last of para encontrar a última ocorrência no string s de qualquer caractere de “xz ‘. A busca é feita a partir do fim de s. O caracter ‘p’ é encontrado na posição 144.
A chamada
« s.find_first_not_of(“heTv lusinodrpayft”);
na linha 37 usa a função find_first_not_of para encontrar o primeiro caractere no string s não contido em “heTv lusinodrpayft’. A busca é feita a partir do início de s.
A chamada
« s.find_last_not_of(‘heTv lusinodrpayft”) « endi;
na linha 42 usa a função find last not of para encontrar o primeiro caractere não contido em “heTv lusinodrpayft’, mas com a busca sendo feita a partir do fim de s.
19.8 Substituindo caracteres em um string
A Fig. 19.7 demonstra as funções string para substituição e eliminação de caracteres.

Fig. 19.7 Demonstrando as funções erase e replace (parte 1 de 2).

1	II Fig. 19.7: figl9_07.cpp	
2	// Demonstrando as	funções erase e replace
3	#include <iostream>	
4		
5	using std::cout;	
6	using std::endl;	

7		
8	#include <string>	

902 C++ COMO PROGRAMAR

```
9
10 using std::string;
1]
int main()
/1 o compilador concatena todas as partes em um só string
string s( “The values in any left subtree
“\nare less than the value in the”
\nparent node and the values in”
“\nany right subtree are greater”
“\nthan the value in the parent node );
// remove todos os caracteres,
// da posição 62 até o fim de s
s.erase( 62 );
// exibe o novo string
cout « String original depois de executar erase:\n” « s
« “\n\nDepois da primeira substituição:\n”;
// substitui todos os espaços por um ponto
int x = s.find( “ “
while ( x < string: :npos
s.replace( x, 1, “.“
x = s . f ind (““, x + 1
36 cout « s « “\n\nDepois da segunda substituição:\n”;
37
// substitui todos os pontos por dois ponto-e-vírgulas
// NOTA: isto vai escrever em cima de outros caracteres
x = s. f ind ( “.“ )
while ( x < string::npos
s.replace( x, 2, “xxxxx;;yyy”, 5, 2 );
x = s.find( “.“, x + 1 );
String original depois de executar erase:
The values in any left subtree
are less than the value in the
Depois da primeira substituição:
The . values . in . any. left. subtree
are. less . than. the . value . iri. the
Depois da segunda substituição:
The; ;alues; ;n; ;ny; ;eft; ;ubtree
are; ;ess; ;han; ;he; ;alue; ;n; ;he
```

Fig. 19.7 Demonstrando as funções erase e replace (parte 2 de 2).

4

12
13
14
15
16
17
18
19
20
21
22
23
22
25
26
27
28
29
30
31
32
33
34
35

}

38
39
40
41
42
43
44
45
46
47
48

cout « s « endl; return 0;

O programa declara e inicializa o string s. A linha 23

s.erase(62);

CAPÍTULO 19 - A CLASSE String E O PROCESSAMENTO EM STREAM DE STRINGS 903
usa a função erase para eliminar todos os caracteres, do elemento 62 até o fim de s.
As linhas 30 a 34
int x s.f ind
while (x < string: :npos
s.replace(x, 1,
x = s f ind (“”, x + 1
usam a função f ind para encontrar cada ocorrência do caractere espaço. Cada espaço é então substituído por um ponto através de uma chamada para a função replace. A função replace recebe três argumentos, o subscrito inicial, o número de caracteres a serem substituídos e o string para substituição. A constante string: : npos representa o máximo comprimento do string. A função f ind retorna s tring: : npos quando o fim de s é atingido.
As linhas 40 a 44
x = s . find (“.“
while (x < string: :npos) {
s.replace(x, 2, “xxxxx;;yyy”, 5, 2);
x = s.find(“.“, x + 1);
usam a função f ind para encontrar cada ponto e a função replace para substituir cada ponto e seu caractere subseqüente por dois ponto-e-vírgulas. Os argumentos passados para replace são o subscrito do elemento onde começa a operação de substituição, o número de caracteres a substituir, um string de caracteres de substituição do qual um substring é usado para substituir caracteres, o elemento no string de caracteres onde começa o substring de substituição e o número de caracteres, no string de caracteres usado para substituição, que serão usados.
19.9 Inserindo caracteres em um string
A classe string fornece funções para inserir caracteres em um string. O programa da Fig. 19.8 demonstra os recursos de insert de string.
```
1 II Fig. 19.8: figl9O8.cpp
2 /1 Demonstrando as funções de inserção em strings.
3 #include <iostream>
4
5 using std::cout;
6 using std: :endl;
7
8 #include <string>
9
10 using std::string;
11
12 int main()
13 {
```

```
14 string si( “início fim” ),
15 s2 ( “meio “ ) , s3 ( “12345678” ) , s4 ( “xx”
16
17 cout « “Strings iniciais:\nsi: “ « si
18 « “\ns2: “ « s2 « “\ns3: “ « s3
19 « “\ns4: “ « s4 « “\n\n”;
20
21 II insere “meio “ na posição 7 de si
22 si.insert( 7, s2 );
```
Fig. 19.8 Demonstrando as funções insert de string (parte 1 de 2).


```
23
24 II insere ‘xx” na posição 3 de s3
25 s3.insert( 3, s4, 0, string::npos );
26
27 cout « “Strings depois da inserção:\nsi: “ « si
28 « “\ns2: “ « s2 « “\ns3: “ « s3
29 « ‘\ns4: “ « s4 « endi;
30
31 return 0;
32
Strings iniciais:
si: inicio fim
s2: meio
s3: i2345678
s4: xx
Strings depois da inserção:
si: inicio meio fim
s2: meio
s3: 123xx45678
s4: xx
```
Fig. 19.8 Demonstrando as funções insert de string (parte 2 de 2).

O programa declara e inicializa quatro strings si, s2, s3, e s4. Cada string é então exibido. A linha 22 sl.insert(7, s2);
usa a função insert para inserir o string s2 antes do elemento 7.
A linha 25
s3.insert(3, s4, 0, string::npos);
usa insert para inserir s4 antes do terceiro elemento de s3. Os dois últimos argumentos especificam o elemento de início de s4 e o número de caracteres de s4 que devem ser inseridos.

1 Dica de desempenho 19.1

f Operações de inserção podem resultar em operações adicionais de administração de memória que podem diminuir o desempenho.

19.10 Conversões para strings char* no estilo da llnguagem C

A classe string fornece funções para a conversão de strings para o estilo de

strings usado na linguagem C. Como mencionado anteriormente, diferentemente de strings no estilo da linguagem C, strings não são necessariamente terminados com nulos. Estas funções são úteis quando uma determinada função recebe um string no estilo de C como argumento. O programa da Fig. 19.9 demonstra a conversão de strings para o estilo de strings usado em C.

Fig. 19.9 Convertendo strings para strings e arrays de caracteres no estilo de C (parte 1 de 2).

i	// Fig. 19.9: figl9O9.cpp	
2	II Convertendo para	strings ao estilo de C.
3	#include <iostream>	
4		
5	using std::cout;	


```
6 using std: :endl;
7
8 #include <string>
9
10 using std::string;
11
12 int main()
13
14 string s( “STRINGS” );
15 const char *ptrl = 0;
16 int len = s.lengthO);
17 char *ptr2 = new char[ len + 1 1; // incluindo o nulo
18
19 II copia caracteres do string para a memória alocada por new
20 s.copy( ptr2, len, O
21 ptr2[ len ] = 0; II acrescenta terminador nulo
22
23 II output
24 cout « “O string s é “ « s
25 « “\ns convertido para um string ao estilo de C é
26 « s.c_str() « “\nptrl é “;
27
28 II Atribui o const char * retornado pela função data O)
29 II ao pointeiro ptrl. NOTA: esta é uma atribuição
```

```
30 II potencialmente perigosa. Se o string for modificado,
31 // o ponteiro ptrl pode se tornar inválido.
32 ptrl = s.data
33
34 for ( int k = 0; k < len; ++k
35 cout « ( ptrl + k ); II usa aritmética de ponteiros
36
37 cout « "\nptr2 é " « ptr2 « endl;
38 delete [1 ptr2;
39 return 0;
40 }
O string s é STRINGS
s convertido para um string ao estilo de C e' STRINGS
ptrl é STRINGS
ptr2 é STRINGS
```

Fig. 19.9 Convertendo strings para strings e arrays de caracteres no estilo de C (parte 2 de 2).

O programa declara s tring como int, e dois ponteiros. O string s é inicializado com "STRINGS", ptrl é inicializado com O, e len é inicializado com o comprimento de s. A memória é alocada dinamicamente e associada ao ponteiro ptr2.

A linha 20

```
s.copy( ptr2, len, O );
```

usa a função copy para copiar s para o array apontado por ptr2. A conversão de string para um string de caracteres no estilo de C é implícita. A linha 21 coloca um caractere nulo de terminação no array ptr2.

A primeira inserção em stream da linha 26

```
« s.cstr()
```

exibe o const char* terminado em nulo retornado de c_str quando o string s é convertido.

A linha 32

```
ptrl s.dataO;
```

atribui um array de caracteres const char* no estilo de C retornado por data, não terminado em nulo, ao ponteiro ptrl. Note que neste exemplo não modificamos o string s. Se o string s fosse modificado. ptrl se tornaria inválido - o que levaria a resultados imprevisíveis.

Note que o array de caracteres retornado por data e o string no estilo de C retornado por cstr têm tempo
de vida limitado. Eles são possuídos pela class string e não devem ser eliminados com delete.

As linhas 34 e 35 usam aritmética de ponteiros para exibir o array apontado por ptrl. Nas linhas 37 e 38, o
string no estilo de C apontado por ptr2 é exibido e a memória alocada para ptr2 é deletada para evitar "perda de memória".

Erro comum de programação 19.6
Não terminar com um caractere nulo o array de caracteres retornado por data ou copy pode levar a
erros durante a execução.
Boa prática de programação 19.1
Sempre que possível, use strings, porque são mais robustos que os strings no estilo de C. Err comum de programação 19.7
Converter um s tring que contém um ou mais caracteres nulos para um string no estilo de Cpode causar erros de lógica. Os caracteres nulos são interpretados como terminadores para strings no estilo de C.
19.11 Iteradores
A classe string fornece iteradores para percorrer strings para frente e para trás. Os iteradores fornecem acesso caracteres individuais com uma sintaxe que é semelhante à das operações com ponteiros. Iteradores não são testados quanto à validade do intervalo. Note que nesta seção damos “exemplos mecânicos” para demonstrar o uso de iteradores. Discutiremos usos mais robustos de iteradores no próximo capítulo. O programa da Fig. 19.10 demonstra os iteradores.
As linhas 14 e 15
string s(“Testando iteradores”);
string::const_iterator ii = s.begin();
declara string s e string: constiterator ii. Um const_iterator é um iterador que não pode modificar o contêiner -através do qual ele está iterando. O iterador ii é inicializado para o início de s com a função begin da classe string. Existem duas versões de begin, uma versão que retorna um iterator para iterar através de um string não-const e uma versão const que retorna um const iterator para iterar através de um string const. O string s é então exibido.
```
1 II Fig. 19.10: figl9_10.cpp
2 II Usando um iterador para exibir um string.
3 #include <iostrearn>
4
5 using std::cout;
6 using std::endl;
7
8 #include <string>
```
Fig. 19.10 Usando um iterador para exibir um string (parte 1 de 2).


```
9
10 using std::string;
11
12 int main()
13
14 string s( ‘Testando iteradores” );
15 string::const_iterator ii = s.begin
```

```
16
17 cout « “s = “ « s
18 « “\n(Usando o iterador ii) s é:
19
20 while ( ii != s.end() ) {
21 cout « i1; II derreferencia o iterador para obter caractere
22 ++il; II avança o iterador para o próximo caractere
23
24
25 cout « endi;
26 return 0;
27
s = Testando iteradores
(Usando o iterador ii) s é: Testando iteradores
```

Fig. 19.10 Usando um iterador para exibir um string (parte 2 de 2).

As linhas 20 a 23

```
while ( ii != s.end() )
cout « i1; II derreferencia o iterador para obter caractere
++il; II avança o iterador para o próximo caractere
```

usam o iterador ii para “percorrer” s. A função end retorna um iterador para a primeira posição após o último elemento de s. Os conteúdos de cada posição são impressos primeiramente derreferenciando o iterador, de forma semelhante à que usamos para derreferenciar um ponteiro, e o iterador é então avançado uma posição, usando o operador ++.

A classe string fornece as funções membro rend e rbegin para acessar caracteres individuais do s tring em ordem inversa, do fim do string para o começo do string. As funções membro rend e rbegin podem retornar reverse iterators e const reverse iterators (dependendo de se o string é ou não const). Pedimos ao leitor para demonstrar isto nos exercícios. Usaremos mais iterators e reverse iterators no Capítulo 20.

® Dica de teste e depura ção 19.1

Use a função membro at de string (ao invés de iteradores) quando você quiser que seja testada a
validade do intervalo.

19.12 Processamento de strings em streams

Além da EIS em streams padrão e da EIS em streams de arquivos, a EIS em streams de C++ inclui recursos para ler de strings na memória e escrever em strings na memória. Estes recursos são freqüentemente chamados de EI S na memória ou processamento de strings em streams.

A leitura a partir de um string é suportada pela classe istringstream. A saída para um string é
suportada pela classe ostringstream. Os nomes istringstream e ostringstreain são na realidade aliases (nomes alternativos). Estes nomes são definidos com os typedefs

typedef basic_istringstream< char > istringstream;
typedef basic_ostringstream< char > ostringstream;
As classes basic istringstream e basic ostringstream fornecem a mesma funcionalidade das classes is tream e os tream mais Outras funções membro específicas para a formatação na memória. Os programas que usam formatação na memória devem incluir os arquivos de cabeçalho <sstream> e <iostreain>.
Uma aplicação destas técnicas é a validação de dados. Um programa pode ler uma linha inteira de uma vez só do stream de entrada para um string. Em seguida, um rotina de validação pode examinar o conteúdo do string e, se necessário, corrigir os dados. Então, o programa pode continuar a fazer a entrada a partir do string, sabendo que os dados de entrada estão no formato adequado.
Fazer saída para um string é uma maneira interessante de tirar partido dos poderosos recursos de formamção de streams de C++. Os dados podem ser preparados em um string para imitar o formato editado na tela. Aquele string pode, então, ser gravado em um arquivo em disco para preservar a imagem da tela.
Um objeto ostringstream usa um objeto string para armazenar os dados que são enviados para a
saída. A função membro str de ostringstream retorna uma cópia string do string.
A Figura 19.11 demonstra um objeto ostringstream. O programa cria o objeto outputString de
ostringstream (linha 18) e usa o operador de inserção em stream para fazer a saída de uma série de strings e valores numéricos para o objeto.

```
1 II Fig. 19.11: figl9ll.cpp
2 /1 Usando um objeto ostringstream alocado dina.micamente.
3 #include <iostream>
4
5 using std::;cout;
6 using std::endl;
7
8 #jnclude <string>
9
10 using std::string;
11
12 #include <sstream>
13
14 using std::ostringstream;
15
16 int main()
17
18 ostringstream outputString;
19 string si ( “Saída de diversos tipos de dados “
20 s2( “usando um objeto ostringstream:” ),
21 s3( “\n double: “
22 s4( “\n int: “
```

```
23 s5( “\nendereço de int: “
24 double d = 123.4567;
25 inti22;
26
27 outputString « si « s2 « s3 « d « s4 « i « s5 « &i;
28 cout « “outputString contém:\n” « outputString.strO;
29
30 outputString « “\nmais caracteres adicionados”;
31 cout « “\n\napós as inserções adicionais no stream,\n”
```

Fig. 19.11 Usando um objeto ostringstream alocado dinamicamente (parte 1 de 2).


```
32 « “outputString contém:\n” « outputString.str()
[e 33 « endi;
32
35 return O;
36
```

Fig. 19.11 Usando um objeto ostringstreain alocado dinamicamente (parte 2 de 2).

A linha 27

```
outputString « si « s2 « s3 « d « s4 « i « s5 « &i;
```

faz a saída de string si. string s2, string s3, doubie d, string s4. int i, string s5 e do endereço de int i, todos para outputString na memória. A linha 28

```
cout « “outputString contém:\n” « outputString.str
```

usa a chamada outputString. str ( ) para exibir uma cópia do string criado na linha 27. A linha 30 demonstra que mais dados podem ser anexados ao string na memória, simplesmente aplicando-se o operador de inserção em stream a outputString. A linha 32 exibe o string outputString após a anexação de caracteres adicionais.

Um objeto istringstream lê dados de um string na memória para variáveis do programa. Os dados são armazenados em um objeto istrings tream como caracteres. A entrada do objeto istrings tream funciona de maneira idêntica à entrada de dados de um arquivo qualquer, em geral, ou da entrada padrão em particular. O fim do string é interpretado pelo objeto istringstreain como um fim de arquivo.

A Fig. 19.12 demonstra a entrada de um objeto istringstream.

As linhas l8e 19

```
string input( Testando leitura 123 4.7 A” );
istringstream inputString( input );
```

criam o string input contendo os dados e o objeto inputString de istringstream construído para conter os dados no string input. O string input contém os dados

```
Testando leitura 123 4.7 A
```

os quais, quando lidos como entrada para o programa, consistem em dois

strings (“Testando”e “leitura”), um valor int (123), um valor double (4.7) e um valor char (‘A’). Estes caracteres são extraídos para as variáveis stringi. string2, i. de c, respectivamente na linha 25.
inputString » stringi » string2 » i » d » c;
A saída dos dados é então feita nas linhas 27 a 33. O programa tenta ler novamente do inputString com o
comando if/else da linha 38. Como não há mais dados, a condição do if (linha 40) é avaliada como false e
a parte eise da estrutura if/eise é executada.

outputString contém:				
Saída de diversos tipos de dados	usando	um	objeto	ostringstream:
doubie: 123.457				
int: 22				
endereço de int: OO68FDOC				
após as inserções adicionais no	streain,			
outputString contém:				
Saida de diversos tipos de dados	usando	um	objeto	ostringstreani:
doubie: 123.457				
int: 22				
endereço de int: OO68FDOC				
mais caracteres adicionados				


```
1 II Fig. 19.12: figl9_12.cpp
2 II Demonstrando a entrada de dados a partir de um objeto istringstream.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <string>
9
10 using std::string;
11
12 #include <sstream>
13
14 using std::istringstream;
15
16 int main()
17
18 string input( “Testando leitura 123 4.7 A’ );
19 istringstream inputString( input );
```

```
20 string stringl, string2;
21 int i;
22 double d;
23 char c;
24
25 inputString » stringl » string2 » i » d » e;
26
27 cout « “Os seguintes itens foram extraídos\n”
28 « “do objeto istringstream:”
29 « “\nstring: “ « stringl
30 « “\nstring: « string2
31 « ‘\n int: “ « i
32 « “\ndouble: “ « d
33 « “\n char: “ « c;
34
35 // tentativa de leitura de stream vazio
36 long x;
37
38 inputString » x;
39
40 if ( inputString.good()
41 cout « “\rz\nvalor long é: “ « x « endl;
42 else
43 cout « “\n\ninputString está vazio” « endl;
44
45 return 0;
46 }
```

```
Os seguintes itens foram extraídos
do objeto istringstream:
string: Testando
string: leitura
int: 123
double: 4.7
char: A
inputString está vazio
```

Fg. 19.12 Demonstrando a entrada de dados a partir de um objeto istringstream.

Resumo

• O gabarito de classe basic string de C++ fornece operações típicas de manipulação de strings, tais como cópia, busca, etc.

• O comando typedef

typedef basic_string< char > string;

cria o tipo string para basic string< char >. Também é fornecido um typedef para o tipo wchar_t. O tipo wchat_t normalmente armazena dois bytes (16 bits) para

suportar outros conjuntos de caracteres. O tamanho de wchart não é fixado pelo padrão.
• Para usar strings, inclua o arquivo de cabeçalho <string> da biblioteca padrão de C++.
• A classe string não fornece conversões de int ou char para string.
• É permitido atribuir um único caractere a um objeto string em um comando de atribuição.
• strings não são necessariamente terminados por nulos.
• O comprimento de um string é armazenado no objeto string e pode ser recuperado através da função membro lenght ou size.
• A maioria das funções membro de string aceita como argumentos uma posição de subscrito inicial e o número de caracteres sobre os quais operar.
• Tentar passar para uma função membro de string um valor maior que o comprimento do string. como o número de caracteres a serem processados, resulta no valor sendo redefinido como o comprimento do resto do string.
• A classe string fornece o operator sobrecarregado e a função membro assign para atribuições de strings.
• O operador subscrito, [ ], possibilita o acesso direto a qualquer elemento de um string.
• A função at oferece um acesso verificado - ultrapassar qualquer dos limites do s tririg dispara uma exceção outof range. O operador subscrito, [ ], não oferece um acesso verificado.
• A classe string fornece os operadores sobrecarregados + e = e a função membro append para realizar concatenação de strings.
• A classe string fornece os operadores sobrecarregados ==, =, <, >, <=, >=, para comparação de strings.
• A função compare de string compara dois strings (ou substrings) e retoma O se os stririgs são iguais, um número positivo se o primeiro string é lexicograficamente maior que o segundo ou um número negativo se o primeiro string é lexicograficamente menor que o segundo.
• A função substr recupera um substring de um string.
• A função swap permuta (troca) o conteúdo de dois strings.
• As funções size e lenght retomam o tamanho ou comprimento de um string. (i.e, o número de caracteres correntemente armazenado no string).
• A função capacity retoma o número total de elementos que podem ser armazenados no string sem aumentar os requisitos de memória do string.
• A função max size retoma o tamanho do maior string possível que pode ser armazenado.
• A função resize muda o comprimento do string.
• As funções find da classe string, find, rfind, find first of, find last of, firid first not of e findlastnotof procuram strings ou caracteres em um stririg.
• O valor string: : pos é freqüentemente usado para indicar o processamento de todos os elementos de um string em funções que requerem um número de caracteres a serem processados.
• A função erase apaga elementos de um strirzg.
• A função replace substitui caracteres em um string.

• A função irisert insere caracteres em um string.
• A função cstr retoma um const char * apontando para um string de caracteres terminado com nulo, no estilo de C, que contém todos os caracteres em um string.
• A função data retorna um const char * apontando para um array de caracteres não-terminado com nulo, no estilo de C, que contém todos os caracteres em um string
• A classe string fornece as funções membro end e begiri para acessar caracteres individuais.
• A classe string fornece as funções membro rend e rbegin para acessar caracteres individuais em um string na ordem inversa, do fim de um string para o começo do mesmo.
• A leitura de um string é suportada pelo tipo istringstream. A saída para um string é suportada pelo tipo os- trings tream.
• Programas que usam formatação na memória devem incluir os arquivos de cabeçalho <sstream> e <iostream>.
• A função membro str de ostringstream retorna uma cópia string do string.

Terminologia

acesso verificado função findfirstof
arquivo de cabeçalho <sstream> função find_last_not_of
arquivo de cabeçalho <string> função findlastof
capacidade função getline
caracteres largos função insert
classe istringstream função lenght
classe ostringstream função max_size
classe string função membro siream de string str
comprimento de um s tring função rbegin
constiterator ftinção rend
constreverseiterator função replace
EIS in-core função resize
E/S na memória função size
exceção lenghterror função substr
exceção out_of_range função swap
exceção range_error funçõesflnd
função access iterator
função at namespace std
função c_s tr operador subscrito,
função capacity operadores de igualdade: =, =
função compare operadores relacionais:>, <, >, <
função copy operadores: +, +=. «, », E 1
função data reverse_iterator
função empty string vazio
função erase tamanho máximo de um string
função f ind ipo wchar_t
função find_first_not_of typedef basic_string<char> string

Erros comuns de programação
19.1 Tentar converter um ínt ou um char para um string através de uma atribuição em uma declaração ou através de um argumento para um construtor é um erro de sintaxe.
19.2 Construir um string que é longo demais para ser representado dispara uma exceção length_error.
19.3 Acessar um subscrito de stringfora dos limites do string usando a função at dispara uma exceção outof range.
19.4 Acessar um elemento além do tamanho de um string usando o operador subscrito é um erro de lógica.
19.5 Não terminar um string com aspas é um erro de sintaxe.

CAPÍTULO 19 - A CLASSE String E O PROCESSAMENTO EM STREAM DE STRINGS 913
19.6 Não terminar com um caractere nulo o array de caracteres retornado por data ou copy pode levar a erros durante a execução.
19.7 Converter um string que contém um ou mais caracteres nulos para um string no estilo de C pode causar erros de lógica. Os caracteres nulos são interpretados como terminadores para strings no estilo de C.
Boa prática de programação
19.1 Sempre que possível, use strings, porque são mais robustos que os strings no estilo de C.
Dica de desempenho
19.1 Operações de inserção podem resultar em operações adicionais de administração de memória que podem diminuir o desempenho.
Dica de teste e depura ção
19.1 Use a função membro at de string (ao invés de iteradores) quando você quiser que seja testada a validade do intervalo.
Exercícios de auto-revisão
19.1 Preencha os espaços em branco em cada um dos seguintes itens:
a) O cabeçalho deve ser incluído para a classe string.
b) A classe string pertence ao namespace
e) A função apaga caracteres de um string.
d) A função encontra a primeira ocorrência de qualquer caractere de uma série de caracteres.
19.2 Diga quais das seguintes afirmações são verdadeiras e quais são falsas. Se uma afirmação é falsa, explique porquê.
a) A concatenação pode ser executada com o operador adição +=.
b) Os caracteres em uni string começam no elemento O.
e) O operador de atribuição, =. copia um string.
d) Um string no estilo de C é um string.
19.3 Encontre o(s) erro(s) em cada um dos seguintes itens e explique como corrigi-lo.
a) string sv(28); II construct sv
string bc(‘z’); // construct bc
b) // assuma que o namespace std é conhecido

const char *ptr = name.dataO; II name é “joe bob’
ptr[3) =
cout « ptr « endi;
Respostas aos exercícios de auto-revisão
19.1 a)string b)std c)erase d)findfirstof
19.2 a) Verdadeiro.
b) Verdadeiro.
e) Verdadeiro.
d) Falso. Um string é um objeto que fornece muitos serviços diferentes. Um string no estilo de C não fornece serviços. Strings no estilo de C são terminados com nulo e strings não.
19.3 a) Não existem construtores para os argumentos passados. Deveriam ser usados outros construtores válidos convertendo os argumentos para strings se necessário.
b) A função data não adiciona um terminador nulo. Em seu lugar use c_str.

914 C++ COMO PROGRAMAR
Exercícios 19.14 1
para mc
19.4 Preencha os espaços em branco em cada um dos seguintes itens:
a) As funções , , e convertem strings para strings no estilo de C. 19.15 1
b) A função é usada para atribuição, uma sei
c) é o tipo de retomo da função rbegin. 19.16 1
d) A função é usada para recuperar um substring. que cad
19.5 Diga quais das seguintes afirmações são verdadeiras e quais são falsas. Se uma afirmação é falsa, explique por quê.
a) stririgs são terminados com nulos. d
b) A função max size retorna o tamanho máximo para um string. po e se
c) A função até capaz de disparar uma exceção outof range.
d) Por default, strings são passados por referência.
Note qu
19.6 Encontre quaisquer erros em cada um dos seguintes itens e explique como corrigi-lo(s). etc
a) std::cout « s.data() « std::endl; /1 s é “heilo” gadasn
b) erase (s . rfind(“x”) , 1) ; // s é “xenon” 19 17 r
c) string& foo(void) teres.O
se o usu
string s (“HelJ.o”);
1/ outros comandos da função 19.18 E
return; é igual 1
‘arar
19.7 (Criptografia simples) Algumas informações na Internet podem ser cifradas com um algoritmo simples conhecido como 19.19 F
“rotl3” - que “gira” cada caractere 13 posições no alfabeto. Assim, ‘a corresponde a ‘n e x corresponde a ‘k’.O rotl3 19.20
é um exemplo de criptografia simétrica por chave. Com a criptografia simétrica

por chave, tanto o encriptador como o decripta do usam a mesma chave. 19.2 1 E
a) Escreva um programa que encripta uma mensagem usando o rotl3. 1922 E
b) Escreva um programa que decripta mensagem embaralhada usando 13 como a chave.
c) Após escrever o programa da parte a) e da parte b), responda sucintamente à seguinte questão: se você não sabe a strto
chave para a parte b), quão difícil você pensa que seria decifrar o código usando quaisquer recursos disponíveis? O 19.23 E
que aconteceria se você tivesse acesso a recursos computacionais muito poderosos (p. ex., supercomputadores Cray?)
No Exercício 19.27, pediremos que você faça isso. 19.24
19.8 Escreva um programa usando iteradores que demonstre ouso das funções rbegin e rend. 19.25 E
19.9 Escreva suas próprias versões das funções data e c_str. 19.26 1
19.10 Escreva um programa que lê vários strings e imprime somente aqueles terminando em “r” ou “ia. Somente devem ser levadas em conta letras minúsculas.
19.11 Escreva um programa que demonstra passar um s tring tanto por referência como por valor.
19.12 Escreva um programa que lê separadamente um nome e um sobrenome e então concatena ambos em um novo string.
19.13 Escreva um programa que joga ojogo da forca. O programa deve pegar uma palavra (que está codificada no programa ou é lida de um arquivo-texto) e exibira seguinte:
Adivinhe a palavra XXXXXX
Cada X representa uma letra. Se o palpite do usuário estiver correto, o programa deve exibir:
Parabéns’ Você adivinhou minha palavra. Quer jogar de novo? sim/não
A resposta apropriada, sim ou não, deve ser lida. Se o palpite do usuário estiver incorreto, exiba a parte apropriada do corpo da
pessoa na forca. 19.27
Após sete palpites errados, o usuário deve estar enforcado. O display deve se parecer com decript
mais fr
/ 1 \ Escrevc
encnpt
/ \ 19.28
Após cada palpite, você quer exibir todos os palpites do usuário.

19.14 Escreva um programa que lê um string e imprime o string de trás para diante. Converta todas as letras maiúsculas para minúsculas e todas as minúsculas para maiúsculas.
19.15 Escreva um programa que usa os recursos de comparação introduzidos

neste capítulo para colocar em ordem alfabética uma série de nomes de animais. Somente letras maiúsculas devem ser usadas para as comparações.
19.16 Escreva um programa que cria um criptograma a partir de um string. Um criptograma é uma mensagem ou palavra em que cada letra é substituída por outra letra. Por exemplo, o string
The birds naxne was squawk
pode ser embaralhado para formar
xms kbypo zhqs fho obrhfu
Note que os espaços não são embaralhados. Neste caso particular, 'T' foi substituído por x' , cada 'a' foi substituído por ,etc. Letras maiúsculas e letras minúsculas deveriam ser tratadas da mesma forma. Use técnicas semelhantes àquelas empregadas no Exercício 19.7.
19.17 Modifique o programa do exercício anterior para permitir que um usuário solucione o criptograma fornecendo dois caracteres. O primeiro caractere especifica a letra no criptograma e o segundo caractere especifica o palpite do usuário. Por exemplo, se o usuário digita r g. então o usuário está dando o palpite que a letra r é na verdade um g.
19.18 Escreva um programa que lê uma sentença e conta o número de palíndromos na sentença. Um palíndromo é uma palavra que
é igual tanto lida da esquerda para direita como da direita para a esquerda. Por exemplo, 'amor" não é um palíndromo, mas
"arara" é um palíndromo.
19.19 Escreva um programa que conta o número de vogais em uma sentença. Exiba a freqüência de ocorrência de cada vogal.
19.20 Escreva um programa que insere os caracteres exatamente no meio de um string.
19.21 Escreva um programa que apaga as seqüências "por" e "POR" de um string
19.22 Escreva um programa que lê uma linha de texto, substitui todos os sinais de pontuação por espaços e, então, usa a função strtok da biblioteca de strings da linguagem C para separar o string em palavras individuais.
19.23 Escreva um programa que lê uma linha de texto e então imprime o texto ao contrário. Use iteradores na sua solução.
19.24 Escreva uma versão recursiva do Exercício 19.23.
19.25 Escreva um programa que demonstra o uso das funções erase que recebem iterators como argumentos.
19.26 Escreva um programa que, a partir do stririg "abcdefghijklmnopqrstuvwxyz { ", gera o seguinte:
a
bcb
cdedc
defgfed
efghihgfe
fghijkj ihgf
ghijklmlkjihg
hijklmnonnilkjih

ijklmriopqponmlkj i
j klmnopqrsrqponmlkj
klmnopqrs tutsrqponmlk
lmnopqrs tuvwvutsrqponml
mnopqrs tuvwxyxwvutsrqponxn
nopqrstuvwxyz { zyxwvutsrqporz
19.27 No Exercício 19.7, pedimos que você escrevesse um algoritmo simples de criptografia. Escreva um programa que tenta
decriptar uma mensagem “rotl3” usando a substituição de freqüência simples (assuma que você não conhece a chave). As letras mais freqüentes na frase encriptada devem ser substituidas pelas letras mais comumente usadas em inglés (a,e,i,o,u,s,t,r,etc.). Escreva as possibilidades em um arquivo, O que tornou fácil decifrar o código? Como pode ser melhorado o mecanismo de encriptação?

19.28 Escreva uma versão do bubble sort que classifica strings. Use a função swap na sua solução.

20
A biblioteca padrão de gabaritos (STL)

Objetivos
• Ser capaz de usar contêineres padronizados da STL, contêineres adaptadores e “quase contêineres”.
• Ser capaz de programar com as dezenas de algoritmos da STL.
• Compreender como algoritmos usam iteradores para acessar os elementos dos contêineres da STL.
• Familiarizar-se com os recursos da STL disponíveis na Tnternet e na World Wide Web.

As formas que um contêiner brilhante pode conter Theodore Roethke
Viaje por todo o universo em um mapa. Miguel de Cervantes
Oh! tendes perseverança condenável, e arte suficiente para poder corromper um santo.
William Shakespeare
Aquele enorme monte de pó chamado “história Augustine Birreil
O historiador é um profeta em reverso. Friedrich von Schlegel
Attempt the end, and never stand to doubt;
Nothing ‘s so hard but search willflnd it out.
Robert Herrick

Prossiga - continue andando. Thomas Morton

CAPÍTULO 20 - A BIBLIOTECA PADRÃO DE GABARITOS (STL) 917
Visão Geral
20.1 Introdução à biblioteca padrão de gabaritos (STL)
20.1.1 Introdução a contêineres
20.1.2 Introdução a iteradores
20.1.3 Introdução a algoritmos
20.2 Contêineres seqüenciais
20.2.1 O contêiner seqüencial vector
20.2.2 O contêiner seqüencial list
20.2.3 O contêiner seqüencial deque
20.3 Contêineres associativos
20.3.1 O contêiner associativo multiset
20.3.2 O contêiner associativo set
20.3.3 O contêiner associativo multimap
20.3.4 O contêiner associativo map
20.4 Adaptadores de contêineres

20.4.1 O adaptador stack
20.4.2 O adaptador queue
20.4.3 O adaptador priority_queue
20.5 Algoritmos
20.5.1 f iii, fill_n, generate e generate_n
20.5.2 equal, mismatch e lexicographical compare
20.5.3 remove, remove if, remove copy e remove copyif
20.5.4 replace, replace if replace copy e replacecopyif
20.5.5 Algoritmos matemáticos
20.5.6 Algoritmos básicos de pesquisa e classificação
20.5.7 swap, iter_swap e swap_ranges
20.5.8 copybackward, merge, unique e reverse
20.5.9 inplacemerge, unique_copy e reversecopy
20.5.10 Operações sobre conjuntos
20.5.11 lowerbound, upper_bound e equal_range
20.5.12 Heapsort
20.5.13 minemax
20.5.14 Algoritmos não-cobertos neste capítulo
20.6 A classe bitset
20.7 Objetos função
Resumo• Terminologia. Erros comuns de programação . Boas práticas de programação. Observações de
engenharia de software • Dicas de desempenho • Dicas de teste e depura ção Exercícios de auto-revisão
Respostas aos exercícios de auto-revisão • Exercícios • Recursos para a STL na Internet e na World Wide
Web . Bibliografia da STL

“inteligente”, corno veremos. Classes de iteradores são projetadas para serem usadas genericamente com qualquer contêiner.

Os contêineies encapsulam algumas operações primitivas, mas os algoritmos da STL são implementados independentemente dos contêineres.
A STL evita new e delete em favor de alocadores para a alocação e desalocação de memória. O programador pode fornecer alocadores para customizar a maneira como um contêiner trata a administração de memória, mas os alocadores default fornecidos pela STL são sufucientes para a maioria das aplicações. Alocadores customizados são um tópico avançado, fora do escopo deste texto.
Este texto pretende ser uma introdução à STL. Ele não é de forma alguma completo ou abrangente. Entretanto, é um capítulo amigável e acessível que deveria lhe convencer do valor da STL e incentivá-lo a estudar mais a mesma. Usamos a mesma abordagem de “código que funciona” que usamos ao longo de todo o livro. Em termos da sua percepção do valor de “reutilizar, reutilizar,

reutilizar", este pode ser um dos capítulos mais importantes do livro. Os contêineres da STL contêm as estruturas de dados mais comuns e de maior valor. Elas são todas definidas como gabaritos, de forma que você possa adaptá-las para conter o tipo de dados relevante para suas aplicações particulares.
No Capítulo 15, estudamos estruturas de dados. Construímos listas encadeadas, filas, pilhas e árvores. Ligamos cuidadosamente objetos com ponteiros. O código baseado em ponteiros é complexo e a menor omissão ou desatenção pode conduzir a sérias violações de acesso à memória e erros de "perda de memória" sem reclamações do compilador. Implementar estruturas de dados adicionais, tais como deques, filas com prioridade, conjuntos, mapas, etc. iria requerer um trabalho adicional substancial.

Observação de engenharia de software 20.2

Evite reinventar a roda; programe com componentes reutilizáveis da biblioteca padrão de C++. A STL contém muitas das estruturas de dados mais populares, como contêineres, e fornece vários algoritmos populares que os programas usam para processar dados nestes contêineres.

Dica de (este e depura ção 20.1

Quando programamos com estruturas de dados e algoritmos baseados em ponteiros, devemos fazer nossa própria depuraão e teste para garantir que nossas estruturas de dados, classes e algoritmos funcionam corretamente. Efácil cometer erros ao manipular ponteiros neste nível tão baixo. "Perdas de memória" e violações de acesso à memo ria são comuns em tal código customizado. Para a maioria dos pro gramadores, e para maioria das aplicações que necessitarão escrever as estruturas de dados definidas como gaba- ritos, pré-empacotadas, da STL são suficientes. Usar o código da STL pode evitar muito tempo de teste e depura ção. Uma precaução a ser tomada com relação a grandes projetos é que o tempo de compilação pode ser significativo.

Cada contêiner da STL tem funções membro associadas. Algumas funcionalidades se aplicam a todos os contêineres da STL. Outras funcionalidades são específicas para determinados contêineres. Ilustramos a maioria das funcionalidades comuns com os gabaritos de classe vector, list e deque. Apresentamos funcionalidades específicas para um contêiner nos exemplos de cada um dos outros contêineres da STL.
Fizemos uma pesquisa extensa de recursos disponíveis na InternetlWorld Wide Web e os incluímos para você
no final deste capítulo. Também fornecemos uma extensa bibliografia de artigos relacionados com a STL.

20.1.1 Introdução a contêineres

Os tipos de contêineres da STL são mostrados na Fig. 20.1. Os contêineres estão divididos em três categorias principais - contêineres de seqüências, contêineres associativos e adaptadores de contêineres. Os contêineres de seqüências são às vezes chamados de contêineres seqüenciais; nonnalmente usaremos o termo contêineres seqüenciais. Os contêineres seqüenciais e os contêineres associativos são chamados coletivamente de contêineres de primeira classe. Existem quatro outros tipos de contêineres que são considerados “quase contêineres” - arrays no estilo de C (discutidos no Capítulo 4), strings (discutidos no Capítulo 19), bitsets para manter conjuntos de indicadores 1/O e valarray para executar operações matemáticas sobre vetores em alta velocidade (esta classe é otimizada

*1
1

para desempenho e não é tão flexível quanto os contêineres de primeira classe). Estes quatro tipos são considerados “quase contêineres” porque apresentam recursos similares aos contêineres de primeira classe, mas não suportam todos os recursos dos contêineres de primeira classe.

Fig. 20.1 Classes contêiner da biblioteca padrão.

A STL foi cuidadosamente projetada de maneira que os contêineres forneçam funcionalidades similares. Há muitas operações genéricas, como a função size. que se aplicam a todos os contêineres e outras operações que se aplicam a subconjuntos de contêineres similares. Isto estimula a extensibilidade da STL com novas classes. As funções comuns a todos os contêineres da STL são ilustradas na Fig. 20.2. [Nota: as funções operador sobrecarregadas operator<, operator<=, operator>, operator>=. operator == eoperator!=não são fornecidas para priorityqueue.

Fig. 20.2 Funções comuns a todos os contêineres da STL (parte 1 de 2).

Classes contêiner da Biblioteca Padrão	Descrição
Contêineres seqüenciais	
vector	inserções e retiradas rápidas no fim acesso direto a qualquer elemento
deque	inserções e retiradas rápidas no início e no fim acesso direto a qualquer elemento
i.ist	lista duplamente encadeada, inserção e retirada rápidas em qualquer lugar

Contêineres associativos	
set	pesquisa rápida, duplicatas não-permitidas
multiset	pesquisa rápida, duplicatas permitidas
map	mapeamento um para um, duplicatas não-permitidas, pesquisa rápida usando chaves
muJ.timap	mapeamento um para muitos, duplicatas não-permitidas, pesquisa rápida usando chaves
Adaptadores de contêineres	
stack	último a entrar, primeiro a sair (LIFO)
queue	primeiro a entrar, primeiro a sair (FIFO)
priorityqueue	o elemento de mais alta prioridade é sempre o primeiro a sair

Funções membro comuns a todos os contêineres da STL	Descrição
construtor default	Um construtor para fazer uma inicialização default do contêiner. Normalmente, cada contêiner tem diversos construtores que fornecem uma variedade de métodos de inicialização para o contêiner.
construtor de cópia	Um construtor que inicializa o contêiner como sendo uma cópia de um contêiner existente do mesmo tipo.

Funções membro
comuns a todos os
contêineres da STL Descrição
destruidor Função destruidor para limpeza depois que um contêiner não é mais necessário.
empty Retorna true se não há nenhum elemento no contêiner; caso contrário, retorna false. niaxsize Retorna o número máximo de elementos de um contêiner.

size Retorna o número de elementos presentemente no contêiner.
operator= Atribui um contêiner a outro.
operator< Retorna true se o primeiro contêiner é menor do que o segundo contêiner; caso contrario, retorna false.
operator<= Retorna true se o primeiro contêiner é menor ou igual ao segundo contêiner; caso contrário, retorna false.
operator> Retorna true se o primeiro contêiner é maior do que o segundo contêiner; caso contrário, retorna false.
operator>= Retorna true se o primeiro contêiner é maior ou igual ao segundo contêiner; caso contrário, retorna false.
operator== Retorna true se o primeiro contêiner é igual ao segundo contêiner;

caso contrário, retorna false.
operator ! = Retorna true se o primeiro contêiner não é igual ao segundo contêiner, caso contrário, retorna false.
swap Permuta os elementos de dois contêineres.
Funções que são encontradas somente em contêineres de primeira classe
begin As duas versões desta função retornam ou um iterator ou um const_iterator que se refere ao primeiro elemento do contêiner.
end As duas versões desta função retornam ou um iterator ou um constiterator que se refere à próxima posição após o fim do contêiner.
rbegin As duas versões desta função retornam ou um reverse_iterator ou um constreverseiterator que se refere ao último elemento do contêiner.
rend As duas versões desta função retornam ou um reverse iterator ou um constreverseiterator que se refere à posição antes do primeiro elemento do contêiner.
erase Apaga um ou mais elementos do contêiner.
clear Apaga todos os elementos do contêiner.
Fig. 20.2 Funções comuns a todos os contêineres da STL (parte 2 de 2).
Os arquivos de cabeçalho para cada um dos contêineres da biblioteca padrão são mostrados na Fig. 20.3. Os conteúdos destes arquivos de cabeçalho estão todos em naxnespace std. [Nota: alguns compiladores C++ ainda não suportam os novos arquivos de cabeçalho. Muitos destes compiladores fornecem sua própria versão dos nomes dos arquivos de cabeçalho. Veja a documentação do seu compilador para mais informações sobre o suporte à STL que seu compilador fornece.]
A Fig. 20.4 mostra os typedefs comuns (para criar sinônimos ou apelidos para nomes de tipos muito longos) encontrados em contêineres de primeira classe. Estes typedefs são usados em declarações genéricas de variáveis, parâmetros para funções e valores de retorno de funções. Por exemplo, value type em cada contêiner 4. é sempre um typedef que representa o tipo de valor armazenado no contêiner.

Arquivos de cabeçalho de contêineres da biblioteca padrão
<vector>
<li st>
<deque>
<queue> contém queue e priorityqueue
<stack>
<map> contém map e inultimap
<set> contém set e znultiset
<bitset>
Fig. 20.3 Arquivos de cabeçalho dos contêineres da biblioteca padrão.
typedef Descrição
valuetype O tipo do elemento armazenado no contêiner.
reference Uma referência ao tipo do elemento armazenado no contêiner.
constreference Uma referência constante ao tipo do elemento armazenado no

contêiner.
Tal tipo de referência pode ser usado somente para ler elementos no contêiner e para executar operações const.
pointer Um ponteiro para o tipo do elemento armazenado no contêiner.
i terator Um iterador que aponta para o tipo do elemento armazenado no contêiner.
constiterator Um iterador constante que aponta para o tipo de elemento armazenado
no contêiner e que pode ser usado somente para ler elementos.
reverseiterator Um iterador reverso que aponta para o tipo de elemento armazenado no
contêiner. Este tipo de iterator é para percorrer um contêiner do fim para o início.
constreverseiterator Um iterador reverso constante para o tipo de elemento armazenado no
contêiner e que pode ser usado somente para ler elementos. Este tipo de iterador é para percorrer o contêiner do fim para o início.
differencetype O tipo do resultado da subtração de dois iteradores que se referem ao
mesmo contêiner (operator- não está definido para iteradores de lists e contêineres associativos).
sizetype O tipo usado para contar itens em um contêiner e indexar ao longo de um contêiner seqüencial (não pode indexar ao longo de uma list).
Fig. 20.4 typedefs comuns encontrados em contêineres de primeira classe.
Dica de desempenho 20.3
_____ A STL geralmente evita herança efunções virtual em favor da programação genérica com gabaritos, para obter um desempenho melhor durante a execução.
Dica de portabilidade 20.1
A STL certamente vai se tornar o meio favorito de programação com contêineres. Programar com a STL
vai aumentar a portabilidade de seu código.

I)ica de desempenho 20.4
Conheça seus componentes da STL. Escolher o contêiner mais apropriado para um determinado problema pode maximizar o desempenho e minimizar os requisitos de memória.
Ao se preparar para usar um contêiner da STL, é importante assegurar-se de que o tipo de elemento sendo armazenado no contêiner suporta um conjunto mínimo de funcionalidades. Quando um elemento é inserido em um contêiner, é feita uma cópia daquele elemento. Por esta razão, o tipo do elemento deve fornecer seu próprio construtor de cópia e operador de atribuição. [Nota: isto é necessário somente se uma cópia membro a membro default não executa uma operação de cópia adequada para o tipo do elementoj. Além disso, os contêineres associativos e diversos algoritmos requerem que os elementos

sejam comparados. Por esta razão, o tipo do elemento deve fornecer um operador de igualdade (==) e um operador menor do que (<).

Observação de engenharia de software 20.3

_____ Os operadores de igualdade e menor do que são tecnicamente desnecessários para os elementos armazenodos em um contêiner a menos que os elementos necessitem ser comparados. Entretanto, quando se cria código a partir de um gabarito, alguns compiladores requerem que todas as partes do gabarito sejam definidas, enquanto outros compiladores requerem somente as partes do gabarito que são realmente usadas no programa.

20.1.2 Introdução a iteradores

Os iteradores têm muitas características em comum com os ponteiros e são usados para apontar para os elementos dos contêineres de primeira classe (e para uns poucos outros propósitos, como veremos). Iteradores armazenam informação sensível aos tipos específicos de contêineres com os quais eles operam, portanto iteradores são implementados apropriadamente para cada tipo de contêiner. Não obstante, certas operações de iteradores são uniformes para todos os contêineres. Por exemplo, o operador derreferenciador (*) derreferencia um iterador de modo que você possa usar o elemento para o qual ele aponta. A operação ++ em um iterador retorna um iterador para o próximo elemento do contêiner (assim como incrementar um ponteiro para um array aponta o ponteiro para o próximo elemento do array).

Os contêineres de primeira classe da STL fornecem funções membro begin () e end () . A função begin O retorna um iterador apontando para o primeiro elemento do contêiner. A função end O retorna um iterador apontando para o primeiro elemento após o final do contêiner (um elemento que não existe). Se um iterador i aponta para um elemento em particular, então ++ i aponta para o próximo elemento e * i refere-se ao elemento apontado por i. O iterador resultante de end () pode ser usado somente em uma comparação quanto à igualdade ou desigualdade, para determinar se o iterador móvel” (i, neste caso) atingiu o fim do contêiner.

Usamos um objeto do tipo iterador para fazer referência a um elemento de contêiner que pode ser modificado. Usamos um objeto do tipo const iterator para fazer referência a um elemento de contêiner que não pode ser modificado.

Usamos iteradores com seqüências (também chamadas de intervalos). Estas seqüências podem estar em contêineres ou podem ser seqüências de entrada ou seqüências de saída. O programa da Fig. 20.5 demonstra a leitura a partir da entrada padrão (uma seqüência de dados de entrada para um programa) usando um istreamiterator e saída para a saída padrão (uma seqüência de dados para saída de um programa) usando um ostream iterator. O programa lê dois inteiros digitados pelo usuário no teclado e mostra a soma dos inteiros. [Nota: nos exemplos neste capítulo, cada uso de uma função da STL e cada definição de um objeto contêiner da STL é precedido pelo prefixo std: : em vez de colocar os comandos using no começo do programa, como mostrado na maioria dos exemplos anteriores. Devido a diferenças nos compiladores e à complexidade do código gerado quando a STL é usada, é difícil se construir um conjunto adequado de comandos using que permita que os programas sejam

compilados sem erros. Para permitir que estes programas sejam compilados na maior variedade de plataformas, optamos pela abordagem com prefixo “std: :“.j

924 C÷+ COMO PROGRAMAR

```
1 II Fig. 20.5: fig2005.cpp
2 II Demonstrando entrada e saída com iteradores.
3 #include <iostreain>
4
5 using std::cout;
6 using std::cin;
7 using std::endl;
8
9 #include <iterator>
10
11 int main()
12
13 cout « “Digite dois inteiros: “;
14
15 std: :istream_iterator< int > inputlnt( cm ),
16 int numberl, nun’iber2;
17
18 nuniberl = *inputlnt; II lê primeiro int da entrada padrão
19 ++inputlnt; II move iterador para próximo valor de entrada
20 nuniber2 = *inputlnt; // lê próximo int da entrada padrão
21
22 cout « “A soma é: “;
23
24 std: :ostreaxniterator< int > outputlnt( cout );
25
26 *outputlnt = numberi + number2; // envia resultado para cout
27 cout « endl;
28 return 0;
29
Digite dois inteiros: 12 25
A soma é: 37
```

Fig. 20.5 Demonstrando iteradores de entrada e saída com streams.

A linha 15

```
std::istreamiterator< int > inputlnt( cm );
```

cria um istream iterator que é capaz de extrair (ler) valores int de um modo seguro quanto ao tipo dc objeto de entrada padrão cm. A linha 18

```
numberi = *inputlnt; // lê primeiro int da entrada padrão
```

desreferencia o iterador inputlnt para ler o primeiro inteiro de cm e atribui aquele inteiro a numberl. Observe o uso do operador derreferenciador * para obter o valor do stream associado com inputlnt; isto é semelhante derreferenciar um ponteiro. A linha 19

```
++inputlnt; // nove iterador para próximo valor de entrada
```

posiciona o iterador inputInt no próximo valor no stream de entrada. A linha 20 number2 = *inputInt; II lê próximo int da entrada padrão

lê o próximo inteiro de inputInt e o atribui a number2.
A linha 24

std::ostreamiterator< int > outputInt(cout);

cria um ostream iterator que é capaz de inserir (enviar) valores int no objeto de saída padrão cout. A linha 26

*outputInt = numberi + number2; // envia resultado para cout

envia um inteiro para cout ao atribuir ao *outputInt a soma de numberl e nuniber2. Note o uso do operador derreferenciador * para usar *outputInt como um Ivalue na instrução de atribuição. Se você quizer produzir outro valor usando outputInt, o iterador deve ser incrementado com ++ (podem ser usados tanto pré-incremento como pós-incremento).

® Dica de teste e depuração 20.2
O operador * (desreferenciador) de qualquer iterador const retorna uma referência const ao elemento do contêiner não permitindo, portanto, o uso de funções membro não-const.

Erro comum de programação 20.1
Tentar desreferenciar um iteradorposicionadofora de seu contêiner é um erro lógico durante a execução.
Especialmente, o iterador retornado por end () não pode ser derreferenciado ou incrementado.

Erro comum de programação 20.2
Tentar criar um iterador não-const para um contêiner const é um erro de sintaxe.

A Fig. 20.6 mostra as categorias de iteradores usados pela STL. Cada categoria fornece um conjunto específico de funcionalidades.

Categoria Descrição

Usado para ler um elemento de um contêiner. Um iterador de entrada pode mover-se somente na direção para a frente (isto é, do início do contêiner para o fim do contêiner), um elemento a cada vez. Interadores de entrada suportam somente algoritmos de um único passo - o mesmo iterador de entrada não pode ser usado para percorrer uma seqüência duas vezes.

Usado para escrever um elemento em um contêiner. Um iterador de saída pode mover-se somente na direção para a frente, um elemento a cada vez. Iteradores de saída suportam somente algoritmos de um único passo - o mesmo iterador de saída não pode ser usado para percorrer uma seqüência duas vezes.

Combina as recursos de iteradores de entrada e de saída e retém sua posição no contêiner (como informação de estado).

Combina os recursos de um iterador para a frente com a habilidade de mover-se na direção para trás (isto é, do final do contêiner em direção ao início do contêiner). Tteradores para a frente suportam algoritmos de múltiplas passagens.

Combina os recursos de um iterador bidirecional com a habilidade de acessar diretamente qualquer elemento do contêiner, isto é, saltar para frente ou para trás um número arbitrário de elementos.

entrada

saída

para a frente

bidirecional

acesso aleatório

Fig. 20.6 Categorias de iteradores.

entrada saída
para a frente
bidirecional
acesso aleatório
Fig. 20.7 Hierarquia das categorias de iteradores.
A Fig. 20.7 ilustra a hierarquia das categorias de iteradores. À medida que você segue a hierarquia de cima parabaixo, cada categoria de iterador suporta toda a funcionalidade das categorias acima dela na figura. Portanto, os tipos mais fracos de iteradores estão no topo e o tipo mais poderoso de iterador está na parte inferior. Note que isto não é uma hierarquia de herança.
A categoria de iterador suportada por cada contêiner determina se aquele contêiner pode ser usado com algoritmos específicos na STL. Contêineres que

suportam iteradores de acesso aleatório podem ser usados com todos os algoritmos na STL. Como veremos, ponteiros para arrays podem ser usados em lugar de iteradores na maioria dos algoritmos STL, incluindo aqueles que necessitam de iteradores de acesso aleatório. A Fig. 20.8 mostra a categoria de iterador suportada por cada um dos contêineres da STL. Observe que somente vectors. deques, lists, sets, multisets, maps e multimaps (isto é, os contêineres de primeira classe) podem ser percorridos com iteradores.

Observação de engenharia de software 20.4

_____ Usar o “iterador mais fraco” que produz um desempenho aceitável ajuda a produzir componentes reutilizáveis ao máximo.

Contêiner	Tipo de iterador suportado
Contêineres seqüenciais	
vector	acesso aleatório
deque	acesso aleatório
list	bidirecional
Contêineres associativos	
set	bidirecional
multiset	bidirecional
map	bidirecional
multimap	bidirecional
Adaptadores de contêineres	
stack	não tem suporte a iteradores
queue	não tem suporte a iteradores
priorityqueue	não tem suporte a iteradores

Fig. 20.8 Tipos de iteradores suportados em cada contêiner da biblioteca padrao.

A Fig. 20.9 mostra os typedefs de iteradores predefinidos que são encontrados nas definições de classes dos contêineres da STL. Nem todo typedef está definido para cada contêiner. Usamos versões const dos iteradores

para percorrer contêineres apenas de leitura. Usamos iteradores reversos para percorrer contêineres do fim para o início.

Fig. 20.9 typedefS de iteradores predefinidos.

Dica de teste e depura ção 20.3

Operações executadas em um cons t iterator retornam refrrências cons t para evitar modificação a elementos do contêiner que está sendo manipulado. Dê preferência ao uso de const iterators em vez de iterators, onde apropriado. Este é um outro exemplo do princípio do menor privilégio.

A Fig. 20.10 mostra as operações que podem ser executadas sobre cada tipo de iterador. Note que as operações para cada tipo de iterador incluem todas as operações precedentes àquele tipo na figura. Note, também, que para os iteradores de entrada e de saída não é possível salvar o iterador e, então, usar o valor salvo mais tarde.

Fig. 20.10 Algumas operações de iteradores para cada tipo de iterador (parte 1 de 2).

typedefS predefinidos para tipos de iteradores	Direção de ++	Recurso ler/escrever
i te rator	para frente	
constiterator	para frente	ler
reverseiterator	para trás	ler/escrever
const_reverse_iterator	para trás	ler

Operação de iterador	Descrição
Todos os iteradores	
++p p++	pré-incrementa um iterador pós-incrementa um iterador
iteradores de entrada	
*p p = p1 p == p1 p ! = pi	derreferencia um iterador para uso como um rvalue atribui um iterador a outro compara iteradores quanto à igualdade compara iteradores quanto à desigualdade
iteradores de saída	
*p p = p1	derreferência um iterador (para uso como um lvalue) atribui um iterador a outro
Iteradores para a frente	iteradores para a frente fornecem todas as funcionalidades tanto dos iteradores de entrada como de saída
Iterudures bidirecionais	
- -p p--	pré-decrementa um iterador pós-decrementa um iterador

Iteradores de acesso aleatório
p += i

p -= i
p+i

p- i

p[i 1 p < pi

p <= pi

p > pi

p >= pi

incrementa o iterador p em i posições
decrementa o iterador p em i posições
fornece como resultado um iterador posicionado em p incrementado em i posições.
fornece como resultado um iterador posicionado em p decrementado em i posições.
retorna uma referência ao elemento distante i posições de p. retorna true se o iterador p for menor do que o iterador p1 (isto é, o iterador p está antes do iterador p1 no contêiner); caso contrário, retorna false. retorna true se o iterador p for menor do que ou igual ao iterador p1 (isto é, o iterador está antes do iterador p1 ou está no mesmo lugar que o iterador p1 no contêiner); caso contrário, retorna false.
retorna true se o iterador p for maior do que o iterador p1 (isto é, o iterador p está após o iterador p1 no contêiner); caso contrário, retorna false. retorna true se o iterador p for maior ou igual ao iterador p1 (isto é, o iterador p está após o iterador p1, ou está no mesmo lugar que o iterador p1 no contêiner); caso contrário, retorna false.

Fig. 20.10 Algumas operações de iteradores para cada tipo de iterador (parte 2 de 2).

20.1.3 Introdução a algoritmos

Um aspecto crucial da STL é que ela fornece algoritmos que podem ser usados genericamente para vários tipos de contêineres. A STL fornece muitos algoritmos que você usará freqüentemente para manipular contêineres. Inserir, deletar, procurar, classificar e outras mais são ações apropriadas para alguns ou todos os contêineres da

STL.

A STL inclui aproximadamente 70 algoritmos padrão. Fornecemos exemplos de código real da maioria destes e resumimos os outros em tabelas. Os algoritmos operam sobre os elementos de contêineres somente indiretamente, através de iteradores. Muitos algoritmos operam em seqüências de elementos definidas por pares de iteradores - um primeiro iterador apontando para o primeiro elemento da seqüência e um segundo apontando para um elemento imediatamente após o último elemento da seqüência. Além disso, é possível criar seus próprios algoritmos novos, que operem de maneira similar, de modo que possam ser usados com os contêineres e iteradores da STL.

A função membro de contêiner begin () retorna um iterador para o primeiro elemento de um contêiner; end () retorna um iterador para a primeira posição além do último elemento de um contêiner. Algoritmos freqüentemente retornam iteradores.
Um algoritmo como f ind () , por exemplo, localiza um elemento e retorna um iterador para aquele elemento. Se o elemento não for encontrado, f ind () retorna o iterador end , que pode ser testado para determinar se um elemento não foi encontrado (o retorno de end O supõe uma pesquisa em todo o contêiner). O algoritmo f ind O pode ser usado com qualquer contêiner da STL.

Observação de engenharia de software 20.5

A STL é implementada concisamente. Até agora, projetistas de classes teriam associado os algoritmos aos contêineres tornando os algoritmos funções membro dos contêineres. A STL adota uma abordagem diferente. Os algoritmos estão separados dos contêineres e operam sobre os elementos dos contêineres só indiretamente, através de iteradores. Esta separação torna mais fácil escrever algoritmos genéricos aplicáveis a muitas outras classes de contêineres.

Os algoritmos da STL criam mais uma oportunidade para reutilização. Usando a rica coleção de algoritmos populares, os programadores podem economizar muito tempo e esforço.
Se um algoritmo usa iteradores menos poderosos, ele também pode ser usado com contêineres que suportam
iteradores mais poderosos. Alguns algoritmos exigem iteradores poderosos; por exemplo, sort exige iteradores de acesso aleatório.
Observação de engenharia de software 20.6
_____ A STL é extensível. É simples adicionar a ela novos algoritmos e fazer isso sem mudanças nos contêineres da STL.
Observação de engenharia de software 20.7
_____ Os algoritmos podem operar sobre contêineres da STL e sobre arrays baseados em ponteiros, no estilo usado em C.
Dica de portabilidade 20.2
_____ Devido aos algoritmos da STL processarem os contêineres só indiretamente, através dos iteradores, um algoritmo pode freqüentemente ser

usado com muitos contêineres diferentes.
A Fig. 20.11 mostra muitos dos algoritmos seqtienciais mutantes - isto é, os algoritmos que resultam em modificações dos contêineres sobre os quais os algoritmos são aplicados.

Fig. 20.11 Algoritmos seqüenciais mutantes
A Fig. 20.12 mostra muitos dos algoritmos seqüenciais não-mutantes - isto é, os algoritmos que não resultam em modificações dos contêineres sobre os quais os algoritmos são aplicados.

Fig. 20.12 Algoritmos seqüenciais não-mutantes.
A Fig. 20.13 mostra os algoritmos numéricos do arquivo de cabeçalho <numeric>.

1 Algoritmos seqüenciai	s mutantes	1
copy()	remove O	reversecopy()
copybackward O	remove copy O	rotate O
filiO	remove copyif (1	rotate_copy()
fiii_n ()	removeif O	stabieyartition O
generate()	repiace()	swap()
generate_n ()	replacecopy ()	swap ranges O
iterswap ()	repiace_copyif ()	transform O
partition()	repiace_if()	unique()
randomshuffie O	reverse O	uniquecopy ()

1 Algoritmos seqüenciais	não-mutantes	1 find_if O mismatch() search()
adjacentfind() count() countif ()	findQ findeach() findend()	
equal()	find_first_of()	search_n()

Algoritmos numéricos do arquivo de cabeçalho <numeric>
accumulate O
inner_product ()

partialsum()
adjacentdifference O
Fig. 20.13 Algoritmos numéricos do arquivo de cabeçalho <numeric>

20.2 Contêineres seqüenciais

A biblioteca padrão de gabaritos de C++ fornece três contêineres seqüenciais - vector, list e deque. A classe vector e a classe deque são ambas baseadas em arrays. A classe list implementa uma estrutura de dados de lista encadeada similar à nossa classe List apresentada no Capítulo 15, mas mais robusta.

Um dos contêineres mais populares na STL é vector. A classe vector é um refinamento da classe Array "meio inteligente" que criamos no Capítulo 8. Um vector pode mudar de tamanho dinamicamente. Diferentemente dos arrays "crus" de C e C++ (ver Capítulo 4), os vectors podem ser atribuídos um ao outro. Isto não é possível com os arrays no estilo usado em C, baseados em ponteiros, porque os nomes daqueles arrays são ponteiros constantes e não podem ser o destino em atribuições. Assim como com os arrays C, a indexação de vector não executa a verificação automática de limites, mas a classe vector fornece este recurso (como veremos adiante) através da função membro at.

Dica de desempenho 20.5

A inserção no fim de um vector é eficiente. O vector simplesmente cresce, se necessário, para acomodar o novo item. E dispendioso inserir (ou retirar) um elemento no meio de um vector - a porção inteira do vector após o ponto de inserção (ou retirada) deve ser movida, porque elementos vector ocupam posições contíguas na memória, assim como um array "cru" em C ou C++.

A Fig. 20.2 apresenta as operações comuns a todos os contêineres da STL. Além destas operações, cada contêiner fornece tipicamente diversos outros recursos. Muitos destes recursos são comuns a diversos contêineres. Entretanto, estas operações nem sempre são igualmente eficientes para cada contêiner. Os programadores precisam então, freqüentemente, escolher o contêiner mais apropriado às suas aplicações.

Dica de desempenho 20.6

Aplicações que requeiram freqüentes inserções e dele ções em ambas as extremidades de um contêiner normalmente usam uma deque de preferência a um vector. Embora possamos inserir e deletar elementos na frente e atrás tanto de um vector como de uma deque, a classe deque é mais eficiente do que vector para fazer inserções e deleções no início.

Dica de desempenho 20.7

'_ Aplicações com freqüentes inserções e dele ções no meio e/ou nos extremos de um contêiner normalmente usam uma list, devido à sua eficiente implementação de inserção e deleção em qualquer lugar na estrutura de dados.

Além das operações em comum descritas na Fig. 20.2, os contêineres seqüenciais têm diversas outras operações em comum - front para retornar uma referência ao primeiro elemento no contêiner, back para retornar uma referência ao último elemento no contêiner, push_back para inserir um novo elemento no final do contêiner e pop_back para remover o último elemento do contêiner.

20.2.1 O coritêiner seqüencial vector

A classe vector fornece uma estrutura de dados com posições de memória contíguas. Isto viabiliza o acesso direto
e eficiente a qualquer elemento de um vector via o operador subscrito [],exatamente como um array “bruto” em
C ou C++. A classe vector é mais comumente usada quando os dados no contêiner devem ser ordenados e facilmente acessíveis via um subscrito. Quando a memória de um vector está esgotada, o vector aloca uma área de memória maior em uma área contígua, copia os elementos originais para a nova memória e libera a memória antiga.

Dica de desempenho 20.8

_____ Escolha o contêiner vector para obter o melhor desempenho com acesso aleatório.

Dica de desempenho 20.9

f Objetos da classe vector fornecem acesso indexado rápido com o operador subscrito sobrecarregado [J porque eles estão armazenados em uma área contígua, como um array “bruto” em C ou C+ +.

Dica de desempenho 20.10

t É muito mais rápido inserir muitos elementos de uma só vez do que um elemento de cada vez.

Uma parte importante de todo contêiner é o tipo de iterador que ele suporta. Isto determina quais algoritmos podem ser aplicados ao contêiner. Um vector suporta iteradores de acesso aleatório - isto é, todas as operações sobre contêineres mostradas na Fig. 20.10 podem ser aplicadas a um iterador de vector. Todos os algoritmos podem operar sobre um vector. Os iteradores para um vector são normalmente implementados como ponteiros para os elementos do vector. Cada um dos algoritmos da STL que aceitam iteradores como argumentos requerem que estes iteradores forneçam um nível mínimo de funcionalidade. Se um algoritmo requer um iterador para a frente, por exemplo, tal algoritmo pode operar em qualquer contêiner que forneça iteradores para a frente, iteradores bidirecionais ou iteradores de acesso aleatório. Desde que o contêiner suporte a funcionalidade mínima do iterador do algoritmo, o algoritmo pode operar sobre o contêiner.

A Fig. 20.14 ilustra diversas funções do gabarito de classe vector. Muitas destas funções estão disponíveis
em todos os contêineres de primeira classe da biblioteca padrão. Você deve incluir o arquivo de cabeçalho <vector> para usar a classe vector.

```
1 // Fig. 20.14: fig2Ol4.cpp
2 // Testando o gabarito de classe vector da Biblioteca Padrão
3 #include <iostream>
4
5 using std: :cout;
6 using std: :cin;
7 using std::endl;
8
```

```
9 #include <vector>
10
11 template < class T >
12 void printVector( const std: :vector< i vec
13
14 int main()
15 {
16 const int SIZE = 6;
17 inta[SIZE]={1,2,3,4,5,6};
18 std: :vector< int > v;
19
20 cout « ‘O tamanho inicial de v é: « v.size()
```

Fig. 20.14 Demonstrando o gabarito de classe vector da biblioteca padrão (parte 1 de 2).


```
21 « “\nA capacidade inicial de v é: « v.capacity
22 v.push_back( 2 ); II o método push_back() existe em
23 v.push_back( 3 ); II todas as coleções seqüenciais
24 v.push_back( 4 );
25 cout « “\nO tamanho de v é: « v.size()
26 « “\nA capacidade de v é: « v.capacity
27 cout « “\n\nConteúdo do array a usando notação de ponteiro:
28
29 for ( int *ptr = a; ptr != a + SIZE; ++ptr
30 cout « *ptr « ‘
31
32 cout « “\nConteúdo do vector v usando notação de iterador: “;
33 printVector( v
34
35 cout « “\nConteúdo do vector v invertido: “;
36
37 std: :vector< int >: :reverse_iterator p2;
38
39 for ( p2 = v.rbeginO; p2 != v.rend(); ++p2
40 cout « *p2 «
41
42 cout « endl;
43 return 0;
44
45
46 template < class T >
47 void printVector( const std: :vector< T > &vec
48
49 std: :vector< T >: :const_iterator p1;
```

50
51 for (p1 = vec.beginO; p1 != vec.endO; pl++
52 cout « *p1 «
53
O tamanho inicial de v é: O
A capacidade inicial de v é: O
O tamanho de v é: 3
A capacidade de v é: 4
Conteúdo do array a usando notação de ponteiro: 1 2 3 4 5 6
Conteúdo do vector v usando notação de iterador: 2 3 4
Conteúdo do vector v invertido: 4 3 2
Fig. 20.14 Demonstrando o gabarito de classe vector da biblioteca padrão (parte 2 de 2).
A linha 18
std: :vector< int > v;
declara uma instância da classe vector chamada v, que armazena valores int. Quando este objeto é instanciad é criado um vector vazio com tamanho (isto é, o número de elementos armazenados no vector) igual a O capacidade (isto é, o número de elementos que podem ser armazenados sem alocar mais memória para o vectoz também igual a O.
As linhas 20 e 21

cout « “O tamanho inicial de v é: “ « v.size()
« “\nA capacidade inicial de v é: “ « v.capacity

demonstram as funções size e capacity, as quais inicialmente retornam O para o vector v, neste exemplo. A função size disponível em todos os contêineres - retorna o número de elementos atualmente armazenados no contêiner. A função capacity retorna o número de elementos que podem ser armazenados no vector antes que o vector se redimensione dinamicamente para acomodar mais elementos.
As linhas 22 até 24
v.pushback(2); II o método push_back() existe em
v.pushback(3); II todas as coleções seqüenciais
v.pushback(4);
usam a função push_back - disponível em todos os contêineres seqüenciais - para adicionar um elemento ao final do vector. Se um elemento é adicionado a um vector cheio, o vector aumenta seu tamanho - algumas implementações da STL fazem o vector dobrar seu tamanho.
1 Dica de desempenho 20.11
_____ Pode ser um desperdício dobrar o tamanho de um vector quando é necessário mais espaço. Por exemplo, um vector com um total de 1.000.000 elementos se redimensiona para acomodar 2.000.000 elementos quando é adicionado um novo elemento. Isto deixa 999.999 elementos sem uso. Os

programadores podem usar resize () para controlar melhoro uso do espaço.
As linhas 25 e 26 usam size e capacity para ilustrar o novo tamanho e capacidade do vector após as operações de push back. A função size retorna 3 - o número de elementos acrescentados ao vector. A função capacity retorna 4 indicando que podemos acrescentar mais um elemento sem alocar mais memória para o vector. Quando adicionamos o primeiro elemento, o tamanho de v tornou-se 1 e a capacidade de v tornou-se 1. Quando adicionamos o segundo elemento, o tamanho de v tornou-se 2 e a capacidade de v tornou-se 2. Quando adicionamos o terceiro elemento, o tamanho de v tornou-se 3 e a capacidade de v tornou-se 4. Se adicionarmos mais dois elementos, o tamanho de v será 5 e a capacidade será 8. A capacidade dobra toda vez que o total do espaço alocado para o vector está cheio e outro elemento é adicionado.
As linhas 29 e 30 demonstram como enviar para a saída o conteúdo de um array usando ponteiros e aritmética de ponteiros. A linha 33 busca a função printVector para produzir os conteúdos de um vector usando iteradores. A definição do gabarito de função printVector começa na linha 46. A função recebe uma referência const para um vector como seu argumento. A linha 49
std: :vector< T >: :const iterator p1;
declara um const_iterator chamado p1 que itera através do vector e envia seu conteúdo para a saída. Um const vector habilita o programa a ler os elementos do vector, mas não permite ao programa modificar os elementos. A estrutura for nas linhas 51 e 52
for (p1 = vec.beginO; p1 != vec.end(); pl++
cout « *p1 «
inicializa p1 usando a função membro begin de vector que retorna um const_iterator para o primeiro elemento no vector (há uma outra versão de begin que retorna um iterator que pode ser usado para contêineres não-const). O laço continua enquanto p1 não tenha ultrapassado o final de vector. Isto é determinado ao comparar p1 com o resultado de vec . end () que retorna um const_iterator (como com begin. há uma outra versão de end que retorna um iterator) indicando a localização após o último elemento do vector. Se p1 é igual a este valor, o fim do vector foi atingido. As funções begin e end estão disponíveis a todos os contêineres de primeira classe. O corpo do laço derreferencia o iterador p1 para conseguir o valor que está armazenado no elemento corrente (apontado pelo iterador) do vector. A expressão pl++ posiciona o iterador no próximo elemento do vector.
® Dica de teste e depura ção 20.4
Somente iteradores de acesso aleatório suportam <. É melhor usar ! = e end O para testar o fim do contêiner

A linha 37
declara um reverse_iterator que pode ser usado para percorrer um vector do fim para o início. Todos os contêineres de primeira classe suportam este tipo de iterador.

As linhas 39 e 40
for (p2 = v.rbeginO; p2 != v.rendO; ++p2
cout « *p2 «
usam uma estrutura for similar àquela na função printVector para percorrer o vector. Neste laço, as funções rbegin (isto é, o iterador para o ponto inicial para percorrer o contêiner do fim para o início) e rend (isto é, o iterador para o ponto final para percorrer o contêiner do fim para o início) delimitam o intervalo de elementos para saída do fim para o início. Como as funções begin e end, rbegin e rend podem retornar um const_reverse_iterator ou um reverse iterator baseadas em se o contêiner é ou não constante.
A Fig. 20.15 ilustra funções que permitem a recuperação e manipulação dos elementos de um vector. A linha 16
std::vector< int > v(a, a + SIZE);
usa um construdctor vector sobrecarregado que aceita dois iteradores como argumentos. Lembre-se de que p011- teiros para um array podem ser usados como iteradores. Esta declaração cria integer vector v e o inicializa com o conteúdo do array integer a da posição a até - mas não incluindo - a posição a + SIZE.

```
1 II Fig. 20.15: fig2Ol5.cpp
2 // Testando as funções de manipulação de elementos
3 // do gabarito de classe vector da Biblioteca Padrão
4 #include <iostream>
5
6 using std::cout;
7 using std::endl;
8
9 #include <vector>
10 #include <algorithm>
li
12 int main()
13 {
14 const int SIZE = 6;
15 inta[SIZE]={l,2,3,4,5,6};
16 std: :vector< int > v( a, a + SIZE );
17 std: :ostreamiterator< int > output( cout,
18 cout « “O vector v contém:
19 std: :copy( v.beginO), v.endO), output );
20
21 cout « “\nPrimeiro elemento de v: “ « v.frontO
22 « “\Último elemento de v: “ « v.backO;
23
24 v[ O ] = 7; II atribui 7 ao primeiro elemento
25 v.at( 2 ) = 10; II atribui 10 ao elemento na posição 2
26 v.insert( v.begin() + 1, 22 ); // insere 22 como segundo elemento
27 cout « “\nConteúdo do vector v após mudanças: “;
28 std::copy( v.beginO, v.endO, output );
```

```
29
30 try{

std: :vector< int >: :reverse_iterator p2;
```

Fig. 20.15 Demonstrando as funções de manipulação de elementos do gabarito de classe vector da
biblioteca padrão (parte 1 de 2).


```
31 v.at( 100 ) = 777; /1 acessa elemento fora do intervalo válido
32
33 catch ( std: :out_of_range e
34 cout « “\nExceção: “ « e.what();
35
36
37 v.erase( v.begin() );
38 cout « “\nConteúdo do vector v após erase: “;
39 std: :copy( v.beginQ, v.endO, output );
40 v.erase( v.beginO, v.end() );
41 cout « “\nApós erase, o vector v
42 « ( v.empty() ? “está” : “não está” ) « “ vazio”;
43
44 v.insert( v.beginQ, a, a + SIZE );
45 cout « “\nConteúdo do vector v antes de clear: “;
46 std: :copy( v.beginO, v.endO, output );
47 v.clearO; 1/ clear chama erase para esvaziar uma coleção
48 cout « “\nApós clear, o vector v
49 « ( v.empty() ? “está” : “não está” ) « “ vazio”;
48
51 cout « endl;
52 return 0;
53
```

Fig. 20.15 Demonstrando as funções de manipulação de elementos do gabarito de classe vector da biblioteca padrão (parte 2 de 2).

A linha 17

```
std: :ostreamiterator< int > output( cout, “ “
```

declara um ostream iterator chamado output que pode ser usado para enviar para a saída via cout valores inteiros separados por espaços simples. Um ostream iterator é um mecanismo de saída seguro quanto ao tipo - que vai produzir somente valores do tipo int ou de um tipo compatível. O primeiro argumento para o construtor especifica o stream de saída e o segundo argumento é um string especificando caracteres separadores para os valores de

saída - neste caso um caractere espaço. Usaremos o ostream iterator para enviar o conteúdo do vector para a saída neste exemplo.
A linha 19
std: :copy(v.beginO, v.endO, output);
usa o algoritmo copy da biblioteca padrão para enviar o conteúdo de vector v para a saída padrão. O algoritmo copy copia cada elemento do contêiner começando com a posição especificada pelo iterador em seu primeiro argumento e até - mas não incluindo - a posição especificada pelo iterador em seu segundo argumento. O primeiro e segundo argumentos precisam satisfazer aos requisitos de iteradores de entrada - isto é, eles devem ser iteradores através dos quais valores podem ser lidos de um contêiner. Além disso, a aplicação de ++ ao primeiro iterador deve, em algum momento, fazer com que o primeiro iterador alcance o segundo argumento iterador no contêiner. Os elementos são copiados para a posição especificada pelo iterador de saída (isto é, um iterador através do qual um

fr

O vector v contém: 1 2 3 4 5 6			
Primeiro elemento de v: 1			
Ultimo elemento de v: 6			
Conteúdo do vector v após mudanças: 7	22 2 10 4	5	6
Exceção: invalid vector<T> subscript			
Conteúdo do vector v após erase: 22 2	10 4 5 6		
Após erase, o vector v está vazio			
Conteúdo do vector v antes de clear: 1 2 3 4 5 Após clear, o vector v está vazio		6	1

valor pode ser armazenado ou enviado para a saída) especificado como o último argumento. Neste caso, o iterad(de saída é o ostream iterator output que está vinvulado a cout, de modo que os elementos são copiadc para a saída padrão. Para usar os algoritmos da biblioteca padrão, você precisa incluir o arquivo de cabeçalh <algorithm>.
As linhas 21 e 22 usam as funções front e back (disponíveis para todos os contêineres seqüenciais) par
determinar o primeiro e o último elementos do vector, respectivamente.
Erro comum de programação 20.3
O vector não deve estar vazio; caso contrário, os resultados das fim ções front e back são indefinido
As linhas 24 e 25
v[O 1 = 7; II atribui 7 ao primeiro elemento
v.at(2) = 10; // atribui 10 ao elemento na posição 2
ilustram dois modos de usar subscritos para percorrer um vector (estes modos

também podem ser usados com o contêineres deque). A linha 24 usa o operador subscrito que é sobrecarregado para retornar ou uma referência a valor na posição especificada ou uma referência constante àquele valor, dependendo de se o contêiner é constante oi não. A função at executa a mesma operação com um recurso adicional - verificar os limites. A função at primem verifica o valor fornecido como um argumento e determina se ele está dentro dos limites do vector. Se não estiver a função at dispara uma exceção out_of_bounds (como demostrado nas linhas 30 a 35). Alguns dos tipos d exceção da STL são mostrados na Fig. 20.16 (os tipos de exceção são comentados no capítulo 13, “Tratamento d exceções”).

Fig. 20.16 Tipos de exceções da STL.

A linha 26

v.insert(v.begin() + 1, 22); II insere 22 como segundo elemento

usa uma das três funções insert que estão disponíveis para todo contêiner seqüencia!. A declaração anterio insere o valor 22 antes do elemento na posição especificada pelo iterador no primeiro argumento. No exemplo, iterador está apontando para o segundo elemento do vector, então 22 é inserido como o segundo elemento e segundo elemento original torna-se o terceiro elemento do vector. As outras versões de insert permitem inseri múltiplas cópias do mesmo valor começando por uma posição particular no contêiner ou inserir uma faixa de valore de um outro contêiner (ou array começando em uma posição específica no contêiner original.

As linhas 37 e 40

v.erase(v. begin());

v.erase(v. begin, v.end());

[Tipos de exceções da STL	Descrição
out_of_range	Indica quando o subscrito está tora do intervalo-por exemplo, quan do um subscrito inválido é especificado para a função membro ai de vector.
invalidargument	Indica que um argumento inválido foi passado para uma função
lengtherror	Indica uma tentativa de criar um contêiner muito longo. string etc.
badalloc	Indica que uma tentativa de alocar memória com new (ou com un alocador) falhou porque não havia memória suficiente disponível.

usam as duas funções erase que estão disponíveis em todos os contêineres de primeira classe. A linha 37 indica que o elemento na posição especificada pelo argumento iterador deveria ser removida do contêlner (neste exemplo o

elemento no início do vector). A linha 40 especifica que todos os elementos no intervalo começando com a posição do primeiro argumento até - mas não incluindo - a posição do segundo argumento devem ser apagados do contêiner. Neste exemplo, todos os elementos são apagados do vector. A linha 42 usa a função empty (disponível para todos os contêineres, incluindo os adaptadores) para confirmar que o vector está vazio.
Erro comum de programação 20.4
Apagar um elemento que contém um ponteiro para um objeto alocado dinamicamente não deleta o objeto.

A linha 44

v.insert(v. beginO, a, a + SIZE);

usa a versão da função insert que usa o segundo e terceiro argumentos para especificar a posição inicial e a posição final de uma seqüência de valores (possivelmente de outro contêiner, mas neste caso do array integer a) que deve ser inserida no vector. Lembre-se de que a posição final especifica a posição na seqüência após o último elemento ter sido inserido; a cópia é executada até - mas não incluindo - esta posição.
Finalmente, a linha 47
v.clearo; II clear chama erase para esvaziar um contêiner

usa a função clear (disponível em todos os contêineres de primeira classe) para esvaziar o vector. Esta função chama a versão de erase usada na linha 40, que é quem realmente executa a operação.
INota: há outras funções que são comuns a todos os contêineres e comuns a todos os contêineres seqüenciais que ainda não ftram cobertas. Abordaremos a maior parte destas funções nas próximas seções. Também abordaremos muitas funções que são específicas para cada contêiner.j
20.2.2 O contêiner seqüencial list
O contêiner seqüencial list fornece uma implementação eficiente para operações de inserção e deleção em qualquer posição no contêiner. Se a maioria das inserções e deleções ocorrem nas extremidades do contêiner, a estrutura de dados deque (Seção 20.2.3) fornece uma implementação mais eficiente. A classe list é implementada como uma lista duplamente encadeada - isto é, cada nodo em list contém um ponteiro para o nodo anterior em list e um para o nodo seguinte em list. Isto habilita a classe list a suportar iteradores bidirecionais que permitem que o contêiner seja percorrido de ambos os modos, para frente e para trás. Qualquer algoritmo que requeira iteradores de entrada, saída, para frente e bidirecionais pode operar sobre uma list. Muitas das funções membro manipulam os elementos do contêiner como um conjunto ordenado de elementos.
Além das funções membro de todos os contêineres da STL na fig. 20.2 e as funções membro comuns de todos os contêineres seqüenciais discutidas na Seção 20.5, a classe list fornece outras oito funções membro - splice. pushfront,

pop front, remove. unique, merge, reverse e sort. A Fig. 20.17 demonstra diversos recursos da classe list. Lembre-se de que muitas das funções apresentadas nas Figs. 20.14 e 20.15 podem ser usadas com a classe list. O arquivo de cabeçalho <list> deve ser incluído para usar a classe list.

Fig. 20.17 Demonstrando o gabarito de classe list da biblioteca padrão (parte 1 de 3).

1	// Fig. 20.17: fig2Ol7.cpp		
2	// Testando a classe	list	da Biblioteca Padrão
3	#include <iostream>		
4			
5	using std: :cout;		
6	using std::endl;		


```
7
8 #include <list>
9 #include <algorithm>
10
11 template < class T >
12 void printList( const std::list< T > &J.istRef );
13
14 int main()
15
16 const jnt SIZE = 4;
17 int a[ SIZE ) = { 2, 6, 4, 8 };
18 std::list< int > values, otherValues;
19
20 values.pushfront( 1 );
21 values.pushfront( 2 );
22 values.pushback( 4 );
23 values.pushback( 3 );
24
25 cout « ‘values contém: “;
26 printList( values );
27 values.sortQ;
28 cout « “\nvalues após sort contém: “;
```

```
29 printList( values );
30
31 otherValues.insert( otherValues.beginO), a, a + SIZE );
32 cout « “\notherValues contém: “;
33 printList( otherValues );
34 values.splice( values.endO, otherValues );
35 cout « “\nApós splice values contém:
36 printList( values );
37
38 values.sortQ;
39 cout « “\nvalues contém:
40 printList( values );
41 otherValues.insert( otherValues.beginO, a, a + SIZE );
42 otherValues.sortO;
43 cout « “\notherValues contém: “;
44 printList( otherValues );
45 values.merge( otherValues
46 cout « “\nApós merge:\n values contém:
47 printList( values );
48 cout « “\n otherValues contém: “;
49 printList( otherValues );
50
51 va1ues.popfront
52 va1ues.popback II todos os contêineres seqüenciais
53 cout « “\nApós pop_front e pop back values contém:\n”;
54 printList( values );
55
56 values.uniqueQ;
57 cout « “\nApós unique values contém: “;
58 printList( values );
59
60 // o método swap está disponível em todos os contêineres
61 values.swap( otherValues );
62 cout « “\nAfter swap:\n values contém:
63 printList( values );
64 cout « “\n otherValues contém:
```

Fig. 20.17 Demonstrando o gabarito de classe list da biblioteca padrão (parte 2 de 3).


```
65 printList( otherValues );
66
67 values. assign( otherValues.beginO, otherValues. end() );
68 cout « “\nApós assign values contém: “;
69 printList( values );
70
```

```
71 values.merge( otherValues );
72 cout « “\nvalues contém: “;
73 printList( values );
74 values.remove( 4 );
75 cout « “\nApós remove( 4 ) values contém: “;
76 printList( values );
77 cout « endi;
78 return O;
79 }
80
81 template < class T >
82 void printList( const std: :list< T > &listRef
83
84 if ( listRef.empty()
85 cout « “lista está vazia’;
86 else {
87 std: :ostream_iterator< T > output( cout, “ “
88 std::copy( 1istRef.begin, listRef.endQ, output );
89 }
90 }
```

```
values contém: 2 1 4 3
values após sort contém: 1 2 3 4
otherValues contém: 2 6 4 8
Após splice values contém: 1 2 3 4 2 6 4 8
values contém: 1 2 2 3 4 4 6 8
otherValues contém: 2 4 6 8
Após merge:
valuescontém: 122234446688
otherValues contém: lista está vazia
Após pop_front e pop_back values contém:
2223444668
Após unique values contém: 2 3 4 6 8
Após swap:
values contém: lista está vazia
otherValues contém: 2 3 4 6 8
Após atribuição values contém: 2 3 4 6 8
values contém: 2 2 3 3 4 4 6 6 8 8
Após remove( 4 ) values contém: 2 2 3 3 6 6 8 8
```

Fig. 20.17 Demonstrando o gabarito de classe list da biblioteca padrão (parte 3 de 3).

A linha 18

```
std::list< int > values, otherValues;
```

instancia dois objetos list capazes de armazenar inteiros. As linhas 20 e 21 usam a função push_front paa inserir inteiros no início de values. A função push front é específica para classes list e deque (não para vector). As linhas 22 e 23 usam função push_back para inserir inteiros no final de values. Lembre-se de que a

função push_back é comum a todos os contêineres seqüenciais.

940 C++ COMO PROGRAMAR
A linha 27
values.sortO;
usa a função membro sort de list para organizar os elementos na list em ordem ascendente. [Nota: isto é diferente do sort nos algoritmos da STL.] Há uma segunda versão da função sort que permite ao programador fornecer uma função predicado binário que recebe dois argumentos (valores na lista), executa uma comparação e retorna um valor bool indicando o resultado. Esta função determina a ordem na qual os elementos da list são classificados. Esta versão poderia ser particularmente útil para uma list que armazena ponteiros em vez de valores. [Nota: demonstramos uma função predicado unária na Fig. 20.28. Uma função predicado unária recebe um único argumento, executa uma comparação usando aquele argumento e retorna um valor bool indicando o resultado. 1
A linha 34
values.splice(values.end() , otherValues);
usa a função splice de list para remover os elementos de otherValues e inseri-los em values antes da posição do iterador especificado como o primeiro argumento. Há outras duas versões desta função. A função splice com três argumentos permite que um elemento seja removido do contêiner especificado como segundo argumento a partir da posição especificada pelo iterador no terceiro elemento. A função splice com quatro argumentos usa os dois últimos argumentos para especificar uma faixa de posições que deveriam ser removidas do contêiner no segundo argumento e colocadas na posição especificada no primeiro argumento.
Após inserir mais elementos na list otherValues e classificar tanto values como otherValues, a
linha 45
values.merge(otherValues);
usa a função membro merge de list para remover todos os elementos de otherValues e inseri-los em ordem classificada em values. Ambas as lists devem estar classificadas na mesma ordem antes que esta operação seja executada. Uma segunda versão de merge permite que o programador forneça uma função predicado que recebe dois argumentos (valores na lista) e retoma um valor bool. A função predicado especifica a ordem de classificação usada por merge.
A linha 51 usa a função pop front de list para remover o primeiro elemento da list. A linha 52 usa
a função pop back (disponível a todos os contêineres seqüenciais) para remover o último elemento da list. A linha 56
values.unique()
usa a função unique de list para remover elementos duplicados da list. A list deve estar em ordem classificada (de modo que todos os duplicados estejam lado a lado) antes que esta operação seja executada, para garantir que todos os duplicados sejam eliminados. Uma segunda versão de unique permite que o programador forneça uma função predicado que recebe dois argumentos

(valores no list) e retorna um valor bool. A função predicado especifica se dois elementos são iguais.
A linha 61
values. swap (otherValues);
usa a função swap (disponível para todos os contêineres) para trocar os conteúdos de values com os conteúdos de o therValue s.
A linha 67
values.assign(otherVa1ues.begin, otherValues.end());


```
10
11 int main()
12
13 std: :deque< double > values;
14 std::ostreamiterator< double > output( cout,
15
16 values.pushfront( 2.2 );
17 values.pushfront( 3.5 );
18 values.pushback( 1.1 );
19
20 cout « values contém:
21
22 for ( int i o; i < values.sjzeO; ++i
23 cout « values[ i ) «
24
25 va1ues.pop_front
26 cout « \riApós pop_front values contém:
27 std::copy ( values.begmnO, values.endO, output );
28
29 values[ 1 ] = 5.4;
30 cout « '\nApós values[ 1 ) = 5.4 values contém: ";
31 std::copy ( values.beginO), values.endO, output );
32 cout « endi;
33 return 0;
34 }
```

```
vaJ.ues contém: 3.5 2.2 1.1
Após pop_front values contém: 2.2 1.1
Após values[ 1 3 = 5.4 values contém: 2.2 5.4
```

Fig. 20.18 Demonstrando o gabarito de classe deque da biblioteca padrão (parte 2 de 2).
Alinha 13
std: :deque< double > values;
instancia uma deque que pode armazenar valores double. As linhas 16 a 18 usam as funções push_front e push back para inserir elementos no início e no final da deque. respectivamente. Lembre-se de que push_back está dRponível a

todos os contêineres seqilenciais, mas push_front está disponível somente para a classe list e a classe deque.
A estrutura for na linha 22
for (int i = 0; i < values.sizeQ; ++i
cout « values[i 3 «
usa o operador subscrito para recuperar o valor em cada elemento do deque para saida. Note ouso da função size na condição para assegurar que não tentaremos acessar um elemento fora dos limites do deque.
A linha 25 usa função pop_front para demonstrar a retirada do primeiro elemento do deque. Lembre-se
de que pop front está disponível somente para classe list e a classe deque (não para classe vector).
A linha 29
values[1 3 = 5.4;
usa o operador subscrito para criar um Ivalue. Isso permite que sejam atribuídos valores diretamente a qualquer elemento do deque.

20.3 Contêineres associativos
Os contêineres associativos da STL destinam-se a fornecer acesso direto para armazenar e recuperar elementos via chaves (freqüentemente chamadas de chaves de pesquisa). Os quatro contêineres associativos são multiset, set. multimap e xnap. Em cada contêiner, as chaves são mantidas em ordem classificada. herar através de um contêiner associativo percorre o mesmo em uma ordem classificada para aquele contêiner. As classes multiset e set fornecem operações para manipular conjuntos de valores nos quais os próprios valores são as chaves - isto é. não há um valor separado associado a cada chave. A principal diferença entre um multiset e um set é que um multiset permite chaves duplicadas e um set não permite. As classes multimap e inap fornecem operações para manipular valores associados com chaves (estes valores às vezes são referidos como valores mapeados). A principal diferença entre um multimap e um map é que um multimap permite chaves duplicadas com valores associados a serem armazenados e um map permite somente chaves únicas com valores associados. Em adição às funções membro comuns a todos os contêineres apresentados na Fig. 20.2, todos os contêineres associativos também suportam diversas outras funções membro, incluindo find, lower bound, upper bound e count. Exemplos de cada contêiner associativo e das funções membro comuns aos contêineres associativos são apresentados nas próximas subseções.
20.3.1 O contêiner associativo multiset
O contêiner associativo muitiset oferece armazenagem e recuperação rápidas de chaves. Um multiset permite chaves duplicadas. A ordenação dos elementos é determinada por um objeto função de comparação. Por exemplo, em um mui tiset de inteiros, os elementos podem ser ordenados em ordem crescente ordenando-se as chaves com o objeto função de comparação iess< int>. O tipo de dado das chaves em todos os contêineres associativos deve suportar

adequadamente a comparação, com base no objeto função de comparação especificado - as chaves ordenadas com less< int > devem suportar a comparação com operator<. Se as chaves usadas em contêineres associativos são de tipos definidos pelo programador, estes tipos devem prover os operadores de comparação apropriados. Um multiset suporta iteradores bidirecionais (mas não iteradores de acesso randômico).

Dica de desempenho 20.14

• Por razões de desempenho, muitisets e sets são tipicamente implementados como estruturas denominadas árvores de pesquisa binária vermelho-preto. Com esta representação interna, a árvore de pesquisa
binária tende a ser balanceada, minimizando desta forma os tempos médios de pesquisa.

A Fig. 20.19 demonstra o contêiner associativo multiset para um multiset de inteiros ordenados em ordem
crescente. O arquivo de cabeçalho <set> deve ser incluído para usar a classe multi set. Contêineres multiset
e set oferecem as mesmas funções membro.

```
1 1/ Fig. 20.19: fig2o_19.cpp
2 // Testando a classe zuultiset da Biblioteca Padrão
3 #include <iostreani>
4
5 using std::cout;
6 using std::endl;
7
8 #include <set>
9 #include <algorithm>
10
11 int main()
12 {
13 const int SIZE 10;
14 int a[ SIZE ) = { 7, 22, 9, 1, 18, 30, 100, 22, 85, 13 };
15 typedef std::multiset< int, std::less< int > > ims;
16 ims intNultiset; II ims significa “integer multiset”
```

Fig. 20.19 Demonstrando o gabarito de classe multiset da biblioteca padrão (parte 1 de 2).


```
17 std: :ostreamiterator< int > output( cout, “ “
18
19 cout « “Atualmente existem “ « intNultiset.count( 15
20 « “ valores iguais a 15 no multiset\n”;
21 intMultiset.insert( 15 );
22 intNultiset.insert( 15 );
23 cout « Após inserts, existem
24 « intMultiset.count( 15
25 « “ valores iguais a 15 no multiset\n”;
```

```
26
27 ims: :const iterator result;
28
29 result = intMultiset.find( 15 ); /1 find retorna iterador
30
31 if ( result != intMultiset.end() ) II se iterador não está no fim
32 cout « “Encontrou o valor 15\n”; II encontrou o valor 15 procurado
34 result = intMultiset.find( 20 );
35
36 if ( result == intMultiset.end() ) II será verdadeiro portanto
37 cout « “Não encontrou o valor 20\n”; II não encontrou 20
38
39 intMultiset.insert( a, a + SIZE ); II soma array a ao multiset
40 cout « “Após insert intMultiset contém:\n”;
41 std: :copy( intldultiset.beginQ, intMultiset.endO, output );
42
43 cout « “\nLimite inferior de 22:
44 « *( intMultiset.lowerbound( 22 ) );
45 cout « “\nLimite superior de 22:
46 « ( intNultiset.upper_bound( 22 ) );
47
48 std::pair< ims::constiterator, ims::constiterator > p;
49
50 p = intMultiset.equal range( 22 );
51 cout « “\nUsando equal_range de 22”
52 « “\n Limite inferior: « ( p.first
53 « “\n Limite superior: “ « ( p.second );
54 cout « endl;
55 return 0;
56 }
```

```
Atualmente existem O valores iguais a 15 no multiset
Após inserts, existem 2 valores iguais a 15 no multiset
Encontrou o valor 15
Não encontrou o valor 20
Após insert intMultiset contém:
1 7 9 13 15 15 18 22 22 30 85 100
Limite inferior de 22: 22
Limite superior de 22: 30
Usando equal_range de 22
Limite inferior: 22
Limite superior: 30
```

Fig. 20.19 Demonstrando o gabarito de classe multiset da biblioteca padrão (parte 2 de 2).

As linhas 15 e 16

```
typedef std::multiset< int, std::less< int > > ims;
ims intMultiset; II ims significa “integer multiset”
```

usam um typedef para criar um novo nome de tipo (alias) para um multiset de inteiros ordenados em ordem ascendente usando o objeto função less< int>. Este novo tipo é então usado para instanciar um objeto integer multiset, denominado intNultiset.

Boa prática de programação 20.1

_____ Usar typedefs para conseguir ler mais facilmente código de programas com tipos com nomes longos (tais como multisets).

O comando de saída na linha 19

cout « “Atualmente existem “ « intMultiset.count(15
« “ valores iguais a 15 no multiset\n’;

usa a função count (disponível a todos os contêineres associativos) para contar o número de ocorrências do valor 15 presentemente no multiset.

As linhas 21 e 22

intMultiset.insert(15);
intMultiset.insert(15);

usam uma das três versões da função insert para adicionar o valor 15 ao multiset duas vezes. Uma segunda versão de insert recebe um iterador e um valor como argumentos e inicia a busca do ponto de inserção a partir da posição especificada pelo iterador. Uma terceira versão de insert recebe dois iteradores como argumentos que especificam um intervalo de valores de um outro contêiner para adicionar ao multiset.

A linha 29

result = intMultiset.find(15); II f ind retorna iterador

usa a função f ind disponível a todos os contêineres associativos) para localizar o valor 15 no multiset. A função f ind retorna um iterator ou um const iterator apontando para a primeira posição em que o valor é encontrado. Se o valor não é encontrado, f ind retorna um iterator ou um const iterator igual ao valor retornado por uma chamada a end.

A linha 39

intMultiset.insert(a, a + SIZE); II soma array a ao xnultiset

usa a função insert para inserir os elementos do array a no multiset. Na linha 41,o algoritmo copy copia os elementos do multiset para a saída padrão. Note que os elementos estão dispostos em ordem ascendente.

As linhas 43 a 46

cout « “\nLimite inferior de 22:
« (intMultiset.lowerbound(22));
cout « “\nLimite superior de 22:
« *(intMultiset.upper_bound(22));

usam as funções lower bound e upper bound (disponíveis a todos os contêineres associativos) para determinar a posição da primeira ocorrência do valor 22 no multiset e a posição do elemento após a última ocorrência do valor 22 no multiset. Ambas as funções retornam iterators ou const_iterators apontando para posição apropriada ou o iterator retornado por end. se o valor procurado não está no multiset.

A linha 48

std: :pair< ims: :constiterator, ims: :const_iterator > p;
instancia uma instância da classe pair chamada p. Objetos da classe pair são usados para associar pares de valores. Neste exemplo, os conteúdos de um pair são dois const iterators para nosso multiset baseado em inteiros. A finalidade de p é armazenar o valor retornado pela função equal range de multiset que

retorna um pair contendo os resultados das operações lower bound e upper bound. O tipo pair contém dois membros de dados public chamados first e second.
A linha 50
p = intMultiset.equal_range(22);
usa a função equal range para determinar o lower bound e o upper bound de 22 no multiset. As linhas 52 e 53 usam p. first e p. second, respectivamente, para acessar o lower bound e o upper_bound. Derreferenciamos os iteradores para enviar para a saída os valores retornados por equal range.

20.3.2 O contêiner associativo set

O contêiner associativo set é usado para armazenagem e recuperação rápidas com chaves únicas. A implementação de um set é idêntica àquela de um multiset, exceto pelo fato de que um set precisa ter chaves únicas. Portanto, se é feita uma tentativa de inserir uma chave duplicada em um set, a duplicata é ignorada como este é o comportamento matemático objetivado por um set, não o identificamos como um erro comum de programação. Um set suporta iteradores bidirecionais (mas não iteradores de acesso aleatório). A Fig. 20.20 demonstra um set de doubles. O arquivo de cabeçalho <set> deve ser incluído para usar a classe set.

```
1 II Fig. 20.20: fig2O2O.cpp
2 // Testando a classe set da biblioteca padrão
3 #include <iostream>
4
5 using std: :cout;
6 using std: :endl;
7
8 #include <set>
9 #include <algorithm>
10
11 int main()
12
13 typedef std::set< double, std::less< double > > double_set;
14 const int SIZE = 5;
15 doublea[SIZE]={2.1, 4.2, 9.5,2.1, 3.7};
16 doubleset doubleSet( a, a + SIZE );
17 std: :ostream iterator< double > output( cout, ‘
18
19 cout « “doubleSet contém: ;
20 std: :copy( doubleSet.begmnO, doubleSet.endO, output );
```

```
21
22 std: :pair< double set: :const_iterator, bool > p;
23
24 p = doubleSet.insert( 13.8 ); II valor não está no conjunto
25 cout « '\n' « *( p.first
26 « ( p.second ? " foi" : " não foi' ) « " inserido";
27 cout « "\ndoubleSet contem:
28 std: :copy( doubleSet.begmnO), doubleSet.endO, output );
29
30 p = doubleSet.insert( 9.5 ); II valor já está no conjunto
31 cout « '\n' « *( p.first
32 « ( p.second ? " foi" : " não foi" ) « " inserido";
33 cout « \ndoubleSet contém: ";
34 std: :copy( doubleSet.beginO, doubleSet.endQ, output );
35
```

Fig. 20.20 Demonstrando o gabarito de classe set da biblioteca padrão (parte 1 de 2).


```
36
37
38

cout « endi; return O;
```

Fig. 20.20 Demonstrando o gabarito de classe set da biblioteca padrão (parte 2 de 2).

A linha 13
typedef std: :set< double, std: :less< double > > double_set;
usa typedef para criar um novo nome de tipo (alias) para um conjunto de valores double ordenados em ordem ascendente, usando a função objeto less< double >.

A linha 16
doubleset doubleSet(a, a + SIZE);
usa o novo tipo double set para instanciar o objeto doubleSet. A chamada para o construtor recebe os elementos no array a entre a e a + SIZE (isto é, o array inteiro) e os insere dentro do set. A linha 20 usa o algoritmo copy para enviar o conteúdo do set para a saída. Observe que o valor 2.1 - que aparece duas vezes no array a somente aparece uma vez em doubleSet. Isto porque o contêiner set não permite duplicatas.

A linha 22
std: :pair< double_set: :const_iterator, bool > p;
define um pair consistindo em um const iterator para um double set e um valor bool. Este objeto armazena o resultado de uma chamada para a função insert de set.

A linha 24
p = doubleSet.insert(13.8); // valor não está no conjunto
usa a função insert para inserir o valor 13 . 8 no set. O pair retornado, p, contém um iterador p. first apontando para o valor 13. 8 no set e um valor bool que é true se o valor foi inserido e false se o valor não foi inserido (porque já estava no set).
20.3.3 O contêiner associativo multimap
O contêiner associativo multimap é usado para armazenagem e recuperação rápidas de chaves e valores associados (freqüentemente chamados de pares chave/valor). Muitos dos métodos usados com multisets e sets são usados também com multimaps e maps. Os elementos de multimaps e maps são pairs de chaves e valores em vez de valores individuais. Quando se insere em um multimap ou map. um objeto pair, que contém a chave e o valor, é usado. A ordenação das chaves é determinada por um objeto função comparadora. Por exemplo, em um multimap que usa inteiros como o tipo de chave, as chaves podem ser classificadas em ordem ascendente ordenando as chaves com o objeto função comparadora less< int >. Chaves duplicadas são permitidas em um multimap. de modo que valores múltiplos podem ser associados com uma chave única. Isto é chamado freqüentemente de um relacionamento um para muitos. Por exemplo, em um sistema de processamento de transações por cartão de crédito, uma conta de cartão de crédito pode ter muitas transações associadas; em uma universidade, um estudante pode fazer vários cursos e um professor pode ensinar muitos estudantes; no exército, um posto

doubleSet	contém:	2.1	3.7	4.2	9.5	
13.8 foi	inserido					
doubleSet	contém:	2.1	3.7	4.2	9.5	13.8
9.5 não f	oi inserido					
doubleSet	contém:	2.1	3.7	4.2	9.5	13.8

(como “soldado raso”) tem muitas pessoas. Um multimap suporta iteradores bidirecionais (mas não iteradores de acesso aleatório). Tal como com multisets e sets, multimaps são implementados típicamente como uma árvore de busca binária vermelho-preta na qual os nodos da árvore são pairs chave/valor. A Fig. 20.21 demonstra o contêiner associativo multimap. O arquivo de cabeçalho map> deve ser incluído para usar a classe multimap.
Dica de desempenho 20.15
_____ Um multimap é implementado de modo a localizar eficientemente todos os pares de valores com uma determinada chave.
A linha 12
typedef std::multimap< int, double, std::less< int > > mmid;

usa typedef para definir um alias para um tipo multimap onde o tipo de chave é int. o tipo de um valor associado é double e os elementos são ordenados em ordem ascendente. A linha 13 usa o novo tipo para instanciar um multimap denominado pairs.

```
1 II Fig. 20.21: fig20_21.cpp
2 II Testando a classe multimap da Biblioteca Padrão
3 #include <iostreaxn>
4
5 using std::cout;
6 using std::endl;
7
8 #include <map>
9
10 int main()
11 {
12 typedef std::multimap< int, double, std::less< int > > mmid;
13 mmid pairs;
14
15 cout « ‘Atualmente existem “ « pairs.count( 15
16 « ‘ pares com a chave 15 no multimap\n’;
17 pairs.insert( mmmd: :value type( 15, 2.7 ) );
18 pairs.insert( mmid::valuetype( 15, 99.3 ) );
19 cout « “Após inserts, existem
20 « pairs.count( 15
21 « “ pares com a chave 15\n’;p
22 pairs.insert( mmid: :value type( 30, 111.11 ) );
23 pairs.insert( mmid: :valuetype( 10, 22.22
24 pairs.insert( mmid: :value type( 25, 33.333 ) );
25 pairs.insert( mmid: :valuetype( 20, 9.345 ) );
26 pairs.insert( mmid::valuetype( 5, 77.54 ) );
27 cout « O multimap pairs contém:\nChave\tValor\n”;
28
29 for ( mmid::const_iterator iter = pairs.begmn
30 iter != pairs.end ++iter
31 cout « iter->first « ‘\t’
32 « iter->second « ‘\n’;
31
34 cout « endl;
35 return 0;
36
```

Fig. 20.21 Demonstrando o gabarito de classe multimap da biblioteca padrão (parte 1 de 2).

Fig. 20.21 Demonstrando o gabarito de classe multimap da biblioteca padrão

(parte 2 de 2).
As linhas 15 e 16
cout « “Atualmente existem “ « pairs.count(15
« “ pares com a chave 15 no multimap\n”;
usam a função count para determinar o número de pares chave/valor com uma chave igual a 15.
A linha 17
pairs.insert(mmid::valuetype(15, 2.7));
usa a função insert para adicionar um novo par chave/valor ao multimap. A expressão mid: : value type 15, 2.7) cria um objeto pair no qual first é a chave (15) do tipo int e second é o valor (2.7) do tipo double. O tipo mmid: : value type é definido na linha 12 como parte do typedef para o multimap.
A estrutura for na linha 29 envia para a saída o conteúdo do multimap, incluindo tanto as chaves como os
valores. As linhas 31 e 32
cout « iter->first « ‘\t’
« iter->second « ‘\n;
usam o const_iterator denominado iter para acessar os membros do par em cada elemento do multimap. Note na saída que as chaves estão em ordem ascendente.
20.3.4 O contêiner associativo map
O contêiner associativo map é usado para armazenagem e recuperação rápidas de chaves ánicas e valores associados. Chaves duplicadas não são permitidas em um map. de modo que somente um único valor pode ser associado a cada chave. Isto é chamado de mapeamento um a um. Por exemplo, uma empresa que usa números únicos para identificar seus empregados tais como 100, 200 e 300 poderiam ter um map que associa o número dos empregados com seus ramais 4321, 4115 e 5217, respectivamente. Com um map você especifica a chave e obtém de volta os dados associados rapidamente. Um map é normalmente chamado de array associativo. Fornecer a chave em um operador de subscrito de map [ ], permite localizar o valor associado com aquela chave no map. Inserções e retiradas podem ser feitas em qualquer lugar em um map.
A Fig. 20.22 demonstra o contêiner associativo map. A Fig. 20.22 usa as mesmas características que a Fig.
20.21, exceto pelo operador subscrito, O arquivo de cabeçalho <map> precisa ser incluído para usar a classe map. As linhas 31 e 32
pairs[25) = 9999.99; II muda valor existente para 25
pairs[40 3 = 8765.43; II insere novo valor para 40

Atualmente existem O pares com a chave 15 Após inserts, existem 2 pares com a chave O multimap pairs contém: Chave Valor		no multimap 15
5 77.54		
10 22.22		
15 2.7		

15 99.3		
20 9.345		
25 33.333		
30 111.11		

usam o operador de subscrito da classe map. Quando o subscrito é uma chave que já está no map. o operador retor uma referência ao valor associado. Quando o subscrito é uma chave que não está no map. o operador insere a cha no map e retorna uma referência que pode ser usada para associar um valor àquela chave. A linha 31 substitui o vai correspondente à chave 25 (anteriormente 33 . 333, como especificado na linha 19) por um novo valor de 9999. 9 A linha 32 insere um novo par chave/valor (denominado criar u,na associação) no map.

1 II Fig. 20.22: fig2022.cpp
2 // Testando a classe map da biblioteca padrão
3 #include <iostreaxn>
4
5 using std::cout;
6 using std::endl;
7
8 #include <map>
9
int main()
typedef std: :map< int, double, std: :less< int > > mid; mid pairs;
15, 2.7));
30, 111.11));
5, 1010.1));
10, 22.22));
25, 33.333));
5, 77.54)); /20, 9.345));
- 15, 99.3)); //
cout « “pairs contém:\nChave\tValor\n”;

pairs . insert pairs . insert pairs . insert pairs . insert pairs . insert pairs . insert pairs . insert

mid: :valuetype(mid: :valuetype(mid: :valuetype(mid: :valuetype(mid: : value type mid: :valuetype(mid: :valuetype(mid: :value type(

pairs . insert

10
11

12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42

duplicata ignorada duplicata ignorada

```
mid: :const_iterator iter;
for ( iter = pairs.begin iter != pairs.endQ; ++iter cout « iter->first « '\t'
« iter->second « '\n';
pairs[ 25 ) = 9999.99; II muda valor existente para 25
pairs[ 40 ] = 8765.43; II insere novo valor para 40
cout « "\nApós operações com subscritos, pairs contém:"
« "\nChave\tValor\n";
for ( iter pairs.begin iter != pairs.end ++iter cout « iter->first « '\t'
« iter->second « '\n';
cout « endl;
return 0;
```

pairs
Chave
5
10
15
20
25
30

contém:
Valor
1010.1
22.22
2.7
9.345
33.333
111.11

Fig. 20.22 Demonstrando o gabarito de classe map da biblioteca padrão (parte 1 de 2).

A STL fornece três dos chamados adaptadores de contêineres - stack. queue e priority queue. Adaptadores não são contêineres de primeira classe porque não implementam verdadeiramente a estrutura de dados na qual elementos podem ser armazenados e também porque não suportam iteradores. O beneficio de uma classe de adaptador é que o programador pode escolher uma estrutura de dados subjacente apropriada. Todas as três classes de adaptadores fornecem funções membro push e pop que implementam o método adequado para inserir um elemento em cada estrutura de dados do adaptador e o método adequado para retirar um elemento de cada estrutura de dados. As conhecida subseções seguintes fornecem exemplos das classes de adaptadores.

20.4.1 O adaptador stack

A classe stack fornece funcionalidade que permite inserções e retiradas em uma mesma extremidade da estrutura de dados subjacente (comumente conhecida como uma estrutura de dados último a entrar, primeiro a sair - LIFO). Um stack pode ser implementado com qualquer um dos contêineres seqüenciais: vector. list e deque. Este exemplo cria três stacks de inteiros usando cada um dos contêineres seqüenciais da Biblioteca Padrão como a estrutura de dados subjacente para representar o stack. Por default, um stack é implementado com um deque. As operações do stack são: push, para inserir um elemento no topo do stack (implementado pela chamada à função push back do contêiner subjacente); pop. para remover o elemento do topo do stack (implementado pela chamada à função pop back do contêiner subjacente); top. para obter uma referência ao elemento do topo do stack (implementada pela chamada à função back do contêiner subjacente); empty. para determinar se o stack está vazio (implementado pela chamada à função empty do contêiner subjacente); e size

para obter o número de elementos no stack (implementado pela chamada à função size do contêiner subjacente).
II)ica de desempenho 20.16
Cada uma das operações comuns de um stack é implementada como uma função mime que chama a fànção apropriada do contêiner subjacente. Isto evita o overhead de uma segunda chamada de função.
Dica de desempenho 20.17
Para melhor desempenho, use as classes deque ou vector como o contêiner subjacente para um stack.
A Fig. 20.23 demonstra o adaptador de classe stack. O arquivo de cabeçalho <stack> deve ser incluído para usar a classe stack.
1 II Fig. 20.23: fig2023.cpp
2 // Testando a classe stack da biblioteca padrão
3 #include <iostream>

lor

Após operações com subscritos, pairs contém:
Chave Valor
5 1010.1
10 22.22
15 2.7

20
25
30
40

9.345
9999. 99
111.11
8765.43

Fig. 20.22 Demonstrando o gabarito de classe map da biblioteca padrão (parte 2 de 2).

20.4 Adaptadores de contêineres

Fig. 20.23 Demonstrando a classe adaptadora stack da biblioteca padrão (parte 1 de 2).

```
4
5 using std::cout;
6 using std::endl;
7
8 #include <stack>
9 #include <vector>
10 #include <list>
11
12 template< class T >
13 void popElements ( T &s );
14
15 int main()
16
17 std::stack< int > intDequeStack; // pilha baseada em deque
18 std::stack< int, std::vector< int > > intVectorStack;
19 std::stack< int, std::list< int > > intListStack;
20
21 for ( int i 0; i < 10; ++i
22 intDequeStack.push( i );
23 intVectorStack.push( i );
24 intListStack.push( i );
25 }
26
27 cout « 'Retirando de intDequeStack:
28 popElements( mntDequeStack );
29 cout « "\nRetirando de intVectorStack: ";
30 popElements ( intVectorStack );
31 cout « "\nRetirando de intListStack: ";
32 popElements ( intListStack );
33
34 cout « endi;
35 return 0;
36
37
38 template< class T >
39 void popElements( T &s
40 {
41 while ( !s.empty() ) {
42 cout « s.top() « '
43 s.popO);
44 }
45 }
```

```
Retirando de intDequeStack: 9 8 7 6 5 4 3 2 1 O
Retirando de intVectorStack: 9 8 7 6 5 4 3 2 1 O
Retirando de intListStack: 9 8 7 6 5 4 3 2 1 O
```

Fig. 20.23 Demonstrando a classe adaptadora stack da biblioteca padrão (parte

2 de 2).
As linhas 17 a 19
std::stack< int > intDequeStack; // pilha baseada em deque std::stack< int, std::vector< int > > mntVectorStack;
std::stack< int, std::list< int > > intListStack;
instanciam três s tacks de inteiros. A linha 17 especifica um stack de inteiros que usa o contêiner default deque como sua estrutura de dados subjacente. A linha 18 especifica um stack de inteiros que usa um vector de

1
CAPÍTULO 20 - A BIBLIOTECA PADRÃO DE GABARITOS (STL) 953
inteiros como sua estrutura subjacente de dados. A linha 19 especifica um stack de inteiros que usa uma list de inteiros como sua estrutura subjacente de dados. As linhas 22 a 24 usam cada uma a função push (disponível em toda classe de adaptador) para colocar um inteiro no topo de cada stack.
A função popElements na linha 38 retira os elementos de cada stack. A linha 42 usa a função top de stack para recuperar o elemento no topo do stack como resultado. A função top não remove o elemento do topo. A linha 43 usa a função pop (disponível em cada adaptador de classe) para remover o elemento no topo do stack. A função pop não retorna nenhum valor.
20.4.2 O adaptador queue
deque A classe queue possibilita inserções no fim da estrutura de dados subjacente e retiradas no início da estrutura de dados subjacente (comumente conhecida como uma estrutura de dados primeiro a entrar primeiro a sair - FIFO). Uma queue pode ser implementada com as estruturas de dados list e deque da STL. Por default, uma queue é implementada com um deque. As operações comuns de queue são: push para inserir um elemento no fim da queue (implementada pela chamada à função push back do contêiner subjacente); pop para remover o elemento do início da queue (implementada pela chamada à função pop_front do contêiner subjacente); front para obter uma referência ao primero elemento na queue (implementada pela chamada à função front do contêiner subjacente): back para obter uma referência ao último elemento na queue (implementada pela chamada à função back do contêiner subjacente); empty para determinar se a queue está vazia (implementada pela chamada à função empty do contêiner subjacente); e size para obter o número de elementos na queue (implementada pela chamada à função size do contêiner subjacente).
IDica de desempenho 20.18
_____ Cada uma das operações comuns de uma queue é implementada como uma função mime que chama a função apropriada do contêiner subjacente. Isto evita o overhead de uma chamada de uma segunda função.
Dica de desempenho 20.19
t Para melhor desempenho, use a classe deque como o contêiner subjacente para uma queue.
A Fig. 20.24 demonstra a classe adaptador ql.leue. O arquivo de cabeçalho <queue> deve ser incluído para usar uma queue.

```
1 II Fig. 20.24: fig20_24.cpp
_____ 2 // Testando o gabarito de classe adaptadora queue da
Biblioteca Padrão
3 #include <iostream>
4
5 using std::cout;
6 using std: :endl;
_____ 7
de 2). 8 #include <queue>
9
10 int main()
11 {
12 std::queue< double > values;
deque 13
14 vaiues.push( 3.2 );
15 vaiues.push( 9.8 );
16 values.push( 5.4 );
têiner default deque 17
usa um vector de
```

Fig. 20.24 Demonstrando os gabaritos de classe adaptadora queue da biblioteca padrão (parte 1 de 2).


```
18 cout « “Retirando de values: “;
19
20 while ( !values.empty()
21 cout « values.front() « ‘; // não remove
22 values.popO; // remove o elemento
23
24
25 cout « endl;
26 return 0;
27
Retirando de values: 3.2 9.8 5.4
```

Fig. 20.24 Demonstrando os gabaritos de classe adaptadora queue da biblioteca padrão (parte 2 de 2).

A linha 12

std: :pieue< double > values;

instancia uma queue que armazena valores double. As linhas 14 a 16 usam função push para adicionar elementos à queue. A estrutura while na linha 20 usa a função empty (disponível em todos os contêineres) para determinar se a queue está vazia.

Enquanto há mais elementos na queue, a linha 21 usa a função front de queue para ler (mas não remover) o primeiro elemento da queue para enviar para a saída. A linha 22 remove o primeiro elemento da queue com a função pop

(disponível em todas as classes de adaptadores).
20.4.3 O adaptador priority_queue
A classe priority queue fornece funcionalidade que permite inserções em ordem classificada na estrutura de dados subjacente, bem como retiradas do início da estrutura de dados subjacente. Uma priority_queue pode ser implementada com estruturas de dados vector e deque. Por default, uma priority ql.leue é implementada com um vector como a estrutura de dados. Ao se adicionar elementos a uma priority queue. os elementos são inseridos em ordem de prioridade, de tal modo que o elemento de mais alta prioridade (isto é, o valor maior) será o primeiro elemento retirado da priority queue. Isto normalmente é realizado usando-se uma técnica de classificação denominada heapsort, que sempre mantém o maior valor (isto é, a mais alta prioridade) na frente da estrutura de dados - tal estrutura de dados é chamada de heap. A comparação de elementos é executada com o objeto função de comparação less< T > por default, mas o programador pode fornecer um comparador diferente.
As operações comuns de priority queue são: push para inserir um elemento na posição apropriada com base na ordem de prioridade da priority_queue (implementada pela chamada à função push_back do contêiner subjacente e então pela reordenação dos elementos usando heapsort): pop para remover o elemento de mais alta prioridade da priority queue (implementada pela chamada à função pop back do contêiner subjacente após remover o elemento do topo do heap): top, para obter uma referência ao elemento no topo da priorityqueue (implementada pela chamada à função front do contêiner subjacente); empty para determinar se a priorityqueue está vazia (implementada pela chamada à função empty do contêiner subjacente); e size para obter o número de elementos na priori ty queue (implementada pela chamada à função si ze do contêiner subjacente).
1 Dica de desempenho 20.20
_____ Cada uma das operações comuns de uma priority queue é implementada como uma função inline
que chama afunção apropriada do contêiner subjacente. Isto evita o overhead de uma segunda função.
Dica de desempenho 20.21
f Para melhor desempenho, use a classe vector como contêiner subjacente para uma priorityqueue.

A Fig 20.25 demonstra a classe de adaptador priority queue. O arquivo de cabeçalho <queue> deve
ser incluído para usar a classe priorityqueue.
```
1 // Fig. 20.25: fig2025.cpp
2 // Testando a classe priority_queue da biblioteca padrão
3 #include <iostream>
4
5 using std: :cout;
6 using std: :endl;
```

```
7
8 #include <queue>
9
10 int main()
11 {
12 std: :priority_queue< double > priorities;
13
14 priorities.push( 3.2 );
15 priorities.push( 9.8 );
16 priorities.push( 5.4 );
17
18 cout « “Retirando de priorities: “;
19
20 while ( !priorities.emptyO) ) {
21 cout « priorities.top() «
22 priorities.popO);
23
24
25 cout « endl;
26 return 0;
27 }
[Retirando de priorities: 9.8 5.4 3.2
```

Fig. 20.25 Demonstrando a classe adaptadora priori ty queue da biblioteca padrão.

A linha 12

```
std: :priority_queue< double > priorities;
```

instancia uma priority queue que armazena valores double e usa um vector como a estrutura de dados subjacente. As linhas 14 a 16 usam a função push para adicionar elementos à priority queue. A estrutura while na linha 20 usa a função exnpty (disponível em todos os contêineres) para determinar se a priorityqueue está vazia. Enquanto houver mais elementos na priority queue, a linha 21 usa a função top de priorityqueue para recuperar o elemento de mais alta prioridade na priority queue como resultado. A linha 22 remove físicamente o elemento de maior prioridade na priority queue com a função pop (disponível em todas as classes de adaptadores).

20.5 Algoritmos

Antes da STL, as bibliotecas de classes de contêineres e algoritmos de diferentes fornecedores eram essencialmente incompatíveis. As bibliotecas de contêineres antigas geralmente usavam herança e polimorfismo, com os overheads associados a chamadas de funções virtual. As bibliotecas antigas tinham os algoritmos construídos nas classes de contêineres, como comportamentos das classes. A STL separa os algoritmos dos contêineres. A STL é implementada para eficiência. Ela evita o overhead das chamadas de funções virtual. Com a STL, os elementos de contêineres são acessados através de iteradores.

Observação de engenharia de software 20.8
Os algoritmos não dependem dos detalhes de implementação dos contêineres sobre os quais eles operam. Enquanto os iteradores de contêineres (ou de arrays) satisfizerem os requisitos do algoritmo, os algoritmos da STL podem trabalhar com quaisquer arrays no estilo de C, baseados em ponteiros, assim como trabalhar com os contêineres da STL (e estruturas de dados definidas pelo usuário).
Observação de engenharia de software 20.9
Algoritmos podem ser facilmente adicionados à STL sem modificar as classes contêineres.
20.5.1 fui, fili_n, generate e generate_n
A Fig. 20.26 demonstra as funções f iii. filin, generate e generate n da biblioteca padrão. As funções
f iii e fui n atribuem a cada elemento de uma série de elementos do contêiner um valor específico. As funções
generate e generate_n usam uma função geradora para criar valores para cada elemento de uma série de
elementos do contêiner. A função geradora não recebe nenhum argumento, retornando um valor que pode ser colocado em um elemento do contêiner.

```
1 II Fig. 20.26: fig2026.cpp
2 /1 Demonstrando os métodos da Biblioteca Padrão
3 II fui, fiiln, generate e generate_n.
4 #include <iostream>
5
6 using std: :cout;
7 using std::endl;
8
9 #include <algorithm>
10 #inciude <vector>
11
12 char nextLetterQ;
13
14 int main()
15
16 std::vector< char > chars( 10 );
17 std: :ostream_iterator< char > output( cout, “
18
19 std: :fill( chars.begmn() , chars.end() , ‘5’ );
20 cout « “Vector chars depois de preenchido com 5s:\n”;
21 std: :copy( chars.beginO), chars.endO, output );
22
23 std: :fill_n( chars.begmnO), 5, ‘A’
24 cout « “nVector chars depois de preenchidos cinco elementos”
25 « “com As:\n”;
26 std: :copy( chars.beginQ, chars.end() , output
```

```
27
28 std: :generate( chars.beginQ, chars.end() , nextLetter );
29 cout « “nVector chars depois da geração das letras A a
30 std::copy( chars.beginQ, chars.endQ, output );
31
32 std::generate_n( chars.beginQ, 5, nextLetter );
33 cout « “\nVector chars depois da geração de K a O para os”
34 « “ primeiros cinco elementos:\n”;
35 std: :copy( chars.begin() , chars.end() , output
36
```

Fig. 20.26 Demonstrando as funções fill, filin, generate e generate_n da biblioteca padrão (parte 1 de 2).


```
37 cout « endi;
38 return 0;
‘am. 39 }
mos 40
iba- 41 char nextLetter()
42 {
43 static char letter = ‘A’;
44 return letter++;
45 }
Vector chars depois de preenchido com 5s:
5555555555
Vector chars depois de preenchidos cinco elementos com As:
AAAAA5 5555
:ões
Vector chars depois da geraçao das letras A a J:
ABCDEFGHIJ
de
aio- Vector chars depois da geração de K a O para os primeiros cinco
elementos:
KLMNOFGHIJ
```

Fig. 20.26 Demonstrando as funções fill, filin, generate e generate n da biblioteca padrão (parte 2 de 2).

A linha 19

```
std: :fill( chars.begin() , chars.end() , ‘5’ );
```

usa a função fill para colocar o caracter 5 em cada elemento do vector chars desde chars begin O até, mas não incluindo, chars end . Observe que os iteradores fornecidos como o primeiro e o segundo argumentos devem ser no mínimo iteradores para a frente (isto é, podem ser usados tanto para leitura de um contêiner quanto para saída para um contêiner na direção para a frente).

A linha 23

```
std: :filln( chars.begin, 5, ‘A’
```

usa a função fill_n para colocar o caractere ‘A’ nos primeiros cinco elementos do

vector chars. O iterador fornecido como o primeiro argumento deve ser no mínimo um iterador de saída (isto é, pode ser usado para saída de um contêiner na direção para a frente). O segundo argumento especifica o número de elementos para preencher. O terceiro argumento especifica o valor para colocar em cada elemento.
A linha 28
std: :generate(chars.begin() , chars.end() , nextLetter)
usa a função generate para colocar o resultado de uma chamada à função geradora nextLetter em cada elemento do vector chars. desde chars begin () até, mas não incluindo, chars . end o. Os iteradores fornecidos como o primeiro e segundo argumentos devem ser no mínimo iteradores para a frente. A função nextLetter (definida na linha 41) começa com o caractere 'A' mantido em uma variável local static. O comando na linha 44
return letter++;
incrementa o valor de letter e retorna o valor anterior de letter cada vez que nextLetter é chamada. A linha 32
rao

std: :generate_n(chars.begin() , 5, nextLetter)

usa a função generate_n para colocar o resultado de uma chamada do gerador de função nextLetter em cinco elementos do vector chars começando em chars . begin . O iterador fornecido como o primeiro argumento deve ser no mínimo um iterador de saída.
20.5.2 equal. mismatch e lexicographical compare
A Fig . 20.27 demonstra a comparação de seqüências de valores quanto à igualdade com as funções equal, mismatch e lexicographical compare da biblioteca padrão.
```
1 // Fig. 20.27: fig2027.cpp
2 // Demonstra as funções da biblioteca padrão
3 // equal, mismatch e lexicographical compare.
4 #include <iostream>
5
6 using std: :cout;
7 using std: :endl;
8
9 #inciude <algorithm>
10 #include <vector>
ii
12 int main()
13
14 const int SIZE = 10;
15 intai[SIZE)={i, 2,3,4,5,6,7,8,9, i0};
16 inta2[SIZE)={i, 2,3,4,1000,6,7,8,9, 10};
17 std::vector< int > vl( ai, ai + SIZE ),
```

```
18 v2( ai, ai + SIZE ),
19 v3( a2, a2 + SIZE );
20 std: :ostreamiterator< int > output( cout, “
21
22 cout « “Vector vi contém:
23 std::copy( v1.begmn, vi.end, output );
24 cout « “\nVector v2 contém: “;
25 std::copy( v2.beginQ, v2.end(), output );
26 cout « “\nVector v3 contém: “;
27 std: :copy( v3.beginQ, v3.endQ, output );
28
29 bool result =
30 std: :equal ( vi .begmnO, vi .endO, v2 .begin() );
31 cout « “\n\nVector vi “ « ( result ? “é” : “não é”
32 « “ igual a v2.\n”;
33
34 result = std::equal( vl.beginO, vl.end(), v3.begin() );
35 cout « “Vector vi “ « ( result ? “é” : “não é”
36 « “ igual a vector v3.\n”;
1
38 std::pair< std::vector< int >::iterator,
39 std::vector< int >::iterator > location;
40 location =
41 std: :mismatch( vi.begin(), vl.endO, v3.begin() );
42 cout « “\nExiste uma diferença entre vi e v3 na
43 « “posição « ( location.first - vl.beginO)
44 « “\nonde vi contém “ « *locationfirst
45 « “ e v3 contém “ « *locationsecond
```

Fig. 20.27 Demonstrando as funções equal, mismatch e lexicographical compare da biblioteca padrão (parte 1 de 2).


```
m 46 « “\n\n”;
ro 47
48 char cl[ SIZE ] = “01”, c2[ SIZE 1 = “ATÉ LOGO”;
49
50 result = std: :lexicographicalcompare(
51 cl, cl + SIZE, c2, c2 + SIZE
52 cout « cl
53 « ( result ? “ é menor do que
54 “ é maior do que ou igual a
55 « c2 « endi;
56
57 return 0;
58 }
Vector vi contém: 1 2 3 4 5 6 7 8 9 10
```

```
Vector v2 contém: 1 2 3 4 5 6 7 8 9 lO
Vector v3 contém: 1 2 3 4 1000 6 7 8 9 10
Vector vi é igual a vector v2.
Vector vi não é igual a vector v3.
Existe uma diferença entre vl e v3 na posição 4
onde vi contém 5 e v3 contém 1000
01 é maior do que ou igual a ATÉ LOGO
```

Fig. 20.27 Demonstrando as funções equal, mismatch e lexicographical compare da biblioteca padrão (parte 2 de 2).

As linhas 29 e 30

```
bool result =
std: :equal( vi .beginO, vi .endO, v2 .begin() );
```

usam a função equal para comparar duas seqüências de valores quanto à igualdade. Cada seqüência não precisa conter, necessariamente, o mesmo número de elementos - equal retorna false se as seqüências não têm o mesmo comprimento. A função operator== faz a comparação dos elementos. Neste exemplo, os elementos no vector vi, de vi . begin () até, mas não incluindo, vi . end , são comparados aos elementos no vector v2 começando em v2 . begin () (neste exemplo, vi e v2 são iguais). Os três argumentos iteradores devem ser no mínimo iteradores de entrada (isto é, podem ser usados para leitura de uma seqüência na direção para a frente). A linha 34 usa a função equal para comparar vectors vi e v3 que não são iguais.

Há uma outra versão da função equal que aceita uma função predicado binária como um quarto parâmetro. A função predicado binária recebe os dois elementos que estão sendo comparados e retorna um valor booi indicando se os elementos são iguais ou não. Isto pode ser útil em seqüências que armazenam ponteiros para valores em vez de valores reais, porque você pode definir uma ou mais comparações. Por exemplo, você pode comparar objetos Employee quanto à idade, código ou localização em vez de comparar objetos inteiros. Pode comparar aquilo a que os ponteiros se referem, em vez de comparar os conteúdos dos ponteiros (isto é, os endereços armazenados nos ponteiros).

As linhas 38 a41

```
std: :pair< std: :vector< int >: :iterator,
std: :vector< int >: :iterator > location;
location =
std: :mismatch( vi .beginQ, vi. endO, v3 .begin() );
```


começam por instanciar um pair de iteradores chamado location com um vector de inteiros. Este objeto armazena o resultado da chamada para mismatch. A função mismatch compara duas seqüências de valores e retorna um pair de iteradores indicando a posição, em cada seqüência, dos elementos diferentes. Se todos os elementos forem iguais, os dois iteradores no pair são iguais ao último iterador para cada seqüência. Os três argumentos iteradores devem ser pelo menos iteradores de entrada. Para determinar a verdadeira posição da não-concordância dos vectors, neste exemplo é usada a expressão location. first -

vi . begin () na linha 43. O resultado desse cálculo é o número de elementos entre os iteradores (isto é análogo à aritmética de ponteiros que estudamos no Capítulo 5). Corresponde ao número do elemento neste exemplo, porque a comparação é executada a partir do começo de cada vector.
Assim como com a função equal, há outra versão da função mismatch que aceita uma função predicado
binária como um quarto parâmetro.
As linhas 50 e 51
result std: :lexicographical_compare(
ci, cl + SIZE, c2, c2 + SIZE);
usam a função lexicographical compare para comparar os conteúdos de dois arrays de caracteres. Os quatro argumentos iteradores desta função devem ser pelo menos iteradores de entrada. Como você sabe, ponteiros em arrays são iteradores de acesso aleatório. Os dois primeiros argumentos de iteradores especificam o intervalo das posições na primeira seqüência. Os dois últimos argumentos de iteradores especificam o intervalo das posições na segunda seqüência. Ao se percorrer as seqüências, se o elemento na primeira seqüência é menor do que o elemento correspondente na segunda seqüência, a função retorna true. Se o elemento na primeira seqüência é maior que ou igual ao elemento na segunda seqüência, a função retorna false. Esta função pode ser usada para colocar seqüências em ordem lexicográfica*. Tipicamente, tais seqüências contêm strings.
20.5.3 remove. remove if, remove_copy e remove_copy_if
A Fig. 20.28 demonstra a remoção de valores de uma seqüência usando as funções remove. remove_if, remove_copy e remove_copy_if da biblioteca padrão.
```
1 Il Fig. 20.28: fig2028.cpp
2 Il Demonstra as funções remove, removeif,
3 Il remove_copy e remove_copyif da biblioteca padrão.
4 #include <iostream>
5
6 using std::cout;
7 using std: :endl;
8
9 #include <algorithm>
iO #include <vector>
ii
12 bool greater9 ( int );
13
i4 int main()
```
Fig. 20.28 Demonstrando as funções remove, remove_if, remove copy e remove_copy_if da biblioteca padrão (parte 1 de 3).
* N. de R.T.: Na ordem em que apareceriam em um dicionário, ou seja, em ordem crescente dos códigos numéricos utilizados para representar os caracteres na memória do computador.

jeto 15 {
se 16 const int SIZE = 10;
17 int a[SIZE] = { 10, 2, 10, 4, 16, 6, 14, 8, 12, 10 };
três 18 std: :ostreamiterator< int > output(cout, “ ‘
19
ao 2
// Remove lOs de v
nha 21 std: :vector< int > v(a, a + SIZE);
[fOS 22 std::vector< int >::iterator newLastElement;
CU- 23 cout « “Vector v antes da remoção de todos os lOs:\n;
24 std: :copy(v.beginQ, v.endO, output);
25 newLastElement = std: :remove(v.beginO, v.endO, 10);
26 cout « “\nVector v depois da remoção de todos os lOs:\n’;
27 std: :copy(v.begin, newLastElement, output);
28
29 // Copia de v2 para c, removendo os lOs
30 std::vector< int > v2(a, a + SIZE);
31 std: :vector< int > c(SIZE, O);
32 cout « “\n\nVector v2 antes da remoção de todos os lOs
33 « “e copia:\n”;
be, 34 std::copy(v2.beginO, v2.endQ, output);
35 std: :remove copy(v2.beginQ, v2.endO, c.beginO, 10);
ao 36 cout « ‘\nVector c depois da remoção de todos os lOs de v2:\n”;
se- 37 std: :copy(c.begin, c.endQ, output);
38
na 1 // Remove elementos maiores do que 9 de v3
ais 40 std::vector< int > v3(a, a + SIZE);
41 cout « “\n\nVector v3 antes da remoção de todos os elementos”
42 « “\nmaiores do que 9:\n’;
43 std::copy(v3.begin, v3.endO, output);
44 newLastElement =
45 std: :remove_if(v3.beginQ, v3.endO, greater9);
46 cout « “\nVector v3 depois da remoção de todos os elementos”
47 « “\nmaiores do que 9:\n’;
48 std: :copy(v3.begin, newLastElement, output);
49
50 II Copia elementos de v4 para c2,
51 // removendo elementos maiores do que 9
52 std::vector< int > v4(a, a + SIZE);
53 std::vector< int > c2(SIZE, O);
54 cout « “\n\nVector v4 antes da remoção de todos os elementos”
55 « \nmaiores do que 9 e copia:\n”;
56 std::copy(v4.beginQ, v4.endO, output);
57 std: : remove copy if(v4 .begin, v4 . endQ,
58 c2.begin, greater9);

```
59 cout « “\nVector c2 depois da remoção de todos os elementos”
60 « ‘\nmaiores do que 9 de v4:\n”;
61 std::copy( c2.beginO, c2.endO, output );
62
63 cout « endl;
64 return 0;
65
66
67 bool greater9( int x
68
69 return x > 9;
70
```

Fig. 20.28 Demonstrando as funções remove, remove if, remove copy e remove copy if da biblioteca padrão (parte 2 de 3).


```
Vector v antes da remoção de todos os lOs:
10 2 10 4 16 6 14 8 12 10
Vector v depois da remoção de todos os lOs:
2 4 16 6 14 8 12
Vector v2 antes da remoção de todos os lOs e copia:
1021041661481210
Vector c depois da remoção de todos os lOs de v2:
2416614812000
Vector v3 antes da remoção de todos os elementos
maiores do que 9:
10 2 10 4 16 6 14 8 12 10
Vector v3 depois da remoção de todos os elementos
maiores do que 9:
2468
Vector v4 antes da remoção de todos os elementos
maiores do que 9 e copia:
1021041661481210
Vector c2 depois da remoção de todos os elementos
naiores do que 9 de v4:
2468000000
```

Fig. 20.28 Demonstrando as funções remove, remove if, remove copy e remove_copy_if da bíblioteca padrão (parte 3 de 3).

A linha 25

```
newLastElement = std: :remove( v.begin, v.endO, 10
```

usa a função remove para eliminar todos os elementos com o valor 10 no intervalo que vai de v.begin O até, mas não incluindo, v. end () do vector v. Os dois primeiros argumentos iteradores devem ser iteradores para a frente, de modo que o algoritmo possa modificar os elementos na seqüência. Esta função não modifica o número de elementos no vector nem destrói os elementos

eliminados, mas move todos os elementos que não são eliminados em direção ao começo do vector. A função retorna um iterador posicionado após o último elemento do vector que não foi excluído. Os elementos da posição do iterador até o final do vector têm valores indefinidos (neste exemplo, cada posição “indefinida” tem valor O).
A linha 35
std: :remove_copy(v2.beginQ, v2.endO, c.begin, 10);
usa a função remove copy para copiar todos os elementos que não têm o valor 10 na fila v2 . begin () até, mas não incluindo, v2 . end () do vector v2. Os elementos são colocados no vector c começando na posição c . begin . Os iteradores fornecidos como os dois primeiros argumentos devem ser iteradores de entrada. O iterador fornecido como terceiro argumento deve ser um iterador de saída, de modo que o elemento que está sendo copiado possa ser inserido na posição da cópia. Esta função retorna um iterador posicionado após o último elemento copiado no vector c. Note na linha 31 o uso do construtor de vector que recebe o número de elementos no vector e o valor inicial daqueles elementos.
As linhas 44 and 45
newLastElement =
std: : remove if(v3 .beginO, v3 .endO, greater9);
usam a função remove if para deletar todos os elementos no intervalo v3 . begin () até, mas não incluindo, v3 . end () do vector v3 para o qual nossa função predicado unária definida pelo usuário greater9 retorna

using std::cout; using std::endl;

9 #include <algorithm>
10 #include <vector>

11
12
13

14
15
16
17
18
19
20
21
22
23
24
25

26
27

b, .i., .LU

true. A função greater9 é definida na linha 67 de modo a retornar true se o valor passado for maior do que 9, e, caso contrário, retornar false. Os iteradores fornecidos como os dois primeiros argumentos devem ser iteradores para a frente; deste modo, o algoritmo pode modificar os elementos na seqüência. Esta função não modifica o número de elementos no vector, mas ela move de fato para o início do vector todos os elementos que não são eliminados. Esta função retorna um iterador posicionado após o último elemento no vector que não foi excluído. Todos os elementos da posição do iterador até o final do vector têm valores indefinidos.
As linhas 57 e 58
std: :remove copyif(v4.begin, v4.endQ,
c2.begin, greater9);
usam a função remove copy_if para copiar todos os elementos no intervalo v4 . begin () até, mas não incluindo, v4 . end O do vector v4 para os quais a função predicado unária greater9 retorna true. Os elementos são colocados no vector c2 a partir da posição c2 . begin O . Os iteradores fornecidos como os primeiros dois argumentos devem ser iteradores de entrada. O iterador fornecido como terceiro argumento deve ser um iterador de saída, de modo que o elemento que está sendo copiado possa ser inserido na posição da cópia. Esta função retorna um iterador posicionado após o último elemento copiado para o vector c2.

20.5.4 replace. replace_if, replace_copy e replace_copy_if

A Fig. 20.29 demonstra a substituição de valores de uma seqüência usando as funções replace, replaceif, replacecopy e replacecopyif da biblioteca padrão.

6
7
8

```
1 Il Fig. 20.29: fig2029.cpp
2 Il Demonstra as funções replace, replace_if, replace_copy
3 Il e replace_copy_if da biblioteca padrão
4 #include <iostream>
5

bool greater9 ( int );
int main()
const int SIZE = 10;
```

```
inta[ SIZE) = { 10,2,10,4,16,6,14,
std: :ostreamiterator< int > output( cout,
// Substitui lOs em vi por iOOs
std::vector< int > vl( a, a + SIZE );
cout « Vector vi antes da substituição de todos os lOs:\n"; std: :copy( v1.begin,
vl.endO, output );
std::replace( vi.beginO, vl.endO, 10, 100 );
cout « "\nVector vi após a substituição de todos os std: : copy( vi .beginO, vi.
endO, output );
28 // copia de v2 para ci, substituindo i0s por iOOs
29 std::vector< int > v2( a, a + SIZE );
30 std::vector< int > ci( SIZE );
31 cout « "\n\nVector v2 antes da substituição de todos os i0s
32 « "e copia:\n";


i0s por iOOs:\n";
```

Fig. 20.29 Demonstrando as funções replace, replace if, replace_copy e replace_copy_if
da biblioteca padrão (parte 1 de 2).


```
33 std::copy( v2.beginO, v2.end(), output );
34 std: : replace_copy( v2 .beginO, v2 .endO,
35 cl.begin() , 10, 100
36 cout « "\nVector cl depois da substituição de todos os lOs em
37 std::copy( c1.begin, cl.endQ, output );
38
39 II Substitui valores maiores do que 9 em v3 por 100
40 std::vector< int > v3( a, a + SIZE );
41 cout « '\n\nVector v3 antes da substituição de valores maiores"
42 « " do que 9:\n";
43 std: :copy( v3.begin, v3.end(), output );
44 std: :replace if( v3.beginQ, v3.endQ, greater9, 100 );
45 cout « "\nVector v3 após a substituição de todos os valores"
46 « "\nmaiores do que 9 por iOOs:\n";
47 std::copy( v3.beginO, v3.endO, output );
48
49 II Copia v4 para c2, substituindo elementos maiores do que 9 por 100
50 std::vector< int > v4( a, a + SIZE );
51 std::vector< int > c2( SIZE );
52 cout « "\n\nVector v4 antes da substituição de todos os valores
53 « "\nmaiores do que 9 e copia:\n";
54 std::copy( v4.beginQ, v4.endO, output );
55 std: :replace copy if( v4.beginQ, v4.endO, c2.begin,
56 greater9, 100 );
```

```
57 cout « “\nVector c2 após a substituição de todos os valores”
58 « “\nmaiores do que 9 em v4:\n”;
59 std: :copy( c2.beginQ, c2.end(), output );
60
61 cout « endl;
62 return 0;
63
64
65 bool greater9( int x
66
67 return x > 9;
68
Vector vi antes da substituição de todos os lOs:
1021041661481210
Vector vi após a substituição de todos os lOs por lOOs:
100 2 100 4 16 6 14 8 12 100
Vector v2 antes da substituição de todos os lOs e copia:
1021041661481210
Vector ci após a substituição de todos os lOs em v2:
100 2 100 4 16 6 14 8 12 100
Vector v3 antes da substituição de valores maiores do que 9:
1021041661481210
Vector v3 após a substituição de todos os valores
maiores do que 9 por lOOs:
100 2 100 4 100 6 100 8 100 100
Vector v4 antes da substituição de todos os valores
maiores do que 9 e copia:
1021041661481210
Vector c2 após a substituição de todos os valores
maiores do que 9 em v4:
100 2 100 4 100 6 100 8 100 100
```

Fig. 20.29 Demonstrando as funções replace, replace if, replace_copy e replace_copy_if da biblioteca padrão (parte 2 de 2).

A linha 24

```
std::replace( vl.beginQ, vl.endO, 10, 100 );
```

usa a função replace para substituir todos os elementos com o valor 10 no intervalo devi begin () até, mas não incluindo, vi end O do vector vi pelo novo valor 100. Os iteradores fornecidos como os dois primeiros argumentos devem ser iteradores para a frente; deste modo, o algoritmo pode modificar os elementos na seqüência.

As linhas 34 e 35

```
std: :repiace copy( v2 .beginQ, v2 .endO,
cl.beginQ, 10, 100 );
```

usam a função replace copy para copiar todos os elementos no intervalo v2 . begin () até, mas não incluindo, v2 .endO do vector v2, substituindo todos os elementos com o valor 10 pelo novo valor 100. Os elementos são copiados para o vector ci a partir da posição ci begin o. Os iteradores fornecidos como os primeiros dois argumentos devem ser iteradores de entrada. O iterador fornecido como terceiro argumento deve ser um iterador de saída, de modo que o elemento que está sendo copiado possa ser inserido na posição da cópia. Esta função retorna um iterador posicionado após o último elemento copiado para o vector c2.
A linha 44
std::replace_if(v3.beginO, v3.endO, greater9, 100);
usa a função replace if para substituir todos os elementos no intervalo v3 . begin O até, mas não incluindo,
v3 end O do vector v3 para os quais a função predicado unária greater9 retorna true. A função greater9
é definida na linha 65 para retornar true se o valor passado for maior do que 9 e faise caso contrário. O valor
100 substitui cada valor maior do que 9. Os iteradores fornecidos como os primeiros dois argumentos devem ser
iteradores para a frente; deste modo, o algoritmo pode modificar os elementos na seqüência.
As linhas 55 e 56
std: :replace copyif(v4 .beginQ, v4 .end(), c2 .beginO,
greater9, 100);
usam a função repiace copy if para copiar todos os elementos no intervalo v4 .begin O) até, mas não incluindo. v4 . end O) do vector v4. Elementos para os quais a função predicado unária greater9 retorna true são substituídos pelo valor 100. Os elementos são colocados no vector c2 começando na posição c2 begin O. Os iteradores fornecidos como os primeiros dois argumentos devem ser iteradores de entrada. O iterador fornecido como terceiro argumento precisa ser um iterador de saída, de modo que o elemento que está sendo copiado possa ser inserido na posição da cópia. Esta função retorna um iterador posicionado após o último elemento copiado para o vector c2.

20.5.5 Algoritmos matemáticos

A Fig. 20.30 demonstra alguns algoritmos matemáticos comuns da STL. incluindo random shuffie. count. countif, mm element. max element, accumulate. for_each e transform.

Fig. 20.30 Demonstrando alguns algoritmos matemáticos da biblioteca padrão (parte 1 de 3).

1	II Fig. 20.30: fig2O3O.cpp
2	// Exemplos dos aigoritmos matemáticos da Biblioteca Padrão.

3	#include <iostream>
4	
5	using std::cout;
6	using std::endl;
7	


```
8 #inciude <aigorithm>
9 #inciude <nuineric> II accumulate está definido aqui
10 #inciude <vector>
11
12 bool greater9( int );
13 void outputSquare( int );
14 int calculateCube( int );
15
16 int main()
17
18 const int SIZE = 10;
19 int ai[] = { 1, 2, 3, 4, 5, 6, 7, 8, , iv
20 std: :vector< int > v( ai, ai ÷ SIZE );
21 std: :ostreamiterator< int > output( cout, “
22
23 cout « Vector v antes de randomshuf fie:
24 std: :copy( v.begin, v.endQ, output );
25 std: :randomshuffie( v.begmnO, v.end() );
26 cout « ‘\nVector v após random_shuffie:
27 std: :copy( v.begmnO, v.endQ, output );
28
29 int a2[] = { 100, 2, 8, 1, 50, 3, 8, 8, 9, 10 };
30 std::vector< int > v2( a2, a2 + SIZE ); A
31 cout « “\n\nVector v2 contém:
32 std::copy( v2.beginO), v2.endO, output );
33 int resuit = std::count( v2.beginO, v2.endO, 8 );
34 std: :cout « “\nQuantidade de elementos iguais a 8: « result;
35 US
36 result = std::count_if( v2.beginO), v2.endO, greater9 );
37 cout « \nQuantidade de elementos maiores do que 9: « result;
38
39 cout « “\n\nEiemento mínimo em Vector v2 é:
40 « *( std::minelement( v2.beginO, v2.end() ) );
41 us
```

```
42 cout « “\nElemento máximo em Vector v2 é:
43 « *( std::max_eiement( v2.beginO), v2.end() ) );
44
45 cout « “\n\nO total dos elementos em Vector v é:
46 « std: :accuinulate( v.beginO, v.end, O );
47
48 cout « “\n\nO quadrado do todos os inteiros em Vector v é:\n”;
49 std::for_each( v.beginO), v.end, outputSquare ); VE
50
51 std::vector< int > cubes( SIZE );
52 std::transform( v.beginO), v.endO), cubes.begino,
53 caicuiateCube );
54 cout « “\n\nO cubo de todos os inteiros em Vector v
55 std::copy( cubes.beginO, cubes.endO, output );
56
57 cout « endl;
58 return 0;
59
60
61 bool greater9( int value ) { return value > 9;
62
63 void outputSquare ( int value ) { cout « value * value «
64
65 int calculateCube( int value ) { return value * value * value;
```

Fig. 20.30 Demonstrando alguns algoritmos matemáticos da biblioteca padrão (parte 2 de 3).


```
Vector v antes de randomshuffle: 1 2 3 4 5 6 7 8 9 10
Vector v após random_shuf fie: 5 4 1 3 7 8 9 10 6 2
Vector v2 contem: 100 2 8 1 50 3 8 8 9 10
Quantidade de elementos iguais a 8: 3
Quantidade de elementos maiores do que 9: 3
Elemento mínimo em Vector v2 é: 1
Elemento máximo em Vector v2 é: 100
O total de todos os elementos em Vector v é: 55
O quadrado de todos os inteiros em Vector v é:
251619496481100364
O cubo de todos os inteiros em Vector v é:
125 64 1 27 343 512 729 1000 216 8
```

Fig. 20.30 Demonstrando alguns algoritmos matemáticos da biblioteca padrão (parte 3 de 3).

A linha 25

```
std: :randomshuffle( v.beginO, v.end() );
```

usa a função random shuffie para ordenar aleatoriamente os elementos no intervalo v . begin O até, mas não incluindo, v. end () no vector v. Esta função

recebe dois iteradores de acesso aleatório como argumentos.
A linha 33
int resuit = std::count(v2.beginO, v2.endO, 8);
usa a função count para contar os elementos com o valor 8 no intervalo v2 . begin () até, mas não incluindo, v2 . end () no vector v2. Esta função requer que seus dois argumentos sejam no mínimo iteradores de entrada.
A linha 36
result = std::count_if(v2.beginO, v2.endO, greater9);
usa a função count if para contar os elementos no intervalo v2 . begin O até, mas não incluindo, v2 . end O no vector v2 para os quais a função predicado greater9 retorna true. A função count if requer que seus dois argumentos iteradores sejam no mínimo iteradores de entrada.
As linhas 39 e 40
cout « "\n\nElemento mínimo em Vector v2 é:
« (std::min_element(v2.beginO, v2.end()));
usam a função mm eiement para localizar o menor elemento no intervalo v2 . begin () até, mas não incluindo, v2 .end() no vector v2. A função retorna um iterador de entrada apontando para o menor elemento ou, se o intervalo está vazio, retorna o próprio iterador. A função requer que seus dois argumentos iteradores sejam no mínimo iteradores de entrada. Uma segunda versão desta função aceita como seu terceiro argumento uma função binária que compara os elementos na seqüência. A função aceita dois argumentos e retoma um valor bool.
Boa prática de programação 20.2
É uma boa prática verificar se o intervalo especWcado em uma chamada para mm element não está
vazio e verificar se o valor de retorno não é um iterador apontando para "depois do final".

As linhas 42 e 43
cout « "\nElemento máximo em Vector v2 é:
« (std::max_element(v2.beginO, v2.end()));
usam a função max element para localizar o maior elemento no intervalo v2 begin () até, mas não incluindo, v2 . end () no vector v2. A função retorna um iterador de entrada apontando para o maior elemento. A função requer que seus dois argumentos sejam no mínimo iteradores de entrada. Uma segunda versão desta função recebe como seu argumento uma função predicado binária que compara os elementos na seqüência. A função binária aceita dois argumentos e retoma um valor bool.
As linhas 45 e 46
cout « "\n\nO total dos elementos em Vector v é:
« std: : accumulate (v.begin, v. end, O);
usam a função accumulate (cujo protótipo está no arquivo de cabeçalho <numeric>) para somar os valores no intervalo v begin () até, mas não incluindo. v. end () no vector v. Os dois argumentos iteradores desta função devem ser no

mínimo iteradores de entrada. Uma segunda versão desta função aceita como seu terceiro argumento uma função genérica que determina como os elementos são acumulados. A função genérica precisa receber dois argumentos e retorna um resultado. O primeiro argumento para esta função é o valor acumulado corrente. O segundo argumento é o valor do elemento corrente na seqüência sendo acumulada. Por exemplo, para acumular a soma dos quadrados de cada elemento, pode-se usar a função

int sumOfSquares(int accumulator, int currentValue
return accumulator + currentValue * currentValue;

que recebe o total anterior como seu primeiro argumento (accumulator) e o novo valor para elesar ao quadrado e somar ao total como seu segundo argumento (currentValue). Quando é chamada a função, ela eleva ao quadrado currentValue. soma o quadrado de currentValue ao accumulator e retorna o novo total.

A linha 49

std: :for_each(v.begin, v.endO, outputSquare);

usa a função for_each para aplicar uma função genérica sobre cada elemento no intervalo v . begin até, mas não incluindo, v. end () no vector v. A função deve receber o elemento corrente como um argumento e não deve modificar tal elemento. A função for_each requer que seus dois argumentos iteradores sejam no mínimo iteradores de entrada.

As linhas 52 e 53

std: :transform(v.begin, v.endO, cubes.begin(),
calculateCube);

usam a função transformpara aplicaruma função genérica sobre todos os elementos no intervalo v.bgin O até, mas não incluindo, v. end () no vector v. A função genérica (o quarto argumento) deve receber o elemento corrente como um argumento, não deve modificar o elemento e deve retornar o valor transformado. A função trans form requer que seus dois primeiros argumentos iteradores sejam no mínimo iteradores de entrada e que seu terceiro argumento seja pelo menos um iterador de saída. O terceiro argumento especifica onde os valores transformados devem ser colocados. Note que o terceiro argumento pode ser igual ao primeiro.

20.5.6 Algoritmos básicos de pesquisa e classificação

A Fig. 20.31 demonstra algumas recursos básicos de pesquisa e classificação da biblioteca padrão, incluindo f ind. findif, sort e binary_search.


```
1 II Fig. 20.31: fig2O3l.cpp
2 // Demonstra recursos de pesquisa e classificação.
3 #include <iostream>
4
5 using std::cout;
6 using std: :endl;
7
8 #include <algorithxn>
```

```
9 #include <vector>
10
11 bool greaterlO ( int value );
12
13 int main()
14
15 const int SIZE = 10;
16 int a[ SIZE ] = { 10, 2, 17, 5, 16, 8, 13, 11, 20, 7 };
17 std: :vector< int > v( a, a + SIZE );
18 std: :ostreamiterator< int > output( cout, “ “
19
20 cout « Vector v contém: “;
21 std: :copy( v.beginO, v.endO, output );
22
23 std::vector< int >::iterator location;
24 location = std::find( v.begin, v.end(), 16 );
25
26 if ( location != v.endO
27 cout « “\n\nEncontrou 16 na posição
28 « ( location - v.begin() );
29 else
30 cout « “\n\n16 não encontrado;
31
32 location = std::find( v.begin, v.endO, 100 );
33
34 if ( location != v.endQ
35 cout « \nEncontrou 100 na posição
36 « ( location - v.begin() );
37 else
38 cout « “\nlOO não encontrado”;
39
40 location = std::find_if( v.beginO, v.endO, greaterlO );
41
42 if ( location ‘= v.end()
43 cout « “\n\nO primeiro valor maior do que 10 é
44 « *l,tjn « \nencontrado na posição
45 « ( location - v.begin() );
46 else
47 cout « “\n\nNenhum valor maior do que 10 foi encontrado’;
48
49 std::sort( v.beginO, v.end() );
50 cout « “\n\nVector v após classificação: “;
51 std::copy( v.begin(), v.endO, output );
52
53 if O std::binary_search( v.begin, v.endO, 13
54 cout « “\n\n13 foi encontrado em v”;
```

```
55 else
56 cout « “\n\nl3 não foi encontrado em v”;
57
```
Fig. 20.31 Algoritmos básicos de pesquisa e classificação da biblioteca padrão (parte 1 de 2).


```
58 if ( std: :binary search( v.beginO, v.endQ, 100
59 cout « “\nlOO foi encontrado em v”;
60 else
61 cout « “\nlOO não foi encontrado em v”;
62
63 cout « endl;
64 return 0;
65
66
67 bool greaterl0( int value ) { return value > 10;
Vector v contem: 10 2 17 5 16 8 13 11 20 7
Encontrado 16 na posição 4
100 não encontrado
O primeiro valor maior do que 10 é 17
encontrado na posição 2
Vector v após classificação: 2 5 7 8 10 11 13 16 17 20
13 foi encontrado em v
100 não foi encontrado em v
```
Fig. 20.31 Algoritmos básicos de pesquisa e classificação da biblioteca padrão (parte 2 de 2).

A linha 24

location = std::find(v.begin, v.endO, 16);

usa a função f ind para localizar o valor 16 no intervalo v . begin () até, mas não incluindo, v. end () no vector v. A função requer que seus dois argumentos iteradores sejam, no mínimo, iteradores de entrada. A função retorna um iterador de entrada que está posicionado ou no primeiro elemento contendo o valor ou um iterador indica o fim da seqüência.

A linha 40

location std::find_if(v.begin, v.end, greaterlO);

usa a função find_if para localizar o primeiro valor no intervalo v begin O até, mas não incluindo, v. end O no vector v para o qual a função predicado unária greaterlO retoma true. A função greaterlO está definida na linha 65 para receber um inteiro e retornar um valor bool indicando se o argumento inteiro é maior do que 10. A função find if requer que seus dois argumentos iteradores sejam no mínimo iteradores de entrada. A função retorna um iterador de entrada que é posicionado ou no primeiro elemento contendo um valor para qual a função predicado retoma true ou um iterador indicando o fim da seqüência.

A linha 49

std::sort(v.beginO, v.end());

usa a função sort para ordenar os elementos no intervalo v . begin O até, mas não incluindo, v. end O no vector v, em ordem ascendente. A função requer que seus dois argumentos iteradores sejam iteradores de acesso aleatório. Uma segunda versão desta função aceita um terceiro argumento que é uma função predicado binária que recebe dois argumentos que são valores na seqüência e retorna um bool indicando a ordem de classificação - se o valor retornado é true, os dois elementos comparados estão ordenados.

Erro com um de programação 20.5

A linha 53
if (std: :binary_search(v.beginQ, v.endO, 13
usa a função binary_search para determinar se o valor 13 está no intervalo v . begin () até, mas não incluindo, v . end () no vector v. A seqüência de valores deve ser primeiro classificada em ordem ascendente. A função binary_search requer que seus dois argumentos iteradores sejam, no mínimo, iteradores para a frente. A função retorna um bool indicando se o valor foi ou não encontrado na seqüência. Uma segunda versão desta função aceita um quarto argumento que é uma função predicado binária que recebe dois argumentos que são valores na seqüência e retorna um bool. A função predicado retorna true se os dois elementos sendo comparados estão ordenados.

20.5.7 swap, iter_swap e swap_ranges

A Fig. 20.32 demonstra as funções swap. iter_swap e swap ranges para permutar elementos.

```
1 II Fig. 20.32: fig2032.cpp
2 II Demonstra iterswap, swap e swap_ranges.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <algorithm>
9
10 int main()
11
12 const int SIZE = 10;
13 inta[SIZE]={1, 2,3,4,5,6,7,8,9, i0};
14 std: :ostreamiterator< int > output( cout, " );
15
16 cout « "Array a contém:\n";
17 std: :copy( a, a + SIZE, output );
18
19 std::swap( a[ O ), a[ 1 1 );
20 cout « "\nArray a após permutação de aIO) e a[1]
21 « "usando swap:\n";
22 std: :copy( a, a + SIZE, output );
24 std: :iterswap( &a[ O ] , &a[ 1 ] )
```

```
25 cout « “\nArray a após permutação de a[0) e ali)
26 « “usando iterswap:\n”;
27 std: :copy( a, a + SIZE, output );
28
29 std: : swap_ranges ( a, a + 5, a + 5 );
30 cout « “\nArray após a permutação dos primeiros cinco elementos\n”
31 « “pelos últimos cinco elementos:\n”;
32 std::copy( a, a + SIZE, output );
33
34 cout « endi;
35 return 0;
36 }
```


Tentar ordenar um contêiner usando um iterador que não seja um iterador de acesso aleatório é um erro de sintaxe. A função sort requer um iterador de acesso aleatório.

Fig. 20.32 Demonstrando swap. iter swap e swap ranges (parte 1 de 2).


```
Array a contém:
1 2 3 4 5 6 7 8 9 10
Array a após permutação de a[0] e a[1] usando swap:
2 1 3 4 5 6 7 8 9 10
Array a após permutação de a[0] e a[l] usando iter_swap:
1 2 3 4 5 6 7 8 9 10
Array a após permutação dos primeiros cinco elementos
pelos últimos cinco elementos:
6 7 8 9 10 1 2 3 4 5
```

Fig. 20.32 Demonstrando swap, iter_swap e swap ranges parte 2 de .

A linha 19

```
std::swap( a[ O ], a[ 1 ] );
```

usa a função swap para permutar dois valores. Neste exemplo, o primeiro e o segundo elementos do array a são permutados. A função recebe como argumentos referências para os dois valores que estão sendo permutados.

A linha 24

```
std: :iter_swap( &a[ O ] , &a 1
```

usa a função iter_swap para permutar os dois elementos. A função aceita dois argumentos iteradores para a frente (neste caso, ponteiros para elementos de um array) e permuta os valores nos elementos aos quais os iteradores se refere.

A linha 29

```
std: :swap_ranges( a, a + 5, a + 5 );
```

usa a função swap ranges para permutar os elementos no intervalo a até, mas não incluindo, a ÷ 5 com os elementos começando na posição a + 5. A função

requer três argumentos iteradores para a frente. Os dois primeiros argumentos especificam o intervalo de elementos na primeira seqüência que será trocado com os elementos na segunda seqüência começando a partir do iterador especificado como terceiro argumento. Neste exemplo, as duas seqüências de valores estão no mesmo array, mas as seqüências podem ser de arrays ou contêineres diferentes.

20.5.8 copy_backward, merge, unique e reverse

A Fig. 20.33 demonstra as funções copy backward, merge. unique e reverse da biblioteca padrão.

Fig. 20.33 Demonstrando copy backward, merge, unique e reverse (parte 1 de 2).

```
1  II Fig. 20.33: fig2033.cpp  II Demonstra
2  diversas funções:  II copy_backward,
3  merge, unique e reverse.  #include
4  <iostream>
5
6  using std: :cout; using std: :endl;
7
8
9  #include <algorithm> #include
10 <vector>
11
12 int main()
13 {
```



```
14 const int SIZE = 5;
15 intai[SIZE]{1,3,5,7,9};
16 inta2[SIZE]={2,4,5,7,9};
17 std::vector< int > vl( ai, ai + SIZE );
18 std::vector< int > v2( a2, a2 + SIZE );
19
20 std: :ostreamiterator< int > output( cout, );
21
22 cout « “Vector vi contém: “;
23 std: :copy( vl.beginQ, vl.endO, output );
24 cout « \nVector v2 contém:
25 std::copy( v2.beginO, v2.endO, output );
26
```

```
27 std::vector< int > results( vl.size() );
28 std: : copy_backward( vi .beginO, vi .end, results .end() );
29 cout « “\n\nApós copybackward, resuits contém: “;
30 std::copy( results.begin, results.endO, output );
31
32 std::vector< int > resuits2( vl.size() + v2.size() );
33 std: :merge( vl.beginO, vl.endQ, v2.beginO, v2.endO,
34 results2.begin() );
35 cout « “\n\nApós intercalação de vi e v2, resuits2 contém:\n’;
36 std::copy( results2.begin, results2.endQ, output );
37
38 std: :vector< int >: :iterator endLocation;
39 endLocation =
40 std: :unique( resuits2 .beginO, resuits2.end() );
41 cout « “\n\nApós unique, results2 contém:\n”;
42 std::copy( resuits2.begin, endLocation, output );
43
44 cout « “\n\nVector vi após reverse: ;
45 std: :reverse( vi.beginQ, vi.end() );
46 std: :copy( vi.begin(), vi.endO, output );
47
48 cout « endi;
49 return 0;
50 }
```

```
Vector vi contém: 1 3 5 7 9
Vector v2 contém: 2 4 5 7 9
Após copybackward, resuits contém: 1 3 5 7 9
Após intercalação de vi e v2, results2 contém:
1234557799
Após unique, results2 contém:
1234579
Vector vi após reverse: 9 7 5 3 1
```

Fig. 20.33 Demonstrando copy backward, merge, unique e reverse (parte 2 de 2).

A linha 28

```
std: :copybackward( vi .beginO, vi .endO, results .end() );
```

usa a função copy_backward para copiar elementos no intervalo vi . begin () até, mas não incluindo, vi . end O no vector vi e colocar os elementos no vector results começando do elemento antes de results . end ()

e trabalhando em direção ao começo do vector. A função retorna um iterador posicionado no último elemento copiado para o vector resuits (isto é, o começo de resuits, porque estamos indo para trás). Os elementos são colocados em resuits na mesma ordem que cmvi. Esta função requer três argumentos iteradores bidirecionais (iteradores que podem ser incrementados e

decrementados para percorrer uma seqüência para frente e para trás, respectivamente). A principal diferença entre copy e copy backward é que o iterador retornado de copy é posicionado após o último elemento copiado e o iterador retornado de copy_backward é posicionado no último elemento copiado (que é, na verdade, o primeiro elemento na seqüência). Além disso, copy também requer dois iteradores de entrada e um iterador de saída como argumentos.

As linhas 33 e 34

```
std: :merge( vi .beginO, vi .endO, v2 .beginO, v2 .end,
results2.begin() );
```

usam a função merge para intercalar duas seqüências de valores classificados em ordem ascendente em uma terceira seqüência classificada em ordem ascendente. A função requer cinco iteradores como argumentos. Os quatro primeiros argumentos devem ser no mínimo iteradores de entrada e o último argumento deve ser no mínimo um iterador de saída. Os dois primeiros argumentos especificam o intervalo de elementos na primeira seqüência ordenada (vi), os dois segundos argumentos especificam o intervalo de elementos na segunda seqüência ordenada (v2) e o último argumento especifica a posição inicial na terceira seqüência (resuits2) onde os elementos serão intercalados. Uma segunda versão dessa função aceita como seu quinto argumento uma função predicado binária que especifica a ordem de classificação.

Note que a linha 32 cria o vector resuits com o número de elementos vi. size () + v2 . size O. Usar a função merge como mostrado aqui requer que a seqüência na qual os resultados serão armazenados seja no mínimo do tamanho das duas seqüências que estão sendo intercaladas. Se não se quiser alocar o número de elementos para a seqüência resultante antes da operação merge. pode-se usar as seguintes declarações:

```
vector< int > resuits2O;
merge ( v1.begin, vl.endO, v2.beginO, v2.endO,
backinserter( resuits2 ) );
```

O argumento back inserter (resuits2) usa a função gabarito back_inserter (arquivo de cabeçalho <iterator>) para o contêiner resuits2. Uma back_inserter chama a função push back default do contêiner para inserir um elemento no final do contêiner. Mais importante, se um elemento é inserido em um contêiner que não tem mais elementos disponíveis, o contêiner cresce em tamanho. Portanto, o número de elementos no contêiner não precisa ser conhecido antecipadamente. Há outras duas funções de inserção - front_inserter (para inserir um elemento no começo de um contêiner especificado como seu argumento) e inserter (para inserir um elemento antes de um iterador fornecido como seu segundo argumento no contêiner fornecido como seu primeiro argumento).

As linhas 39 e 40

```
endLocation =
std: :unique( resuits2.begin, resuits2.end() );
```

usam a função unique sobre a seqüência classificada de elementos no intervalo

resuits2 . begin O até, mas não incluindo, resuits2 . end O no vector resuits2. Após essa função haver sido aplicada a uma seqtiência ordenada com valores duplicados, somente uma única cópia de cada valor permanece na seqüência. A função recebe dois argumentos que precisam ser no mínimo iteradores para a frente. A função retorna um iterador posicionado após o último elemento na seqüência de valores únicos. Os valores de todos os elementos no contêiner após o último valor único são indefinidos. Uma segunda versão dessa função recebe como um terceiro argumento uma função predicado binária especificando como comparar dois elementos quanto à igualdade.
A linha 45
std: :reverse(vi.beginO, vi.end());

usa a função reverse para colocar em ordem inversa todos os elementos no intervalo vi . begin () até, mas não incluindo, vi . end () no vector vi. A função recebe dois argumentos que devem ser no mínimo iteradores bidirecionais.

20.5.9 inplace merge, unique_copy e reverse_copy

O programa da Fig. 20.34 demonstra as funções inpiace merge, unique copy e reverse copy da biblioteca padrão. [Nota: o programa a seguir não compila no Borland C++j.

```
1 II Fig. 20.34: fig2034.cpp
2 II Demonstra diversas funções:
3 // inpiacemerge, reverse_copy e unique_copy.
4 #include <iostream>
5
6 using std::cout;
7 using std::endi;
8
9 #include <aigorithni>
10 #inciude <vector>
11 #include <iterator>
12
13 int main()
14 {
15 const int SIZE = 10;
16 intal[SIZE]={1, 3,5,7,9,1,3,5,7, 9};
17 std::vector< int > vi( ai, ai + SIZE );
18
19 std: :ostreamiterator< int > output( cout, " "
20
21 cout « "Vector vi contém:
22 std::copy( v1.begin, vl.endO, output );
23
24 std: :inpiace_merge( vl.begmnO), vl.begin() + 5, vi.end() );
25 cout « "\nApos inpiacemerge, vi contém: ";
26 std::copy( vl.beginQ, vi.end(), output );
```

```
27
28 std::vector< int > resuitsl;
29 std: :unique_copy( vl.beginO), vl.endO),
30 std::back_inserter( resuitsi ) );
31 cout « “\nApós uniquecopy, resuitsi contém:
32 std::copy( resuitsl.beginQ, resuitsi.end, output );
33
34 std::vector< int > results2;
35 cout « “\nApós reversecopy, resuits2 contém: “;
36 std: :reverse_copy( vi.beginO), vl.endO),
37 std::backinserter( resuits2 ) );
38 std::copy( resuits2.beginO), resuits2.endO, output );
39
40 cout « endi;
41 return 0;
42
Vector vi contém: i 3 5 7 9 1 3 5 7 9
Após inpiacemerge, vi contém: 1 1 3 3 5 5 7 7 9 9
Após uniquecopy, resuitsi contém: 1 3 5 7 9
Após reverse_copy, resuits2 contém: 9 9 7 7 5 5 3 3 1 1
```
Fig. 20.34 Demonstrando inpiace_merge, unique_copy e reverse_copy.


```
1 II Fig. 20.35: fig2035.cpp
2 // Demonstra as funções includes, set_difference,
3 II set_intersection, setsymmetricdifference e set_union.
4 #include <iostream>
5
6 using std::cout;
7 using std::endl;
8
9 #inciude <algorithm>
10
ii int main()
12 {
13 const int SIZE1 = 10, SIZE2 = 5, SIZE3 = 20;
14 intai[SIZE1] ={i, 2,3,4,5,6,7,8,9, iO};
15 inta2[SIZE2]{4,5,6,7,8};
16 int a3[ SIZE2 ] = { 4, 5, 6, li, 15 };
17 std: :ostreamiterator< int > output( cout, “ );
18
19 cout « “ai contém: “;
20 std::copy( ai, ai + SIZEi, output );
21 cout « \na2 contém: ‘;
22 std: :copy( a2, a2 + SIZE2, output );
23 cout « “\na3 contém: ;
```

```
24 std::copy( a3, a3 + SIZE2, output );
25
26 if ( std::includes( ai, ai + SIZE1, a2, a2 + SIZE2
27 cout « “\nai inclui a2’;
28 else
29 cout « “\nai não inclui a2”;
30
31 if ( std::includes( ai, ai + SIZEi, a3, a3 + SIZE2
32 cout « “\nal inclui a3’;
33 else
34 cout « “\nai não inclui a3”;
35
36 int difference[ SIZE1 ];
37 int *ptr = std::setdifference( ai, ai + SIZE1,
38 a2, a2 + SIZE2, difference );
39 cout « “\nsetdifference de ai e a2 é:
40 std: :copy( difference, ptr, output );
41
42 int intersection[ SIZE1 ];
43 ptr = std::setintersection( ai, ai + SIZE1,
44 a2, a2 + SIZE2, intersection );
45 cout « “\nsetintersection de ai e a2 é:
46 std::copy( intersection, ptr, output );
47
48 int symmetricdifference[ SIZEi ];
49 ptr = std::setsymmetricdifference( ai, ai + SIZE1,
50 a2, a2 + SIZE2, syinmetricdifference );
51 cout « ‘\nsetsymmetricdifference de ai e a2 é:
52 std::copy( symmetricdifference, ptr, output );
53
54 int unionSet[ SIZE3 );
55 ptr = std::set_union( ai, ai + SIZE1,
56 a3, a3 + SIZE2, unionSet );
57 cout « ‘\nsetunion de ai e a3 é:
```

Fig. 20.35 Demonstrando as operações set da biblioteca padrão (parte 1 de 2).


```
58 std::copy( unionSet, ptr, output );
59 cout « endi;
60 return 0;
61
ai contém: 1 2 3 4 5 6 7 8 9 10
a2 contém: 4 5 6 7 8
a3 contém: 4 5 6 ii 15
ai inciui a2
```

ai não inciui a3
setdifference de ai e a2 é: 1 2 3 9 10
setintersection de ai e a2 é: 4 5 6 7 8
setsymmetricdifference de ai e a2 é: 1 2 3 9 10
set_uniondeai ea3 é: 123456789 iO ii i5
Fig. 20.35 Demonstrando as operações set da biblioteca padrão (parte 2 de 2).
As linhas 37 e 38
int *ptr = std::setdifference(ai, ai + SIZE1,
a2, a2 + SIZE2, difference);
usam a função set_difference para determinar os elementos do primeiro conjunto de valores ordenados que : não estão no segundo conjunto de valores ordenados (os dois conjuntos de valores devem estar em ordem ascendente). Os elementos que são diferentes são copiados no quinto argumento (neste caso, o array difference). Os dois primeiros argumentos iteradores devem ser no mínimo iteradores de entrada para o primeiro conjunto de valores. Os próximos dois argumentos iteradores devem ser no mínimo iteradores de entrada para o segundo conjunto de valores. O quinto argumento deve ser no mínimo um iterador indicando onde armazenar uma cópia dos valores que são diferentes. A função retorna um iterador de entrada posicionado imediatamente após o último valor copiado no conjunto para o qual o quinto argumento aponta. Uma segunda versão da função set difference aceita um sexto argumento que é uma função predicado binária indicando a ordem na qual os elementos foram originalmente classificados. As duas seqüências devem estar ordenadas usando-se a mesma função de comparação.
A linha 43 e 44
ptr = std::setintersection(ai, ai + SIZEi,
a2, a2 + SIZE2, intersection);
usam a função set intersection para determinar os elementos do primeiro conjunto de valores ordenados que estão no segundo conjunto de valores ordenados (os dois conjuntos de valores devem estarem ordem ascendente). Os elementos comuns a ambos os conjuntos são copiados para dentro do quinto argumento (neste caso, o array intersection). Os dois primeiros argumentos iteradores devem ser no mínimo iteradores de entrada para o primeiro conjunto de valores. Os dois argumentos iteradores seguintes devem ser no mínimo iteradores de entrada para o segundo conjunto de valores. O quinto argumento deve ser no mínimo um iterador de saída indicando onde armazenar a cópia dos valores que são diferentes. A função retorna um iterador posicionado imediatamente após o último valor copiado para o conjunto para o qual o quinto argumento aponta. Uma segunda versão da função set_intersection recebe um sexto argumento que é uma função predicado binária indicando a ordem na qual os elementos estavam originalmente classificados. As duas seqüências devem estar ordenadas usando a mesma função de comparação.
As linhas 49 e 50
ptr = std::setsymmetricdifference(ai, ai + SIZE1,
a2, a2 + SIZE2, symmetricdifference);
usam a função set symmetricdifference para determinar os elementos no

primeiro conjunto que não estejam no segundo conjunto e os elementos no segundo conjunto que não estão no primeiro conjunto (os dois

conjuntos de valores devem estarem ordem ascendente). Os elementos que são diferentes são copiados de ambos os conjuntos para dentro do quinto argumento (neste caso, o arranjo symmetric_difference). Os dois primeiros argumentos iteradores devem ser no mínimo iteradores de entrada para o primeiro conjunto de valores. Os dois argumentos iteradores seguintes devem ser no mínimo iteradores de entrada para o segundo conjunto de valores. O quinto argumento deve ser no mínimo um iterador indicando onde armazenar a cópia dos valores que são diferentes. A função retorna um iterador posicionado imediatamente após o valor copiado para dentro do conjunto para o qual quinto argumento aponta. Uma segunda versão da função set symmetric_difference recebe um sexto argumento que é uma função predicado binária indicando a ordem na qual os elementos estavam originalmente classificados. As duas seqüências devem estar classificadas usando a mesma função de comparação.
As linhas 55 e 56
```
ptr = std::set_union( ai, ai + SIZE1,
a3, a3 + SIZE2, unionSet );
```
usam a função set_union para criar um conjunto de todos os elementos que estão em um, ou em outro ou nos dois conjuntos ordenados (os dois conjuntos de valores devem estar em ordem ascendente). Os elementos são copiados de ambos os conjuntos para o quinto argumento (neste caso, o array unionSet). Elementos que aparecem em ambos os conjuntos são somente copiados do primeiro conjunto. Os dois primeiros argumentos devem ser no mínimo iteradores de entrada para o primeiro conjunto de valores. Os dois argumentos devem ser no mínimo iteradores de entrada para o segundo conjunto de valores. O quinto argumento deve ser no mínimo um iterador de saída indicando onde armazenar os elementos copiados. A função retoma um iterador de saída posicionado imediatamente após o último valor copiado no conjunto para o qual o quinto argumento aponta. A segunda versão da função set_union recebe um sexto argumento que é uma função predicado binária indicando a ordem em que os elementos estavam originalmente classificados. As duas seqüências devem estar ordenadas usando-se a mesma função de comparação.

20.5.11 lower bound, upper_bound e equal_range

A Fig. 20.36 demonstra as funções lower bound, upper_bound e equal range da biblioteca padrão.

```
1 II Fig. 20.36: fig2036.cpp
2 // Demonstra iowerbound, upper_bound e equal range
3 II para uma seqüência de valores ordenados.
4 #include <iostream>
5
6 using std::cout;
7 using std::endi;
```

```
8
9 #inciude <algorithin>
10 #include <vector>
ii
12 int main()
13
14 const int SIZE = 10;
15 intai[]{2, 2,4,4,4,6,6,6,6,8);
16 std::vector< int > v( ai, ai + SIZE );
17 std: :ostreamiterator< int > output( cout, " ' );
18
19 cout « 'Vector v contém:\n";
20 std: :copy( v.begmnO, v.endQ, output );
21
22 std: :vector< int >: :iterator lower;
23 lower = std::iower_bound( v.begmnO), v.endO, 6 );
24 cout « "\n\nLimite inferior de 6 é o elemento
25 « ( iower - v.begmn() ) « " do vector v";
```

Fig. 20.36 Demonstrando iower bound, upper bound e equal range (parte 1 de 2).


```
26
27 std: :vector< int >: :iterator upper;
28 upper = std: :upper_bound( v.beginO, v.endQ, 6 );
29 cout « "\nLimite superior de 6 é o elemento
30 « ( upper - v.begin() ) « " do vector v";
31
32 std::pair< std::vector< int >::iterator,
33 std: :vector< int >: :iterator > eq;
34 eq = std::equal_range( v.begin, v.endO, 6 );
35 cout « "\ntjsando equal_range:\n"
36 « " Limite inferior de 6 é o elemento
37 « ( eq.first - v.begin() ) « " do vector v";
38 cout « "\n Limite superior de 6 é o elemento
39 « ( eq.second - v.begin() ) « " do vector v";
40
41 cout « "\n\nUsando lowerbound para localizar o primeiro ponto\n"
42 « "no qual 5 pode ser inserido em ordem";
43 lower = std::lowerbound( v.begin, v.endO, 5 );
44 cout « "\n Limite inferior de 5 é o elemento
45 « ( lower - v.begin() ) « " do vector v";
46
47 cout « "\n\nUsando upper bound para localizar o último ponto\n"
48 « "no qual 7 pode ser inserido em ordem";
```

```
49 upper = std::upperbound( v.beginO, v.endO, 7 );
50 cout « “\n Limite superior de 7 é o elemento
51 « ( upper - v.begin() ) « “ do vector v”;
52
53 cout « “\n\nUsando equal_range para localizar o primeiro e o\n”
54 « “último ponto nos quais 5 pode ser inserido em ordem”;
55 eq = std::equal_range( v.begin, v.endO, 5 );
56 cout « “\n Limite inferior de 5 é o elemento
57 « ( eq.first - v.begin() ) « “ do vector v”;
58 cout « “\n Limite superior de 5 é o elemento
59 « ( eq.second - v.begin() ) « “ do vector v”
60 « endl;
61 return 0;
62
Vector v contém:
2244466668
Limite inferior de 6 é o elemento 5 do vector v
Limite superior de 6 é o elemento 9 do vector v
Usando equal_range:
Limite inferior de 6 é o elemento 5 do vector v
Limite superior de 6 é o elemento 9 do vector v
Usando lower_bound para localizar o primeiro ponto
no qual 5 pode ser inserido em ordem
Limite inferior de 5 é o elemento 5 do vector v
Usando upper_bound para localizar o último ponto
no qual 7 pode ser inserido em ordem
Limite superior de 7 é o elemento 9 do vector v
Usando equal_range para localizar o primeiro e o
ultimo ponto nos quais 5 pode ser inserido em ordem
Limite inferior de 5 é o elemento 5 do vector v
Limite superior de 5 é o elemento 5 do vector v
```

Fig. 20.36 Demonstrando lower bound, upper bound e equal range (parte 2 de 2).

A linha 23
lower = std: :lower_bound(v.beginO, v.endO, 6);
usa a função lower bound para determinar a primeira posição em uma seqüência de valores ordenados na qual o terceiro argumento poderia ser inserido na seqüência. de modo que, ainda assim, a seqüência estaria classificada em ordem ascendente. Os dois primeiros argumentos iteradores devem ser no mínimo iteradores para a frente. O terceiro argumento é o valor para o qual se quer determinar o limite inferior. A função retorna um iterador para a frente apontando para a posição onde a inserção pode ocorrer. Uma segunda versão da função lower bound aceita como um quarto argumento uma função predicado

binária indicando a ordem na qual os elementos estavam originalmente classificados.
A linha 28
upper = std::upper_bound(v.beginO, v.endO, 6);
usa a função upper_bound para determinar a última posição em uma seqüência ordenada de valores na qual o terceiro argumento poderia ser inserido na seqüência, de modo que, ainda assim, a seqüência estaria classificada em ordem ascendente. Os dois primeiros argumentos iteradores devem ser no mínimo iteradores para a frente. O terceiro argumento é o valor para o qual se quer determinar o limite superior. A função retorna um iterador para a frente apontando para a posição na qual a inserção pode ocorrer. Uma segunda versão da função upper bound aceita como quarto argumento uma função predicado binária indicando a ordem na qual os elementos estavam original- mente classificados.
A linha 34
eq = std::equal_range(v.begin, v.endO, 6);
usa a função equal range para retornar um pair de iteradores para a frente contendo os resultados combinados de executar tanto uma operação lower bound como uma upper_bound. Os dois primeiros argumentos iteradores devem ser iteradores para a frente. O terceiro argumento é o valor ao qual se quer aplicar equal range. A função retorna um pair de iteradores para o limite inferior (eq. first) e o limite superior (eq. second), respectiva- mente.
As funções lower bound, upper bound e equal range são freqüentemente usadas para determinar
a posição de inserção em seqüências ordenadas. A linha 40 usa lower_bound para localizar o primeiro ponto no
qual 5 pode ser inserido na ordem em v. A linha 46 usa upper bound para localizar o último ponto no qual 7
pode ser inserido na ordem em v. A linha 52 usa equal range para localizar o primeiro e último pontos no qual
5 pode ser inserido na ordem em v.
20.5.12 Heapsort
A Fig. 20.37 demonstra as funções da Biblioteca Padrão para executar o algoritmo de classificação heapsort. Heapsort é um algoritmo de classificação no qual um array de elementos é organizado em uma árvore binária especial chamada de heap. O fator-chave para o bom desempenho de um heap é que o maior elemento esteja sempre no topo do heap e os valores dos filhos de qualquer nodo na árvore binária sejam sempre menores que ou iguais ao valor daquele nodo. Um heap organizado desta maneira é freqüentemente chamado de maxheap. O heapsort é geralmente estudado em disciplinas de Ciência da Computação chamadas de “Estruturas de dados” e “Algoritmos.”

Fig. 20.37 Usando funções da biblioteca padrão para executar um heapsort (parte 1 de 3).

1	II Fig. 20.37: fig20_37.cpp
2	// Demonstrando push_heap, pop_heap, make_heap e sort_heap.
3	#include <iostream>
4	
5	using std: :cout;


```
6 using std::endl;
7
8 #include <algorithm>
9 #include <vector>
10
11 int main()
12
13 const int SIZE = 10;
14 int a[ SIZE ) = { 3, 100, 52, 77, 22, 31, 1, 98, 13, 40 );
15 int i;
16 std: :vector< int > v( a, a + SIZE ), v2;
17 std: :ostream jterator< int > output( cout, " " );
18
19 cout « "Vector v antes de make_heap:\n";
20 std: :copy( v.beginO, v.end(), output );
21 std::make_heap( v.beginQ, v.endO );
22 cout « "\nVector v após make_heap:\n";
23 std: :copy( v.beginO), v.endO, output );
24 std: :sort_heap( v.begmn() , v.end() );
25 cout « "\nVector v após sort_heap:\n";
26 std: :copy( v.begin, v.endO, output );
27
28 // executa o heapsort com push_heap e pop_heap
29 cout « "\n\nArray a contém:
30 std: :copy( a, a ÷ SIZE, output );
31
32 for ( i = 0; i < SIZE; ++i
33 v2.push_back( a[ i 1 );
34 std: :push_heap( v2 .begmnO), v2 .end() );
35 cout « "\nv2 após push_heap(a[" « i « "1):
36 std::copy( v2.beginO), v2.end(), output );
37
38
39 for ( i = 0; i < v2.sizeQ; ++i
```

```
40 cout « “\nv2 após “ « v2[ O ] « “ ser retirado do heap\n”;
41 std::pop_heap( v2.begmnQ, v2.endO - i );
42 std::copy( v2.begmn, v2.end(), output );
43 }
44
45 cout « endi;
46 return 0;
47 }
Vector v antes de make_heap:
3 100 52 77 22 31 1 98 13 40
Vector v após make_heap:
100 98 52 77 40 31 1 3 13 22
Vector v após sort_heap:
1 3 13 22 31 40 52 77 98 100
Array a contém: 3 100 52 77 22 31 1 98 13 40
v2 após push heap(a[0]) : 3
v2 após pushheap(a[1]): 100 3
v2 após pushheap(a[2fl: 100 3 52
v2 após push_heap(a[3)): 100 77 52 3
```

Fig. 20.37 Usando funções da biblioteca padrão para executar um heapsort (parte 2 de 3).


```
v2 após pushheap(a[4]): 100 77 52
v2 após push_heap(a[5]): 100 77 52
v2 após push_heap(a[6]): 100 77 52
v2 após push_heap(a[7]): 100 98 52
v2 após pushheap(a[8]): 100 98 52
v2 após push heap(a[9]) : 100 98 52
v2 após 100 ser retirado do heap
98 77 52 22 40 31 1 3 13 100
v2 após 98 ser retirado do heap
77 40 52 22 13 31 1 3 98 100
v2 após 77 ser retirado do heap
52 40 31 22 13 3 1 77 98 100
v2 após 52 ser retirado do heap
40 22 31 1 13 3 52 77 98 100
v2 após 40 ser retirado do heap
31 22 3 1 13 40 52 77 98 100
v2 após 31 ser retirado do heap
22 13 3 1 31 40 52 77 98 100
v2 após 22 ser retirado do heap
13 1 3 22 31 40 52 77 98 100
v2 após 13 ser retirado do heap
3 1 13 22 31 40 52 77 98 100
```

v2 após 3 ser retirado do heap
1 3 13 22 31 40 52 77 98 100
v2 após 1 ser retirado do heap
1 3 13 22 31 40 52 77 98 100

3 22
3 22 31
3 22 31 1
77 22 31 1 3
77 22 31 1 3 13
77 40 31 1 3 13 22

Fig. 20.37 Usando funções da biblioteca padrão para executar um heapsort (parte 3 de 3).

A linha 21

std: :make_heap(v.beginO, v.end());

usa a função make heap para obter uma seqüência de valores no intervalo v . begin () até, mas não incluindo, v . end () e criar um heap que pode ser usado para produzir uma seqüência ordenada. Os dois argumentos devem ser iteradores de acesso aleatório, de modo que esta função trabalhará somente com arrays, vectors e deques. Uma segunda versão desta função aceita como terceiro argumento uma função predicado binária para comparar valores.

A linha 24

std: :sort_heap(v.beginO, v.end()

usa a função sort_heap para ordenar uma seqüência de valores no intervalo v . begin () até, mas não incluindo, v. end O que são organizados em um heap. Os dois argumentos iteradores devem ser iteradores de acesso aleatório. Uma segunda versão dessa função aceita como terceiro argumento uma função predicado para comparar valores.

A linha 34

std: :push_heap(v2 .begin, v2 .end());

usa a função push_heap para adicionar um novo valor a um heap. Tomamos um elemento do array a de uma vez, acrescentamos aquele elemento ao final do vector v2 e executamos a operação push_heap. Se o elemento acrescentado é o único elemento no vector, o vector já é um heap. Caso contrário, a função

pushheap rearranja os elementos do vector em um heap. Cada vez que push heap é chamada, ela assume que o último

elemento corrente do vector (isto é, aquele que é acrescentado antes da chamada à função push heap) é o elemento que está sendo acrescentado ao heap e que todos os outros elementos no vector estão organizados realmente como um heap. Os dois argumentos iteradores para push heap devem ser iteradores de acesso aleatório. Uma segunda versão dessa função aceita como terceiro argumento uma função predicado binária para comparar os valores.
A linha 41
std::pop_heap(v2.begin, v2.end() - i);
usa pop_heap para remover o elemento do topo do heap. Essa função assume que os elementos no intervalo especificado por seus dois argumentos iteradores de acesso aleatório já são um heap. Remover repetidamente o elemento do topo de um heap resulta em uma seqüência de valores ordenados. A função pop_heap permuta o primeiro elemento do heap (v2 . begin O neste exemplo) com o último elemento do heap (o elemento antes de v2 end () - i neste exemplo) e, então, assegura que os elementos até, mas não incluindo, o último elemento ainda formam um heap. Note na saída do programa que, após as operações pop heap. o vector está classificado em ordem ascendente. Uma segunda versão dessa função aceita como terceiro argumento uma função predicado binária para comparar valores.
2O5.13 mm e max
Os algoritmos mm e max determinam o mínimo de dois elementos e máximo de dois elementos, respectivamente.
O programa da Fig. 20.38 demonstra mm e max para valores int e char, [Nota: o compilador Microsoft Visual
C++ não suporta os algoritmos mm e max da STL, porque entram em conflito com funções do mesmo nome das
Microsoft Foundation Classes - as classes reutilizáveis da Microsoft para criação de aplicativos para Windows. O
programa da Fig. 20.38 foi compilado com o Borland C++.j

```
1 II Fig. 20.38: fig2038.cpp
2 // Demonstrando mm e max
3 #include <iostream>
4
5 using std::cout;
6 using std::endi;
7
8 #include <algorithin>
9
10 int main()
11
12 cout « 'O mínimo de 12 e 7 é:
```

```
13 « std::min( 12, 7 );
14 cout « "\nO máximo de 12 e 7 é:
15 « std::max( 12, 7 );
16 cout « "\nO mínimo de 'G' e 'Z é:
17 « std::min( 'G', 'Z' );
18 cout « "\nO máximo de 'G' e 'Z' é:
19 « std: :max( 'G', 'Z' ) « endi;
20 return O;
21 }
Omínimodel2e7é: 7
O máximo de 12 e 7 é: 12
O mínimo de 'G' e 'Z' é: G
O máximo de 'G' e 'Z' é: Z
```

Fig. 20.38 Demonstrando os algoritmos mm e max.

20.5.14 Algoritmos não-cobertos neste capítulo

A Fig. 20.39 comenta os algoritmos que não são cobertos neste capítulo.

Algoritmo	Descrição
inneryroduct	Calcula a soma dos produtos de duas seqüências tomando elementos correspondentes em cada seqüência, multiplicando aqueles elementos e acumulando os resultados em um total.
	Começando com o segundo elemento em uma seqüência, calcula a diferença (usando o operador-) entre o elemento corrente e o anterior e armazena o resultado. Os dois primeiros argumentos iteradores de entrada indicam o intervalo de elementos no contêiner e o terceiro argumento iterador de saída indica onde os resultados devem ser armazenados. Uma segunda versão deste algoritmo aceita como quarto argumento uma função para executar um cálculo usando o elemento corrente e o elemento anterior.
	Calcula um total acumulado (usando o operador +) dos valores em uma seqüência. Os dois primeiros argumentos iteradores de entrada indicam o intervalo dos elementos no contêiner e o iterador de saída especificado como terceiro argumento indica onde os resultados devem ser armazenados. Uma segunda versão deste algoritmo aceita como quarto argumento uma função binária que executa um cálculo usando o valor corrente na seqüência e o total acumulado atual.
	Usa três iteradores de acesso aleatório para particionar uma seqüência de elementos. O primeiro e o último argumentos representam o intervalo de elementos. O segundo argumento é a posição do elemento particionador. Após esse algoritmo ser executado, todos os elementos à esquerda do elemento particionador são menores do que aquele elemento e todos os elementos à

direita do elemento particionador são maiores que ou iguais àquele elemento. Uma segunda versão deste algoritmo aceita como quarto argumento uma função binária de comparação.
Esse algoritmo é similar ao nth element. mas requer iteradores bidirecionais menos potentes, o que o torna mais flexível do que nth_element. O algoritmo partition requer dois iteradores bidirecionais indicando o intervalo dos elementos para particionamento. O terceiro elemento é uma função predicado unária que ajuda a particionar os elementos de tal modo que todos os elementos na seqüência para os quais o predicado é true estão à esquerda (em direção ao começo da seqüência) de todos os elementos para os quais o predicado é false. E retornado um iterador bidirecional, indicando o primeiro elemento na seqüência para o qual a função predicado retorna false.
stable_partition Este algoritmo é similar a partition. exceto pelo fato de que os elementos para os quais a função predicado retorna true são mantidos em sua ordem original e elementos para os quais a função predicate retorna false são mantidos em sua ordem original.
next_yermutation Próxima permutação lexicográfica de uma seqüência.
prevyermutation Permutação lexicográfica anterior de uma seqüência.
rotate Usa três argumentos iteradores para a frente para girar a seqüência indicada pelo
primeiro e último argumentos pelo número de posições calculado subtraindo-se o primeiro argumento do segundo argumento. Por exemplo, a seqüência 1, 2, 3, 4, 5 girada por duas posições seria 4, 5, 1, 2, 3.
rotatecopy Este algoritmo é idêntico a rotate. exceto pelo fato de que os resultados são armazenados em uma seqüência separada indicada pelo quarto argumento - um
iterador de saída. As duas seqüências devem ter o mesmo número de elementos.

adj acent_difference
partialsum
nthelement
partition

Fig. 20.39 Algoritmos não-cobertos neste capítulo (parte 1 de 2).

Fig. 20.39 Algoritmos não-cobertos neste capítulo (parte 2 de 2).
206 A classe bitset
A classe bitset toma mais fácil criar e manipular bit sets (conjuntos de bits). Bit sets são úteis para representar um conjunto de bits usados como indicadores (flags). O tamanho de um bitset é fixado durante a compilação. A declaração

bitset< size > b;

cria o bitset b, no qual cada bit é inicialmente 0. O comando

b.set(bitNuniber);

coloca o bit número bitNumber do bitset b em 1. A expressão b. set () coloca todos os bits de b em 1. O comando
b.reset(bitNuinber);
coloca o bit número bitNumber do bitset b em O. A expressão b. reset () coloca todos os bits de bem O. O

comando

b.flip(bitNumber);

complementa o bit número bitNuinber do bitset b (por exemplo, se o bit é 1. flip o coloca em O). A expressão b. flip () complementa todos os bits de b. O comando

b[bitt4uinber];
retorna uma referência ao bit número bitNuniber do bitset b. Similarmente,

faz vál do
faz ret ou
ret
ft't

ret

cor

cxc

exe
O'-

A

de

des

b.at(bitNuxtiber);

Algoritmo	Descrição
adj acent f ind	Este algoritmo retorna um iterador de entrada indicando o primeiro de dois elementos adjacentes idênticos em uma seqüência. Se não há elementos adjacentes idênticos, o iterador é posicionado na posição end da seqüência.
partialsort	Usa três iteradores de acesso aleatório para ordenar parte de uma seqüência. O primeiro e o último argumentos indicam a seqüência de elementos. O segundo argumento indica a posição final para a parte ordenada da seqüência. Por default, os elementos são ordenados usando-se o operator < (uma função predicado binária pode ser também fornecida). Os elementos do segundo argumento iterador até o final da seqüência ficam em uma ordem indefinida.
partialsortc opy	Usa dois iteradores de entrada e dois iteradores de acesso aleatório para classificar parte da seqüência indicada pelos dois argumentos iteradores de entrada. Os resultados são armazenados na seqüência indicada pelos dois argumentos iteradores de acesso aleatório. Por default, os elementos são ordenados usando- se o operador < (uma função predicado binária também pode ser fornecida). O número de elementos ordenados é o menor entre o número de elementos no resultado e o número de elementos na seqüência original.
stablesort	O algoritmo é similar ao sort exceto pelo fato de que todos os elementos iguais são mantidos em sua ordem original.

faz uma verificação de validade do bit número bitNumber primeiro. Então, sebitNumber está no intervalo válido, at retorna uma referência ao bit. Caso contrário, at dispara uma exceção out_of_range. O comando
b.test(bitNunber);
faz uma verificação de validade do bitNuinber primeiro. Então, se bitNuniber está no intervalo válido, test retorna true se o bit é igual a 1 e false se o bit é igual a 0. Caso contrário, test dispara uma exceção out_of_range. A expressão
b.size()
retorna o número de bits do bitset b. A expressão
b.count()
retorna o número de bits que estão ligados (iguais a 1) no bitset b. A expressão
b . any O
retorna true se qualquer bit está ligado no bitset b. A expressão
b.none()
retorna true se nenhum dos bits de bitset b está ligado. As expressões
b == bi
b != bi
comparam os dois bitsets quanto à igualdade e desigualidade, respectivamente. Cada um dos operadores de atribuição sobre bits &=. = e = podem ser usados para combinar bitsets. Por
exemplo,
b & bi;

executa um E lógico bit a bit entre os bitsets b e bi. O resultado é armazenado em b. O OU lógico bit a bit e o OU lógico exclusivo bit a bit (XOR) são executados por
b bi;
b A b2;
A expressão
b »= n;
desloca os bits em bitset b para a direita por n posições. A expressão
b «= n;
desloca os bits em bitset b para a esquerda por n posições. As expressões
b.tostring()
b.to_ulong()
converte o bitset b para um string e um unsigned).ong, respectivamente.

A Fig. 20.40 revisita o Crivo de Eratóstenes para encontrar os números primos que comentamos no Exercício 4.29. Um bitset é usado em lugar de um array para implementar o algoritmo. O programa exibe todos os números de 2 a 1023 e então permite ao usuário digitar um número para determinar se aquele número é primo.

```
1 II Fig. 20.40: fig2O4O.cpp
2 // Usando um bitset para demonstrar o Crivo de Eratóstenes.
3 #include <iostream>
4
5 using std::cin;
6 using std::cout;
7 using std: :endl;
8
9 #include <iomanip>
10
11 using std::setw;
12
13 #include <bitset>
14 #include <cmath>
15
16 int main()
17 {
18 const int size = 1024;
19 int i, value, counter;
20 std::bitset< size > sieve;
21
22 sieve.flipO);
23
24 // executa o Crivo de Eratóstenes
25 int finalBit = sqrt( sieve.size() ) + 1;
26
```

```
27 for ( i = 2; i < finalBit; ++i
28 if ( sieve.test( i
29 for ( int j = 2 * i; j < size; j += i
30 sieve.reset( j );
31
32 cout « “Os números primos no intervalo de 2 a 1023 são:\n’;
33
34 for ( i = 2, counter = 0; i < size; ++i
35 if ( sieve.test( i ) ) {
36 cout « setw( 5 ) « i;
37
38 if ( ++counter % 12 == O
39 cout« ‘\n;
40 }
41
42 cout « endi;
43
44 II obtém um valor digitado pelo usuário para determinar se é primo
45 cout « “\nDigite um valor de 2 a 1023 (-1 para terminar) : “;
46 cm » value;
47
48 while ( value -1 ) {
49 if ( sieve[ value 1
50 cout « value « “ é um número primo\n;
51 else
52 cout « value « não é um número primo\n;
```

Fig. 20.40 Demonstrando a classe bitset e o Crivo de Eratóstenes (parte 1 de 2).


```
53
54 cout « “\nDigite um valor de 2 a 1023 (-1 para terminar)

59 1

}

cm » value;

return 0;
```

```
Digite um valor de 2 a 1023 (-1 para terminar) : 389 389 é um numero primo
Digite um valor de 2 a 1023 (-1 para terminar): 88
88 não é um numero primo
Digite um valor de 2 a 1023 (-1 para terminar): -1
```

Fig. 20.40 Demonstrando a classe bitset e o Crivo de Eratóstenes (parte 2 de 2).

A linha 20

std: :bitset< size > sieve;

cria um bitset de bits size (size é 1024 neste exemplo). Ignoramos os bits nas posições O e 1 neste programa. Por default, todos os bits no bitset são colocados em zero. O código

```
// executa o Crivo de Eratóstenes
int finalBit = sqrt( sieve.size() ) + 1;
for ( i = 2; i < finalBit; ++i
if ( sieve.test( i
for ( int j =2 * i; j < size; j +=i

sieve.reset( j
```

determina todos os números primos de 2 a 1023. O inteiro finalBit é usado para determinar quando o algoritmo foi completado. O algoritmo básico é que um número é primo se ele não tem outros divisores além de 1 e ele mesmo. Começando com o número 2, uma vez que sabemos que um número é primo, podemos eliminar todos os múltiplos daquele número. O número 2 é divisível somente por 1 e por ele mesmo, então ele é primo. Portanto, pode-se eliminar 4, 6, 8 e assim por diante. O número 3 é divisível somente por 1 e ele mesmo. Portanto, pode-se eliminar todos os múltiplos de 3 (lembre-se de que todos os números pares já foram eliminados).

55
56
57
58

Os	números		primos		intervalo		de 2	a 1023	são:			
	2	3	5	7	11	13	17	19	23	29	31	37
	41	43	47	53	59	61	67	71	73	79	83	89
	97	101	103	107	109	113	127	131	137	139	149	151
	157	163	167	173	179	181	191	193	197	199	211	223
	227	229	233	239	241	251	257	263	269	271	277	281
	28	29	30	31	31	31	33	337	34	34	35	35

	3	3	7	1	3	7	1		7	9	3	9
	367	373	379	383	389	397	401	409	419	421	431	433
	439	443	449	457	461	463	467	479	487	491	499	503
	509	521	523	541	547	557	563	569	571	577	587	593
	599	601	607	613	617	619	631	641	643	647	653	659
	661	673	677	683	691	701	709	719	727	733	739	743
	751	757	761	769	773	787	797	809	811	821	823	827
	829	839	853	857	859	863	877	881	883	887	907	911
	919	929	937	941	947	953	967	971	977	983	991	997
1009		1013	1019 1021								

990 C++ COMO PROGRAMAR

20.7 Objetos função

Objetos função e adaptadores de funções são fornecidos para tornar a STL mais f]exível. Um objeto função contém uma função que pode ser tratada sintática e semanticamente como uma função usando operator o. Objetos função e adaptadores da STL estão definidos no cabeçalho <functional>. Um objeto função pode também encapsular dados com a função nele incluída. Os objetos função padrão são colocados mime para obter melhor desempenho. Os objetos função da STL são mostrados na Fig. 20.4 1.

Objetos função Tipo

divides< T > aritmético
equalto< T > relacional
greater< T > relacional
greaterequal< T > relaciona!
less< T > relacional
lessequal< T > relacional
logicaland< T > lógico
logicalnot< T > lógico
logicalor< T > lógico
minus< T > aritmético
modulus< T > aritmético
negate< T > aritmético
notequalto< T > relaciona!
plus< T > aritmético

multiplies< T > aritmético
Fig. 20.41 Objetos função na biblioteca padrão.
O programa da Fig. 20.42 demonstra o uso do algoritmo numérico accumulate (comentado na Fig. 20.30) para
calcular a soma dos quadrados dos elementos em um vector. O quarto argumento para accumulate é um objeto
função binária ou uma função ponteiro para uma função binária que aceita dois argumentos e retorna um resultado.
A função accumulate é demonstrada duas vezes - uma vez com uma função no ponteiro para uma função binária
e outra vez com um objeto função.
1 II Fig. 20.42: fig20_42.cpp
2 // Demonstrando funções objeto.
3 #include <iostream>
4
5 using std: :cout;
6 using std::endl;
7
8 #include <vector>
9 #include <algorithm>
10 #include <numeric>
11 #include <functional>
12
13 // função binária que soma o quadrado de seu segundo
14 II argumento e o total acumulado em seu primeiro
15 II argumento e retorna a soma
16 int sumSquares(int total, int value
17 { return total + value * value;
18
19 II gabarito de classe função binária que define um
Fig. 20.42 Demonstrando um objeto função binária (parte 1 de 2).

// operatorOsobrecarregado que soma o quadrado
II de seu segundo argumento e o total acumulado
// em seu primeiro argumento e retorna a soma
template< class T >
class SuxnSquaresClass : public std: :binary function< T, T, T >
public:
const T operator() (const T &total, const T &value
return total + value * value;
int main()
const int SIZE = 10;
int al[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
std: :vector< int > v(al, al + SIZE);

```
std: :ostreamiterator< int > output( cout, “
int result = 0;
cout « “O vector v contém:\n”;
std::copy( v.beginO), v.end0, output );
result =
std: :accumulate( v.begin() , v.end() , 0, sumSquares );
cout « “\n\nSoma dos quadrados dos elementos no vector v usando
« “\na função binária sumSquares: “ « result;
45
46 result = std: :accumulate( v.begin() , v.end0 , 0,
47 SumSquaresClass< int > );
48 cout « “\n\nSoma dos quadrados dos elementos no vector
49 « “\nobjeto função binária do tipo
50 « “SumSquaresClass< int >: “ « result « endl;
51 return 0;
52
O vector v contem:
1 2 3 4 5 6 7 8 9 10
Soma dos quadrados dos elementos no vector v usando
a função binária sumSquares: 385
Soma dos quadrados dos elementos no vector v usando o
objeto função binária do tipo SumSquaresClass< int >: 385
```

Fig. 20.42 Demonstrando um objeto função binária (parte 2 de 2).

As linhas l6e 17

```
int sumSquares( int total, int value )
return total + value * value;
```

definem a função sumSquares. que eleva ao quadrado seu segundo argumento (value), acrescenta o quadrado ao seu primeiro argumento (total) e retorna a soma. A função accumulate passará cada um dos elementos da seqüência sobre o qual ela itera como segundo argumento para sumSquares no exemplo. Na primeira chamada a suniSquares. o primeiro argumento será o valor inicial de total (que é fornecido como o terceiro argumento a ser acumulado; O neste programa). Todas as chamadas subseqüentes a sumSquares recebem como primeiro argumento a soma corrente retornada pela chamada anterior a sumSquares. Quando accuniulate termina, ela retorna a soma dos quadrados de todos os elementos na seqüência.

991

20
21
22
23
24
25
26

27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44

v usando o”

992 C++ COMO PROGRAMAR
As linhas 23 a 29
template< class T >
class SumSquaresClass public std::binary_function< T, T, T >
publie:
const T operator() (const T &total, const T &value
return total + value * value;
definem a classe SumSquaresClass que herda da classe binary function (no arquivo de cabeçalho <functional>). Classes que herdam de binary function definem a função operator () sobrecarregada com dois argumentos. A classe SuinSquaresClass é usada para definir objetos função para os quais as funções sobrecarregadas operator O executam a mesma tarefa que a função sumSquares. Os três parâmetros de tipo (T) para o gabarito binary function são o tipo do primeiro argumento para operator O , o tipo do segundo argumento para operator () e o tipo de retomo de operator o, respectivamente. A função accumulate passará os elementos da seqüência sobre a qual ela itera como segundo argumento para a função operator () do objeto da classe SumSquaresClass que é passado para o algoritmo accumulate. Na primeira chamada a operator O o primeiro argumento será o valor inicial de total (o qual é fornecido como terceiro argumento para accumulate: O neste programa). Todas as chamadas subseqüentes a operator () recebem como primeiro argumento o resultado retomado pela chamada anterior a operator O . Quando accumulate termina, retorna a soma dos quadrados de todos os elementos na seqüência.
Aslinhas4l e42
result =

std: :accuniulate(v.begin() , v.end() , O, sumSquares
chamam a função accumulate com um ponteiro para a função em sumSquares como seu último argumento. O comando nas linhas 46 e 47
result = std::accumulate(v.beginO, v.endO, O,
SumSquaresClass< int >);
chama a função accumulate com um objeto da classe SumSquaresClass como um último argumento. A expressão SumSquaresClass< int > () cria uma instância da classe SumSquares que é passada para accumulate que envia ao objeto a mensagem (invoca a função) operator () . A declaração precedente poderia ser escrita como duas instruções separadas, como segue:
SumSquaresClass< int > sumSquaresOb
result = std::accumulate(v.begin, v.endO, O, sumSquaresObj);
A primeira linha define um objeto da classe SumSquaresClass. Tal objeto então é passado para a função accumulate e a mensagem operator é enviada.
Observação de engenharia de software 20.10
Diferentemente dos ponteiros para função, um objeto função pode também encapsular dados.
Resumo
• Usar a STL pode economizar tempo e esforços consideráveis e resultar em programas da mais alta qualidade.
• A escolha de qual contêiner da biblioteca padrão usar em uma aplicação específica é baseada freqüentemente em considerações de desempenho.

• Iteradores de acesso aleatório combinam as capacidades de um iterador bidirecional com a habilidade de acessar diretamente qualquer elemento de um contêiner, isto é, pular para frente ou para trás por um número arbitrário de elementos.
• A categoria de iterador suportada por cada contêiner determina se aquele contêiner pode ser usado com algoritmos específicos na STL. Contêineres que suportam iteradores de acesso aleatório podem ser usados com todos os algoritmos na STL.
• Ponteiros para arrays podem ser usados em lugar de iteradores em todos os algoritmos da STL.
• A STL inclui aproximadamente 70 algoritmos padrão. Algoritmos seqüenciais mutantes resultam em modificações nos elementos do contêiner. Algoritmos seqüenciais não-mutantes não modificam os elementos do contêiner.
• As funções f iii e fui n atribuem a cada elemento em um intervalo de elementos de contêiner um valor específico.
• As funções generate e generate n usam uma função geradora para criar valores para cada elemento em um intervalo de elementos de contêiner.
• A função equal compara duas seqüências de valores quanto à igualdade.
• A função mismatch compara duas seqüências de valores e retorna um pair de iteradores indicando a posição em cada seqüência dos elementos diferentes. Se todos os elementos são iguais, o pair contém o resultado da função erid para cada seqüência.

• A função iexicographicai compare compara os conteúdos de duas seqüências para determinar se uma seqüência é menor que outra seqüência (similar a uma comparação de strings).
• As funções remove e remove copy deletam todos os elementos em uma seqüência que coincidem com um valor especificado. As funções remove if e remove copy if deletam todos os elementos em uma seqüência para os quais a função predicado unária passada às funções retoma true.
• As funções repiace e replace copy substituem todos os elementos em uma seqüência iguais a um valor especificado. As funções repiace if e repiace copy if substituem por um novo valor todos os elementos na seqüência para os quais a função predicado unária passada para as funções retoma true.
• A função random shuffie ordena aleatoriamente os elementos de uma seqüência.
• A função count conta os elementos com o valor especificado em uma seqüência. A função count if conta os elementos em uma seqüência para os quais a função predicado unária fornecida retorna true.
• A função mm eiement localiza o menor elemento em uma seqüência. A função max eiement localiza o maior elemento em uma seqüência.
• A função accumuiate soma os valores em uma seqüência. A segunda versão desta função recebe um ponteiro para uma função genérica que aceita argumentos e retorna um resultado. A função genérica determina como os elementos em uma seqüência são acumulados.
• A função for_each aplica uma função genérica sobre cada elemento em uma seqüência. A função genérica aceita um argumento (que ela não deve modificar) e retorna void.
• A função trans form aplica uma função genérica sobre cada elemento em uma seqüência. A função genérica aceita um argumento (que ela pode modificar) e retorna o resultado transformado.
• A função find localiza um elemento em uma seqüência e, se o elemento é encontrado, retorna um iterador para o elemento; caso contrário, f ind retorna um iterador indicando o fim da seqüência. A função find_if localiza o primeiro elemento para o qual a função predicado unária fornecida retorna true.
• A função sort organiza os elementos de uma seqüência em ordem classificada (ordem ascendente, por default, ou a ordem indicada por uma função predicado binária fornecida).
• A função binary search determina se um elemento está presente em uma seqüência ordenada.
• A função swap permuta dois valores.
• A função iter_swap permuta dois valores referenciados por iteradores.
• A função swap ranges permuta os elementos em duas seqüências de elementos.
• A função copy_backward copia os elementos de uma seqüência e os coloca em outra seqüência começando do último elemento na segunda seqüência e trabalhando em direção ao início da segunda seqüência.

• Iteradores de acesso aleatório combinam as capacidades de um iterador bidirecional com a habilidade de acessar diretamente qualquer elemento de um contêiner, isto é, pular para frente ou para trás por um número arbitrário de elementos.
• A categoria de iterador suportada por cada contêiner determina se aquele contêiner pode ser usado com algoritmos específicos na STL. Contêineres que suportam iteradores de acesso aleatório podem ser usados com todos os algoritmos na STL.
• Ponteiros para arrays podem ser usados em lugar de iteradores em todos os algoritmos da STL.
A STL inclui aproximadamente 70 algoritmos padrão. Algoritmos seqüenciais mutantes resultam em modificações nos elementos do contêiner. Algoritmos seqüenciais não-mutantes não modificam os elementos do contêiner.
• As funções f iii e fui n atribuem a cada elemento em um intervalo de elementos de contêiner um valor específico.
• As funções generate e generate_n usam uma função geradora para criar valores para cada elemento em um intervalo de elementos de contêiner.
• A função equal compara duas seqüências de valores quanto à igualdade.
• A função mismatch compara duas seqüências de valores e retoma um pair de iteradores indicando a posição em cada seqüência dos elementos diferentes. Se todos os elementos são iguais, o pair contém o resultado da função end para cada seqüência.
• A função iexicographicai compare compara os conteúdos de duas seqüências para determinar se uma seqüência é menor que outra seqüência (similar a uma comparação de strings).
• As funções remove e remove copy deletam todos os elementos em uma seqüência que coincidem com um valor especificado. As funções remove if e remove copy if deletam todos os elementos em uma seqüência para os quais a função predicado unária passada às funções retoma true.
• As funções repiace e repiace copy substituem todos os elementos em uma seqüência iguais a um valor especificado. As funções repiace if e repiace copy if substituem por um novo valor todos os elementos na seqüência para os quais a função predicado unária passada para as funções retorna true.
• A função random shuffle ordena aleatoriamente os elementos de uma seqüência.
• A função count conta os elementos com o valor especificado em uma seqüência. A função count_if conta os elementos em uma seqüência para os quais a função predicado unária fornecida retoma true.
• A função mm eiement localiza o menor elemento em uma seqüência. A função max element localiza o maior elemento em uma seqüência.
• A função accumuiate soma os valores em uma seqüência. A segunda versão desta função recebe um ponteiro para uma função genérica que aceita argumentos e retorna um resultado. A função genérica determina como os elementos em uma seqüência são acumulados.
• A função for_each aplica uma função genérica sobre cada elemento em uma

seqüência. A função genérica aceita um argumento (que ela não deve modificar) e retoma void.
• A função transform aplica uma função genérica sobre cada elemento em uma seqüência. A função genérica aceita um argumento (que ela pode modificar) e retoma o resultado transformado.
• A função f ind localiza um elemento em uma seqüência e, se o elemento é encontrado, retoma um iterador para o elemento; caso contrário, f ind retorna um iterador indicando o fim da seqüência. A função find if localiza o primeiro elemento para o qual a função predicado unária fornecida retoma true.
• A função sort organiza os elementos de uma seqüência em ordem classificada (ordem ascendente, por default, ou a ordem indicada por uma função predicado binária fomecida).
• A função binary search determina se um elemento está presente em uma seqüência ordenada.
• A função swap permuta dois valores.
• A função iter_swap permuta dois valores referenciados por iteradores.
• A função swap ranges permuta os elementos em duas seqüências de elementos.
• A função copy backward copia os elementos de uma seqüência e os coloca em outra seqüência começando do último elemento na segunda seqüência e trabalhando em direção ao início da segunda seqüência.

• A função merge intercala duas seqüências ordenadas ascendentemente em uma terceira seqüência ordenada. Observe que merge também opera com seqüências não-ordenadas, mas não produz uma seqüência ordenada.

• Uma back inserter usa a capacidade default do contêiner para inserir um elemento no final do contêiner. Quando um elemento é inserido em um contêiner que não tem mais elementos disponíveis, o contêiner cresce em tamanho. Há dois outros insersores frontinserter e inserter. Um front inserter insere um elemento no início de um contêiner (especificado como seu argumento) e um inserter insere um elemento antes da posição indicada pelo iterador fornecido corno seu segundo argumento, no contêiner fornecido como seu primeiro argumento.

A função unique remove todos os valores duplicados de uma seqüência ordenada.

A função reverse coloca na ordem inversa todos os elementos de uma seqüência.

A função inplace merge intercala duas seqüências de elementos no mesmo contêiner.

• A função unique copy faz uma cópia de todos os elementos únicos em uma seqüência ordenada. A função reversecopy faz uma cópia na ordem inversa dos elementos de uma seqüência.

• A função includes compara dois conjuntos ordenados de valores para determinar se cada elemento do segundo conjunto está incluído no primeiro conjunto. Sendo assim. includes retorna true; caso contrário, includes retorna false,

• A função set_difference determina os elementos do primeiro conjunto de valores ordenados que não estão no segundo conjunto de valores ordenados (os dois conjuntos de valores devem estar em ordem ascendente e usando a mesma função de comparação).

• A função set intersection determina os elementos do primeiro conjunto de valores ordenados que estão no segundo conjunto de valores ordenados (os dois conjuntos de valores devem estar em ordem ascendente e usando a mesma função de

comparação).

A função set symmetric difference determina os elementos no primeiro conjunto que estão no segundo conjunto e os elementos no segundo conjunto que não estão no primeiro conjunto (os dois conjuntos de valores devem estar em ordem ascendente e usando a mesma função de comparação).

• A função set union cria um conjunto de todos os elementos que estão em um ou em ambos os conjuntos (os dois conjuntos de valores devem estar em ordem ascendente e usando a mesma função de comparação).

• A função lower bound determina a primeira posição em uma seqüência ordenada na qual o terceiro argumento pode ser inserido na seqüência de modo que a seqüência ainda estaria classificada em ordem ascendente.

• A função upper bound determina a última localização em uma seqüência ordenada de valores na qual o terceiro argumento poderia ser inserido na seqüência de modo que a seqüência ainda estaria classificada em ordem ascendente.

• A função equal range retorna um pair de iteradores para a frente contendo os resultados combinados da execução de duas operações, uma lower_bound e uma upperbound.

Heapsori é um algoritmo de classificação no qual um array de elementos é

organizado em uma árvore binária especial denominada heap. A chave do bom desempenho de um heap é que o maior elemento está sempre no topo do heap e os valores dos filhos de qualquer nodo da árvore binária são sempre menores que ou iguais ao valor daquele nodo. Um heap organizado desta maneira é freqüentemente chamado de maxheap.

• A função make heap aceita uma seqüência de valores e cria um heap que pode ser usado para produzir uma seqüência ordenada.

A função sort heap classifica uma seqüência de valores que já estão organizados em um heap.

A função push_heap adiciona um valor novo a um heap. push heap assume que o último elemento atualmente no contêiner é o elemento que está sendo adicionado ao heap e que todos os outros elementos no contêiner já estão organizados como heap.

• A função pop_heap remove o elemento do topo do heap. Esta função assume que os elementos já estão organizados como sheap.

• A função mm determina o mínimo de dois valores. A função max determina o máximo de dois valores.

• A classe bitset torna mais fácil criar e manipularbit nets (conjuntos de bits). Conjuntos de bits são úteis para representar um conjunto de indicadores (flags) booleanos. O tamanho de um bitset é fixado durante a compilação.

Terminologia
<algorithm> erase()
<deque> filiO
<functional> filin O
<iterador> findO
<list> for each()
<map> front()
<numeric> generate O
<set> generaten O
<stack> inplacemerge O
<vector> insertO
accumuiate () intervalo
adaptador is tream i terator
adjacentdifference O iterswap()
adjacentf ind () iterador
algoritmo iterador bidirecional
algoritmo de classificação iterador de acesso aleatório

algoritmo de seqüencial não-mutante iterador de entrada
algoritmos de seqüência mutante iterador de saída
array associativo iterador para a frente
assign () iterador reverso
atribuição iexicographical_compare O
back () iowerbound ()
begin O makeheap O
biblioteca padrão de gabaritos (STL) mapeamento um para um
bibliotecas de classes específicas para uma plataforma max O
bibliotecas de classes independentes de plataforma maxelement O
binarysearch O rnaxsize O
classe de adaptador de contêiner priorityqueue inerge O
classe de adaptador de contêiner queue mm O
classe de adaptador de contêiner stack mineiement O
classes de adaptador de contêiner mismatch O
colocar em ordem inversa o conteúdo de um contêiner namespace std
const_iterator ntheiement
const_reverseiterator objeto função
contêiner operator ! = O
contêiner associativo operator< O
contêiner associativo map operator<= O
contêiner associativo multimap operator O
contêiner associativo multiset operator> O
contêiner associativo set operator>= O
contêiner de seqüência ostream_iterator
contêiner seqüencial partiaisort O
contêiner seqüencial deque partialsortcopy O
contêiner seqüencial list par tial_sum O
contêiner seqüencial vector partition ()
contêineres de primeira classe pop O
copy() popback()
copybackward() popfront()
count() popheap()
countif () primeiro a entrar, primeiro a sair (FIFO)
criar uma associação programação genérica
deque<T> push(.)
deque<T>: :iterator pushback()
empty O push_front O
end() pushheap()
equal() randomshuffle()
equa 1_range O rbegin O

CAPÍTULO 20- A BIBLIOTECA PADRÃO DE GABARITOS (STL) 997
remove () setunion O
remove copy() size O
remove copyif O size_type

removeif () sort O
rend () sort_heap O
replace() string
replace_copy () string ()
replacecopyif O struct less<T>
replace_if O swap()
reverse () swap range O
reversecopy O top O
reverseiterator trans form O
rotate O último a entrar, primeiro a sair (LIFO)
rotatecopy() unique()
seqüência upperbound O
set_difference O vai array
set_intersection () vaiuetype
setsymmetricdifference O
Erros comuns de programação
20.1 Tentar derreferenciar um iterador posicionado fora de seu contêiner é um erro lógico durante a execução. Especialmente, o iterador retornado por end O não pode ser derreferenciado ou incrementado.
20.2 Tentar criar um iterador não-const para um contêiner const é um erro de sintaxe.
20.3 O vector não deve estar vazio; caso contrário, os resultados das funções front e back são indefinidos.
20.4 Apagar um elemento que contém um ponteiro para um objeto alocado dinamicamente não deleta o objeto.
20.5 Tentar ordenar um contêiner usando um iterador que não seja um iterador de acesso aleatório é um erro de sintaxe. A função sort requer um iterador de acesso aleatório.
Boas práticas de programação
20.1 Usar typedefs para conseguir ler mais facilmente código de programas com tipos com nomes longos (tais como rnultisets).
20.2 E uma boa prática verificar se o intervalo especificado em uma chamada para mm element não está vazio e verificar se o valor de retomo não é um iterador apontando para “depois do final”.
Dicas de desempenho
20.1 Para qualquer aplicação particular, diversos contêineres da STL diferentes podem ser apropriados. Selecione o contêiner mais apropriado que obtém o melhor desempenho (ou seja, o equilíbrio entre velocidade e tamanho) para aquela aplicação. A eficiência foi um aspecto considerado crucial no projeto da STL.
20.2 Os recursos da biblioteca padrão são implementados para operar eficientemente em uma ampla variedade de aplicações. Para algumas aplicações com requisitos especiais de desempenho, pode ser necessário escrever suas próprias
implementações.
20.3 A STL geralmente evita herança e funções virtual em favor da programação

genérica com gabaritos, para obter um desempenho melhor durante a execução.

20.4 Conheça seus componentes da STL. Escolher o contêiner mais apropriado para um determinado problema pode maximizar o desempenho e minimizar os requisitos de memória.
20.5 A inserção no fim de um vector é eficiente. O vector simplesmente cresce, se necessário, para acomodar o novo item. E dispendioso inserir (ou retirar) um elemento no meio de um vector-a porção inteira do vector após o ponto de inserção (ou retirada) deve ser movida, porque elementos vector ocupam posições contíguas na memória, assim como um array “cru” em C ou C++.
20.6 Aplicações que requeiram freqüentes inserções e deleções em ambas as extremidades de um contêiner normalmente usam uma deque de preferência a um vector. Embora possamos inserir e deletar elementos na frente e atrás tanto de um vector como de uma deque, a classe deque é mais eficiente do que vector para fazer inserções e deleções no início.

20.7 Aplicações com freqüentes inserções e deleções no meio e/ou nos extremos de um contêiner, normalmente usam uma list, devido à sua eficiente implementação de inserção e deleção em qualquer lugar na estrutura de dados.
20.8 Escolha o contêiner vector para obter o melhor desempenho com acesso aleatório.
20.9 Objetos da classe vector fornecem acesso indexado rápido com o operador subscrito sobrecarregado [1 porque eles estão armazenados em uma área contígua, como um array “bruto” em C ou C++.
20.10 E muito mais rápido inserir muitos elementos de uma só vez do que um elemento de cada vez.
20.11 Pode ser um desperdício dobrar o tamanho de um vector quando é necessário mais espaço. Por exemplo, um vector com um total de 1.000.000 elementos se redimensiona para acomodar 2.000.000 elementos quando é adicionado um novo elemento. Isto deixa 999.999 elementos sem uso. Os programadores podem usar resize () para controlar melhor o uso do espaço.
20.12 Depois que um bloco de armazenagem é alocado para uma deque, em diversas implementações o bloco não é desalocado até que a deque seja destruída. Isto torna a operação de uma deque mais eficiente do que se a memória fosse repetida- mente alocada, desalocada e realocada. Mas isso significa que a deque. muito provavelmente, vai usar a memória mais ineficientemente (do que um vector, por exemplo).
20.13 Inserções e deleções no meio de uma deque são otimizadas para minimizar o número de elementos copiados para manter a ilusão de que os elementos do deque são contíguos.
20.14 Por razões de desempenho, multisets e sets são tipicamente implementados como estruturas denominadas árvores de pesquisa binária vermelho-preto. Com esta representação interna, a árvore de pesquisa binária tende a ser balanceada, minimizando desta forma os tempos médios de

pesquisa.
20.15 Um multimap é implementado de modo a localizar eficientemente todos os pares de valores com uma determinada chave.
20.16 Cada uma das operações comuns de um stack é implementada como uma função mime que chama a função apropriada do contêiner subjacente. Isto evita o overhead de uma segunda chamada de função.
20.17 Para melhor desempenho, use as classes deque ou vector como o contêiner subjacente para um stack.
20.18 Cada uma das operações comuns de uma queue é implementada como uma função mime que chama a função apropriada do contêiner subjacente. Isto evita o overhead de uma chamada de uma segunda função.
20.19 Para melhor desempenho, use a classe deque como o contêiner subjacente para uma queue.
20.20 Cada uma das operações comuns de uma priority queue é implementada como uma função mime que chama a função apropriada do contêiner subjacente. Isto evita o overhead de uma segunda função.
20.21 Para melhor desempenho, usar a classe vector como contêiner subjacente para uma priorityqueue.
Dicas de portabilidade
20.1 A STL certamente vai se tomara meio favorito de programação com contêineres. Programar com a STL vai aumentar a portabilidade de seu código.
20.2 Devido aos algoritmos da STL processarem os contêineres só indiretamente, através dos iteradores, um algoritmo pode freqüentemente ser usado com muitos contêineres diferentes.
Observações de engenharia de software
20.1 A abordagem da STL permite que programas genéricos sejam escritos de maneira que o código não dependa do contêiner subjacente. Tal estilo de programação é chamado de progranwção genérica.
20.2 Evite reinventar a roda; programe com componentes reutilizáveis da biblioteca padrão de C++. A STL contém muitas das estruturas de dados mais populares, como contêineres, e fornece vários algoritmos populares que os programas usam para processar dados nestes contêineres.
20.3 Os operadores de igualdade e menor do que são tecnicamente desnecessários para os elementos armazenados em um contêiner, a menos que os elementos necessitem ser comparados. Entretanto, quando se cria código a partir de um gabarito, alguns compiladores requerem que todas as partes do gabarito sejam definidas, enquanto outros compiladores requerem somente as partes do gabarito que são realmente usadas no programa.
20.4 Usar o “iterador mais fraco” que produz um desempenho aceitável ajuda a produzir componentes reutilizáveis ao máximo.
20.5 A STL é implementada concisamente. Até agora, projetistas de classes teriam associado os algoritmos aos contêineres tornando os algoritmos funções membro dos contêineres. A STL adota uma abordagem diferente. Os algoritmos estão separados dos contêineres e operam sobre os elementos dos contêineres só indiretamente, através de iteradores. Esta separação torna mais fácil escrever algoritmos genéricos aplicáveis a muitas outras classes de contêineres.

20.6 A STL é extensível. E simples adicionar a ela novos algoritmos e fazer isso sem mudanças nos contêineres da STL.
20.7 Os algoritmos podem operar sobre contêineres da STL e sobre arrays baseados em ponteiros, no estilo usado em C.

20.8 Os algoritmos não dependem dos detalhes de implementação dos contêineres sobre os quais eles operam. Enquanto os iteradores de contêineres (ou de arrays) satisfizerem os requisitos do algoritmo, os algoritmos da STL podem trabalhar com quaisquer arrays no estilo de C, baseados em ponteiros, assim como trabalhar com os contêineres da STL (e estruturas de dados definidas pelo usuário).
20.9 Algoritmos podem ser facilmente adicionados à STL sem modificar as classes contêineres.
20.10 Diferentemente dos ponteiros para função, um objeto função pode também encapsular dados.
Dicas de teste e depura ção
20.1 Quando programamos com estruturas de dados e algoritmos baseados em ponteiros, devemos fazer nossa própria depuração e teste para garantir que nossas estruturas de dados, classes e algoritmos funcionam corretamente. E fácil cometer erros ao manipular ponteiros neste nível tão baixo. Perdas de memória" e violações de acesso à memoria são comuns em tal código customizado. Para a maioria dos programadores, e para maioria das aplicações que necessitarão escrever, as estruturas de dados definidas como gabaritos, pré-empacotadas, da STL são suficientes. Usar o código da STL pode evitar muito tempo de teste e depuração. Uma precaução a ser tomada com relação a grandes projetos é que o tempo de compilação pode ser significativo.
20.2 O operador * (derreferenciador) de qualquer iterador const retorna uma referência const ao elemento do contêiner, não permitindo, portanto, o uso de funções membro não-const.
20.3 Operações executadas em um const_iterator retornam referências corist para evitar modificação a elementos do contêiner que está sendo manipulado. Dê preferência ao uso de const_iterators em vez de iterators. onde apropriado. Este é um outro exemplo do princípio do menor privilégio.
20.4 Somente iteradores de acesso aleatório suportam <. E melhor usar = e end () para testar o fim do contêiner.
Exercícios de auto-revisão
20.1 (VIF) A STL faz uso abundante de herança e funções virtual.
20.2 Os dois tipos de contêineres da STL são contêineres seqüenciais e contêineres _____
20.3 A STL evita usar new e delete dando preferência ao uso de para possibilitar diversos meios de controlar a alocação e a desaloção de memória.
20.4 Os cinco principais tipos de iterador são _____ _____, , _____ e
20.5 (V/F) Um ponteiro é uma forma generalizada de iterador.
20.6 (V/F) Os algoritmos da STL podem operar sobre arrays baseados em

ponteiros ao estilo de C.
20.7 (V/F) Os algoritmos da STL são encapsulados como funções dentro de cada classe de contêiner.
20.8 (V/F) O algoritmo remove não diminui o tamanho do vector dos quais os elementos estão sendo removidos.
20.9 Alocação e desalocação de memória são executadas na STL com objetos
20.10 Os três adaptadores de contêiner da STL são _____ _____ e _____
20.11 (V/F) A função membro de contêiner end () retoma a posição do último elemento do contêiner.
20.12 Os algoritmos da STL operam sobre elementos de contêiner indiretamente usando _____
20.13 O algoritmo sort requer um iterador _____
Respostas aos exercícios de auto-revisão
20.1 Falso. Estes foram evitados devido a razões de desempenho.
20.2 Associativos.
20.3 Alocadores.
20.4 Entrada, saída, para a frente, bidirecional e acesso aleatório.
20.5 Falso. Na realidade é vice-versa.
20.6 Verdadeiro.

20.7 Falso. Os algoritmos da STL não são funções membro. Eles operam indiretamente sobre contêineres através de iteradores.
20.8 Verdadeiro.
20.9 Alocador.
20.10 stack, queue, priority_qiieue.
20.11 Falso. Na realidade, retoma a posição imediatamente após o final do contêiner.
20.12 Iteradores.
20.13 Acesso aleatório.
Exercícios
20.14 Escrever uma função gabarito palindrome que aceita como parâmetro um const vector e retoma true ou false dependendo de se o vector é lido ou não da mesma forma da frente para trás e de trás para diante (por exemplo, um vector contendo 1,2, 3, 2, 1 é um palíndromo e um vector contendo 1, 2,3,4 não é).
20.15 Modificar o programa da Fig. 20.29, o Crivo de Eratóstenes, de modo que se o número que o usuário digitar no programa
não for primo, o programa exiba os fatores primos do número. Lembre-se de que os fatores de um número primo são somente 1
e o próprio número primo. Todo número que não é primo tem uma faturação prima única. Por exemplo, considere o número 54.
Os fatores de 54 são 2, 3, 3 e 3. Quando estes valores são multiplicados juntos, o resultado é 54. Para o número 54, os fatores
primos resultantes devem ser 2 e 3.
20.16 Modificar o Exercício 20.15 de modo que se o número que o usuário

digitar no programa não for primo, o programa exiba os fatores primos do número e o número de vezes que o fator primo aparece na fatoração prima única. Por exemplo, o resultado para o número 54 deve ser
A fatoração prima única de 54 é: 2 * 3 * 3 * 3
Recursos para a STL na Internet e na World Wide Web
A seguinte é uma coleção de recursos para a STL existentes na Internet e na World Wide Web. Estes sites incluem tutoriais, referências, FAQs, artigos, livros, entrevistas e software.
Tutorwis
http: //www. cs .brown . edu/people/jak/prograniming/stl-tutorial/tutorial .html Este tutorial sobre a STL está organizado por exemplos, filosofia, componentes e extensões da STL. Você encontrará
exemplos de código usando os componentes da STL, explanações úteis e diagramas para ajuda.
http: //web. ftech. net/honeyg/artic1es/eff_st1 . htin
Este tutorial sobre a STL fornece informações sobre componentes, contêineres, adaptadores de slreurns e contêineres,
além de como transformar e selecionar valores, filtrar e transformar valores e objetos.
http: //www.xraylith.wisc . edu/khan/software/st1/os_examp1es/examp1es .html
Este site é feito para pessoas que estão aprendendo a STL. Você encontrará uma introdução à STL e exemplos do
ObjectSpace STL Tool Kit.
Referências
http: //www. sgi. com/Technology/STL/other_resources . html
Este site tem uma lista de muitos sites na Web relativos à STL e uma lista de livros sugeridos sobre a STL.
http: //www. cs . rpi edu/projects/STL/stl/stl . html
Esta é a homepage da referência on-line da biblioteca padrão de gabaritos do Rensselaer Polytechnic Institute. Você
achará explicações detalhadas sobre a STL, assim como links para outros recursos úteis para informação acerca da STL.
http: //www. sgi com/Technology/STL/

lores. O Silicon Graphics Standard Template Library Programmer's Guide é um recurso útil para informações sobre a STL. Você pode fazer o download da STL a partir deste site, achar a última informação, documentação de projeto e links para
outros recursos da STL.
http: //www. dinkumware com/refcpp html
Este sitc contém informações úteis sobre a biblioteca padrão ANSI/ISO de C++ e contém informações extensas sobre a
biblioteca padrão de gabaritos (STL).
http: //www. roguewave com/products/xplatform/stdlib/
A página da Web da biblioteca padrão de C++ da Rogue Wave Software. Você

pode fazer download de artigos relacionados com a versão deles da biblioteca padrão de C++.
FAQs
alse
ctor ftp://butler.hpl.hp.com/stl/stl.faq
Este site FTP é uma página FAQ para a STL mantida por Marian Corcoran, membro do comitê ANST e uma “expert” em
c++.
grama
ente! Artigos, livros e entrevistas
atores http: //www. sgi com/Technology/STL/other resources html
Este site tem uma lista de mais de 15 sites na Web relativos à STL e uma breve lista de livros sugeridos sobre a STL.
exiba
ultado http://www.byte.com/art/951O/sec12/art3.htm
O site do Byte Magazine tem uma cópia de um artigo sobre a STL escrito por Alexander Stepanov. Stepanov, um dos criadores da biblioteca padrão de gabaritos, fornece informações sobre o uso da STL em programação genérica.
http: //www. sgi com/Technology/STL/drdobbs-interview.html
Uma entrevista com Alexander Stepanov que tem algumas informações interessantes acerca da criação da biblioteca
padrão de gabaritos. Stepanov fala sobre como a STL foi concebida, programação genérica, o acrônimo STL” e mais.
toriais, ANSI/ISO C++ Standard
http://www.ansi.org/
Você pode comprar uma cópia completa do documento do padrão C++ neste site.
Software
intrará
http: //www. cs rpi edu/-musser/stl .html
O site RPI STL inclui informação sobre como a STL difere de outras bibliotecas de C++ e como compilar programas que usem a STL, lista dos principais arquivos de include para a STL, exemplos de programas que usam a STL, classe contêineres
ineres, da STL e categorias de iteradores da STL. Ele também fornece uma lista de compiladores compatíveis com a STL, sites FTP para código-fonte da STL e materiais relacionados.
1 http://www.mathcs.sjsu.edu/facu1ty/horstinan/safest1 .htnsJ.
los do Download do SAFESTL.ZIP, uma ferramenta projetada para encontrar erros em programas que usam a STL.
http: //www. objectspace. com/ jgl/
Object Space fornece informação sobre portar programas escritos em C++ para Java, Você pode fazer o download grátis de suas bibliotecas de classe portáveis Standards<ToolKit>. Aspectos-chave do “conjunto de ferramentas” incluem contêineres, iteradores, algoritmos, alocadores, strings e exceções.
http: //www. cs rpi edu/wiseb/st1-bor1and - html

“Using the Standard Template Library with Borland C++”. Este site e uma referência útil para pessoas que usam o
e. Você compilador Borland C++. O autor tem seções sobre advertertências e incompatibilidades.
IaSTL.
http://msdn.microsoft.com/visua1cI
Esta é a homepage do Microsoft Visual C++. Aqui você pode encontrar as mais recentes notícias, atualizações, recursos
técnicos, exemplos e downloads de Visual C++.

http: //www borland. cozn/bcppbuilder/
Esta é a homepage do Borland C++ Builder. Aqui você pode encontrar diversos recursos para C++. incluindo diversos grupos de notícias sobre C++, informações sobre as últimas melhorias de produtos, FAQs e muitos outros recursos para programadores que usam o C++ Builder.
Bibliografia da STL
(Am97) Ammeraal, L., STLfor C++ Programmers, New York, NY: John Wiley, 1997.
(G195) Glass, G., e B. Schuchert, The STL <Primer>, Upper Saddle River, NJ: Prentice Hail PTR, 1995.
(He97) Henricson, M., e E. Nyquist, Industrial Strength C++: Ruies e Recornmendations, Upper Saddle River, NJ: Prentice Hall, 1997.
(Jo99) Josuttis, N., The C++ Standard Library:A Tutorial and Handbook, Reading, MA: Addison-Wesley, 1999.
(Ko97) Koenig, A., e B. Moo, Ruminations on C++, Reading, MA: Addison-Wesley, 1997.
(Mu94) Musser, D. R., e A. A. Stepanov, “Algorithm-Oriented Generic Libraries,” Software Practice e Experience, Vol. 24, No. 7, July 1994.
(Mu96) Musser, D. R., e A. Saini, STL Tutorial e Reference Guide: C++ Programming wjth the Standard Template Libraty, Reading, MA: Addison-Wesley, 1996.
(Ne95) Nelson, M., C+ + Programmer’s Guide to the Standard Template Library, Foster City, CA: Programmers Press, uma divisão de IDG Books Worldwide, mc., 1995.
(Po97) Pohl, 1., C++ Distilled: A ConciseANSi/ISO Reference and Siyle Guide, Reading, MA: Addison-Wesley, 1997.
(Po97a) Pnhl, 1., Object-Oriented Programming Using C++, Second F.dition. Reading, MÁ: Addison-Wesley, 1997.
(RoOO) Robson, R., Using the STL: The C++ Standard Template Library, Springer Verlag, 2000.
(Sc99) Schildt, H., STL Programmingfrom the Ground Up, Osborne McGraw-Hill, 1999.
(Sr94) Stroustrup, B., “Making a vector Fit for a Standard,” The C++ Report, October 1994.
(Sr94a) Stroustrup, B., The Design and Evolution of C++, Reading, MA: Addison-

Wesley, 1994.
(Sr97) Stroustrup, B., The C++ Programming Language, Third Edition, Reading, MA: Addison-Wesley, 1997.
(St95) Stepanov, A., e M. Lee, “The Standard Template Library”, Internet Distribution, Publicado em ftp: II butier hpl hp. com/stl, July 7, 1995.
(Vi94) Vilot, M. J., “An Introduction to the Standard Template Library,” The C++ Report, Vai. 6, No. 8, October 1994.

rsos
21

L. 24,

Acréscimos à linguagem		
padrão	C++	

Objetivos

• Compreender e usar o tipo de dados bool.
• Ser capaz de usar operadores de coerção: staticcast,
uma const_cast e reinterpretcast.
• Compreender o conceito de ambientes de nomes.
• Compreender e usar informações sobre tipos durante a execução (RTFT)e os operadores typeide dynainiccast.
• Compreender as palavras-chave operadores.
• Compreender construtores explicit.
• Usar membros mutable em objetos corist.
• Compreender e usar os operadores ponteiros para membros de classes *e_>*
• Compreender o papel das classes base virtual na herança múltipla.

O que há em um nome? O que chamamos de rosa sob qualquer outro nome soaria tão doce.
William Shakespeare, Romeu e Julieta

Oh, diamante!, diamante! mal tendes conhecimento do dano causado.
Sir Tsaac Newton

sê verdadeiro contigo mesmo
William Shakespeare, Hamlet

A sorte está lançada.
Júlio César

1004 C++ COMO PROGRAMAR
Visão Geral
21.1 Introdução
21.2 O tipo de dados bool
21.3 O operador static_cast
21.4 O operador const_cast
21.5 O operador reinterpret_cast
21.6 Ambientes de nomes
21.7 Informação sobre tipo durante a execução - RTTI
21.8 Palavras-chave operadores
21.9 Construtores explicit
21.10 Membros de classe mutable
21.11 Ponteiros para membros de classe (.* e _>*)
21.12 Herança múltipla e classes base virtual
21.13 Observações finais

Resumo• Terminologia Erros comuns de programação Boas práticas de programação Dica de desempenho . Dicas de portabilidade Observação de engenharia de software Dicas de teste e depura ção Exercícios de auto-revisão Respostas aos exercícios de auto-revisão • Exercícios

21.1 Introdução

Agora examinaremos alguns recursos de C++ padrão, incluindo o tipo de dados bool, operadores de coerção, ambientes de nomes (namespace) informações sobre tipos durante a execução (RTTI) e palavras-chave operadores. Também discutiremos operadores ponteiros para membros de classes e classes base virtual.

21.2 O tipo de dados bool

C++ padrão fornece o tipo de dados bool, cujos valores podem ser false ou true, como alternativa preferencial ao estilo antigo de uso de O para indicar falso e um valor diferente de zero para indicar verdadeiro, O programa da Fig.21.l demonstra o tipo de dados bool.

```
1 II Fig. 21.1: fig2lOl.cpp
2 II Demonstrando o tipo de dado bool.
3 #include <iostream>
4
5 using std::cout;
6 using std: :eridl;
7 using std::cin;
8 using std: :boolalpha;
9
10 int main()
11
12 bool boolean = false;
13 intx=O;
14
```

Fig. 21.1 Demonstrando o tipo de dados primitivo bool (parte 1 de 2).


```
15 cout « “boolean é “ « boolean
16 « “\nDigite um inteiro: “;
17 cm » x;
18
19 cout « “o inteiro “ « x « “ é”
20 « ( x ? “ diferente de zero “ : “ zero
21 « “e interpretado como “;
22
23 if(x)
24 cout « “true\n”;
25 else
26 cout « “false\n’;
27
28 boolean = true;
```

```
29 cout « “boolean é “ « boolean;
30 cout « “\nboolean exibido com o manipulador boolalpha é
31 « boolalpha « boolean « endl;
32
33 return 0;
34
boolean é O
Digite um inteiro: 22
o inteiro 22 é diferente de zero e interpretado como true
boolean é 1
boolean exibido com o manipulador boolalpha é true
```

Fig. 21.1 Demonstrando o tipo de dados primitivo bool (parte 2 de 2).

tão, A linha 12

```
dobool
```

```
boolean = false;
```

declara a variável boolean como do tipo bool e inicializa boolean como false. A variável x é declarada e inicilizada com 0. A linha 15

```
cout « “boolean é “ « boolean
```

exibe o valor de boolean. O valor o é exibido em lugar da palavra-chave false. O default para display de valores bool é o valor numérico correspondente.

O valor dcx (lido na linha 17) é usado como a condição de um if/else na linha 23. Se x é 0, a condição é false. Caso contrário, a condição é true. Note que valores negativos são diferentes de zero e, portanto, true.

A linha 28 atribui true para boolean. O valor de boolean (1) é exibido na linha 29. Uma variável bool

é exibida como O ou 1 por default. O operador de inserção em stream, «, foi sobrecarregado para exibir bools como inteiros.

As linhas 30 e 31

```
cout « “\nboolean exibido com o manipulador boolalpha é
« boolalpha « boolean « endl;
```

usam “o” manipulador de stream boolalpha para preparar o stream de saída para exibir valores bool como os strings “true” e “false”. O manipulador boolalpha também pode ser usado para entrada.

Ponteiros, ints, doubles, etc. também podem ser convertidos implicitamente para bools. Valores zero são

convertidos para false e valores diferentes de zero são convertidos para true. Por exemplo, a expressão

```
bool dc = false + x * 2 - b && true;
```

atribuiria true a dc assumindo-se que x é 3 e b é true. Note que a avaliação do lado direito da expressão de atribuição dá o valor 5, mas este valor é implicitamente convertido para true.

Boa prática de programação 21.1

Quando criar variáveis de estado para indicar verdade ou falsidade, dê preferência ao uso de bools em

vez de ints.
Boa prática de programação 21.2
Usar true e false em lugar de valores zero e diferentes de zero torna os programas mais claros.
21.3 O operador static_cast
C++ padrão contém quatro operadores de coerção para serem usados de preferência ao “velho estilo” de coerção que tem sido usado em C e C++. Os novos operadores de coerção são menos poderosos e mais específicos do que o estilo antigo de coerção, o que dá ao programador um controle mais preciso. A coerção é perigosa, podendo freqüentemente ser uma fonte de erros; assim, os operadores de coerção no novo estilo são também mais fáceis de detectar e pesquisar usando-se ferramentas automatizadas. Uma outra vantagem das coerções no novo estilo é que os quatro operadores têm finalidades completamente diversas, enquanto que com o velho estilo de coerção a filosofia era “um operador de coerção para tudo”.
C++ fornece o operador static cast para conversão entre tipos. A verificação de tipos é feita durante a
compilação. O operador static cast executa conversões padrão (por exemplo, void * para char , int para float, etc.) e suas inversas. A Fig. 21.2 demonstra o operador staticcast.
Erro comum de programação 21.1
Executar uma coerção ilegal com o operador static cast é um erro de sintaxe. Coerções ilegais incluem fazer uma conversão de tipos const para tipos não-const, fazer coerção de ponteiros e referências entre tipos que não são relacionados por herança public e fazer coerção para um tipo para o qual não há um construtor ou operador de conversão apropriado para executar a conversão.
O programa declara as classes BaseClass e DerivedClass. Cada classe define uma função membro f. As linhas 23 e 24
```
double d = 8.22;
int x = static cast< int >( d );
```
declaram e inicializam tanto d como x. O operador static_cast converte d de double para int. O operador static_cast pode ser usado para a maioria das conversões entre tipos de dados primitivos, tais como int, float, double, etc.
```
1 // Fig. 21.2: fig2lO2.cpp
2 II Demonstrando o operador static_cast.
3 #include <iostreain>
4
5 using std: :cout;
6 using std::endl;
7
8 class BaseClass {
9 public:
10 void f( void ) const { cout « “BASE\n”; }
11 };
12
13 class DerivedClass : public BaseClass {
```

Fig. 21.2 Demonstrando o operador static cast (parte 1 de 2).

-


```
14 public:
15 void f( void ) const { cout « “DERIVADA\n;
16
17
18 void test( BaseClass *
19
20 int main()
21 (
22 Il usa static_cast para uma conversão
23 double d = 8.22;
24 int x = staticcast< int >( d );
25
26 cout « ‘d é « d « \nx é « x « endi;
27
28 BaseClass * basePtr = new DerivedClass;
29 test( basePtr ); // chama test
30 delete basePtr;
31
32 return 0;
33 )
34
35 void test( BaseClass * basePtr
36
37 DerivedClass *derjvedptr;
38
39 // faz coerção de ponteiro p/classe base em ponteiro p/classe derivada
40 derivedPtr = static_cast< DerivedClass * >( basePtr );
41 derivedPtr->f O); // invoca a função f de DerivedClass
42
```

Fig. 21.2 Demonstrando o operador static_cast (parte 2 de 2).

Observação de engenharia de software 21.1

_____ Com os acréscimos dos novos operadores de coerção (p.ex., static cast) ao C++ padrão, os opera- dores de coerção no estilo antigo (estilo de C) são considerados obsoletos.

Boa prática de programação 21.3

Use o operador static cast, mais seguro e confiável, em preferência ao operador de coerção no
estilo de C.

Na linha 28, um novo objeto DerivedClass é atribuído ao ponteiro para BaseClass basePtr e passado para a função test na linha 29. O endereço

passado para test é recebido pelo ponteiro basePtr. O ponteiro derivedPtr de DerivedClass é declarado na linha 37. A linha 40
derivedPtr = static cast< DerivedClass * >(basePtr);
usa static_cast para tazer um downcasting de BaseClass * para DerivedClass . Embora (como vimos no Capítulo 9) fazer um downcasting de um ponteiro de uma classe base para um ponteiro de uma classe derivada seja uma operação potencialmente perigosa, static_cast permite essa coerção. A função f é invocada usando-se derivedPtr (linha 41).

d é 8.22
xé 8
DERIV ADA

21.4 O operador const_cast

C++ fornece o operador const_cast para eliminar os atributos de const e volatile. O programa da Fig. 21.3 demonstra o uso de const_cast.

```
1 // Fig. 21.3: fig2l_03.cpp
2 II Demonstrando o operador const_cast.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 class ConstCastTest
9 public:
10 void setNumber( int );
11 int getNumber() const;
12 void printNumber() const;
13 private:
14 int number;
15 };
16
17 void ConstCastTest::setNuinber( int num ) { number num;
18
19 int ConstCastTest::getNumber() const { return number;
20
21 void ConstCastTest::printNumber() const
22
23 cout « “\nNumber após modificação: “;
24
25 // a expressão number-- geraria um erro de compilação
```

```
26 // elimina atributo de const para permitir a modificação
27 const_cast< ConstCastTest * >( this )->number--;
28
29 cout « nunber « endl;
30
31
32 int main()
33 {
34 ConstCastTest x;
35 x.setNuxnber( 8 ); // atribui 8 ao dado private number
36
37 cout « “Valor inicial de number: « x.getNumberO;
38
39 x.printNuniberO;
40 return 0;
41 }
Valor inicial de number: 8
Nurnber após modificação: 7
```

Fig. 21.3 Demonstrando o operador const_cast.

As linhas 8 a 15 declaram a classe ConstCastTest que contém três funções membro e a variável private nuniber. Duas das funções membro são declaradas const. A função setNuniber define o valor de number. A
função getNumber retorna o valor de nuznber.

L

A função membro const printNumber modifica o valor de number na linha 27 daFig. constcast< ConstCastTest * >(this)->nuinber--;

Na função membro const printNumber. o tipo de dados do ponteiro this é const ConstCastTest*.

O comando anterior retira o atributo const do ponteiro this com o operador const_cast. Agora, o tipo do ponteiro this para o restante do comando é ConstCastNumber . Isto permite que number seja modificado, O operador const cast não pode ser usado para retirar diretamente o atributo const de uma variável constante.

21.5 O operador reinterpret_cast

C++ fornece o operador reinterpret cast para coerções não-padrão (por exemplo, coerção de um tipo de ponteiro para outro tipo de ponteiro, etc.). O operador reinterpret cast também pode ser usado para coerções padrão (i. e., double para int, etc.). A Fig. 21.4 demonstra o uso do operador reinterpret_cast.

```
1 Il Fig. 21.4: fig2104.cpp
2 Il Demonstrando o operador reinterpret_cast.
3 #include <iostreain>
4
5 using std::cout;
6 using std::endl;
```

```
7
8 int main()
9{
10 int x = 120, *ptr =
11
12 cout « *remnterpretcast<char *>( ptr ) « endi;
13
14 return 0;
15 }
```
x
Fig. 21.4 Demonstrando o operador reinterpretcast.
O programa declara um inteiro e um ponteiro. O ponteiro ptr é inicializado como endereço de x. A linha 12 cout « *reinterpretcast<char *>(ptr) « endl;
usa o operador reinterpret cast para fazer a coerção de ptr (do tipo int *) para char . O endereço
retornado é derreferenciado.
® Dica de teste e depura ção 21.1
É fácil usar reinterpret cast para executar manipulações perigosas que podem levar a sérios
erros durante a execução.
Dica de portabilidade 21.1
ivate
nber. A Usar reinterpret cast pode fazer com que os programas se comportem de modo diferente em diferentes plataformas.

21.6 Ambientes de nomes
4 Um programa inclui muitos identificadores definidos em diferentes escopos. Às vezes, o escopo de uma variável se “sobrepõe ao” (i.e., colide com o) escopo de uma variável com o mesmo nome em um escopo diferente, criando um problema potencial. Tais sobreposições podem ocorrer em muitos níveis. A sobreposição de identificadores ocorre com freqüência com bibliotecas fornecidas por terceiros que usam os mesmos nomes para identificadores globais (tais como funções). Quando isto ocorre, geralmente são gerados erros de compilação.
Boa prática de programação 21.4
Evite começar identificadores com o caractere sublinhado, o que pode levar a erros de ligação.
C++ padrão tenta resolver este problema com ambientes de nomes (namespaces). Cada namespace define um escopo onde identificadores e variáveis são colocados. Para usar um membro de um namespace. o nome do membro deve ser qualificado com o nome do naxnespace e o operador binário de resolução de escopo (: :) como segue:
namespace_nome: : membro
ou um comando using deve aparecer antes de o nome ser usado; comandos using são tipicamente colocados no começo do arquivo no qual membros do namespace são usados. Por exemplo, o comando

us ing namespace nome_do_namespace
no começo de um arquivo-fonte especifica que os membros do nainespace nome_do_namespace podem ser usados no arquivo sem preceder cada membro com o nome_do_namespace e o operador de resolução de escopo (: :).

Boa prática de programação 21.5
Preceda um membro pelo nome do seu namespace e o operador de resolução de escopo (: :) se existe possibilidade de um conflito de escopo.
Nem todos os namespaces são garantidamente únicos. Dois fornecedores independentes podem inadvertidamente usar o mesmo namespace. A Fig. 21.5 demonstra o uso de namespaces.
A linha 4
using namespace std; II usa o ambiente de nomes std
informa ao compilador que o namespace std está sendo usado. Os conteúdos do arquivo de cabeçalho <iostream> são todos definidos como sendo parte do namespace std. [Nota: a maioria dos programadores de C++ considera uma prática de programação sobre escrever um comando using como o da linha 4, porque todo o conteúdo do ambiente de nomes é incluído. 1

```
1 II Fig. 21.5: fig2lO5.cpp
2 // Demonstrando ambientes de nomes.
3 #include <iostream>
4 using namespace std; II usa o ambiente de nomes std
5
6 int mylnt = 98; II variável global
7
8 namespace Example {
9 const double P1 = 3.14159;
10 const double E = 2.71828;
11 int mylnt = 8;
12 void printValuesO;
13
14 namespace Inner { II ambiente de nomes aninhado
15 enum Years { FISCAL1 = 1990, FISCÃL2, FISCAL3 };
```

Fig. 21.5 Demonstrando o uso de namespaces (parte 1 de 2).


```
16 }
17 )
18
19 naxnespace { II ambiente de nomes sem nome
20 double d = 88.22;
21
22
23 int main()
24
25 II exibe o valor de d do ambiente de nomes sem nome
```

```
26 cout « “d = “ « d;
27
28 II exibe a variável global
29 cout « “\n(global) mylnt = « mylnt;
30
31 II exibe os valores do ambiente de nomes Example
32 cout « \nPI = ‘ « Example::PI « \nE =
33 « Exainple: :E « “\nmylnt =
34 « Example: :mylnt « “\nFISCAL3 =
35 « Example::Inner::FISCAI3 « endl;
36
37 Example::printValuesO; II invoca a função printValues
38
39 return 0;
40
41
42 void Example::printValues()
43 {
44 cout « \nEm printValues:\n” « “mylnt =
45 « mylnt « \nPI = « P1 « \nE =
46 « E « “\nd = « d « “\n(global) mylnt
47 « : :mylnt « ‘\nFISCAL3 =
48 « Inner::FISCAL3 « endl;
49 }
```

```
d = 88.22
(global) mylnt = 98
P1 = 3.14159
E = 2.71828
mylnt = 8
FISCAL3 = 1992
Em printValues:
mylnt = 8
P1 = 3.14159
E = 2.71828
d = 88.22
(global) mylnt = 98
FISCAL3 = 1992
```

Fig. 21.5 Demonstrando o uso de nomespaces (parte 2 de 2).

O comando using namespace especifica que os membros de um namespace serão usados freqüentemente ao longo de um programa. Isto permite ao programador acessar todos os membros do namespace e escrever comandos mais concisos, tais como

```
cout « d =“ « d;
```


em vez de

std::cout « “d = « d;
Sem a linha 4, cada cout e endi na Fig. 21.5 teria que ser qualificado com std: : .
O comando using namespace pode ser usado para namespaces predefinidos (por exemplo, std) ou namespaces definidos pelo programador.
As linhas 8 a 17
nainespace Example {
const double P1 = 3.14159;
const double E = 2.71828;
int mylnt = 8;
void printValues
namespace Inner { II ambiente de nomes aninhado
enum Years { FISCAL1 = 1990, FISCAL2, FISCAL3 };
usam a palavra-chave narnespace para definir o nainespace Example. O corpo de um namespace é delimitado por chaves ({ }). Diversamente dos corpos de classes, os corpos de namespaces não terminam em ponto-e- vírgula. Os membros de Exainple consistem em duas constantes (P1 e E), um int (mylnt), uma função (printValues) e um namespace aninhado (Inner). Note que o membro mylnt tem o mesmo nome que a variável global ntylnt. Variáveis que têm o mesmo nome devem ter escopos diferentes - caso contrário, ocorrerão erros de sintaxe. Um namespace pode conter constantes, dados, classes, namespaces aninhados, funções, etc. As definições de namespaces devem ocupar o escopo global ou estar aninhadas dentro de outros namespaces.
As linhas 19 a 21
namespace { II ambiente de nomes sem nome
double d = 88.22;
criam um namespace não-nomeado contendo o membro d. Os membros de namespaces não-nomeados ocupam o namespace global, são acessáveis diretamente e não têm que ser qualificados com o nome de um names- pace. Variáveis globais também são parte do namespace global e são acessáveis em todos os escopos que seguem a sua declaração em um arquivo.
Observação de engenharia de software 21.2
Cada unidade de compilação separada tem seu próprio e único namespace não-nomeado, ou seja, o
namespace não-nomeado substitui o especficador de ligação static.
A linha 26 exibe o valor de dO membro d é diretamente acessável como parte do namespace não-nomeado. A linha 29 exibe o valor da variável global mylnt.As linhas 32 a 35
cout « “\nPI = “ « Example::PI « “\nE =
« Example: :E « “\nmylnt =
« Example::mylnt « “\nFISCAL3 =
« Example::Inner::FISCAL3 « endi;
exibem os valores de P1. E. mylnt e FISCAL3. P1. E e mylnt são membros de Example e, portanto, são qualificados com Exanple: :. O membro mylnt deve ser qualificado porque existe uma variável global com o mesmo nome. Caso contrário, é exibida a variável global. FISCAL3 é um membro do namespace aninhado Inner e é qualificado com Example: : Inner::.

A função printValues é um membro de Example e pode acessar diretamente outros membros do mesmo namespace sem usar um qualificador de namespace. O cout na linha 44 exibe mylnt, P1, E, d, a variável global mylnt e FISCAL3. Repare que P1 e E não são qualificados com Example, d ainda está acessável, a versão global de mylnt foi qualificada com o operador de resolução de escopo (: :) e FISCAI3 foi qualificado com

Inner: :. Quando acessar membros de um namespace aninhado, os membros devem ser qualificados com o nome do namespace (a menos que você esteja dentro do namespace aninhado).
A palavra-chave using também pode ser usada para permitir que um membro individual de um namespa- ce seja usado. Por exemplo, a linha
using Example::PI;
permitiria que P1 fosse usado sem a qualificação de nainespace. Isto é feito tipicamente quando somente um membro de um naxnespace é usado com freqüência. namespaces podem ter nomes alternativos (aliases). Por exemplo, o comando
namespace CPPHTP3E = CPlusPlusHowToProgram3E;
cria o alias CPPHTP3E para CPlusPlusHowToPrograin3E.
Erro comum de programação 21.2
Colocar main em um namespace é um erro de sintaxe.
Observação de engenharia de software 21.3
_____ Em programas grandes, idealmente cada entidade deve ser declarada em uma classe, função, bloco ou
el namespace. Isto ajuda a deixar claro o papel de cada entidade.
21.7 Informação sobre tipo durante a execução - RTTI
A informação sobre tipo durante a execução (RTTI, run-time tvpe information) fornece um meio de determinar o tipo de um objeto durante a execução. Nesta seção, são discutidos dois importantes operadores de RTTI: typeid e dynamiccast. O programa da Fig. 21.6 demonstra typeid e o programa da Fig. 21.7 demonstra dynamic_cas t.
® Dica de teste e depura ção 21.2
De modo a poder usar R777, alguns compiladores requerem que os recursos de RTI'i sejam habilitados.
Verifique a documentação do seu compilador sobre o uso de RTI'I.

```
1 II Fig. 21.6: fig2l_06.cpp
2 // Demonstrando o recurso typeid de RTTI.
3 #include <iostreani>
4
5 using std::cout;
6 using std::endl;
7
8 #include <typeinfo>
9
```

```
10 template < typenaxne T >
11 T maximum( T valuel, T value2, T value3
12
13 T max = valuel;
14
15 if ( value2 > max
O 16 max = value2;
17
18 if ( value3 > max
19 max = value3;
20
```

Fig. 21.6 Demonstrando typeid (parte 1 de 2).


```
21 // obtém o nome do tipo (i.e., int ou double)
22 const char *dataType = typeid( T ).nameO;
23
24 cout « dataType « “s foram comparados.\nO maior
25 « dataType « é
26
27 return max;
28
29
30 int main()
31 {
32 inta=8,b=88, c=22;
33 double d = 95.96, e = 78.59, f = 83.89;
34
35 cout « maximum( a, b, c ) « “\n”;
36 cout « maximum( d, e, f ) « endl;
37
38 return 0;
39
```

Fig. 21.6 Demonstrando typeid (parte 2 de 2).

A linha 8 inclui o arquivo de cabeçalho <typeinfo>. Quando se usa o resultado de typeid, <typeinfo> é obrigatório, O programa define um gabarito de função maximum que recebe três parâmetros do tipo de dado T especificado e determina e retorna o maior dos três. A palavra-chave typename é usada no lugar da palavra-chave class. Nesta situação, typename se comporta de maneira idêntica a class.

A linha 22

```
const char *dataType = typeid( T ).nameO;
```

usa a função name para retornar um string no estilo de C, definido pela implementação, representando o tipo de dados de T. O operador typeid retorna

uma referência para um objeto type info. Um objeto type info é um objeto mantido pelo sistema que representa um tipo. Note que o string retornado por name é mantido pelo sistema e não deve ser deletado pelo programador.
Boa prática de programação 21.6
Usar typeid em testes do tipo switch é um mau uso de RTrl. Em vez disso, use funções virtual.
O operador dynamic cast assegura que as conversões adequadas sejam feitas durante a execução (ou seja, o compilador não pode verificar se é ou não uma conversão adequada). O operador dynamic_cast é freqüentemente usado para fazer downcasting (coerção) de um ponteiro de uma classe base para um ponteiro de uma classe derivada. O programa da Fig. 21.7 demonstra dynamiccast.
1 II Fig. 21.7: fig2lO7.cpp
2 // Demonstrando dynamic_cast.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
Fig. 21.7 Demonstrando dynamic cast (parte 1 de 3).

ints foram comparados.		
O maior	int é	88
doubles	foram	comparados.
O maior	double	é 95.96

7
8 const double P1 = 3.14159;
9
10 class Shape
11 public:
12 virtual double area() const { return 0.0;
13
14
15 class Circle : public Shape
16 public:
17 Circle(int r = 1) { radius = r;
18
19 virtual double area() const
20
21 return P1 * radius * radius;
22

```
23 protected:
24 int radius;
25
26
27 class Cylinder : public Circle
28 public:
29 Cylinder( int h = 1 ) { height = h;
30
31 virtual double area() const
32
33 return 2 * * radius * height +
34 2 * Circle::areaO;
O>é 35 }
doT 36 private:
have 37 int height;
38
39
40 void outputShapeArea( const Shape * ); // protótipo
41
42 int niain()
Iode 43 {
foe 44 Circle circle;
pelo 45 Cylinder cylinder;
46 Shape *ptr = 0;
47
48 outputShapeArea( &circle ); II exibe a área do círculo
49 outputShapeArea( &cylinder ); II exibe a área do cilindro
50 outputShapeArea( ptr ); II tenta exibir a área
51 return 0;
ja,o 52
asse 54 void outputShapeArea( const Shape *shapePtr )
55
56 const Circle *circleptr;
57 const Cylinder *cylinderPtr;
58
59 // faz coerção de Shape * para um Cylinder *
60 cylinderPtr dynamic_cast< const Cylinder * >( shapePtr );
61
62 if ( cylinderPtr != O ) II setrue, invoca area()
63 cout « Área do cilindro: « shapePtr->areaO;
```

Fig. 21.7 Demonstrando dynamic cast (parte 2 de 3).


```
64 else { II shapePtr não aponta para um cilindro
65
66 II faz coerção de shapePtr para uni Circie *
```

```
67 circlePtr = dynaxnic_cast< const Circle * >( shapePtr );
68
69 if ( circlePtr != O ) // se true, invoca area()
70 cout « “Área do círculo: “ « circlePtr->area
71 else
72 cout « “Nem um círculo nem um cilindro.”;
73 }
74
75 cout « endl;
76
Área do círculo: 3.14159
Área do cilindro: 12.5664
Nem um círculo nem um cilindro.
```

Fig. 21.7 Demonstrando dynamic cast (parte 3 de 3).

O programa define a classe base Shape (linha 10) que contém a função virtual area, uma classe derivada Circle (linha 15) que herda publicamente de Shape e uma classe derivada Cylinder (linha 27) que herda publicamente de Circle. Tanto Circle como Cylinder sobrescrevem a função area.

Na função main, nas linhas 44 a 46, são instanciados um objeto da classe Circie chamado circie, um objeto da classe Cylinder chamado cylinder, e um ponteiro para um Shape chamado ptr é declarado e inicializado com zero. As linhas 48 a 50 chamam a função outputShapeArea (definida na linha 54) três vezes. Cada chamada a outputShapeArea exibirá um de três resultados - a área de um Circle, a área de um Cylinder ou a indicação de que o objeto Shape não é nem um Circle nem um Cylinder. A função outputShapeArea recebe como argumento um ponteiro para um Shape - a primeira chamada recebe o endereço de circle, a segunda chamada recebe o endereço de cylinder e a terceira chamada recebe um ponteiro para a classe base Shape chamado ptr.

A linha 60

cylinderPtr = dynamic cast< const Cylinder * >(shapePtr);

efetua dinamicamente a conversão de shapePtr (um const Shape *) para um const Cylinder * usando o
operador de coerção dynamic cast. Como resultado, ou o endereço do objeto cylinder ou O é atribuído a
cylinderPtr, para indicar que o Shape não é um Cylinder. Se o resultado da coerção não é o, é exibida a
área do Cylinder.

A linha 67

circlePtr = dynamic_cast< const Circle * >(shapePtr);

faz a conversão dinâmica de shapePtr para um const Circle * (downcast) usando o operador de coerção dynamiccast. Como resultado, ou o endereço do objeto circie ou O é atribuído a circlePtr, para indicar que o Shape não é um Circle. Se o resultado da coerção não é o, é exibida a área do Circle.

Erro comum de programação 21.3

Tentar usar dynamic cast sobre um ponteiro do tipo void * é um erro de sintaxe.

Observação de engenharia de software 21.4

_____ Riu foi concebido para ser usado com hierarquias de herança polimómficas (com funções virtual).

CAPÍTULO 21 - AcRÉsciMos À LINGUAGEM PADRÃO C ++ 1017

21.8 Palavras-chave operado res

C++ padrão fornece palavras-chave operadores (Fig. 21.8) que podem ser usadas no lugar de vários operadores C++. Palavras-chave operadores podem ser úteis para teclados de programadores que não suportam certos caracteres, tais como!, &. ‘, 1 etc.

O programa na Fig. 21.9 demonstra o uso de palavras-chave operadores. Este programa foi compilado com o Microsoft Visual C++, que requer o arquivo de cabeçalho <iso646 h> para usar palavras-chave operadores. Outros compiladores podem diferir do acima; assim, verifique a documentação do seu compilador para determinar qual arquivo de cabeçalho deve ser incluído (eventualmente, o compilador pode não necessitar da inclusão de qualquer arquivo de cabeçalho para usar estas palavras-chave).

O programa declara e inicializa dois inteiros, a e b. Operações lógicas e sobre bits são executadas com a e b usando as várias palavras-chave operadores. O resultado de cada operação é exibido.

Fig. 21.9 Demonstrando as palavras-chave operadores (parte 1 de 2).

Operador Palavra-chave operador Descrição
Operadores lógicos
&& and E lógico
1 or OU lógico
not negação lógica
Operadores de desigualdade
= noteq desigualdade
Operado res sobre bits
& bitand E sobre bits
bitor OU inclusivo sobre bits
xor OU exclusivo sobre bits
- compi complemento de bits
Operadores de atribuição sobre bits
&= and_eq atribuição E sobre bits
= oreq atribuição OU inclusivo sobre bits
xoreq atribuição OU exclusivo sobre bits
Fig. 21.8 Palavras-chave operadores como alternativa para símbolos de operadores.
1 II Fig. 21.9: fig2lO9.cpp
2 // Demonstrando as palavras-chave operadores.
3 #include <iostreani>
4
5 using std::cout;

```
6 using std::endl;
7 using std: :boolalpha;
8
9 #include <iso646.h>
10
11 int main()
12
13 inta=8, b=22;
14
15 cout « boolalpha
16 « a and b: « ( a and b
17 « \n a or b: « ( a or b
18 « \n not a: « ( not a
19 « "\na not_eq b: « ( a not_eq b
```


Fig. 21.9 Demonstrando as palavras-chave operadores (parte 2 de 2).

21.9 Construtores explicit

No Capítulo 8, "Sobrecarga de operadores", discutimos que qualquer construtor que é chamado com um argumento pode ser usado pelo compilador para executar uma conversão implícita, na qual o tipo recebido pelo construtor é convertido em um objeto da classe na qual o construtor está definido. A conversão é automática e o programador não precisa usar um operador de coerção. Em algumas situações, conversões implícitas são indesejáveis ou sujeitas a erros. Por exemplo, nossa classe Array na Fig. 8.4 define um construtor que aceita um único argumento int. O objetivo deste construtor é criar um objeto Array contendo o número de elementos especificado pelo argumento int. Contudo, este construtor pode ser mal-utilizado pelo compilador para executar uma conversão implícita. O programa da Fig. 21.10 usa uma versão simplificada da classe Array do Capítulo 8 para demonstrar uma conversão implícita imprópria.

```
II Fig 21.10: array2.h
1/ Uma classe Array simples (para inteiros) #ifndef ARRAY2H
#define ARP.AY2H

6 #include <iostream>

7

8 using std: :ostream;
```

```
9

10 class Array
11 friend ostream &operator«( ostream &, const Array & );
12 public:
13 Array( int = 10 ); II construtor default/de conversão
14 -ArrayO; II destruidor

20
21
22
23
24
25
26
27
28
29

« ‘\na bitand b:
« “\na bit_or b:
« “\n a xor b:
« “\n compl a:
« “\na and_eq b:
« “\n a or_eq b:
« “\na xor eq b:
return 0;

« ( a bitand b
« ( a bitor b
« ( a xor b
« ( compl a
« ( a and_eq b
« ( a or_eq b
« ( a xor eq b ) « endl;

}

a and b: true
a or b: true not a: false
a not_eq b: false
a bitand b: 22
a bit or b: 22 a xor b: O
compl a: -23 a andeq b: 22 a or_eq b: 30 a xor_eq b: 30
```

1
2
3
4
5

Fig. 21.10 Construtores de um único argumento e conversões implícitas - array2 .h (parte 1 de 2).

15 private:
16 int size; II tamanho do array
17 int *ptr; II ponteiro para o primeiro elemento do array
18
19
20 #endif
Fig. 21.10 Construtores de um único argumento e conversões implícitas - array2 .h (parte 2 de 2).
21 II Fig 21.10: array2.cpp
22 II Definições de funções membro para a classe Array
23 #include <iostream>
24
25 using std::cout;
26 using std::ostream;
27
28 #include <cassert>
29 #include “array2.h’
30
31 II Construtor default para a classe Array (tamanho default 10)
32 Array: :Array(int arraySize
33
34 size = (arraySize > O ? arraySize : 10);
35 cout « Construtor de Array chamado para
36 « size « elementos\n”;
37
38 ptr = new int[size]; II cria espaço para array
39 assert(ptr != O); II termina se memória não foi alocada
umeiliu 40
strutoré 41 for (int i = 0; i < size; i++)
42 ptr[i] = 0; // inicializa array
dornao 43 }
ujeitas a 44
int.O 45 II Destruidor para a classe Array
wmento 46 Array: :Array() { delete [] ptr; }
47
icita., 48 II Operador de saída sobrecarregado para a classe Array

nversao 49 ostream &operator«(ostream &output, const Array &a
50 {
51 int i;
52
53 for (i = 0; i < a.size; i++
54 output « a.ptr[i) «
55
56 return output; II permite usar cout « x « y;
57 }
Fig. 21.10 Construtores de um único argumento e conversões implícitas - array2 .cpp.

Fig. 21.10 Construtores de um único argumento e conversões implícitas - fig2l 10 cpp (parte 1 de 2).

58	// Fig 21.10: fig2l_l0.cpp	
59	II Programa de teste	para a classe Array simples
60	#include <iostream>	
61		
62	using std::cout;	
63		

64 #include “array2.h”
65
66 void outputArray(const Array &);
67
68 int main()
69
70 Array integersi (7);
71
72 outputArray(integersi); II envia Array integersi para a saída
73
74 outputArray(15); II converte 15 para um Array e envia para a saída
75
76 return 0;
77 }
78
79 void outputArray(const Array &arrayToOutput

```
80
81 cout « “O array recebido contém:\n”
82 « arrayToOutput « “\n\n”;
83
Construtor de Array chamado para 7 elementos
O array recebido contém:
0000000
Construtor de Array chamado para 15 elementos
O array recebido contém:
000000000000000
```

Fig. 21.10 Construtores de um unico argumento e conversoes imphcitas - fig2l 10. cpp (parte 2 de 2).

A linha 70 em main

```
Array integersi ( 7 );
```

define o objeto integersi do tipo Array e chama o construtor de parâmetro único com o valor int 7 para especificar o número de elementos no Array. Modificamos o construtor de Array de modo que exiba uma linha de texto indicando que o construtor de Array foi chamado e o número de elementos que foram alocados ao Array. A linha 72

```
outputArray( integersl ); Il envia Array integersi para a saída
```

chama a função outputArray (definida na linha 79) para exibir o conteúdo do Array. A função outputArray recebe como seu argumento um const Array& para o Array e, então, exibe o Array usando o operador de inserção em stream « . A linha 74

```
outputArray( 15 ); Il converte 15 para um Array e envia para a saída
```

chama a função outputArray com o valor int 15 como argumento. Não existe função outputArray que aceite um int como argumento; assim, o compilador testa a classe Array para verificar se existe um construtor de conversão que pode converter um int em um Array. Como a classe Array fornece um construtor de conversão, o compilador usa esse construtor para ciar um objeto Array temporário contendo 15 elementos e passa o objeto Array temporário para a função outputArray para exibir o Array. A saída mostra que construtor de conversão de Array foi chamado para um Array de 15 elementos, bem como o conteúdo do Array.

C++ fornece a palavra-chave explicit para suprimir conversões implícitas através de construtores de
conversão. Um construtor que é declarado explicit não pode ser usado em conversão implícita. O programa da Fig. 21.11 demonstra um contrutor explicit.


```
1 Il Fig. 21.11: array3.h
2 /1 Classe Array simples (para inteiros)
3 #ifndef ARRAY3H
4 #define ARRAY3H
5
6 #include <iostream>
```

```
7
8 using std::ostream;
9
saída 10 class Array
11 friend ostream &operator«( ostream &, const Array & );
12 public:
13 explicit Array( int = 10 ); // construtor default
14 -ArrayO; II destruidor
15 private:
16 int size; // tamanho do array
17 int *ptr; II ponteiro para o primeiro elemento do array
18
19
20 #endif
```

Fig. 21.11 Demonstrando um construtor explicit - array3 .h.

```
21 II Fig. 21.11: array3.cpp
22 II Definições de funções membro para a classe Array
23 #include <iostreani>
24
25 using std::cout;
26 using std::ostream;
27
28 #include <cassert>
29 #include “array3.h”
30
31 II Construtor default para a classe Array (tamanho default 10)
para 32 Array: :Array( int arraySize
alinha 33
ray. 34 size = ( arraySize > O ? arraySize : 10 );
35 cout « Construtor de Array chamado para
36 « size « elementos\n”;
37
rra 38 ptr = new int[ size ]; II cria espaço para o array
dorde 39 assert( ptr = O ); // termina se memória não alocada
40
41 for ( int i = 0; i < size; i++
42 ptr[ i ] = 0; // inicializa array
43 }
yque
itorde 45 // Destruidor para a classe Array
‘ersão 46 Array: :.-Array() { delete [] ptr;
b
O O 48 // Operador de saída sobrecarregado para a classe Array
versao 49 ostream &operator«( ostream &output, const Array &a
50 {
resde 51 int i;
```

grama 52
Fig. 21.11 Demonstrando um construtor explicit - array3 . cpp (parte 1 de 2).


```
53 for ( i O; i < a.size; i++
54 output « a.ptr[ i ] «
55
56 return output; II permite usar cout « x « y;
57 }
```
Fig. 21.11 Demonstrando um construtor explicit - array3 . cpp (parte 2 de 2).
```
58 II Fig. 21.11: fig2l_1l.cpp
59 II Programa de teste para a classe Array simples
60 #include <iostream>
61
62 using std: :cout;
63
64 #include “array3.h”
65
66 void outputArray( const Array & );
67
68 int main()
69
70 Array integersi ( 7 );
71
72 outputArray( mntegersi ); II envia o Array mntegersi para a saída
73
74 II ERRO: construção não permitida
75 outputArray( 15 ); II converte 15 para um Array e envia para a saída
76
77 outputArray( Array( 15 ) ); II realmente quer fazer isto!
78
79 return 0;
80
81
82 void outputArray( const Array &arrayToOutput
83
84 cout « O array recebido contém:\n”
85 « arrayToOutput « “\n\n’;
86
```
Mensagem de erro do compilador Borland C++ de linha de comando Fig2 1_li. cpp:
```
Error E2064 Fig2l l1.cpp 18: Cannot initialize ‘const Array &‘
wjth ‘int’ in function main()
Error E2340 Fig2l l1.cpp 18: Type mismatch in parameter 1 (wanted
const Array &‘, got ‘int’) in function main()
2 errors in Compile ***
```

Mensagem de erro do compilador Microsofi Visual C+ +
Compiling...
Fig2lll . cpp
Fig2l_11.cpp(18) : error C2664: ‘outputArray’ : cannot convert
parameter 1 from ‘const int’ to ‘const class Array &
Reason: cannot convert from const int to ‘const class Array’ No construtor could take the source type, or construtor overload resolution was ambiguous
Fig. 21.11 Demonstrando um construtor explicit - fig2l 11 . cpp.

A única modificação em relação ao programa da Fig. 21.10 foi a adição da palavra-chave explicit na declaração do construtor de parâmetro único na linha 13. Quando o programa é compilado, o compilador produz uma mensagem de erro indicando que o valor inteiro passado para outputArray na linha 75 não pode ser convertido para um const Array&. A mensagem de erro do compilador é mostrada na janela de saída. A linha 77 ilustra como criar um Array de 15 elementos e passá-lo para outputArray usando o construtor explicit.
Erro com um de programação 21.4
Tentar invocar um construtor explicit para uma conversão implícita é um erro de sintaxe.
IErro comum de programação 21.5
Usar a palavra-chave explicit em membros de dados ou funções membro outras que não um construtor de parâmetro único é m erro de sinta.e.
Observação de engenharia de soflware 21.5
_____ Use a palavra-chave explicit em construtores que não deveriam ser usados pelo compilador para efetuar conversões implícitas.
21.10 Membros de classe mutable
ída
Na Seção 21.4, introduzimos o operador const_cast que permitiu retirar o atributo const. C++ fornece o
laída especificador de classe de armazenamento mutable como alternativa a const cast. Um membro de dados mutable é sempre modificável, mesmo em uma função membro const ou em um objeto const. Isso reduz a necessidade de retirar o atributo const.
Dica de portabilidade 21.2
_____ O efeito de tentar modificar um objeto que foi definido como constante, independente do fato de aquela modificação ter sido possibilitada por um const cast ou por uma coerção no estilo de C, varia de
compilador para compiladoic
Tanto mutable como const cast permitem que um membro de dados seja modificado; cada um é usado em um contexto diferente. Para um objeto const sem membros de dados mutable, o operador const_cast deve ser usado toda vez que um membro de dados deve ser modificado. Isto reduz enormemente a possibilidade de que um membro de dados seja modificado acidentalmente, porque o membro de dados não é modificável permanente- mente. Operações envolvendo const cast são tipicamente ocultadas na implementação de uma

função membro. O usuário de uma classe pode não estar ciente de que um membro está sendo modificado.

Observação de engenharia de software 21.6

_____ Membros mutable são úteis em classes que têm detalhes de implementação “secretos” que não contribuem para o valor lógico de um objeto.

O programa da Fig. 21.12 demonstra o uso de um objeto mutable. O programa define a classe TestNutable (linha 8) que contém um construtor, duas funções e o membro de dados value que é private mutable. A linha 11

void modifyValue() const { value++; }

define a função modifyValue como uma função const que incrementa o membro de dados mutable. va- lue. Normalmente, uma função membro const não pode modificar membros de dados, a menos que o objeto sobre o qual a função opera - aquele para o qual this aponta - seja convertido (usando const_cast) para um

tipo não cons t. Como value é mutable, esta função cons t é capaz de modificar os dados. A função getValue (linha 12) é uma função const que retorna value. Note que getValue poderia mudar value, porque value émutable.

```
1 II Fig. 21.12: fig2l_12.cpp
2 // Demonstrando o especificador de classe de armazenamento mutable.
3 #include <iostream>
4
5 using std::cout;
6 using std: :endl;
7
8 class TestMutable {
9 public:
10 TestMutable( int v = O ) { value = v; )
11 void modifyValue() const { value++; }
12 int getValue() const { return value;
13 private:
14 mutable int value;
15
16
17 int main()
18
19 const TestMutable t( 99 );
20
21 cout « “Valor inicial: “ « t.getValueO;
22
23 t.modifyValueQ; II modifica o membro mutable
24 cout « “\nValor modificado: “ « t.getValue() « endl;
25
26 return 0;
27 }
```

Valor inicial: 99

Valor modificado: 100
Fig. 21.12 Demonstrando um membro de dados mutable.
A linha 19 declara o objeto t como const TestMutable e o inicializa com 99. A linha 21 exibe o conteúdo de
value. A linha 23 chama a função membro const modifyValue para adicionar um a value. Note que, tanto
t como modifyValue, são const. A linha 24 exibe o conteúdo de value (100) para provar que o membro
mutable foi de fato modificado.
21.11 Ponteiros para membros de classe (.* e _>*)
C++ fornece os operadores . * e _>* para acessar membros de classes. Ponteiros para membros de classe não são
o mesmo tipo de ponteiros que discutimos anteriormente. Tentar usar o operador -> ou o operador * com um
ponteiro para membro gera erros de sintaxe. O programa da Fig. 2 1.12 demonstra os operadores ponteiros para
membros de classe.
Erro comum de programação 21.6
Tentar usar o operador -> ou o operador * com um ponteiro para membro de classe é erro de sintaxe.


```
e 1 II Fig. 21.13 fig2ll3.cpp
e 2 // Demonstrando os operadores . e ->
3 #i-nclude <iostream>
4
5 using std::cout;
6 using std: :endl;
7
8 class Test {
9 public:
10 void function() { cout « “função\n;
11 int value;
12
13
14 void arrowStar( Test *
15 void dotStar( Test * );
16
17 int main()
18
19 Test t;
20
21 t.value = 8;
22 arrowStar( &t );
23 dotStar( &t );
24 return 0;
```

```
25 }
26
27 void arrowStar( Test *tptr
28
29 void ( Test: :*memPtr ) O = &Test: :function;
30 ( tptr_>*memPtr ) Q; // invoca a função indiretamente
31 }
32
33 void dotStar( Test *tptr
34
35 int Test: :*vPtr = &Test: :value;
36 cout « ( ) .*vptr « endl; II accessa o valor
37 }
```

função
8

e Fig. 21.13 Demonstrando os operadores . * e _>*.
O programa declara a classe Test que fornece a função membro public function e o membro de dados
público value. A função function exibe a palavra “function”. As linhas 14 e 15 apresentam os protótipos
das funções arrowS tar e dotS tar. Nas linhas 19 a 21,0 objeto t é instanciado e o membro de dados value de
t é definido como 8. As linhas 22 e 23 chamam as funções arrowStar e dotStar; cada chamada passa o
endereço de t.
A linha 29

```
void ( Test: :*memptr ) () = &Test: :function;
```

na função arrowS tar declara e inicializa memPtr como um ponteiro para um membro da classe Tes t que é uma função com uni resultado void e sem parâmetros. Começamos examinando o lado esquerdo da atribuição. Primeiro, void é o tipo de retorno da função membro. Os parênteses vazios indicam que esta função membro não recebe argumentos. Os parênteses do meio especificam um ponteiro memPtr que aponta para um membro da classe Test. Os parênteses em torno de Test: : memPtr são necessários. Note que, se Test: : não for especificado, memPtr é um ponteiro para função padrão. Em seguida, examinaremos o lado direito da atribuição.

Erro comum de programação 21.7
Declarar um ponteiro de função membro sem colocar o nome do ponteiro entre parênteses é um erro de sintaxe.
Erro comum de programação 21.8
Declarar um ponteiro de função membro sem preceder o nome do ponteiro por um nome de classe seguido pelo operador de resolução de escopo (: :) é um

erro de sintaxe.
O lado direito da atribuição usa o operador endereço (&) para obter o deslocamento dentro da classe para a função membro function (que deve retomar void e não aceita argumentos). O ponteiro memPtr é inicializado com este deslocamento. Note que tanto o lado esquerdo como o lado direito da atribuição na linha 29 não se referem a nenhum objeto específico. Somente o nome da classe é usado com o operador binário de resolução de escopo (: :). Sem &Test: :, o lado direito da atribuição na linha 29 é um ponteiro de função padrão. A linha 30
tPtr_>*memPtr) o; // invoca a função indiretamente
invoca o nome da função membro armazenado em memPtr (i.e., function) usando o operador ->. A linha 35 int Test: :*vPtr = &Test: :value;
declara e inicializa vPtr como um ponteiro para um membro de dados int da classe Test. O lado esquerdo da atribuição especifica o nome do membro de dados value. Note que sem Test: :. Ptr se torna um ponteiro int
* para o endereço de int va].ue.
A próxima linha
cout « (*jp) .*vPtr « endl; // accessa o valor
usa o operador . * para acessar o membro cujo nome está em vPtr. Note que em um código cliente podemos usar somente operadores ponteiros para membro nos membros que sejam acessáveis naquele escopo. Neste exemplo, tanto value como function são public. Em uma função membro da classe, todos os membros da classe são acessáveis.
Erro comum de programação 21.9
Colocar espaço(s) entre os caracteres de . * ou >* é um erro de sintaxe.
Erro comum de programação 21.10
Inverter a ordem dos símbolos em . * ou -> é um erro de sintaxe.
21.12 Herança múltipla e classes base virtual
No Capítulo 9, discutimos a herança múltipla, o processo pelo qual uma classe herda de duas ou mais classes. A herança múltipla é usada, por exemplo, na biblioteca padrão de C++ para formar a classe iostream (Fig. 2 1.14).
A classe ios é a classe base tanto de ostream como de istream. cada uma das quais é formada por herança simples. A classe iostream herda tanto de ostream como istream. Isto possibilita a objetos de iostream oferecer tanto a funcionalidade de istream como a de ostream. Em hierarquias de herança múltipla, a situação descrita na Fig. 2 1.14 é chamada de herança losango (diamond).
jos
istream ostream
iostream
Fig. 21.14 Herança múltipla para formar a classe iostream.

Como as classes ostream e istream herdam de ios, existe um problema potencial para iostreain. A classe
- iostream poderia conter objetos da superclasse duplicados (por exemplo, ios é

herdado tanto por ostream v de como por istreain). Poderia acontecer um problema quando um ponteiro de iostream é convertido para um ponteiro de ios. Poderiam existir dois objetos ios. Qual, então, seria usado? Tal situação seria ambígua e resultaria em um erro de sintaxe. A Fig. 2 1.15 demonstra este tipo de ambigüidade, mas através de conversão implícita e - não de conversão do ponteiro para a classe de cima; naturalmente, na realidade iostreaxn não sofre do problema uido que mencionamos. Nesta seção, explicaremos como o uso de classes base virtual resolve o problema de sub-objetos duplicados.

nção Dica de desempenho 21.1

com -k .

Subobjetos duplicados consomem memo ria.

em a

```
1 // Fig. 21.15: fig2ll5.cpp
2 II Tentando chamar polimorficamente uma função
3 II herdada de duas classes base por herança múltipla.
a35 4 #include <iostream>
5
6 using std: :cout;
7 using std::endl;
joda 8
int 9 class Base
10 public:
11 virtual void print() const = 0; II virtual pura
12
13
US 14 class DerivedOne : public Base
15 public:
16 1/ sobrescreve a função de impressão
aveis. 17 void print() const { cout « 'DerivedOne\n'; }
18
19
20 class DerivedTwo : public Base
21 public:
22 // sobrescreve a função de impressão
23 void print() const { cout « "DerivedTwo\n";
24 };
25
26 class Multiple : public DerivedOne, public DerivedTwo
27 public:
28 II qualifica qual a versão da função de impressão
,es.A 29 void print() const { DerivedTwo::print
1.14).
a por 32 int main()
osde 33 {
iiulti- 34 Multiple both; II instancia objeto Multiple
```

```
35 DerivedOne one; 1/ instancia objeto DerivedOne
36 DerivedTwo two; II instancia objeto DerivedTwo
37
38 Base *array[ 3 ];
39 array[ O ) = &both; // ERRO - ambíguo
40 array[ 1 ) = &one;
41 array[ 2 ) = &two;
42
```

Fig. 21.15 Tentando chamar polimorficamente uma função herdada de multiplicação (parte 1 de 2).


```
43 // invoca função de impressão polimorficamente
44 for ( int k = 0; k < 3; k++
45 array[ k 3 -> printO;
46
47 return 0;
48
```

Mensagem de erro do compilador Borland C++ com linha de comando

```
Borland C++ 5.5 for Win32 Copyright (c) 1993, 2000 Borland
Fig2l_15 . cpp:
Error E2034 Fig2l_15.cpp 39: Cannot convert ‘Multiple *‘ to Base *‘ in function
main()
1 errors in Compile ***
```

Mensagem de erro do compilador Microsofi Visual C+ +

```
Compiling...
Fig2l_15 . cpp
Fig2l_15.cpp(39) : error C2594: ‘=‘ : anbiguous conversions from
‘class Multiple *‘ to ‘class Base *‘
```

Fig. 21.15 Tentando chamar polimorticamente uma função herdada de multiplicação (parte 2 de 2).

O programa define a classe Base que contém a função virtual print. As classes DerivedOne e Derive- dTwo herdam publicamente de Base e sobrescrevem print. A classe DerivedOne e a classe DerivedTwo contém cada uma um subobjeto Base.

A classe Multiple herda de DerivedOne e DerivedTwo. A função print é sobrescrita para chamar a print de DerivedTwo. Note a qualificação para especificar a versão correspondente a qual sub objeto chamar.

Em main. é criado um objeto de cada classe da hierarquia. Um array de ponteiros Base * também é declarado. Cada elemento do array é inicializado com o endereço de um objeto. Ocorre um erro quando o endereço de both (que é do tipo herdado multiplamente Multiple) é implicitamente convertido para Base . O objeto both contém subobjetos herdados multiplamente de Base e isto, naturalmente, torna as chamadas a print ambíguas. Um laço for é escrito para chamar polimorficamente print para cada um dos objetos apontados por array.

O problema dos objetos duplicados é resolvido com a herança virtual. Quando

uma classe base é herda- da como virtual, somente um subobjeto aparecerá na classe derivada - um processo chamado de herança de uma classe base virtual. O programa na Fig. 21.16 revisa o programa da Fig. 21.15 para usar uma classe base virtual.

A classe Base é definida e contém a função virtual pura print. A classe DerivedOne herda de Base 41 com a linha

class DerivedOne : virtual public Base

e a classe DerivedTwo herda de Base com a linha

class DerivedTwo : virtual public Base

```
1 II Fig. 21.16: fig2l_16.cpp
2 // Usando classes base virtuais.
3 #include <iostream>
4
5 using std::cout;
6 using std: :endl;
```

Fig. 21.16 Usando classes base virtual (parte 1 de 2).


```
7
8 class Base
9 public:
10 II construtor default implícito
11
12 virtual void print() const = 0; II virtual pura
13 };
14
15 class DerivedOne : virtual public Base
16 public:
17 II construtor default implícito
18 // chama o construtor default base
19
20 // sobrescreve a função de impressão
21 void print() const { cout « DerivedOne\n”;
22
23
24 class DerivedTwo virtual public Base
25 public:
26 II construtor default implícito
27 // chama o construtor default base
28
29 // sobrescreve a função de impressão
30 void print() const { cout « ‘DerivedTwo\n;
31
32
33 class Multiple public DerivedOne, public DerivedTwo
34 public:
```

```
35 II construtor default implícito chama
36 // os construtores default de DerivedOne e DerivedTwo
37
38 II qualifica qual a versão da função de impressão
39 void print() const ( DerivedTwo: :printO;
40
41
42 int main()
43
44 Multiple both; II instancia objeto Multiple
45 DerivedOne one; II instancia objeto DerivedOne
46 DerivedTwo two; II instancia objeto DerivedTwo
47
48 Base *array[ 3 ];
49 array[ O 1 = &both;
50 array[ 1 1 = &one;
51 array[ 2 1 = &two;
52
53 II invoca a função de impressão polimorficamente
54 for ( int k 0; k < 3; k++
55 array[ k 1 -> printi);
56
57 return 0;
58 }
DerivedTwo
DerivedOne
DerivedTwo
```
Fig. 21.16 Usando classes base virtual (parte 2 de 2).

Ambas as classes herdam de Base - cada uma contém um subobjeto de Base. A classe Multiple herda tanto DerivedOne como de DerivedTwo. Somente um subobjeto de Base é herdado pela classe Multiple. compilador agora permite que ocorra a conversão (Multiple * para Base *). Em main. um objeto é criado p cada classe na hierarquia. Também é declarado um array de ponteiros para Base. Cada elemento do arra inicializado com o endereço de um objeto. Note que a conversão (upcast) do endereço de both para Base * ag é permitida. Um laço for percorre array e chama print polimorficamente para cada objeto.

Projetar hierarquias com classes base virtual é simples se são usados os construtores default para classes base. Os dois exemplos anteriores usam construtores default gerados pelo compilador. Se uma classe b virtual fornece um construtor, o projeto se torna mais complicado porque a classe mais derivada deve iniciali a classe base virtual.

Em nossos dois exemplos, Base. DerivedOne, DerivedTwo e Multiple são, cada uma, as clas mais derivadas. Se estivermos criando um objeto de Base, Base é a classe mais derivada. Se estivermos crian um objeto de DerivedOne (ou de

DerivedTwo). DerivedOne (ou DerivedTwo) é a classe mais deriva Se estivermos criando um objeto de Multiple. Multiple é a classe mais derivada. Não importa quão abaixo hierarquia uma classe esteja, ela é, portanto, a classe mais derivada e a responsável pela inicialização da classe b virtual. No Exercício 21.17, solicitamos que leitor treine o conceito de classe mais derivada.

Observação de engenharia de software 21.7

_____ Fornecer um construtor default para classes bases virtual simplifica o projeto da hierarquia.

21.13 Observações finais

Sinceramente, esperamos que você tenha apreciado apreender C++ e programação orientada a objetos neste cur O futuro parece claro. Desejamos-lhe sucesso para alcançá-lo.

Apreciaríamos muito seus comentários, críticas, correções e sugestões para melhorar o texto. Por favor, dir
toda sua correspondência para nosso e-mau:
deitel@deitel . com

Boa sorte!

Resumo

• C++ padrão fornece o tipo de dados bool (com valores false ou true) como uma alternativa preferida ao velho estilo usar 0 para indicar falso e um valor diferente de zero para indicar verdadeiro.

• O “manipulador de stn’am boolalpha prepara o arcam de saída para exibir valores bool como os strings “true false.

• C++ padrão introduz quatro novos operadores de coerção para serem usados preferencialmente em vez do velho estilo coerção de C e C++.

• C++ fornece o operador static cast para conversão entre tipos. A verificação de tipo é efetuada durante a compilaç

• O operador const_cast retira o atributo const dos objetos.

• O operador reiriterpret cast é fornecido para coerções não-padrão entre tipos não-relacionados.

• Cada namespace define um escopo onde são colocados identificadores e variáveis. Para usar um membro de um name pace. o nome do membro deve ser qualificado com o nome do namespace e o operador binário de resolução de esco
(: :) ou um comando using deve ocorrer antes de o nome ser usado.

• Um nainespace pode conter constantes, dados, classes, namespaces aninhados, funções, etc. As definições de name paces devem ocupar o escopo global ou estar aninhadas dentro de outros nainespaces.

• Membros de um naxnespace não-nomeado ocupam o nainespace global.

• A informação sobre tipos durante a execução (RTTI) fornece um meio de determinar o tipo de um objeto durante a execução.

• O operador typeid é usado durante a compilação para retomar uma referência para um objeto typeinfo. Um objeto typeinfo é um objeto mantido pelo sistema que representa um tipo.

• O operador dynamic cast assegura que as conversões apropriadas ocorram durante a execução. O resultado de um dynamiccast para operações de coerção inválidas é O.
• C++ padrão fornece palavras-chave operadores que podem ser usadas no lugar de vários operadores de C++.
• C++ fornece a palavra-chave explicit para suprimir conversões implícitas feitas através de construtores de conversão. Um construtor que é declarado explicit não pode ser usado em uma conversão implícita.
• Um membro de dados mutable é sempre modificável, mesmo em uma função membro const ou um objeto const.
• C++ fornece os operadores * e -> para acessar membros de classes através de ponteiros para aqueles membros.
• A herança múltipla pode criar sub objetos duplicados que podem ser resolvidos com herança virtual. Quando uma clase base é herdada como virtual, somente um subobjeto aparecerá na classe derivada-um processo chamado de herança de classe base virtual.
Terminologia
* nainespace aninhado
<typeinfo> namespace anônimo
_>* namespace global
and not
and_eq not_eq
bitand operador ponteiro para membro de classe
bitor or
bool oreq
boolalpha palavras-chave operadores
classe base virtual ponteiro para função membro
classe mais derivada ponteiro para membro de dados
compl reinterpretcast
constcast RTTI (run-time lype information)
conversão explícita staticcast
conversão implícita sub objeto
downcast (conversão para tipo inferior na hierarquia) true
dynamic_cast typeinfo
explicit typeinfo . h
false typeid
herança losango USing
informação sobre o tipo variáveis globais
mutable virtual
name xor
namespace xoreq
Erros comuns de programação
21.1 Executar uma coerção ilegal como operador static cast é um erro de sintaxe. Coerções ilegais incluem fazer uma conversão de tipos const para tipos não-const. fazer coerção de ponteiros e referências entre tipos que não são relacionados por herança public e fazer coerção para um tipo para o qual não há

um construtor ou operador de conversão apropriado para executar a conversão.
21.2 Colocar main em um namespace é um erro de sintaxe.
21.3 Tentar usar dynamic cast sobre um ponteiro do tipo void * é um erro de sintaxe.
21.4 Tentar invocar um construtor explicit para uma conversão implícita é um erro de sintaxe.
21.5 Usar a palavra-chave explicit em membros de dados ou funções membro outras que não um construtor de parâmetro único é um erro de sintaxe.
21.6 Tentar usar o operador -> ou o operador * com um ponteiro para membro de classe é erro de sintaxe.
21.7 Declarar um ponteiro de função membro sem colocar o nome do ponteiro entre parênteses é um erro de sintaxe.

21.8 Declarar um ponteiro de função membro sem preceder o nome do ponteiro por um nome de classe seguido pelo operador de resolução de escopo (: :) é um erro de sintaxe.
21.9 Colocar espaço(s) entre os caracteres de . * ou >‘ é um erro de sintaxe.
21.10 Inverter a ordem dos símbolos em * ou ->‘ é um erro de sintaxe.

Boas práticas de programação

21.1 Quando criar variáveis de estado para indicar verdade ou falsidade, dê preferência ao uso de bools em vez de ints.
21.2 Usar true e false em lugar de valores zero e diferentes de zero toma os programas mais claros.
21.3 Use o operador static cast, mais seguro e confiável, em preferência ao operador de coerção no estilo de C.
21.4 Evite começar identificadores com o caractere sublinhado, o que pode levar a erros de ligação.
21.5 Preceda um membro pelo nome do seu namespace eu operador de resolução de escopo (: :) se existe possibilidade de um conflito de escopo.
21.6 Usar typeid em testes do tipo switch é um mau uso de RTTI. Em vez disso, use funções virtual.

Dica de desempenho

21.1 Subobjetos duplicados consumem memória.

Dicas de portabilidade

21.1 Usar reinterpret_cast pode fazer com que os programas se comportem de modo diferente em diferentes plataformas.
21.2 O efeito de tentar modificar um objeto que foi definido como constante, independente do fato de aquela modificação ter sido possibilitada por um const_cast ou por uma coerção no estilo de C, varia de compilador para compilador.

Observações de engenharia de software

21.1 Com os acréscimos dos novos operadores de coerção (p.ex.. static cast) ao C++ padrão, os operadores de coerção no estilo antigo (estilo de C) são considerados obsoletos.
21.2 Cada unidade de compilação separada tem seu próprio e único namespace

não-nomeado, ou seja, o namespace não- nomeado substitui o especificador de ligação static.
21.3 Em programas grandes, idealmente cada entidade deve ser declarada em uma classe, função, bloco ou namespace. Isto ajuda a deixar claro o papel de cada entidade.
21.4 RTIl foi concebido para ser usado com hierarquias de herança polimórficas (com funções virtual).
21.5 Use a palavra-chave explicit em construtores que não deveriam ser usados pelo compilador para efetuar conversões implícitas.
21.6 Membros mutable são úteis em classes que têm detalhes de implementação “secretos” que não contribuem para o valor lógico de um objeto.
21.7 Fornecer um construtor default para classes bases virtual simplifica o projeto da hierarquia.
Dicas de teste e depura ção
21.1 É fácil usar reinterpret cast para executar manipulações perigosas que podem levar a sérios erros durante a execução.
21.2 De modo a poder usar RTTI. alguns compiladores requerem que os recursos de RTTI sejam habilitados. Verifique a documentação do seu compilador sobre o uso de RTTI.
Exercícios de auto-revisão
21.1 Preencha os espaços em branco de cada um dos seguintes:
a) O operador qualifica um membro com seu namespace.

operador b) O operador permite que o atributo const de um objeto seja retirado.
c) O operador permite a conversão entre tipos.
21.2 Diga qual das seguintes frases é verdadeira ou falsa. Se uma afirmação éfalsa, explique por quê.
a) namespaces são garantidos como sendo únicos.
h) namespaces não podem ter namespaces como membros.
c) O tipo de dados bool é um tipo de dados primitivo.
e ints. Respostas aos exercícios de auto-revisão
21.1 a) resolução de escopo binário (: :).
ilidade de b) constcast.
c) estilo de C. dynamic cast, static cast ou reinterpretcast.
21.2 a) Falsa. Um programador pode inadvertidamente escolher o mesmo namespace como um outro.
b) Falsa. namespaces podem ser aninhados.
c) Verdadeira.
Exercícios
21.3 Preencha os espaços em branco em cada um dos seguintes itens.
a) O operador _____ é usado para determinar o tipo de um objeto em tempo de execução.
b) A palavra-chave especifica que um namespace ou um membro de um namespace está sendo
splatafor- usado.

c) O operador _____ é a palavra-chave operador para OU lógico.
icação ter d) O especificador de armazemamento _____ permite que um membro de um objeto const seja modificado. ador.
21.4 Diga qual das seguintes frases são verdadeiras e quais são falsas. Se uma afirmação éfalsa, explique por quê.
a) A validade de uma operação static_cast é verificada durante a compilação.
b) A validade de uma operação dynamic cast é verificada durante a execução.
e) O nome typeid é uma palavra-chave.
le coerção d) A palavra-chave explicit pode ser aplicada a construtores, funções membro e membros de dados.
- 21.5 Qual o valor de cada expressão? (Nota: algumas expressões podem gerar erros; se esse for caso, diga qual é a causa do erro).
iacenao- a) cout « false;
b) cout « (bool b 8);
>ace. Isto e) cout « (a = true); // a é do tipo int
d) cout « (*ptr + true && p); II *ptr é 10 e p é 8.88
- e) // *ptr é O e m é false
onversoes boolk (*ptr*2 1 (true+24));
f) bool s = true + false;
iraovalor g) cout « boolalpha « false « setw(3) « true;
21.6 Escreva um namespace Moeda que defina os membros constantes Um, Dois, Cinco, Dez, Vinte, Cinquenta e Cem. Escreva dois pequenos programas que usam Moeda. Um programa deve tornar todas as constantes disponíveis e o outro programa deve tornar somente Cinco disponível.
21.7 Escreva um programa que usa o operador reinterpret_cast para fazer a coerção de diferentes tipos de ponteiros para int. Alguma das conversões resulta em erro de sintaxe?
durante a
21.8 Escreva um programa que usa o operador static cast para fazer a coerção de alguns tipos de dados primitivos para
‘erifique a int. O compilador permite as coerções para int?
21.9 Escreva um programa que demonstra a “conversão para cima” (upcasting) de uma classe derivada para uma classe base. Use o operador static_cast para executar a conversão.
21.10 Escreva um programa que cria um construtor explicit que aceita dois argumentos. O compilador permite isto? Remova explicit e tente uma conversão implícita. O compilador permite isto?
21.11 Qual é o benefício de um construtor explicit?

2 1.12 Escreva uma programa que cria uma classe contendo dois construtores. Um construtor deve aceitar um único argumento int. O segundo construtor deve aceitar um argumento char . Escreva um programa de teste que constrói diversos objetos diferentes, com cada objeto tendo um tipo diferente passado para o construtor. Não use explicit. O que acontece? Agora use explicit somente para o construtor que aceita um int. O que acontece?

21.13 Dados os seguintes namespaces. responda se cada afirmação é verdadeira ou falsa. Justifique as respostasfalsas.

```
1 #include <string>
2 namespace Misc {
3 using namespace std;
4 enuin Countries { POLONIA, SUICA, ALEMANHA,
5 AUSTRIA, REPUBLICATCHECA };
6 int kilometers;
7 string s;
8
9 namespace Temp {
10 short y = 77;
11 Car car; II assume que a definição existe
12
13
14
15 namespace ABC
16 using namespace Misc: :Temp;
17 void *function( void ‘, int );
18 )
```

a) A variável y é acessável dentro do namespace ABC.
b) O objeto s é acessável dentro do namespace Temp.
c) A constante POLONIA não é acessável dentro do namespace Temp.
d) A constante ALEMANHA é acessável dentro do namespace ABC.
e) A função function é acessável no namespace Temp.
f) O namespace ABC é acessável para Misc.
g) O objeto carro é acessável para Misc.

21.14 Compare e contraste muta1e e const cast. Dê ao menos um exemplo de quando um é preferível em relação ao outro. Nota: este exercício não requer que seja escrito código de programação.

21.15 Escreva um programa que usa const_cast para modificar uma variável const. (Sugestão: use um ponteiro em sua solução para apontar para o identificador const.

2 1.16 Que problema resolvem as classes base virtual?

21.17 Escreva um programa que usa classes bases virtual. A classe no topo da hierarquia deve fornecer um construtor que aceite pelo menos um argumento (ou seja, não fornece um construtor default). Que desafios isto apresenta para a hierarquia de herança?

21.18 Encontre o(s) erro(s) em cada um dos seguintes itens. Quando possível, explique como corrigir cada erro.

a) manespace Name {
int x, y;
mutable int z;
b) int integer = const_cast< int >(double);
c) namespace PCM(l1l, “alô”); // constrói o namespace
d) explicit int x = 99;

Apêndice A
Tabela de precedência dos operadores
Os operadores são apresentados em ordem decrescente de precedência, de cima para baixo.

Fig. A.1 Tabela de precedência de operadores (parte 1 de 2).

o e

e [e

Operador		Tipo	Associatividade 1
.		resolução de escopo binário resolução de escopo unário	esquerda para a direita
O		parênteses	esquerda para a direita
[]		subscrito de array	
.		seleção de membro via objeto	
->		seleção de membro via ponteiro	
++ -		pós-incremento unário pós-decremento unário	
typeid		informação sobre tipo durante a execução	
dynainic_cast< tipo>		coerção com verificação de tipo durante a execução	
staticcast< tipo>		coerção com verificação de tipo durante a compilação	
reinterpretcast<	tipo>	coerção para conversões não-padrão	
constcast< tipo>		retira o atributo const com coerção	
++		pré-incremento unário	direita para a esquerda
- -		pré-decremento unário	
+		mais unário	
-		menos unário	
!		negação lógica unária	

-		complemento sobre bits unário	
tipo)		coerção ao estilo de C unário	
sizeof		determina o tamanho em bytes	
&		endereço	
*		derreferencia	
new		alocação dinâmica de memória	
new [3		alocação dinâmica de array	
delete		desalocação dinâmica de memória	
dele te [3		desalocação dinâmica de array	
*		ponteiro para membro via objeto	esquerda para a direita
		ponteiro para membro via ponteiro	
*		multiplicação	esquerda para a direita
/		divisão	
%		módulo (resto da divisão inteira)	
+		adição	esquerda para a direita
-		subtração	

« »
<
<=
>
>
&
A
&&
II
*- 1=
&=
«=
»=

deslocamento sobre bits para a esquerda
deslocamento sobre bits para a direita
relacional menor do que
relacional menor do que ou igual a
relacional maior do que
relacional maior do que ou igual a
relacional igual a
relacional não igual a
AND (E) sobre bits
OR (OU) exclusivo sobre bits

OR (OU) inclusivo sobre bits
AND (E) lógico
OR (OU) lógico
condicional ternário
atribuição
atribuição com adição
atribuição com subtração
atribuição com multiplicação
atribuição com divisão
atribuição com módulo
atribuição com AND sobre bits
atribuição com OR exclusivo sobre bits
atribuição com OR inclusivo sobre bits
atribuição com deslocamento sobre bits para a esquerda atribuição com deslocamento sobre bits para a direita
vírgula esquerda para a direita

esquerda para a direita

esquerda para a direita

esquerda para a direita

Operador Tipo Associatividade 1

esquerda para a direita esquerda para a direita esquerda para a direita esquerda para a direita esquerda para a direita direita para a esquerda direita para a esquerda

Fig. A.1 Tabela de precedência de operadores (parte 2 de 2).

Apêndice B

Conjunto de caracteres ASCII

Conjunto de Caracteres ASCII

Fig. B.1 O conjunto de caracteres ASCII

Os dígitos à esquerda da tabela são os dígitos da esquerda do equivalente decimal (0 a 127) do código do caractere e os dígitos do topo da tabela são os dígitos da direita do código do caractere. Por exemplo, o código do caractere “F” é 70 e o código do caractere “&“ é 38.

O 1 2 3 4 5 6 7 8 9

O
2
3
4
5
6
7
8
9
1 O
11
12

flui	so h	st x	et x	eo t	en q	ac k	be l	bs	ht
ni	vt	ff	cr	so	si	di e	dc l	dc 2	dc 3
dc 4	na k	sy n	et b	ca n	e m	su b	es c	fs	gs
rs	us	sp		‘	#	$	%	&	‘
(	)	*	+	,	-	.	/	O	1
2	3	4	5	6	7	8	9	:	;
<	=	>	?	@	A	B	C	D	E
F	G	H	1	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y
Z	[	\	3	‘		‘	a	b	e
d	e	f	g	h	i	j	k	1	m
n	o	p	q	r	s	t	u	v	w
x	y	z	{		}		de i		

Apêndice C
Sistemas de numeração
Objetivos
• Compreender os conceitos básicos dos sistemas de numeração, tais como base, valor posiciona! e valor do símbolo.
• Compreender como trabalhar com números representados nos sistemas de numeração binário, octa! e hexadecimal.
• Ser capaz de abreviar números binários como números octais ou hexadecimais.
• Ser capaz de converter números octais e hexadecimais em números binários.
• Ser capaz de fazer conversões de números decimais para seus equivalentes binários, octais e hexadecimais e vice-versa.

• Compreender a aritmética binária e como números binários negativos são representados usando a notação de complemento de dois.
Aqui estão os únicos números ratificados.
Wil!iam Shakespeare
A natureza tem uma espécie de sistema coordenado aritmético-geométrico, pois ela possui todos os tipos de modelos. O que percebemos da natureza está em modelos, e todos os modelos da natureza são belos.
Ocorre-me que o sistema daquela natureza deve ser uma beleza real, pois em química descobrimos que as associações são sempre em belos números inteiros - não há frações.
Richard Buckminster Fu!!er

4

APÊNDICE C - SISTEMAS DE NUMERAÇÃO 1039
Visão Geral
C.1 Introdução
C.2 Abreviando números binários como números octais e hexadecimais
C.3 Convertendo números octais e hexadecimais em números binários
C.4 Convertendo do sistema binário, octal ou hexadecimal para o sistema decimal
C.5 Convertendo do sistema decimal para o sistema binário, octal ou hexadecimal
C.6 Números binários negativos: notação em complemento de dois
Reswno . Terminologia Exetrícios de auto-revisão Respostas aos exercícios de auto-revisão • Ese,vícios
C.1 Introdução
Neste apêndice, apresentamos os principais sistemas de numeração que os programadores utilizam, especialmente quando estão trabalhando em projetos de software que exigem muita interação com hardware em “nível de máquina”. Projetos como esses incluem sistemas operacionais, software de redes de computadores, compiladores, sistemas de bancos de dados e aplicações que exigem alto desempenho.
Quando escrevemos um inteiro como 227 ou -63 em um programa, assume-se que o número está no sistema de numeração decimal (base 10). Os dígitos no sistema de numeração decimal são 0, 1, 2, 3,4, 5, 6, 7, 8 e 9. O menor dígito é O e o maior dígito é 9 um a menos que a base 10. Internamente, os computadores usam o sistema de numeração binário (base 2). O sistema de numeração binário tem apenas dois dígitos, ou seja O e 1. Seu menor digito é O e seu maior digito é 1 - um a menos que a base 2. A Fig. C.l resume os dígitos usados nos sistemas de numeração binário, octal, decimal e hexadecimal.
Como veremos, os números binários tendem a ser muito maiores do que seus equivalentes decimais. Os programadores que trabalham com linguagens de montagem (assemblers) e com linguagens de alto nível, que permitem que eles desçam ao “nível da máquina”, acham complicado trabalhar com números binários. Assim, dois outros sistemas de numeração - o sistema de numeração

octal (base 8) e o sistema de numeração hexadecimal (base 16) - são populares, principalmente porque tornam conveniente a abreviação de números binários.
No sistema de numeração octal, os dígitos variam de O a 7. Como tanto o sistema de numeração binário quanto
o sistema octal possuem menos dígitos que o sistema decimal, seus dígitos são os mesmos que seus correspondentes do sistema decimal.
O sistema hexadecimal apresenta um problema porque ele exige dezesseis dígitos - o menor dígito é O e o maior dígito tem valor equivalente a 15 (um a menos que a base 16). Por convenção, usamos as letras de A a F para representar os dígitos correspondentes aos valores decimais de 10 a 15. Desta forma, em hexadecimal podemos ter números como 876 consistindo apenas de dígitos semelhantes aos decimais, números como 8A55F consistindo em dígitos e letras, e números como FFE consistindo apenas em letras. Ocasionalmente, um número hexadecimal é grafado como uma palavra comum como FACE ou CADA - isso pode parecer estranho aos programadores acostumados a trabalhar com números. A Fig. C.2 resume cada um dos sistemas de numeração.
Cada um desses sistemas de numeração usa a notação posicional - cada posição na qual é escrito um dígito possui um valor posicional diferente. Por exemplo, no número decimal 937 (o 9, o 3 e o 7 são chamados de valores dos símbolos ou valores dos algarismos), dizemos que o 7 é escrito na posição das unidades, o 3 é escrito na posição das dezenas e o é escrito na posição das centenas. Observe que cada uma dessas posições é uma potência da base (base 10) e que essas potências começam°

DIgito binário Dígito octal Digito decimal Dígito Hexadecimal

Fig. C.1 Dígitos dos sistemas de numeração binário, octal, decimal e hexadecimal (parte 1 de 2).

Fig. C.1 Dígitos dos sistemas de numeração binário, octal, decimal e hexadecimal (parte 2 de 2).

Dígito decimal 9 3 7
Nome da posição Centenas Dezenas Unidades
Valor posicional 100 10 1
Valor posicional como
uma potência da base 102 10’ 10°
Fig. C.3 Valores posicionais no sistema de numeração decimal.
Para números decimais maiores, as próximas posições à esquerda seriam a posição dos milhares ti u eievauo a terceira potência), aposição das dezenas de milhar (10 elevado à quarta potência), a posição das centenas de milhar (10 elevado à quinta potência), a posição dos milhões (10 elevado à sexta potência),

a posição das dezenas de milhões (10 elevado à sétima potência) e assim por diante.
No número binário 101, dizemos que o 1 da extremidade da direita está escrito na posição da unidade, o O está escrito na posição do dois e o 1 da extremidade esquerda está escrito na posição do quatro. Veja que cada uma dessas posições é uma potência da base (base 2) e que essas potências começam em O e aumentam de 1 em 1 à medida que nos movemos para a esquerda no número (Fig. C.4).
Para números binários mais longos, as próximas posições à esquerda seriam a posição do oito (2 elevado à
terceira potência), a posição do dezesseis (2 elevado à quarta potência), a posição do trinta e dois (2 elevado à quinta potência), aposição do sessenta e quatro (2 elevado à sexta potência) e assim por diante.
No número octal 425, dizemos que o 5 está escrito na posição das unidades, 02 está escrito na posição do oito
e o 4 está escrito na posição do sessenta e quatro. Veja que cada uma dessas posições é uma potência da base (base
8) e que essas potências começam em O e aumentam de 1 em 1 à medida que nos movemos para a esquerda no
número (Fig. C.5).

Dígito binário Dígito octal	Digito decimal	D í	gito Hexadecimal	
4	4	4		
5	5	5		
6	6	6		
7	7 8 9	7 8 9 A B C D E F	(valor decimal (valor decimal (valor decimal (valor decimal (valor decimal (valor decimal	1 0) II) 1 2) 1 3) 1 4) 1 5)

Atributo Binário Octal De	cimal Hexadecimal	
Base 2 8 10 Menor dígito O O O Maior dígito 1 7 9	16 O F	
Fig. C.2 Comparação entre os sistemas de numeração binário,	octal, decimal e hexadecimal.	
Valores posicionais no sistema de numeração		

decimal		

Para números octais mais longos, as próximas posições à esquerda seriam aposição do quinhentos e doze (8
elevado à terceira potência), a posição do quatro mil e noventa e seis (8 elevado à quarta potência), a posição do trinta e dois mil, setecentos e sessenta e oito (8 elevado à quinta potência) e assim por diante.
No número hexadecimal 3DA, dizemos que o A está escrito na posição das unidades, o D está escrito na posição do dezesseis e o 3 está na posição do duzentos e cinqüenta e seis. Veja que cada uma dessas posições é uma potência da base (base 16) e que essas potências começam em O e aumentam de 1 em 1 à medida que nos movemos para a esquerda no número (Fig. C.6).

Fig. C.6 Valores posicionais no sistema de numeração hexadecimal.
Para números hexadecimais mais longos, as próximas posições seriam aposição do quatro mile noventa e seis (16
elevado à terceira potência), aposição do trinta e dois mil, setecentos e sessenta e oito (16 elevado à quarta potência)
e assim por diante.
C.2 Abreviando números binários como números octais e hexadecimais
O principal uso dos números octais e hexadecimais em computação é abreviar representações binárias longas. A Fig. C.7 destaca o fato de que números binários longos podem ser expressos de uma forma concisa em sistemas de numeração com bases maiores do que a do sistema de numeração binário.

Valores posicionais no sistema de numeração binário		1
Digito binário 1 O Nome da posição Quatro Dois Valor posicional 4 2 Valor posicional como uma potência da base (2) 22 2’	1 Unidade 1 2°	
Fig. C.4 Valores posicionais no sistema de numeração binário.		
Valores posicionais no sistema de numeração octal		1
Dígito octal 4 2 Nome da posição Sessenta e quatro Oito Valor posicional 64 8 Valor posicional como uma potência da base (8) 82 81	5 Unidade 1 8°	
Fig. C.5 Valores posicionais no sistema de numeração octal.		
1 Valores posicionais no sistema de numeração hexadecimal		
Dígito hexadecimal 3 D Nome da posição Duzentos e Dezesseis cinqüenta e seis Valor posicional 256 16 Valor posicional como uma potência da base (16) 162	A Unidade	

161	1 16°	

Fig. C.7 Equivalentes decimais, binários, octais e hexadecimais.

Um relacionamento particularmente importante com o sistema de numeração binário, que tanto o sistema de numeração octal quanto o hexadecimal possuem, é que suas bases (8 e 16, respectivamente) são potências da base do sistema de numeração binário (base 2). Examine o número binário com 12 dígitos a seguir e seus equivalentes em octal e hexadecimal. Veja se você pode determinar como esse relacionamento torna conveniente exprimir números binários como números octais e hexadecimais. A resposta vem após os números.

Número binário Equivalente octal Equivalente hexadecimal

100011010001 4321 8D1

Para ver como o número binário é convertido facilmente em um número octal, simplesmente divida o número binário de 12 dígitos em grupos de três bits consecutivos cada um e escreva aqueles grupos sobre os dígitos correspondentes do número octal, como se segue

100 011 010 001

4 3 2 1

Observe que o dígito octal escrito sob cada grupo de três bits corresponde exatamente ao equivalente octal daquele número binário de 3 dígitos, de acordo com o que mostra a Fig. C.7.

O mesmo tipo de relacionamento pode ser observado na conversão de números do sistema binário para o hexadecimal. Em particular, divida o número binário de 12 dígitos em grupos de quatro bits consecutivos cada um e escreva aqueles grupos sobre os dígitos correspondentes do número hexadecimal, como se segue

1000 **1101 0001**

8 0 1

Observe que o dígito hexadecimal que você escreveu sob cada grupo de quatro bits corresponde exatamente ao equivalente hexadecimal daquele número binário de quatro dígitos, como mostra a Fig. C.7.

C.3 Convertendo números octais e hexadecimais em números binários

Na seção anterior, vimos como converter números binários em seus equivalentes octais e hexadecimais, formando grupos de dígitos binários e simplesmente rescrevendo esses grupos como seus valores octais e hexadecimais equivalentes. Esse processo pode ser usado na ordem inversa para produzir o número binário equivalente a um número octal ou hexadecimal dado.

Número decimal	Representação binária	Representação octal	Representação hexadecimal
O	O	O	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

Por exemplo, o número octal *653* é convertido em binário simplesmente escrevendo-se o 6 como seu número binário equivalente de 3 dígitos 110, o 5 como seu binário de 3 dígitos equivalente 101 e o 3 como seu binário de 3

dígitos equivalente 011 para formar o número binário de 9 dígitos 110101011.
O número hexadecimal FAD5 é convertido em binário simplesmente escrevendo-se o F como seu número binário equivalente de 4 dígitos 1111,0 A como seu binário de 4 dígitos equivalente 1010, o D como seu binário de 4 dígitos equivalente 1101 e o 5 como seu binário de 4 dígitos equivalente 0101 para formar o número binário de 16
dígitos 111110101 1010101.

C.4 Convertendo do sistema binário, octal ou hexadecimal para o sistema decimal

Por estarmos acostumados a trabalhar em decimal, freqüentemente é útil converter um número binário, octal ou hexadecimal em decimal, para ter uma noção do que o número "realmente" vale. Nossos diagramas na Seção C. 1 expressam os valores posicionais em decimais. Para converter um número em decimal, a partir de outra base, multiplique o equivalente decimal de cada dígito por seu valor posicional e some esses produtos. Por exemplo, o número binário 110101 é convertido no decimal 53 de acordo com o que mostra a Fig. C.8.

Fig. C.8 Convertendo um número binário em decimal.

Para converter o octal 7614 no decimal 3980, aplicamos a mesma técnica, usando dessa vez os valores posicionais octais apropriados, como mostra a Fig. C.9.
Para converter o hexadecimal AD3B no decimal 44347, aplicamos a mesma técnica, usando dessa vez os valores posicionais hexadecimais apropriados, como mostra a Fig. CIO.

Fig. **C.10** Convertendo um número hexadecimal em decimal.

Convertendo um núm	**ero binário em decimal**							1
Valores posicionais:	**32**	**16**		**8**		**4**	**2**	**1**
Valores dos algarismos:	**1**	**1**		O		1	O	1
Produtos:	**1*32=32**	**1*1616**		0*8=0		**1*4 4**	**0*2= 0**	**1*1 1**
Soma:	**=32+16+0+**		**4+**	**0+153**				

Convertendo um número octal em decimal				1
'valores posicionais: 512 64		8	**1**	
Valores dos algarismos: 7 6		1	4	
Produtos: **7*5123584 6*64=384**		**1*8=8**	**4*14**	
Soma: = **3584 + 384 + 8 + 4** =	**3980**			
Fig. C.9 Convertendo um número octal em decimal.				
Convertendo um número hexadecimal em decimal				
Valores posicionais: 4096 256		16	1	
Valores dos algarismos: **A D** Produtos: **A*4096=40960 D*256=3328**		3 **3*16=48**	**B B*1=11**	
Soma: = **40960** + **3328** + 48 **+ 11**	= **44347**			

C.5 Convertendo do sistema decimal para o sistema binário, octal ou hexadecimal

As conversões das seções anteriores são conseqüências naturais das convenções da notação posicional. Converter do ___sistema decimal para o sistema binário, octal ou hexadecimal também segue essas convenções.

Suponhaque desejamos converter o número decimal 57 para o sistema binário. Começamos escrevendo os valores posicionais das colunas, da direita para a esquerda, até alcançarmos a coluna cujo valor posicional seja maior do que o número decimal. Não precisamos daquela coluna, portanto a descartamos. Assim, escrevemos inicialmente:

Valores posicionais: 64 32 16 8 4 2 1

A seguir, descartamos a coluna com valor 64, restando:

Valores posicionais: 32 16 8 4 2 1

A seguir, trabalhamos a partir da coluna da extremidade esquerda em direção à direita. Dividimos *57* por 32 e observamos que há uma vez 32 em 57, com resto 25, portanto escrevemos 1 na coluna 32. Dividimos 25 por 16 e observamos que há uma vez 16 em 25, com resto 9 e escrevemos 1 na coluna 16. Dividimos 9 por 8 e observamos que há uma vez 8 em 9, com resto 1. As duas próximas colunas produzem quocientes com resto zero quando divididas por seus valores posicionais, portanto escrevemos Os nas colunas 4 e 2. Finalmente, dividindo 1 por 1 obtemos 1, portanto escrevemos 1 na coluna 1. Isso leva a:

Valores posicionais: 32 16 8 4 2 1

Valores dos algarismos: 1 1 1 O O 1

e assim o valor decimal 57 é equivalente ao binário 111001.

Para converter o número decimal 103 para o sistema octal, começamos escrevendo os valores das colunas até alcançarmos a coluna cujo valor posicional seja maior do que o número decimal. Não precisamos dessa coluna, portanto a descartamos. Assim, escrevemos inicialmente:

Valores posicionais: **512 64 8 1**

A seguir, descartamos a coluna com valor 512, restando:

Valores posicionais: 64 8 1

A seguir, trabalhamos a partir da coluna da extremidade esquerda em direção à direita. Dividimos 103 por 64

e observamos que há uma vez 64 em 103, com resto 39, portanto escrevemos 1 na coluna 64. Dividimos 39 por

8 e observamos que há quatro vezes 8 em 39, com resto 7 e escrevemos 4 na coluna 8. Finalmente, dividimos

7 por 1 e observamos que há sete vezes 1 em 7, não ficando resto algum, portanto escrevemos 7 na coluna 1.

Isso leva a:

Valores posicionais: 64 8 1

Valores dos algarismos: 1 4 7

e assim o valor decimal 103 é equivalente ao octal 147.

Para converter o número decimal 375 para o sistema hexadecimal, começamos escrevendo os valores das colunas até alcançarmos a coluna cujo valor posicional seja maior do que o número decimal. Não precisamos dessa coluna, portanto a descartamos. Assim, escrevemos inicialmente:

Valores posicionais: 4096 256 16 1

A seguir, descartamos a coluna com valor 4096, restando:

Valores posicionais: 256 16 1

A seguir, trabalhamos a partir da coluna da extremidade esquerda em direção à direita. Dividimos 375 por *256* e observamos que há uma vez 256 em 375, com resto 119, portanto escrevemos 1 na coluna *256.* Dividimos 119 por 16 e observamos que há sete vezes 16cm 119, com resto 7 e escrevemos 7 na coluna 16. Finalmente, dividimos 7 por 1 e observamos que há sete vezes 1 em 7, não ficando resto algum, portanto escrevemos *7* na coluna 1. Isso leva a:

Valores posicionais: 256 16 1

Valores dos algarismos: 1 7 7

e assim o valor decimal 375 é equivalente ao hexadecimal 177.

C.6 Números binários negativos: notação em complemento de dois

A análise deste apêndice concentrou-se nos números positivos. Nesta seção, explicamos como os computadores representam números negativos usando a *notação em complemento de dois.* Em primeiro lugar, explicamos como é formado o complemento de dois de um número binário e depois mostramos por que ele representa o valor negativo de um determinado número binário.

Considere um equipamento com inteiros de 32 bits. Suponha

```
int valor = 13;
```

A representação em 32 bits **do valor é**

00000000 00000000 00000000 00001101

Para formar o negativo de valor, formamos inicialmente o complemento de um, aplicando o operador de complemento sobre bits de C++ **(-),** que também é chamado de *operador NOT sobre bits:*

complementoDeUmDeValor = ‘valor;

Internamente, valor é agora valor com cada um de seus bits invertidos - os uns se tornam zeros e os zeros se tornam uns, como segue:

valor:

00000000 00000000 00000000 00001101

‘valor (i.e., complemento de um de valor):

11111111 11111111 11111111 11110010

Para formar o complemento de dois de valor, simplesmente adicionamos um ao complemento de um de valor. Assim, O complemento de dois de valor é:

11111111 11111111 11111111 11110011

Agora, se isso é realmente igual a -13, devemos ser capazes de adicionar o binário 13 a ele e obter o resultado 0. Vamos tentar fazer isso:

00000000 00000000 00000000 00001101

+ 11111111 11111111 11111111 11110011

00000000 00000000 00000000 00000000

O bit transportado (vai um) da coluna da extremidade esquerda é descartado e realmente obtemos zero como resultado. Se adicionássemos o complemento de um de um número ao próprio número, o resultado seria todo Is. O

segredo de obter um resultado todo em zeros é que o complemento de dois vale 1 a mais do que o complemento de um. A adição de 1 faz com que cada coluna resulte em zero, transportando o valor 1 para a próxima coluna. O valor é transportado para a esquerda, de uma coluna para outra, até que seja descartado do bit da extremidade esquerda e assim o número resultante fica todo consistindo em zeros.

Na verdade, os computadores realizam uma subtração como

```
x = a - valor;
```

adicionando o complemento de dois de valor a a como se segue:

```
x = a + (-‘valor + 1);
```

Suponha que a é 27 e valor é 13, como antes. Se o complemento de dois de valor é realmente o negativo de valor, adicionar a a deve produzir o resultado 14. Vamos tentar fazer isso:

a(i.e., 27) 00000000 00000000 00000000 00011011

+(-valor + 1) + 11111111 11111111 11111111 11110011

00000000 00000000 00000000 00001110

que é realmente igual a 14.

Resumo

Quando escrevemos um inteiro como 19 ou 227 ou -63 em um programa, o número é considerado automaticamente como estando no sistema de numeração decimal (base 10). Os dígitos no sistema de numeração decimal são 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9. O menor dígito é O e o maior dígito é 9-um a menos que a base 10.

• Internamente, os computadores usam o sistema de numeração binário (base 2). O sistema de numeração binário tem apenas dois dígitos, que são O e 1. Seu menor dígito é O e seu maior dígito é 1 - um a menos que a base 2.

• O sistema de numeração octal (base 8) e o sistema de numeração hexadecimal (base 16) se tornaram populares principalmente porque são convenientes para exprimir números binários de uma forma abreviada.

• Os dígitos do sistema de numeração octal variam de 0 a 7.

• O sistema de numeração hexadecimal apresenta um problema porque exige dezesseis dígitos - o menor dígito com valor 0 e o maior dígito com um valor equivalente a 15 em decimal (um a menos que a base 16). Por convenção, usamos as letras A a F para representar os dígitos hexadecimais correspondentes aos valores decimais 10 a 15.

• Cada sistema de numeração usa notação posicional - cada posição na qual um dígito é escrito tem um valor posicional diferente.

• Um relacionamento importante que tanto o sistema de numeração octal quanto o sistema de numeração hexadecimal possuem com o sistema binário é que as bases dos sistemas octal e hexadecimal (8 e 16, respectivamente) são potências da base do sistema de numeração binário (base 2).

• Para converter um número octal em um número binário, simplesmente substitua cada dígito octal pelo binário equivalente de três dígitos.

• Para converter um número hexadecimal em um número binário, simplesmente substitua cada dígito hexadecimal pelo binário equivalente de quatro dígitos.

• Por estarmos acostumados a trabalhar com números decimais, freqüentemente é útil converter um número binário, octal ou hexadecimal em decimal para ter uma melhor noção do que o número "realmente" vale.
• Para converter um número de outra base para um número decimal, multiplique o equivalente decimal de cada dígito por seu valor posicional e some esses produtos.
Os computadores representam números negativos usando a notação em complemento de dois.

Para formar o negativo de um valor, forme inicialmente seu complemento de um aplicando o operador de complemento sobre bits de C++ **().** Isso inverte os bits do valor. Para formar o complemento de dois de um valor, simplesmente adicione um ao complemento de um do valor.

Terminologia

base sistema de numeração hexadecimal
conversões sistema de numeração na base 10
dígito sistema de numeração na base 16
notação em complemento de dois sistema de numeração na base 2
notação em complemento de um sistema de numeração na base 8
notação posicional sistema de numeração octal
operador de complemento sobre bits valor do símbolo
sistema de numeração binário valor negativo
sistema de numeração decimal valor posicional

Exercícios de auto-revisão

C.1 As bases dos sistemas de numeração decimal, binário, octal e hexadecimal são _____ ,_____ ,_____ e
C.2 Em geral. as representações decimal, octal e hexadecimal de um determinado número binário contêm (mais/menos) dígitos do que o número binário.
C.3 (Verdadeiro/Falso) Um motivo popular para o uso do sistema de numeração decimal é que ele forma uma notação conveniente para exprimir números binários de uma forma abreviada, simplesmente substituindo-se um dígito decimal por um grupo de quatro dígitos binários.
C.4 A representação (octal/hexadecimalldecimal) de um valor binário muito grande é a mais concisa (das alternativas fornecidas). C.5 (Verdadeiro/Falso) O maior dígito em qualquer base vale um a mais que a base.
C.6 (Verdadeiro/Falso) O menor dígito em qualquer base vale um a menos que a base.
C.7 O valor posicional do dígito da extremidade direita de qualquer número nos sistemas binário, octal, decimal e hexadecimal é sempre
C. O valor posicional do dígito à esquerda do dígito da extremidade direita nos sistemas binário, octal, decimal ou hexadecimal é sempre
C.9 Preencha as lacunas na tabela a seguir com os valores posicionais das quatro posições da direita em cada um dos sistemas de numeração indicados.
decimal 1000 100 10 1
hexadecimal 256
binário
octal 512 8
C.1O Converta o binário 110101011000 para os sistemas octal e hexadecimal.
C.I 1 Converta o hexadecimal FACE para o sistema binário.
C.12 Converta o octal 7316 para o sistema binário.
C.13 Converta o hexadecimal 4FEC para o sistema octal. (Sugestão: em primeiro lugar, converta 4FEC para o sistema binário e depois converta aquele número binário para o sistema octal).
C.14 Converta o binário 1101110 para o sistema decimal.
C.15 Converta o octal 317 para o sistema decimal.
C.16 Converta o hexadecimal EFD4 para o sistema decimal.
C.17 Converta o decimal 177 para os sistemas binário, octal e hexadecimal.

C.18 Mostre a representação binária do decimal 417. Depois mostre os complementos de um e de dois de 417.
C.19 Qual o resultado quando o complemento de um de um número é adicionado ao próprio número?

Respostas aos exercícios de auto-revisão

C.1 10. 2, 8, 16.
C.2 Menos.
C.3 Falso.
C.4 1-lexadecimal.
C.5 Falso o maior dígito em qualquer base vale um a menos que a base.
C.6 Falso - o menor dígito em qualquer base é zero.
C.7 1 (a base elevada à potência zero).
C.8 A base do sistema de numeração.
C.9 Preencha as lacunas na tabela a seguir com os valores posicionais das quatro posições da direita em cada um dos sistemas de numeração indicados.
decimal 1000 100 10

hexadecimal 4096 256 16
binário 8 4 2
octal 512 64 8
C.10 Octal 6530; Hexadecimal D58.
C.11 Binário 1111 1010 1100 1110.
C.12 Binário 111 011 001 110.
C.13 Binário0l00 111 111 101 100: 0cta147754.
C.14 Decimal 2+4+8+32+64=110.
C.15 Decimal 7+1*8+3*64=7+8+192=207.
C.16 Decimal **4+13*16+15*256+14*4096=61396.**
C.l7 Decimal 177
para binário:
256 128 64 32 16 8 4 2
128 64 32 16 8 4 2
(1*128)+(0*64) +(1*32) + (1*16)+ (0*8) + (0*4)+(0*2)+(1*1)
10110001
para octal:
512 64 8 1
64 8 1
(2*64)+ (6*8) +(1*1)
261
para hexadecimal:
256 16 1
16 1
(11*16)+(1*1)
(B*16) + (1*1)
B1
C.18 Binário:
5122561286432168421
256 128 64 32 16 8 4 2 1
(1*256) + **(1*128)** + **(0*64)** + **(1*32)** + **(0*16)** + **(0*8)** + **(0*4)** + **(0*2)** + **(1*1)**
110100001

Complemento de um: 001011110
Complemento de dois: 001011111
Verificação: o número binário original + seu complemento de dois
110100001
001011111
000000000
C.19 Zero.

Exercícios

C.20 Algumas pessoas argumentam que muitos de nossos cálculos seriam mais fáceis no sistema de numeração de base 12 porque 12 é divisível por muito mais números do que 10 (para base 10). Qual o menor dígito na base 12? Qual pode ser o maior símbolo para o dígito da base 12? Quais os valores posicionais das quatro posições da direita de qualquer número no sistema de numeração da base 12?
C.21 Como o maior valor de símbolo nos sistemas de numcração analisados se relaciona com o valor posicional do primeiro dígito à esquerda do dígito mais à direita de qualquer número nesses sistemas de numeração?
C.22 Complete a tabela de valores posicionais a seguir para as quatro posições da direita em cada um dos sistemas de numeração indicados:
decimal 1000 100 10 1
basefl 6
base 13 ...169 ..
base3 27 ..
C.23 Converta o binário 100101111010 em octal e em hexadecimal.
C.24 Converta o hexadecimal 3A7D em binário.
C.25 Converta o hexadecimal 765F em octal. (Sugestão: converta inicialmente 765F em binário e depois converta aquele número binário em octal.)
C.26 Converta o binário 1011110 em decimal.
C.27 Converta o octal 426 em decimal.
C.28 Converta o hexadecimal FFFF em decimal.
C.29 Converta o decimal 299 em binário, em octal e em hexadecimal.
C.30 Mostre a representação binária do decimal 779. Depois mostre os complementos de um e de dois de 779.
C.31 Qual o resultado da soma do complemento de dois de um número com o próprio número?

C.32 Mostre o complemento de dois do valor inteiro -1 em um equipamento com inteiros de 32 bits.

Apêndice D

Recursos sobre C++ na Internet e na Web

Este apêndice contém uma lista de recursos valiosos para C++ na Internet e na World Wide Web. Incluem FAQs (respostas a perguntas feitas freqüentemente), tutoriais, como obter o ANSI/ISO C++ padrão, informações sobre compiladores C++ populares e como obter compiladores gratuitamente, demos, livros, tutoriais, ferramentas de software, artigos, entrevistas, conferências, jornais e revistas, cursos on-line, grupos de notícias e recursos para carreira profissional.

Para obter mais informações sobre o American National Standards Institute (ANSI) ou para adquirir documentos padrão, visite o ANSI em http: //www. ansi .org/.

Di Recursos

http://www.progsource.com/index. htinl

The Programmer's Source é uma grande fonte de informações sobre muitas linguagens de programação, inclusive C++. Você encontrará listas de ferramentas, compiladores, software, livros e outros recursos para

C++.

http: //www. **intranet. ca/-sshah/booklist. html#C++**

The Programmer's Book List tem uma seção de livros sobre C++ com mais de 30 títulos.

http: //www. genitor, *comi* **resources** .htm

O site Developer Resources tem links para compiladores C++, ferramentas úteis de C++, código- fonte do *CY C++ Users Journal* e publicações.

http: //www.possibility. com/Cpp/CppCodingStandard.html

O site *The C+ + Coding Standard* tem uma grande quantidade de informações sobre a linguagem de programação C++ bem como uma grande lista de recursos sobre C++ na Web.

http: //help-site. com/cpp .htm).

Help-site.com fornece links para recursos sobre C++ naWeb.

http: //www. glenmccl .com/tutor. htm

Este site é uma boa referência para usuários com conhecimento de *CIC++*. Os tópicos são acompanhados por

explicações detalhadas e exemplos de código.

http: //www .prograinmersheaven. com/zone3/cat353/index .htm

Este site oferece uma vasta coleção de bibliotecas para C++. Estas bibliotecas estão disponíveis para download

gratuito.

http: //www .prograinmersheaven. com/zone3/catl55/index .htm

Este site oferece ferramentas e bibliotecas para *CIC++*.

http: //w. programmersheaven. com/wwwboard/board3/wwwboard. asp

Este quadro de mensagens permite aos usuários divulgar perguntas sobre programação em C++. Estas bibliotecas estão disponíveis para download gratuito.

http: //www hal9k. com/ cug/

Este site oferece recursos, jornais, *shareware,freeware,* etc.

http: //developer earthweb. com/directories/pages/dir. c .developmenttools htm Um site popular na Web para programadores, *Developer.com* oferece uma ampla lista de recursos para programadores usando C e C++.

http: //www.devx.com

DevX é um site abrangente com recursos para programadores. A seção oferece as últimas novidades, ferramentas e técnicas para diversas linguagens de programação. A seção *C++ zone* do site é dedicada a C++.

D.2 Tutoriais

http: //info desy .de/gna/html/cc/index html

Este tutorial *Introduction lo Object-Orientred Programming Using C++* está disponível para download ou você pode

se registrar em um curso baseado na Web. Verifique os livros recomendados sobre programação orientada a objetos e a linguagem de programação C++.
http: //uu-gna.mit. edu: 8001/uu-gna/text/cc/Tutorial/tutorial .html
Este tutorial *Introduction to Object-Oriented Programming Using C++* é dividido em 10 capítulos, cada um
com um conjunto de exercícios e soluções para os exercícios.
http: //www. icce. rug .nl/docs/cplusplus/cplusplus .html
Este tutorial, escrito por um professor universitário, foi projetado para programadores de C que querem apreender **C++.**
http: //www.rdw. tec.mn.us/
O *Red Wing/Winona Technical Coliege* oferece cursos de C++ on-line valendo créditos.
http: //www. zdu. com/zdu/catalog/progranuning .htm
A *ZD Net University* oferece vários cursos *on-line* relacionados com a linguagem de programação C++.
http: //library. advanced. org/3074/
Este tutorial foi projetado para programadores de Pascal que querem aprender C++.
ftp: //rtfm.mit. edu/pub/usenet/news .answers/C-faq/learn-c-cpp-today
Este site tem uma lista de tutoriais sobre C++. Também contém informações sobre vários compiladores C++.
http: //www. icce .rug .nh/docs/cplusplus/cplusplus .htm
Um site para usuários que já conhecem C e querem aprender C++.
http: //www. cprogranuning. com/tutorial/html
Este site inclui um tutorial passo a passo que inclui exemplos de código.
http://www.progranunersheaven.com/zone3/cat34/index. htm
Este site contém uma lista de assuntos de tutoriais. Os níveis dos tutoriais variam desde principiantes até
especialistas.

D.3 FAQs

http: //reality. sgi .com/austern/std-c++/faq .html
Este é um site de FAQs dedicado a perguntas sobre C++ padrão ANSI/ISO, o projeto da linguagem de programação C++ e às últimas mudanças na linguagem.
http: //www. trtnphrst. demon .co .uk/cpplibsl .html
Este é um FAQ sobre as bibliotecas de C++. Você encontrará uma extensa lista de respostas às perguntas feitas
freqtientemente sobre as bibliotecas padrão de C++.

http: //pneuma phys ualberta. ca/-.burris/cpp htm
O *The Internet Link Exchange* é um Outro grande recurso para informações sobre C++. Este site tem links para FAQs relacionados a comp. lang. c++ e as bibliotecas padrão de **C++.**
http: //www.math.uio.no/nett/faq/C-faq/faq.html
A lista comp. lang. e de perguntas feitas freqüentemente (FAQs) e respostas.
http: //lglwww. epfl ch/'wo1f/c std. htm].
A lista de FAQs sobre o padrão ANSI/ISO para a linguagem de programação C.
http: //www. cerfnet. com/-mpcline/C++-FAQs-Lite/
Este site tem muitas FAQs divididas em 35 categorias.
http: //www. faqs org/faqs/by-newsgroup/comp/comp. lang. html
Este site consiste em uma série de links para FAQs e tutoriais reunidos do grupo de notícias Comp Lang C++.
http: //www. cerfnet. com/-mpc1ine/C++-FAQs-Lite/
Este é um site de FAQ com uma ampla gama de tópicos. Cada tópico inclui diversas perguntas com respostas.
http: //www. eskimo. coin/-scs/C-faq/top html
Esta lista de FAQ contém tópicos tais como ponteiros, alocação de memória e *strings.*

DA Visual C-i-+

http://ehesworth.com/pv/languages/c/visualcpptutorial. htin
Este é um bom tutorial para principiantes que estejam aprendendo o Microsoft Visual C++. O tutorial dá ao usuário uma visão geral sucinta de C++.

D.5 comp.lang.c++

http: //weblab. research att. com/phoaks/comp/lang/c++/ resourceso html UAU! Este site é um tremendo recurso de informações relacionadas com comp. lang. c++. O título da página, *People Helping OneAnotherKnow Stuff,* resume tudo que é este site. Você encontrará links para mais de 40 recursos adicionais de informações sobre C++.

http: //www. r2m. com/windev/cpp-compiler html

Este contém links para muitos sites relacionados com C++.

http: //home istar. ca/-.stepanv/

Este site tem muitos links para sites com artigos e informações relacionados com a programação C++. Tópicos listados neste site incluem gráficos orientados a objetos, o padrão ANSI C++, a biblioteca padrão de gabaritos (STL), recursos sobre MFC e tutoriais.

http: //kom. net/-'dbrick/newspage/cornp. lang. c++ html

Visite este site para se conectar a grupos de notícias relacionados à hierarquia eomp lang c++.

http: //www. austinlinks com/CPlusPlus/

O site da Quadralay Corporation tem links para recursos sobre C++, incluindo Visual C++/bibliotecas MFC, informações sobre programação em C++, recursos para carreiras profissional em C÷+ e uma lista de tutoriais e outras ferramentas on-line para ajudá-lo a aprender C++.

http: /1db. csie ncu. edu. tw/-kant c/C/chapter2 21 .html

Este site da Web tem a lista das funções da biblioteca padrão ANSI C.

http: //wwwcnl .**cem. ch/asd/geant/geant4yublic/codingstandards/coding/coding2** .html

Um recurso excelente e extenso sobre informações acerca do padrão C++

http: //ibd. ar. com/ger/comp. lang. .html

O Green Eggs Report **lista mais de 100 URLs dentro de comp. lang** .C++.

http: //www. ts umu .se/-maxell/C++/

Este site fornece exemplos de código para algumas das classes C++.

http: //www. quadralay. com/CPlusPlus/

Este é um grande recurso para informações sobre programação C++ .aprendizado de C++, carreiras em C++ e outras informações relacionadas com C++.

http: //www. research. att. *comi* **bs/homepage** .htznl

Esta é a homepage *de Bjarne Stroutrup,* criador da linguagem de programação C++. Ele fornece uma lista de recursos **C++,** FAQs e outras informações úteis sobre **C++.**

http: //www. cygnus .coxn/misc/wp/draft/index .htxnl

Este site tem a "minuta de trabalho" do padrão ANSI **C++** em formato HTML (dezembro de 1996).

http: //www .austinlinks .com/cPlusPlus/

Este site tem uma lista de recursos **C++** incluindo livros sugeridos, recursos para carreira profissional, informações sobre a linguagem de programação **C++** e links para sites com listas de recursos **C++.**

ftp: **//research. att. com/dist/c++std/WP/CD2/**

Este site tem a minuta atual do padrão ANSI/ISO para **C++.**

http: //ai **kaist. ac .kr/** -ymkim/Program/c++ .html

Este site oferece tutoriais, bibliotecas, compiladores populares, FAQs e grupos de notícias.

http: //www. cyberdiem. com/vin/learn .html

Learn C/C++ Today é o título deste site, que oferece diversos tutoriais avançados sobre *CIC++.*

http: //www. trumphurst. com/cpplibsl .html

A *The C++ Libraries FAQ* é compilada por programadores profissionais para o uso e benefício de outros **programadores de** C++. A biblioteca é atualizada regularmente e é uma boa fonte de informações atualizadas.

http: //www. experts-exchange .**com/comp/lang/cplusplus/**

The Experts Exchange é um recurso gratuito para profissionais *hi gh-tech* que desejem compartilhar informações com seus colegas. Sócios podem afixar perguntas e respostas.

http: //www. **execpc. com/ -ht/vc** .htm

Este site é uma compilação de links sobre programação em C++ que incluem sites de informações genéricas, tutoriais, revistas e bibliotecas.

http: //cplus .about. **com/compute/cplus/**

Este é o site *About.com* para as linguagens de programação *CIC++.* Você encontrará tutoriais, *freeware/ shareware,* dicionários, empregos, revistas e muitos outros itens relacionados.

http: //pent2l infosys .**tuwien .ac. at/cetus/**
oocplusplus .htinl#oocylusylusgeneralnewsgroups

Neste site você encontrará uma explicação geral sobre C++. Este site contém grupos de notícias.
news : comp. lang. c++
Este é um grupo de notícias dedicado a questões sobre a linguagem orientada a objetos C++.

http: //cuiwww. unige .ch/OSG/Vitek/Compilers/YearB6/msg00046 .html "O padrão C em máquinas segmentadas".
http: //www. csci .**csusb** .edu/dick/c++std/
Este site tem links para a minuta do padrão C++ ANSI/ISO e o grupo Usenet comp. std. c++ que fornece novas informações sobre o padrão.

news : comp. lang. c++ .moderated
Este é um grupo de notícias mais voltado a aspectos técnicos dedicado à linguagem C++.
http: / /www .progsource .com/index. htxnl
The Programmer's Source é um grande recurso de informações sobre muitas linguagens de programação inclusive C++. Você encontrará listas de ferramentas, compiladores, software, 1ivros e outros recursos para C++. A lista de compiladores está organizada por plataforma.
http: //www. cygnus .com/misc/gnu-win32/
O ambiente de desenvolvimento GNU está disponível isento de custo no site Cygnus na Web.
http: //www. remcomp. com/lcc-win32/
Você pode fazer download do compilador LCC-Win32 para Windows 95/NT, sem custo, deste site da Web.
http: //www .microsoft. com/visualc/
A *homepage* do *Microsoft Visual C++* fornece informações sobre produtos, *overviews,* materiais suplementares e informações sobre como encomendar o compilador Visual C++.
http://www.com/products/languages/watccpl. html
Notícias e informações sobre o produto para o *Watcom C/C++ versão 11.0* da Powersoft. Não pode ser feito download do compilador a partir deste site da Web. São fornecidas informações para aquisição.
http://netserv.bor1and.com/bor1andcpp/cppcomp/turbocpp.htm1
O site na Web para o compilador *Borland Turbo C++ Visual Edition* para Windows.
http: //www. symantec. com/scpp/fs scpp72 95. html
Symantec C++ 7.5 para Windows 95 e Windows NT.
http: //www .metrowerks .com/products/
Metrowerks CodeWarrior para Macintosh ou Windows.
http: //www. faqs .org/faqs/by-newsgroup/comp/comp. compilers .html
Este é um site que criou uma lista de FAQs gerados dentro do grupo de notícias comp. compilers.
http: //www.ncf. carleton. ca/%7Ebg283/
Este é um compilador C++ para DOS chamado de *Miracle C compiler.* O compilador é grátis para download, mas o código-fonte não fica disponível enquanto você não pagar a taxa de registro.
http: //www .borland. com/bcppbuilder/
Este é um link para o *Borland* C++ *Builder5.5.* Uma versão de linha de comando grátis está disponível para download.
http: //www. compilers .net/
Compilers.net é um site projetado para ajudá-lo a encontrar compiladores.
http: //sunset .backbone .olemiss .edu/%7Ebobcook/eC/
Este compilador C++ é projetado para usuários principiantes em C++ que queiram passar de Pascal para C++.
http: //developer .intel .com/vtune/conipilers/cpp/
O copilador *Intel C++.* Plataformas suportadas são Windows 98, NT e 2000.
http: //www. kai .com/Çplusplus/index .htxnJ.
O *compilador Kai C++* está disponível gratuitamente para um período de demonstração de 30 dias.

0.6 Compiladores

0.7 Ferramentas de desenvolvimento

http: //www. quintessoft. comi
A *Quintessoft Engineering, mc.* oferece o Code Navigator para C++, uma ferramenta de desenvolvimento em C++ para Windows 951NT. Você encontrará informações sobre o produto, comentários dos usuários, downloads gratuitos de edições de demonstração e informações de preços para o produto.

D.8 Biblioteca padrão de gabaritos

Tutoriais

http: //www. cs brown .edu/people/jak/prograinming/stl-tutorial/
tutorial html

Este tutorial sobre a STL está organizado em exemplos, filosofia, componentes e estendendo a STL. Você encontrará exemplos de código usando os componentes da STL, explicações e diagramas úteis.

http: //web. ftech. net/honeyg/artic1es/eff stl htm

Este tutorial sobre a STL oferece informações sobre os componentes da STL, contêineres, adaptadores de *stream* e iteradores, transformação e seleção de valores, filtragem e transformação de valores, e objetos.

http: //www .xraylith wisc edu/-khan/software/stl/osexaniples/
examples htinl

Este é um site útil para pessoas que estejam ainda aprendendo sobre a STL. Você encontrará uma introdução à STL e exemplos do *ObjectSpace STL Tool Kit.*

Referências

http: //www. sgi .com/Technology/STL/other resources .html

Este site tem uma lista de muitos sites da Web relacionados com a STL e uma lista de livros sobre a STL recomendados.

http: //www. cs .rpi edu/proects/STL/st1/st1 .html

Esta é a homepage do *Standard Template Library Online Reference* do Rensselaer Polytechnic Tnstitute. Você encontrará explicações detalhadas sobre a STL, bem como links para outros recursos úteis para informações sobre a STL.

http: //www. sgi com/Technology/STL/

O *Silicon Graphics Standard Template Library Programmer's Guide* é um recurso útil para informações sobre a STL. Você pode fazer download da STL deste site e encontrar as informações mais recentes, documentação de projeto e links para outros recursos para a STL.

http: //www. dinkumware ***comirefcpp*** .html

Este site contém informação útil sobre a biblioteca C++ do padrão ANSI/ISO e contém extensa informação sobre a biblioteca padrão de gabaritos (STL).

http: //www. roguewave com/products/xplatfortn/stdlib/

A página da Web da *Rogue Wave Software's Standard C++ Library.* Você pode fazer download de artigos relacionados com a versão deles da biblioteca padrão de C++.

FAQs

ftp: //butler .hpl .hp. com/stl/stl. faq

Este site de FTP é uma folha de FAQ para a STL mantida por Marian Corcoran, membro do comitê ANSI e
uma especialista em C++.

Artigos, livros e entrevistas

http: //www. sgi com/Technology/STL/other resources html

Este site relaciona muitos sites da Web relacionados com a STL e uma pequena lista de livros sobre a STL recomendados,

http://www.byte.com/art/9510/sec12/art3. htzn

O site da *Bvte Magazine* tem uma cópia de um artigo sobre a STL escrito por Alexander Stepanov. Stepanov, um dos criadores da biblioteca padrão de gabaritos. oferece informações sobre o uso da STL em programação genérica.

http: //www. sgi com/Technology/STL/drdobbs-interview.html

Uma entrevista com Alexander Stepanov que tem algumas informações interessantes sobre a criação da biblioteca padrão de gabaritos. Stepanov fala sobre como a STL foi conceituada, programação genérica, o acrônimo "STL" e mais.

Padrão ANSI/ISO C++

http://www.ansi.org/

Você pode comprar uma cópia do documento padrão de C÷+ neste site.

Software

http: //www. cs rpi .edu/musser/stJ. html

O site RPI STL inclui informação sobre como a STL difere de outras bibliotecas de C++ e como compilar programas que usam a STL, lista dos principais arquivos de #include da STL, exemplos de programas que usam a STL, classes contêíneres da STL e categorias de iteradores da STL. Ele também oferece uma lista de compiladores compatíveis com a STL, sites de FTP para código-fonte STL e materiais relacionados.

http://www.inathcs.sisu.edu/facu1ty/horstman/safest1.htm1

Faça download do SAFESTL.ZIP. uma ferramenta projetada para encontrar erros em programas que usam a STL.

http://www.objectspace.com/jgh

Object Space oferece informação sobre o porte de C++ para Java. Você pode fazer download gratuito da classe portável *Standards<ToolKit>* deles. Os principais destaques do *toolkit* são contêineres, iteradores, algoritmos, alocadores, *strings* e exceções.

http: //www. *cs* rpi edu/-wiseb/stl-borland htnil

"Using the Standard Template Library with Borland C++ Este site *é* uma referência útil para pessoas que usam o compilador Borland C++. O autor tem seções sobre advertências e incompatibilidades.

http: //msdn nicrosoft. com/visualc/

Esta é a homepage do *Microsofi Visual C+ +.* Aqui você pode encontrar as notícias mais recentes sobre o Visual C++, atualizações, recursos técnicos, exemplos e downloads.

http: //www .borland. com/bcppbuilder/

Esta é a homepage do *Borland C++ Builder.* Aqui você pode encontrar uma variedade de recursos para C++, incluindo diversos grupos de notícias sobre C++, informações sobre as mais recentes melhorias do produto, FAQs e muitos outros recursos para programadores que usam o C++ Builder.

Bibliografia

(A192) Allison, C., "Text Processing 1," *The C Users Journal,* Vol. 10, No. 10, October 1992, pp. 23-28.

(A192a) Allison, C., "Text Processing II," *The C Users Journal,* Vol. 10, No. 12, December 1992, pp. 73-77.

(A193) Allison, C., "Code Capsules: A C++ Date Class, Part 1," *The C Users Journal,* Vol. 11, No. 2, February 1993, pp. 123-131.

(A194) Allison, C., "Conversions and Casts," *The C/C++ (Jsers Journal,* Vol. 12, No. 9, September 1994, pp. 67-85.

(Am95) Almarode, J., "Object Security," *Smalltalk Report,* Vol. 5, No. 3 November/December 1995, pp. 15-17. (A n90) ANSI, *American National Standardforlnformation Systems-Programming Language C (ANSI Document* lar *ANSI/ISO 9899: 1990),* New York, NY: American National Standards Institute, 1990.

iue (An94) *American National Standard, Programming Language C++.* (A aprovação e o trabalho técnico de de- de senvolvimento estão sendo conduzidos pelo Accredited Standards Committee X3, Information Technology e seu Technical Committee X3J16, Programming Language C++, respectivamente. Para maiores deta lhes entre em contato com X3 Secretariat, 1250 Eye Street, NW, Washington, DC 20005, EUA.)

(An92) Anderson, A. E., and W. J. Heinze, *C++ Programming and Fundamental Concepts,* Englewood Cliffs, na NJ: Prentice Hali, 1992.

(Ba92) Baker, L., *C Mathematical Function Handbook,* New York, NY: McGraw HilI, 1992.

(Ba93) Bar-David, T., *Object-Oriented Design for C++,* Englewood Cliffs, NJ: Prentice Hail, 1993.

(Be94) Beck, K., "Birds, Bees, and Browsers-Obvious Sources of Objects," *The Smalltalk Report,* Vol. 3, No. 8 June l994,p. 13.

(Be93a) Becker, P., Conversion Confusion, *C++ Report,* October 1993, pp. 26-28.

(Be93) Becker, P., "Shrinking the Big Switch Statement," *Windows Tech Journal,* Vol. 2, No. 5, May 1993, pp. 26-33.

ue (Bd93) Berard, E. V., *Essays on Object Oriented Software Engineering: Volume!,* Englewood Cliffs, NJ: Prentice Hail, 1992.

(Bi95) Binder, R. V., "State-Based Testing," *Object Magazine,* Vol. 5, No. 4, August 1995, pp. 75-78.

(Bi95a) Binder, R. V., "State-Based Testing: Sneak Paths and Conditional Transitions," *Object Magazine,* Vol. 5, No. 6, October 1995, pp. 87-89.

(B192) Blum,A., *NeuralNetworksin C++:An Object-OrientedFrameworkforBuilding ConnectionistSystems,* New York, NY: John Wiley & Sons, 1992.

(Bo91) Booch, G., *Object-Oriented Design with Applications,* Redwood City, CA: The Benjamin/Cummings Publishing Company, mc., 1991.

(Bo94) Booch, G., *Object-Oriented Analysis and Design,* Second Edition, Reading, MA: Addison-Wesley Publishing Company, 1994.

(Bo96) Booch, G., *Object Solutions,* Reading, MA: BenjaminlCummings, 1996.

(Ca92) Cargill, T., *Programming Style,* Reading, MA: Addison-Wesley Publishing Company, 1992.

(Ca95) Carroli, M. D., and M. A. Ellis, *Designing and Coding Reusable C++,* Reading, MA: Addison-Wesley Publishing Company, 1995.

(Co95) Coplien, J. O., and D. C. Schmidt, *Pattern Languages of Program Design,* Reading, MA: AddisonWesley Publishing Company, 1995.

(C++98) ANSI/ISO/IEC: *International Standard: Programming Languages-C++*. 1SO/IEC 14882:1998(E). Publicado pelo the American Nationai Standards Institute, New York, NY: 1998.
(De90) Deitei, H. M., *Operating Systems,* Second Edition, Reading, MA: Addison-Wesley, 1990.
(DeOO) Deitei, H. M., and P. J. Deitei, *Java How to Program,* Third Edition, Upper Saddle River, NJ: Prentice Hali, 2000.
(DeOOa) Deitei, H. M., and P. J. Deitei, *The Java Multimedia Cber Classroom,* Third Edition, Upper Saddie River, NJ: Prentice Hali, 2000.
(DeOi) Deitei, H. M., and P. J. Deitei, *CHow to Program* (Third Edition), Upper Saddle River, NJ: Prentice Haii, 2000.

(Du9 1) Duncan, R., "inside C++: Friend and Virtual Functions, and Muitiple Inheritance," *PC Magazine,* Vol. 10, No. 17, October *15,* 1991, pp. 417-420.
(Ei90) Eliis, M. A., and B. Stroustrup, *TheAnnotated C++ Refrrence Manual,* Reading, MA: Addison-Wesley, i990.
(Em92) Embley, D. W.; B. D. Kurtz; and S. N. Woodfield, *Object-Oriented SystemsAnalysis,* Englewood Ciiffs, NJ: Yourdon Press, 1992.
(En90) Entsminger, G., The Tao of Objects: *A Beginner's Guide to Object-Oriented Programming,* Redwood City, CA: M&T Books, 1990.
(F193) Fiamig, B., *Practi cal Data Structures in C++,* New York, *NY:* John Wiiey & Sons, 1993.
(Ga95) Gamma, E,; R. Heim; R. Johnson; and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software,* Reading, MA: Addison-Wesiey Pubiishing Company, 1995.
(Ge89) Gehani, N., and W. D. Roome, *The Concurrent C Programming Language,* Summit, NJ: Silicon Press, 1989.
(Gi92) Giancoia, A., and L. Baker, "Bit Arrays with C++÷," *The C Users Journal,* Voi. 10, No. 7, Juiy, 1992, pp. 2 1-26.

(G195) Giass, G., and B. Schuchert, *The STL <Primer>,* Upper Saddle River, NJ: Prentice Hali PTR, i995.
(Go95) Gooch, T., "Obscure C++," *Inside Microsofi Visual C++,* Voi. 6, No. ii, November 1995, pp. 13-15.
(Ha90) Hansen, T. L., *The C++ AnswerBook,* Reading, MA: Addison-Wesiey, 1990.
(He97) Henricson, M., and E. Nyquist, *Industrial Strength C++: Rules and Recommendations,* Upper Saddle River, NJ: Prentice Haii, 1997.
(Ja93) Jacobson, 1., "Is Object Technoiogy Software's Industrial Piatform?" *!EEE Software Magazine,* Vol, 10, No. 1, January 1993, pp. 24-30.
(Ja89) Jaeschke, R., *Portability and the C Language,* Indianapolis, IN: Hayden Books, 1989.
(Ke88) Kernighan, B. W., and D. M. Ritchie, *The CProgramming Language* (Second Edition), Englewood Ciiffs, NJ: Prentice Haii, 1988.
(Kn92) Knight, A., "Encapsulation and Information Hiding' *The Smalltalk Report,* Vol. i, No. 8, June 1992, pp. 19-20.
(Ko90) Koenig, A., and B. Stroustrup, "Exception Handling for C++ (revisado)," *Proceedings ofthe USENIX C++ Conference,* San Francisco, CA, Aprii 1990.
(Ko91) Koenig, A., "What is C++ Anyway?" *Journal of Object-Oriented Prograinming,* April/May 1991, pp. 48-52.
(Ko94) Koenig, A., Impiicit Base Ciass Conversions," *C++ Report,* Vol. 6, No. *5,* June 1994, pp. 18-19.
(Ko97) Koenig, A., and B. Moo, *Ruminations on C++,* Reading, MA: Addison-Wesiey, 1997.
(Kr9 1) Kruse, R. L.; B. P. Leung; and C. L. Tondo, *Data Structures and Program Design in C,* Englewood Ciiffs, NJ: Prentice Haii, 1991.
(Le92) Lejter, M.; 5. Meyers; and 5. P. Reiss, "Support for Maintaining Object-Oriented Programs," *ÍEEE Transactions on Software Engineering,* Voi. 18, No. 12, December 1992, pp. 1045-1052.
(Li91) Lippman, 5. B., *C++ Primer* (Second Edition), Reading, MA: Addison-Wesiey Publishing Company, 1991.
(Lo93) Lorenz, M., *Object-Oriented Software Deveiopment: A Practical Guide,* Englewood Ciiffs, NJ: Prentice Hail, 1993.
(Lo94) Lorenz, M., "A BriefLook at Inheritance Metrics," *The Smalltalk Report,* Vol. 3, No. 8, June 1994, pp. 1, 4-5.
(Ma93) Martin, J.. *Principies of Object-Oriented Analysis and Design,* Englewood Ciiffs, NJ: Prentice Hali, 1993.

(Ma95) Martin, R. C., *Designing Object-Oriented C++ Applications Using the Booch Method,* Englewood Cliffs, NJ: Prentice 1-lali, 1995.
(Ma93a) Matsche, J. J., "Object-Oriented Programming in Standard C," *Object Magazine,* Vol. 2, No. *5,* January/ February 1993, pp. 7 1-74.
(Mc94) McCabe, T. J., and Á. H. Watson, "Combining Comprehension and Testing in Object-Oriented Development," *Object Magazine,* Vol. 4, No. 1, May/April 1994, pp. 63-66.
(Me88) Meyer, B., *Object-OrientedSoftware Construction,* C. A. R. Hoare Series Editor, Englewood Cliffs, NJ: Prentice Hali, 1988.

(Me92) Meyer, B., *Advances in Object-Oriented Software Engineering,* Editado por D. Mandrioli e B. Meyer, Englewood Cliffs, NJ: Prentice Hail, 1992.
(Me92a) Meyer, B., *Eiffel: The Language,* Englewood Cliffs, NJ: Prentice Hail, 1992.
(Me92b) Meyers, S., *Effective C++: 50 Specific Ways to Improve Your Programs and Designs,* Reading, MA: Addison-Wesley Publishing Company, 1992.
(Me95) Meyers, S., *More Effective C++. 35 New Ways to Improve Your Programs and Designs,* Reading, MÁ: Addison-Wesley Publishing Company, 1995.
(Me95a) Meyers, 5., "Mastering User-Defined Conversion Functions," *C/C++ Users Journal,* Vol. 13, No. 8, Augusi 1995, pp. 57-63.
(Mu93) Murray, R., *C++ Strategies and Tactics,* Reading, MA: Addison-Wesley Publishing Company, 1993.
(Mu94) Musser, D. R., and A. A. Stepanov, "Algorithm-Oriented Generic Libraries," *Software Practice and Experience,* Vol. 24, No. 7, July 1994.
(Mu96) Musser, D. R., and A. Saini, *STL Tutorial and Reference Guide: C++ Programming with the Standard Template Librarv,* Reading, MA: Addison-Wesley Publishing Company, 1996.
(Ne95) Nelson, M., *C++ Programmer's Guide to the Standard Template Library,* Foster City, CÁ: Programmers Press, 1995.
(Ne920) Nerson, J. M., "Applying Object-Oriented Analysis and Design," *Communications ofthe ACM,* Vol. 35, No. 9, September 1992, pp. 63-74.
(Ni92) Nierstrasz, O.; S. Gibbs; and D. Tsichritzis, "Component-Oriented Software Development," *Communications oftheACM,* Vol. 35, No. 9, September 1992, pp. 160-165.
(P190) Pinson, L. J., and R. S. Wiener, *Applications of Object-Oriented Programming,* Reading, MA: AddisonWesley, 1990.
(Pi93) Pittman, M., "Lessons Learned in Managing Object-Oriented Development," *IEEE Software Magazine,* Vol. lO, No. 1, January 1993, pp. 43-53.
(P192) Plauger, P. J., *The Standard C Library,* Englewood Cliffs, NJ: Prentice Hail, 1992.
(P193) Plauger, D., "Making C++ Safe for Threads," *The C Users Journal,* Vol. 11, No. 2, February 1993, pp. 58-62.
(Po97) PohI, 1., *C+ +Djstilled: A Concise ANSI/ISO Reference and Siyle Guide,* Reading, MÁ: Addison-Wesley, 1997.

(Po97a) PohI, 1., *Object-Oriented Programming Using C++,* Second Edition, Reading, MÁ: Áddison-Wesley Publishing Company, 1997.
(Pr92) Press, W. H., et ai, *Numerical Recipies in C,* Second Edition, Cambridge, MÁ: Cambridge University Press, 1992.
(Pr93) Prieto-Diaz, R., "Status Report: Software Reusability," *IEEE Software,* Vol. 10, No. 3, May 1993, pp. 61-66.
(Pr92) Prince, T., "Tuning Up Math Functions," *The C Users Journal,* Vol. 10, No. 12, December 1992.
(Pr95) Prosise, J., "Wake Up and Smell the MFC: Using the Visual C++ Classes and Ápplications Framework," *Microsoft Systems Journal,* Vol. lO, No. 6, June 1995, pp. 17-34.
(Ra90) Rabinowitz, H., and C. Schaap, *Portable C,* Englewood Cliffs, NJ: Prentice Hali, 1990.
(Re9 1) Reed, D. R., "Moving from C to C++," *Object Magazine,* Vol. 1, No. 3, September/October 1991, pp. 46-60.
(Ri78) Ritchie. D. M.: S. C. Johnson: M. E. Lesk: and B. W. Kernighan. "UNIX Time-Sharing System: The C Programming Language," *The Beli System Technical Journal,* Vol. 57, No. 6, Part 2, JuIy-Áugust 1978, pp. 1991-2019.
(Ri84) Ritchie, D. M., "The UNIX System: The Evolution of the UNIX Time-Sharing System," *AT&T Beil Laboratories Technicaliournal,* Vol. 63, No. 8, Part 2, October 1984, pp. 1577-1593.

(Ro84) Rosler, L., "The UNIX System: The Evolution of C-Past and Future," *AT&T Laboratories Technical Journal,* Vol. 63, No. 8, Part 2, October 1984, pp. 1685-1699.
(Ro00) Robson, R., *Using the STL: The C++ Standard Template Librarv,* SpringerVerlag, 2000.
(Ru92) Rubin, K. S., and A. Goldberg, "Object BehaviorAnalysis," *Communications of theACM,* Vol. 35, No. 9, September 1992, pp. 48-62.
(Ru91) Rumbaugh, J.; M. Blaha; W. Premerlani; F. Eddy; and W. Lorensen, *Ohject-Oriented Modeling and Design,* Englewood Cliffs, NJ: Prentice Hail, 1991.
(Sa93) Saks, D., "Inheritance," *The C Users Journal,* May 1993, pp. 8 1-89.
(Sc99) Schildt, H., *STL Programmingfrom the Ground Up,* Osborne McGraw-Hill, 1999.
(Se92) Sedgwick, R., *Algorithms in C++,* Reading, MA: Addison-Wesley, 1992.
(Se92a) Sessions, R., *Class Construction in C and C++,* Englewood Cliffs, NJ: Prentice Hali, 1992.
(Sk93) SkelIy, C., "Pointer Power in C and C++," *The C Users Journal,* Vol. li, No. 2, February 1993, pp. 93-98.

(Sm92) Smaer, S., and S. J. Melior, *Object Lifecycles: Modeling the World in States,* Englewood Cliffs, NJ: Yourdon Press, 1992.
(Sm90) Smith, J. D.. *Reusabilitv & Software Construction in C & C++,* New York, *NY:* John Wiley & Sons, 1990.
(Sn93) Snyder, A., "The Essence of Objects: Concepts and Terms," *IEEE Software Magazine,* Vol. 10, No. 1, January 1993, pp. 31-42.
(St95) Stepanov, A., and M. Lee, "The Standard Template Library," Internet Distribution, Published at ftp: *II* butler.hpl.hp.com/stl, July 7, 1995.
(St84) Stroustrup, B., "The UNIX System: Data Abstraction in *C,"AT&TBeII Laboratories Technicaliournal,* Vol. 63, No. 8, Part 2, October 1984, pp. 1701-1732.
(St88) Stroustrup, B., "What is Object-Oriented Programming?" *IEEE Software,* Vol. 5, No. 3, May 1988, pp. 10-20.
(St88a) Stroustrup, B., "Parameterized Types for C++, *Proceedings ofthe USENÍX C++ Conference,* Denver, CO, October 1988.
(St9 1) Stroustrup, B., *The C+ + Programming Language* (Second Edition), Reading, MA: Addison-Wesley Series in Computer Science, 1991.
(St93) Stroustrup, B., "Why Consider Language Extensions?: Maintaining a Delicate Balance:' *C++ Report,* September 1993, pp. 44-51.
(St94) Stroustrup, B., "Making a vector Fit for a Standard," *The C++ Report,* October 1994.
(St94a) Stroustrup, B., *The Design Evolution of C++,* Reading, MÁ: Addison-Wesley Publishing Company, 1994.
(St97) Stroustrup, B., *The C++ Programming Language,* Third Edition, Reading, MA: Addison-Wesley Publishing Company, 1997.
(Ta94) Taligent mc., *Taligent 's Guide to Designing Programs: Well-Mannered Object-Oriented Design in C+ +,* Reading, MÁ: Addison-Wesley Publishing Company, 1994.
(Ta92) Taylor, D., *Object-Oriented Information Systems,* New York, NY: John Wiley & Sons, 1992.
(To89) Tondo, C. L., and 5. E. Gimpel, *The CAnswerBook,* Englewood Cliffs, NJ: Prentice HaIl. 1989.
(Ur92) Urlocker, Z., "Polymorphism Unbounded," *Windows Tech Journal,* Vol. 1, No. 1, January 1992, pp. 11-16.
(Va95) Van Camp, K. E., "Dynamic Inheritance Using Filter Classes," *C/C++ Users Journal,* Vol. 13, No. 6, June 1995, pp. 69-78.
(Vi94) Vilot, M. J., "An Introduction to the Standard Template Library," *The C++ Report,* Vol. 6, No. 8. October 1994.
(Vo9 1) Voss, G., *Object-Oriented Programming: An Introduction,* Berkeley, CÁ: Osbourne McGraw-Hill, 1991.
(Vo93) Voss, G., "Objects and Messages," *Windows Tech Journal,* February 1993, pp. 15-16.
(Wa94) Wang, B. L., and J. Wang, "Is a Deep Class Hierarchy Considered Harmful?" *Object Magazine,* Vol. 4, No. 7. November-December 1994, pp. 35-36.
(We94) Weisfeld, M., "Án Álternative to Large Switch Statements," *The C Users Journal,* Vol. 12, No. 4, April 1994, pp. 67-76.
(We92) Weiskamp, K., and B. Flamig, *The Complete C++ Primer,* Second Edition, Orlando, FL: Ácademic Press, 1992.

(Wi93) Wiebel, M., and S. Halladay, "Using OOP Techniques Instead of switch in C++," *The C Users Journal,* Vol. 10, No. lO, October 1993, pp. 105-112.
(Wi88) Wiener, R. S., and L. J. Pinson, *An introduction to Object-Oriented Programming and C++,* Reading, MA: Addison-Wesley, 1988.
(Wi92) Wilde, N., and R. Huitt, "Maintenance Support for Object-Oriented Programs," *ÍEEE Transactions on Software Engineering,* Vol. 18, No. 12, December 1992, pp. 1038-1044.
(W193) Wilde, N.; P. Matthews; and R. Huitt, "Maintaining Object-Oriented Software," *IEEE Software Magazine,* Vol. 10, No. 1, ianuary 1993, pp. 75-80.
(Wi96) Wilson, G. V., and P. Lu, *Parailel Programming Using C++,* Cambridge, MA: MIT Press, 1996.
(Wt93) Wilt, N., "Templates in C÷+," *The C Users Journal,* May 1993, pp. 33-5 1.
(Wi90) Wirfs-Brock, R.; B. Wilkerson; and L. Wiener, *Designing Object-Oriented Software,* Englewood Cliffs, Ni: Prentice Hail, 1990.
(Wy92) Wyatt, B. B.; K. Kavi; and S. Hufnagel, "Parallelism in Object-Oriented Languages: A Survey," *IEEE Software,* Vol. 9, No. 7, November 1992, pp. 56-66.
(Ya93) Yamazaki, S.; K. Kajihara; M. Ito; and R. Yasuhara, "Object-Oriented Design of Telecommunication Software," *ÍEEE Software Magazine,* Vol. 10, No. 1, January 1993, pp. 81-87.

Indice

Símbolos

- operador unário 125
(NOT lógico) 146-149, 1016-1017
‘=77-78
= (operador de desigualdade) 77-78, 518-519, 1016-1017
67-68, 856
* operador do pré-processador 860-861
#* operador do pré-processador 860-

861

$ prompt da linha de comando do
UNIX 868-869
% operador módulo 74-75, 348
% prompt 868-869
& (operador dc endereço) 321-324,
1016-1017
& e * operadores como inversos 323- 324

& em uma lista de parâmetros 226-227
& para declarar referência 224-225
&& (operador AND lógico) 146-147, 219, 1016-1017
&= (operador AND sobre bits com atribuição) 986-987, 1016-1017
O em uma chamada de função 194-

196

(1 operador 194-196
* (operador de derreferência de
ponteiro) 322-324
* operador 74-75
* * operador de exponenciação em
BASIC 518-519
+ operador 72-75
++operador 125

++ operador sobre um iterador 923-924
++ operador de incremento unário 125
+= operador 124-125. 518-519, 893-

894

* operador 1024-1026
.h arquivos de cabeçalho 199-201
/ operador 74-75
/1 comentário de uma só linha 66-67
(operador unário de resolução de
escopo) 228-229, 475-476, 1012-
1013, 1026

<(operador menor que) 77-78
«deslocamento à esquerda 72-73
«operador 66-68
<= (operador menor que ou igual a) 77- 78
= operador 72-75, 148-149
== (operador de igualdade) 77-78, 148- 149, 518-519
> (operador maior que) 77-78
-> (seleção de membro através de ponteiro) 1024-1025

-> operador 1025
>= (operador maior que ou igual a) 77- 78

» deslocamento à direita 73-74 ?: (operador condicional ternário) 105- 106, 219
\ seqüência de escape com caractere barra invertida 68-69
\" seqüência de escape com caractere aspas 68-69
\a alarme 68-69
\n seqüência de escape (nova linha) 68-
69, 642-643
\r retorno do carro 68-69
\t68-69, 179-180
(operador OR exclusivo sobre bits)
1016-10 17
-= (operador OR exclusivo sobre bits com atribuição) 986-987, 1016-1017
1 119-120, 1016-1017
= (operador OR inclusivo sobre bits com atribuição) 986-987, 1016-1017
II (operador OR lógico) 146-148, 219, 1016- 1017
‘\O’ 355-356
\n’ 354-355

Numéricos

<numeric> 928-929, 968, 970
O ponteiro 773-774
OX 657-65 8
Ox 657-658

A

a pilha está corrompida 708-709
a repetição termina 109-111
a STL em programação genérica 1000- 1001, 1055-1056
A tartaruga e a lebre 379-380 a UML 81-82,84

a.out 61-63
aberto 728-729
abordagem de blocos de construção 56-
57, 60-61

abort 478-479, 700-701, 704-705,
708-709, 713-7 14, 86 1-862, 875-876 abortar um programa 699-700
“abordagem de código que funciona”
50-51

“abortar” 115
“aridade” de um operador 518-519
“array inteligente” 272-273
abreviando expressões de atribuição
124-125

abreviaturas semelhantes ao inglês 55-
56

abrir um arquivo inexistente 732-733
abrir um arquivo para leitura 73 1-732
abrir um arquivo para saída 731-732
abstração 82-83
abstração de dados 5 1-52, 452, 479- 480, 524-525, 589-590
ação 50-51, 105-107, 109-110, 151-152, 236-237, 479-480
ação de sair 237-23 8
accumulate 929-930, 967-968, 990- 991

acessando membros union 880-881 Acessando os membros de um objeto através de cada tipo de *handie* para o objeto 407-408
acessibilidade de membros da classe base em classe derivada 577-578
acesso a função 413-414
acesso a membro através de um *handie*
407-408

acesso a membro de classe 406-407

acesso a membro de uma estrutura 406-
407. 808-809
acesso a membro private de uma classe 412
acesso a membros de dados e funções membro não-static 478
acesso à memória inválido 875-876
acesso a variável global 228-229
acesso aos dados do chamador 224-225
acesso detault para membros de struct é publie 412-413
acesso direto 762-763
acesso indexado 940-941
acesso public por default 808
acesso verificado 892-893
acionador (software) de dispositivo 611-
612
ações a serem executadas 100-101, 110-
III
acrescentar a saída a um outro *string*
358-359
acrescentar dados a um arquivo 730-732
acumulador 381
Ada 5 8-59
adaptador 950-951
adaptador de contêiner 919-920, 926- 927, 950-951
adaptador de função 989-990
adiamento indefinido 345-346
adição 53-54, 73-76
adicionar um inteiro a um ponteiro 338-
339
adicionar uma nova conta a um arquivo
746-747
adjacent_differerice 929-930, 985
adjacentfind 929-930, 985-986
adjustfield 662-665
adulteração 230-231
Adulteração para possibilitar ligação segura quanto ao tipo 231-232
advertência 68-69
ajuste da precisão 652-653
ajuste de indicadores 657-658
ajustes originais de formatos 663-664
algoritmo 100-101, 104-106. 110-111. `116, 124-125,760,917-918,918-919, 927-928, 955`
`algoritmo` de "preparar-se para ir trabalhar" 100-101
algoritmo de avaliação 803-805 algoritmo de avaliação de expressão pós-fixa 791-792
algoritmo de avaliação pos-fixo 799-
800, 803-805
algoritmo de conversão de infixa para pós-fixa 791-792, 799-800, 803-805
algoritmo de distribuição 348
algorilmo de embaralhamento 812-813
algoritmo de passagem única 925-926
algoritmo de seqüência não-mutante
`928-929`

algoritmo em pseudocódigo 116
algoritmo numérico 990-991
algoritmo padrão 927-928
Algoritmos básicos de classificação e pesquisa da Biblioteca Padrão 969
algoritmos da biblioteca padrão 200-

201

algoritmos da STL 680-68 1
algoritmos de classificação 969
algoritmos de procura 969
algoritmos de seqüência mutante 928-

929
algoritmos genéricos 928-929
algoritmos matemáticos 965-966
algoritmos numéricos 928-929
algoritmos separados de contêiner 928-
929
<algorithm> 199-201. 935-936
alias 226-227, 322-325, 907-908
alias para o nome de um objeto 428-
429
alias para um tipo 92 1-922 alinhamento 76 1-762, 809-8 10, 880- 881
alinhamento em limite 76 1-762 alinhamento na memória 809-8 lO, 880- 881
alocação de espaço 544-545
alocação de memória 200-201, 918-919
alocação de memória dinâmica 714-715
alocação dinâmica de memória 471,
473-474, 76 1-764, 877-878
alocação dinâmica de memória no estilo de C 877-878
alocador 918-9 19, 936
alocador default 9 18-919
alocador personalizado 918-919
alocar dinamicamente arrays de caracteres para armazenar *strings*
575-577
alocar dinamicamente um array de inteiros 53 1-532, 887-888
alocar e desalocar memória dinamicamente 420-421
alterando o fluxo de controle 144-145
ambiente 191
ambiente C++ 6 1-62
ambiente chamador 73 2-733
ambiente de desenvolvimento de programas 61-62
ambiente de desenvolvimento GNU
1054- 1055
ambiente de programação 190-191
ambiente de programação C++ 190
ambiente hospedeiro 874
ambiente multiusuário 778
ambigüidade 194 196
American National Standards Committee on Computers and Information Processing (X3) 56
American National Standards institute
(ANSI) 50-51, 481-482

American Standard Code for
Information Interchange 360-361
análise 84
análise de texto 390-391
Análise dos resultados de um exame
122- 123
análise e projeto estruturados de
sistemas 59-60
and 1016-1017
AND lógico 1016-1017
AND lógico (&&) 146-147, 185-186,
816-817
AND sobre bits 1016-1017
AND sobre bits(&)8l4-815
AND sobre bits com atribuição 1016-
1017
andeq 1016-1017
aninhamento 104-106, 129-130, 155-
156
aninhamento de estruturas de controle
102-104, 120-121
ANSI 50-51, 56. 48 1-482
ANSI C 56, 65-66
ANSI/ISO 9899: 199056
any 986-987
aparência de blocos de construção 152-
153
apelido 322-325
aplicativos de acesso instantâneo 739-
740
aplicativos distribuídos cliente/servidor
54-55
append 893-894
apresentação de caracteres 199-201
aproximação de números em ponto
flutuante 120-121
área indefinida na memória 810-811

argumento 190-193
argumento da referência 324-325
argumento de função 191-193
argumento de função default 4 19-420
argumento de linha de comando 871-
872
argumento de macro 857-858
argumento default 228-229, 4 17,420
argumento para uma macro 857-858
argumentos default com construtores
4 17,420
argumentos mais à direita (finais) 228-
229
argumentos na ordem correta 194-195
argumentos passados para construtores
de objetos membro 460-461
argv[] 870-871
aritmética 63
aritmética de inteiros 516
aritmética de ponteiros 338-342, 933-
934
aritmética de ponteiros é dependente de
máquina 338-339

aritmética de ponteiros sobre um array de caracteres 339-340
aritmética de ponto flutuante *516*
armazenagem livre 714-715
ARPA 63-64
ARPAnet 63-64
arquivo 727-729, 735, 808
arquivo de acesso aleatório 727, 739- 742, 744-746

arquivo de acesso seqüencial 727, 729- 730, 733-734, 738-740
arquivo de cabeçalho 199-201 408-409,
411-412,583-584, 62 1-622, 857,
873-874, 928-931, 935-937, 941-944,
946, 949-955, 968, 970, 974-976
arquivo de cabeçalho <cassert> 861-
862,478-479
arquivo de cabeçalho <cctype> 200- 201, 329-330, 858-859, 825
arquivo de cabeçalho <csdtlib> 713- 714

arquivo de cabeçalho <csignal> 875- 876

arquivo de cabeçalho <cstdio> 858- 859

arquivo de cabeçalho <cstdlib> 478- 479, 874, 877-878
arquivo de cabeçalho <exception>
706-707
arquivo de cabeçalho <fstream> 728- 729

arquivo de cabeçalho <iomanip h>
119-120
arquivo de cabeçalho <iomanip> 857, 639-640, 65 1-652, 657-658
arquivo de cabeçalho <iostream>
67-68, 200-201, 639-640, 728-729,
857, l0l0-l0l2
arquivo de cabeçalho <iostreain. h>
139- 140

arquivo de cabeçalho <memory> 714- 715

arquivo de cabeçalho **<new>** 711-713

arquivo de cabeçalho <s tdexcept>
716-7 17
arquivo de cabeçalho de entrada/saída
em *streams* <iostream> 67-68
arquivo de cabeçalho definido pelo
programador 199-201
arquivo de contas a receber 727-729
arquivo de estoques 727-729
arquivo de folha de pagamento 727-729
arquivo de implementação 484-485
arquivo de n bytes 729-730
arquivo de texto 745-746
arquivo de transações 756
arquivo em disco 803
arquivo objeto 62 1-622
arquivo objeto pré-compilado 484-485
arquivo seqüencial 727-730, 734-735
arquivo-fonte 873-874
arquivos de cabeçalho 432-434

arquivos de cabeçalho da biblioteca padrão 199-201, 856
arquivos de cabeçalho de contêineres da biblioteca padrão 92 1-922
arquivos de cabeçalho no "estilo antigo"
199-200

arquivos de cabeçalho padrão 191-193
arquivos de código-fonte 408-409
arquivos-fonte múltiplos 408-409
array "bruto" 480-481
Array "inteligente"929-930
array 262-264, 33 1-332, 480-483, 762-
763
Array 9 17-918
array associativo 949-950
array automático 265-267
array baseado em ponteiro no estilo de
C 955-956
array bidimensional 292-298, 3 12-313,
345-346, 760
array bidimensional deck 345-346
array de caracteres 273-274, 343-344,
355-356, 522-523, 904-905
array de caracteres como um ***string*** 274-
275

array de ponteiros para funções 387-388
array de *strings* 345-346
array de *strings* suit 345-346
array dinâmico 877-878
array local automático 276-277
array m por n 292-293
array multidimensional 294
array no estilo deC 9 19-920
array unidimensional 3 26-327, 329,
335-336
arrays baseados em ponteiros 262
arrays com múltiplos subscritos 292-
293, 295
arrays e funções 279-281
arrays passados por referência 282-283
arrays que conhecem seu tamanho 480-
481
arredondando 119-120
arredondar 192-193
árvore 482-483, 760-761, 780-781, 787-
788
árvore balanceada 787-788
árvore binária 760, 780-78 1, 784-785, 981, 983-984
árvore binária com duplicatas 793-794
árvore de pesquisa binária 780-781,
785-787, 793-794
***ASCJI (American Standard Code for
information Interchange)*** 139-140,
360-361, 645-646
aspas 67-69
aspas vazias *(string* nulo) 740-741

asserção 478-479
assert 478-479, 532-534, 543-544, 575-577,699-700,711-712, 861-862
<assert.h> 200-201
assign 892-893, 940-941

assinatura 198-199, 230-231, 546-547, 573,577-578,610-611,622,624-625
assinatura de uma função 198-199
assíncrono 697-698
associa da direita para a esquerda 81-
82, 126-127, 139-140
associa da esquerda para a direita 126-
127
associação l62-l63, 949-950
associações 83-84. 161-162
associações entre classes 162-163
associatividade 8 1-82, 148-149
associatividade da direita à esquerda
81-82

associatividade de operadores 75-76
associatividade não-alterada por
sobrecarga 518-519
asterisco (*) 74-75, 184-185
at 893-894, 907-908, 929-930, 936,
986-987
AT&T 59-60
atender 299-300, 404-405, 412
atexit 874
ativações 302-304
atividade 237-238, 302-304
atof 829-830
atoi 801-805, 829-831
atol 829-830
ator 159-160
atribuição de array 480-481
atribuição de objetos de classes 413-
414,446-447
atribuição de ponteiro 340-34 1
atribuição de ponteiros para classe
derivada a ponteiros para classe base
568-569
atribuição de string 891-892
atribuição de *tring* de caracteres a
objetos string 543-544
atribuição de um objeto a outro 430-431
Atribuição de um objeto a outro com
cópia default membro a membro
430-431

atribuição de uma estrutura a uma
estrutura do mesmo tipo 808-809
atribuição de uma union a uma outra
union do mesmo tipo 880-881
atribuição membro a membro 516-5 17
atribuição membro a membro default
5 16-5 17
atribuir novamente a uma referência
227-228
atribuir um iterador a outro 927
atribuiu o valor de 79-80
atributo 396, 400-401
atributo “const” 1023-1024
atributos 82-84, 161-162, 234-235, 298-
299, 366-367
atributos de classes 366
atributos de uma variável 208-209
atualizar registros no lugar 739-740

atualiLar um registro *757*
áudio 639-640
auto 208-210

auto-atribuição 469-470, 534, *544-545*
auto2tr 714-716
autodocumentação 70-71
automóvel 186, 188
auxflio à depuração 859-860
avaliação da esquerda para a direita 75-77

avaliador pós-fixo 802-803, 805-806
avaliando expressões 774-775, 79 1-792
avaliando uma expressão pós-fixa 792-793
avanço do formulário (‘ \f’) 825, 827- 828

B

B 56
Babbage, Charles 58-59
back 930-931, 936, 95 3-954
backinserter 974-976
badcast 716-717
bad_exception 716-717
badtypeid 7 16-717
badbit 645-646, 732-733
baralho 345-346
baralho simulado 345-346
barra *(“1”)* 236-237
barra de ativação 302-304
barra invertida **(\)** 67-68, 858-859
barra invertida seguida por zero (\O) 274-275
base de *stream* 65 1-652
base de uma pilha 774-775
base e 192-193
base especificada para um *stream* 661-662
basefield 66 1-663, 665-666
BASIC 760-761, 796, 805-806
basicistringstreans 907-908
basicostringstreain 907-908
basicstring 890
BCPL 56
begin 906-907, 920-921, 923-924, 927-928, 933-934
Bell Laboratories 56-57
bibliografia 65-66
biblioteca de matemática 19 1-193, 200-201
biblioteca de tratamento de sinais 875-876
biblioteca de utilitários genéricos
<cstdlib> 478-479, 829-830, 86 1-862, 874, 877-878
biblioteca padrão 190-191, 917-918
biblioteca padrão de C++ 190-191, 535-536
biblioteca padrão de gabaritos (STL) 200-201, 480-481, 629-630, 680-681, 917-918, 1051-1052

bibliotecas 6 1-62
bibliotecas de classes 60-61, 135-136, 170, 190-191, 408-409, 431-432, 565-566, 583-584
bibliotecas de classes independentes de plataforma 918-919
bibliotecas padrão de classes 481-482
bibliotecas privadas 61,63
binário 181-182
binaryfunction 992
binarysearch 968, 970, 971
bit 66 1-662, 727, 808
bitand 1016-1017
bitor 1016-1017
<bitset> 200-201, 92 1-922
bits de estado 645-666

bitset 9 19-920, 985-988
bloco 194-196, 209-214, 421-422
bloco catch 700-705
bloco de construção aninhado 153-156
bloco de construção sobreposto 153-154
bloco de dados 83 8-839
bloco de memória 838-839, 878-879, 940-941
bloco é abandonado (saída do bloco) 209-2 10
bloco está ativo 209-2 10
bloco externo 211-212
bloco interno 211-212
bloco try 700-704, 706-707, 710-711
bloco try externo 706-707
bloco try que o inclui 704-705
blocos aninhados 211-212
"blocos de construção" 83-84
blocos de construção empilhados 153-156
Boas práticas de programação 60-61, 64-65
Bohm,C. 101-103, 154-155
Booch, Grady 84
bool 436-437, 1004-1005
Borland C++ 61, 63, 230-23 1, 874
Borland C++Builder 1001-1002, 1056
branco 103-104, 184-185
hubble sort 283-284, 310-311,334-335, 349-350
Buhble sort com chamada por referência 334-335
bucket sort 317
buferização de saída 667-668
buffer é esvaziado 640-641
buffer é preenchido 640-641
Byron, Lord 58-59
byte 727-729, 813-8 14

C

C & C++ Multimedia Cyber Classroom Third Edition 50-53
C 50-52, 54-56
C clássico 56
C How tu Program 5 1-52

C tradicional 56
cstr 905-906
C++ How To Program 52-53
C++ 54-56, 65-66
cabeçalho catch 704-705
cabeçalho de função 194-196, 206-208, 222-223, 335-336
cabeçalho whiJ.e 138-139
cabeçalhos de colunas 263-264
cadeia de chamadas 698-699
caixa 96-97
caixa automático 159-160. 7 39-740
caixa do sistema 160
calcula o valor de 1 185-186
calcula o valor de p a partir da série infinita 185-186
calcula os rendimentos de um vendedor 178-179
Calculando a soma de dois inteiros lidos do teclado com cm e o operador de

extração de *stream* 645-646
Calculando a soma dos elementos de um array 269-270
Calculando fatoriais com uma função recursiva 215-216
Calculando juros compostos com for 136-137
`calcular recursivamente o valor mínimo` em um array 220-221
cálculodamédia 110-111, 116
cálculos 74-75, 101-103
cálculos aritméticos 74-75
cálculos matemáticos 58-59, 190-191
cálculos monetários 135-136
calloc 877-878
caminhos de decisão 238
campainha 68-69
campo 727-729
campo de bits 813-814, 82 1-825
campo de bits como membro de uma estrutura 822-823
campo de bits sem nome 824-825
campo de bits sem nome com largura zero 824-825
campo de tipo 750-751
campos ajustados 660-661
campos de bits economizam espaço 824-825
campos maiores do que os valores sendo impressos 660-661
capacidade de um string 897, 899
capacity 932-933
captura todos os tipos de exceções 697-698, 704-705
capturar todas as exceções de um certo tipo 697-698
caractere 354-355, 727, 808
caractere apóstrofe (‘) 354-355
caractere barra vertical (1) 119-120
caractere de escape 67-68
caractere de impressão, incluindo espaço 825

caractere de preenchimento 65 1-652, 654-655, 660-661
caractere delimitador 356-357, 361-362
caractere nova linha 141-142
caractere nulo (‘ \O ‘) 274-275, 329- 330, 344-345, 355-358, 362-363, 65 1-652, 760-761, 792-793
caractere nulo de término 274-276, 355- 358, 362-363, 905-906
caractere nulo de término, ‘O’, de um *string* 462-465
caractere sublinhado (_) 70-71
caractere til (-) 404-405, 420-421
caracteres de controle 827-828
caracteres de espaço em branco 80-81, 103-104, 141-142, 274-275, 825, 827-828, 856
caracteres de preenchimento 657-661
caracteres de teste 200-201
caracteres especiais 70-71, 354-355
caracteres representados como códigos numéricos 360-361
característica *defriend* é concedida, não obtida 465-466
característica *defriend* não é simétrica 465-466
característica *defriend* não é transitiva 465-466
características comuns 584-585, 595- 597

carregador 61-62
carregando 61, 63
carregar 61. 63, 381
casamento aceitável para o tipo de uma exceção 705
case com comandos múltiplos l40-141
caso de uso 159-160, 164-165, 366
caso default 137, 140-142, 203-204
caso(s) base 214, 218-220
<cassert> 200-201
cassino 199-201, 206-208
cassino de jogo 199-201
catalogando 431-432
catch (captura) qualquer exceção 705
catch (captura) todas as exceções 716-717

catch (captura) um objeto de classe base 705
catch (captura) uma exceção7l0-711
catch 700-701
catch (. . .) { throw; } 724-725
catch(. . .) 705-707,716-717
catch(exception e) 716-717
CC 61, 63
CD-ROM 52-53
ceil 192-193
CERN 64-65
cerr (erro padrão sem buifer) 63, 639-641, 728-729

<cfloat> 200-201
chamada de função 190-191, 193-196
chamada de função aninhada 699-700, 702-704
chamada de função membro SI 6 chamada de função virtual 630-632

chamada de função virtual ilustrada 631

chamada por referência 223-224, 278- 281, 325-326, 328-329, 33 1-332, 334-335, 430-431, 533-534
chamada por referência com ponteiros 225-226, 324-325
chamada por referência com referências 324-325
chamada por referência const 320, 325- 326

chamada por referência e desempenho 224-225
chamada por referência e segurança 224-225
chamada por referência simulada 279-281. 331-332
chamada por valor 223-224, 278-279, 324-327, 329, 331-332, 430-431, 533-534, 810-811
chamada por valor por default 33 1-332
chamada recursiva 214, 218-219
chamada recursiva de função 774-775
chamadas de construtor 417,420
chamadas de função membro encadeadas 469-470
chamadas de função membro freqüentemente concisas 405-406
chamadas de função membro para objetos corist 452-453
chamadas de funções da biblioteca de matemática 248-249
chamadas dos destruidores em ordem inversa à das chamadas aos construtores 42 1-422, 577-579

chamador 190-191, 194-195 chamando funções por referência 324-325
char 70-71, 138-139, 199-200, 813- 814, 904-905
char * 355-356
char ** 832-833
chave 913-914, 942-943
chave à direita) 67-69, 72-73
chave à direita (}) de término de um bloco 211-212
chave à direita O) de término de uma definição de classe 411-412
chave à esquerda ({) 67-68, 70-7 1
chave de pesquisa 288-291, 942-943
chave de registro 727-729, 757
chaves ({)) 68-69, 80-81, 107-109, 118-1l9, 140-141
chaves duplicadas 942-943, 947-948

chaves em uma estrutura do/while 143- 144
chaves únicas 942-943, 946, 948-949
chefe 190-191
chegada de mensagem pela rede 697-698
Chinês 480-481
ciclo de execução de uma instrução 383-384
científica 662-663
cm *(stream* de entrada padrão) 63
cm 7 1-72, 639-641, 645-646, 728-729, 732-733
cm. clear 665-666
cm . eof 647-649, 665-666
cm. get 138-140, 648-649
cm . getline 356-357
cm. tie 667-668
circuito integrado de silício 50-SI
círculo cheio com ponta de seta anexa 236
clareza 50-51,64-65.71-72,271
class 190, 232-233, 396-397, 400-401. 680-681, 727-729
classe 51-52, 56-57, 82-84, 155-156, 262. 857
classe abstrata 609-6 12, 622, 624-625, 629-630
classe adaptadora priority_queue 954-955
classe adaptadora stack 95 1-952
classe Array 525-526
classe base 564-568, *595-599,* 610-611, 620-621
classe base abstrata 609-612, 615, 619-620, 622, 624-625
classe base catch 705
classe base direta 577-579, 588-589
classe base Employee 611-612
classe base indireta 577-579, 588-589
classe base ios 639-640, 665-666
classe base private 577-579
classe base protected 577-579
classe base public 577-5 78, 705
classe ComnsissionWorker 566-567
classe Complex 448-449, *556-55* 7
classe composta 584-585
classe contêiner4l3-414, 465-466, 482-483, 680-681, 689-690,919-920
classe Date 449-450, 460-461, 546-547

classe de armazenamento 208-209, 210- 211,420-421,872-873
classe de armazenamento automática 209-210, 262, 276-277
classe de armazenamento estática 209-211

classe de auto-referência 760-763
classe de exceção 716-7 17
classe derivada 564-566, 5 77-579, *595*- 597,610-611,615,619-620

a

classe derivada catch 705
classe Employee 460-462, 465, 566-567,611-612
classe gabarito 680, 683-684, 890
classe HugeInt 559-560
classe HugeInteger 449-450
classe IntegerSet 514
classe ios 656-657, 665-666
classeiostream 640-641, 665-666, 1026- 1027
classe istream639-641, 647-649, 665-666, 728-729, 735, 740-741, 744-746, 750-751, 907-908, 1026-1027
classe ieradora 462-465
classe mais derivada 1029-1030
classe Node 760-761
classe ostream 639-640
classe PhoneNumber 562
classe pilha 680
classe pilha de float 680
classe pilha de int 680
classe pilha de string 680
classe Point 570-572, 585
classe Polynomial 562
classe *proxy* 408-409, 482-485
classe Quadrilateral 565-566, 609-611)
classe queue 482-48 3
classe Rational 448-449
classe RationaiNumber 562
classe Rectangle 449-450, 565-566, 609-610
classe SavingsAccount 514
classe string 452, 462-465, 481-482, 535-536
classe string da biblioteca padrão 199-201, 462-465, 890
classe ThreeDimensionalShape 609-610
classe TicTacToe 450
classe Time 400-401,449-450
classe TwoDimensionalShape 609-610
classes 83-84, 396
classes base virtual 1004, 1026-1029
classes concretas 609-6 12
classes contêiner ordenadas 531-532
classes contêineres da biblioteca padrão 919-920
classes de coleções 482-483
classes de exceção definidas pelo usuário 7l6-7l7
classes de exceção derivadas de uma classe base comum 711-712
classes de exceção padrão 716-717
classes de processamento de arquivos 64 1-642
classes de *síream* 728-729
classes genéricas 684-685

classes matemáticas 516
classes "pré-empacotadas" 190-191
classes proprietárias 583-584
classificação com árvore binária 786-787, 806

classificação de uma árvore 786-7 87
classificação por seleção com recursividade 220-22!, 317
cláusula case 141-142
clear 937
cliente 40 1-402, 408-409, 484-485
cliente de uma classe 424-425, 478-479
cliente de uma fila 481-482
<climits> 200-201
clog (erro padrão com buifer) 639-64!, 728-729
dose 733-734
<cmath> 191-193, 198-199, 200-201
COBOL (COmmon Business Oriented Language) 58-59
Code Navigator para C++ 1054-1055
CodeWarrior para Macintosh e Windows 1054-1055
código cliente 610-611
código de caractere 360-361
código de função não-modificável 406-407
código de operação 38!, 798-799
código de operação da SML 381
código em ling uagem de máquina 137, 774-775
código legado 5 1-52, 328, 862-863, 868-869, 877-878
código legado em C 452, 856-85 8, 862-863, 868-869, 877-878
código não-otimizado 803-805
código objeto 61-63, 408-409
código objeto de uma classe 408-409
código otimizado 804-805
código personalizado 192-193
código portável 56-57
código reentrante 406-407
código seqüência 608-609
código-fonte 408-409, 583-584
código-fonte de uma classe 408-409
coeficiente 562
coerção “para cima” 1029-1030
coerção 340-341. 573
coerção de argumentos 198-199
coerção explícita, 567-568
coerção ilegal 1005-1006
coerção no “estilo antigo” 1005-1006
coerções não-padrão 1008-1009
coerções padrão 1008-1009
colaboração 363-364
colaborações 234-235, 298-299, 363-367, 432-434
colchetes 236-237
colchetes **([])** 263-264
colchetes angulares (<e >) 232-233, 680-681, 856

colocando *strings* em ordem alfabética 360-36!
colocar em evidência as características comuns 584-585
“colocar em evidência” atributos e comportamentos comuns 584-585
colocar em zero indicadores designados 665-666
coluna 292-293
comando 65-68
comando break 140-142, 144-146, 186, 188, 706, 878-879
comando break em uma estrutura for 144-145
comando composto 107-110, 118-119, 194-196
comando continue 144-145, 186, 188

comando continue em uma estrutura for 145-146
comando de atribuição 72-73, 126 comando de saída compilado
condicionalmente 859-860
comando de Simple 796
comando end de Simple 796, 798-799
comando executável 7 1-72
comando goto 101-103, 153-154, 211- 212, 878-879
comando goto em Simple 796-799 comando if/goto em Simple 796-
797, 800-802
comando incondicional goto em Simple 796-797
comando input de Simple 796
comando let em Simple 796, 799-80!.
805-806
comando print de Simple 796, 798- 799
comando rem em Simple 796, 798-799 comando return 66-69, 190-l9l,
194-197, 214, 324-325, 804-805
comando return em um tratador catch 706
comando Se em pseudocódigo 104-105 comando separado em várias linhas 80-
81
comando throw 706-707
comando using 1009-1010
comando vazio 108-109
comandos de Simple 796
comandos entre chaves 194-196
combinação de teclas que indica fim de arquivo 73 3-734, 868
combinando estruturas de controle de duas maneiras 149-150
comentário 66-67, 7 1-72, 803
comentário de uma única linha 66-67
comitê técnico X3J II 56
como o sistema deve ser construído
159- 160
comp.lang.c 1051-1052

comp.lang.c++ 1051-1053
comparação de arrays 480-481
comparação de ponteiros 340-341
comparação de strings 893-894
comparações 284-285
comparando blocos de memória 838-
839
Comparando inicialização de arrays static e automática 276-277
comparando *strings* 356-357
comparando strings 893-894
comparando uniões 880-88 1
compare 895-896
compartilhamento de tempo 54-55, 58- 59
compartilhar os recursos de um computador 54-55
compilação condicional 856, 858-859 compilação do comando if/goto
799-800
compilador 55-56. 6 1-62, 68-69, 118- 119,774-775, 805-806
compilador Borland Turbo C++ Visual Edition for Windows 1054-1055
compilador gera instrução SML 799- 800
compilador LCC-Win32 para Windows 95/NT 1054-1055
compilador otimizador 137, 2 10-211.
452-453
compilador(es) C++ 61,63, 1050-1051
compiladores para C++ 1050
compilando 56, 760. 87 3-874
compilando um programa com
múltiplos arquivos-fonte 872-873
compilar 61, 63, 408-409
compJ, 1016-1017
complemento de um 814-815, 819
complemento do construtor 420-421
complemento sobre bits 1016-1017
complexidade 564
complexidade de software 564
complexidade exponencial 2 19-220
componente 190, 43 1-432. 565-566,
917-918
componente reutilizável padronizado
565-566
componentes 59-60, 83-84
componentes de software reutilizáveis
56-57

componentes reutilizáveis 60-61, 431- 432, 565-566
comportamento 82-84, 160, 396. 400- 401, 404-405, 412, 480-481
comportamento inesperado 705
comportamento não-polimórfico 610-
611
comportamento polimórfico com referências 620-621
comportamentos public 40 1-402

composição 162-163, 406-407, 485-
487, 564-565, 584-585, 777
composição em vez de herança 605-606
comprimento de um *string* 274-275,
355-356, 895-896
comprimento de um *.uh.tring* 545-546
comprimento máximo de um *string* 897, 899
compromisso espaço-tempo 744-746
computação 52-53
computação cliente/servidor 54-55
computação conversacional 71-72
computação distribuída 54-55
computação interativa 71-72
computação pessoal 54-55
computação por 'força bruta" 185-186
computador 52-53
computador Apple 54-5 5
computador cliente 54-55
computador DEC PDP-7 56
computador pessoal 52-53,61,63
Computador Pessoal IBM 54-5 5, 654-
655
computadores isolados 64-65
comutação de pacotes 63-64
comutativo 519-520
concatenação 72-73, 544-545
concatenação de strings 89 1-892
concatenar 893-894
concatenar dois objetos lista encadeada
790-791
concatenar *strings* 544-545
concatenar *strings* literais 900-901
conceitos básicos de computadores 51-
52
condição 77-78, 103-106, 144-146,
236-237
condição de continuação do laço 128-
133, 142-145
condição de continuação do laço falha
2 19-220
condição de guarda 236-23 8
condição de término 2 15-216, 272-273
condição dependente 148-149
condição excepcional 141-142
condição mais à esquerda 148-149
condição simples 146-149
condições complexas 146-147
conectado ao dispositivo de entrada
padrão 640-641
conectado ao dispositivo de erro padrão
640-641
conectado ao dispositivo de saída
padrão 640-641
conectando um *stream* de saída a um
stream de entrada 667-668
conexão em rede 639
conflito de escopos 1010-1011
conhece um 585
conjunto 412-414, 423-424, 446-447

conjunto de caracteres 96-97. 142-143. 360-361, 727-729
conjunto de caracteres ASCII 97-98. 354-355, 639
const 223-224, 226-227, 279-282, 324-325. 522-523, 857, 1007-1008
const char * 329-330
const com parâmetros de função 329
const int * **const** 332-333
constiterator 906-907, 920-924, 927, 933-934, 945-949
constreference 922-923
constreverseiterator 907-
908, 92 1-923, 927, 933-934

constante 19 1-193, 798-799
constante com nome 267-268
constante de caractere 354-355
constante de enumeração 206-208. 859- 86(1
constante de ponto flutuante sem sufixo
875-876
constante simbólica 856-859, 86 1-862
constante simbólica NDEBUG 86 1-862
constante simbólica P1 857-858
Constante simbólica predefinida
DATE 86 1-862
constante simbólica predefinida
FILE 861-862
constante simbólica predefinida
LINE 861-862
constante simbólica predefinida
TINE 861-862
constantes simbólicas predefinidas 861-
862
construindo seu próprio compilador
760, 796
construindo um compilador 797-798
construtor 416-417, 422-423, 434-435
construtor chamado recursivamente
533-534
construtor da classe base 577-579
construtor de argumento único 534-
535, 543-544, 1018-1020, 1022-
1023
construtor de classe derivada 577-579 construtor de classe derivada chama o
construtor da classe base 577-579 construtor de classe derivada chama o
construtor default da classe implicitamente 577-579
construtor de conversão 534-535. 543- 545, 582-583, 1020-1021
construtor de cópia 530-534, 920-923 construtor de cópia da classe string
890-891
construtor de cópia para um objeto disparado 710-711
construtor de um único argumento
1017-1018
construtor de union 880-881

construtor default 4 17,420, 462-465, 530-533, 549-550, 686-688, 920-921
construtor default de objeto membro
462-465
construtor em uma união 880-881
construtor **explicit** 1020-1021
construtor ofstream 731-732
construtores de argumento único e conversões implícitas 1018-1019 construtores de cópia na passagem de parâmetros em chamadas por valor
545

construtores de objetos globais 421-422 construtores e destruidores chamados
automaticamente 420-421
construtores não podem ser virtuais
62 1-622
construtores sobrecarregados 416-417
consumo de memória 629-630
conta de poupança 134-135
contador 110-111, 115, 121-122, 179 - 180, 209-2 10
contador de instruções 800-801
contador de laço 113-114, 126-127
contagem baseada em zero 130-131
Contando conceitos alfabéticos 138
contando de um em um para cima 113-
114
contêiner 200-201, 760, 906-907, 917- 920, 955
contêiner associativo 9 19-920, 922-923, 926-927. 942-943, 945-946
contêiner associativo multimap 946-
947

contêiner associativo **set** 946
contêiner de primeira classe 92 1-924, 926-927, 933-934, 937
contêiner seqüencial 9 19-920, 926-927, 929-930, 936, 939-940
contêiner seqüencial deque 940-941
contêiner subjacente 951-952
conteúdo dinâmico 57-58
Controlando a impressão de ieros após a vírgula e casa decimal com valores float 658-659
Controlando a precisão de valores em ponto flutuante 653-654
controle centralizado 63-64
controle de programa 100-101
converge para o caso base 219-220
conversão de classe derivada para classe base 705
conversão de objeto da classe derivada para objeto da classe base 582-583 conversão de ponteiros 573
conversão entre um tipo primitivo e uma classe 544-545
conversão explícita 118-119
conversão implícita 118-119, 543-544, 1017-1018, 1020-1023, 1026-1027

conversão implícita entre tipos primitivos definida pelo compilador
544-545
conversão implícita imprópria 10 18-
1019
conversão perigosa 573
conversões entre tipos primitivos 535-
536

conversões entre tipos primitivos por coerção 535-536
conversões implícitas 1018-1019 conversões implícitas através de construtores de conversão 1020-1021 conversões implícitas definidas pelo usuário 543-544
conversões padrão 1005-1006 conversor de infixa para pós-fixa 805-
806
converte entre tipos 534-535 converte entre tipos definidos pelo usuário e tipos primitivos 535-5 36 converte letras minúsculas 200-201
convertendo de um tipo de dados mais alto para um tipo de dados inferior
199-200
convertendo entre classes e tipos primitivos 413-414, 446-447 Convertendo **strings** para *strings* no estilo de C e arrays de caracteres
904-905
Convertendo um *string* para maiúsculas
329-330
cópia do argumento 329
cópia membro a membro 430-431, 532- 533

cópia membro a membro default 430-
431,532-533,543-544,922-923
cópia não-incrementada de um objeto
55 1-552
copia um *string* 343-344
copia um *string* usando notação de array 343-344
copia um *string* usando notação de ponteiro 343-344
copiando *strings* 343-344
Copiando um *string* usando notação de array e notação de ponteiro 343-344
copy 905-906, 928-929, 935-936
copy_backward 928-929, **97** 1-973
Corcoran, Marian 1000-1001, 1055- 1056

corpo com múltiplos comandos 80-81 corpo de um laço 128-129, 132-133,
186, 188
corpo de uma definição de classe 400-
401

corpo de uma estrutura while 109-110
corpo de uma função 67-69, 194-196
corpo do else 105-106
corpo do while 109-110
correção 63

correio eletrônico 63-64
correto no sentido matemático 150-152
cos 192-193

co-seno 192-193
co-seno trigonométrico 192-193
courit 929-930, 942-943, 945-946, 965-966, 986-987
countif 929-930, 965-968
cout («) (o *stream* de saída padrão) 63, 67-68, 70-72,639-641, 643-644,728-729
cout put 645-646
cout. setf 662-663
cout **write** 650-651
CPU 53-54, 6 1-63
craps 206-209
cria dinamicamente a quantidade exata de espaço 462-465
criação dinâmica de objetos 44 1-442 Criando e destruindo objetos
dinamicamente 302-304
Criando e percorrendo uma árvore binária 781-782
Criando e testando manipuladores de
stream não-parametrizados definidos
pelo usuário 655-656
Criando e usando uma função definida pelo programador 193-194
criando novos tipos de dados 479-480
Criando um arquivo de acesso aleatório seqüencial mente 741-743
Criando um arquivo seqüencial 729-730
criando uma associação 949-950
Criando uma estrutura, inicializando seus membros e imprimindo a
estrutura 398-399
criar novos objetos dinamicamente 435-436

criar novos tipos 480-481
criar seus próprios tipos de dados 72-73
criar um arquivo de acesso aleatório
740-74 1
criptografia 182-183, 913-9 15
criptografla com chave simétrica 913-914
criptógrafo 913-914
criptograma 914-915
crítico para a missão 699-700
Crivo de Eratóstenes 317, 987-988, 1000
<**csetjmp**> 699-700
<**csignal**> **875-876**
<cstddef> 357-358
<**cstdio**> 200-201
<cstdlib> 199-201, 829-830, 201-202
<**cstring**> **200-201,** 545
<**ctime**> **200-201,** 205-206
ctor 543-544
<ctype **h**> 200-201
<ctrl>-c 875-876
<ctrl>-d 139-140,647-649.654-655,
732-733

<ctrl>-z 139-140, 647-649, 654-655,
732-733
cursor 68-69
cursor da tela 68-69
cursos de C++ on-line valendo créditos
1050-1051
dado de seis faces 20 1-202
dados 52-53
dados "brutos" 739-740
dados de saída 63
dados em uma classe abstract 622,
624-625
dados não-inicializados 399-400
dados public 402-403

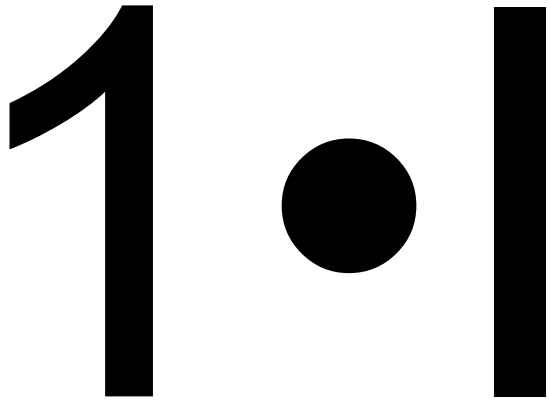
data 905-906
Date com operadores de incremento sobrecarregados 547-548
DECPDP-I1 56
decimal por default 66 1-662
decisão 50-51, 104-106, 155-156
decisão lógica 52-53
declaração antecipada 438-440, 494-495
declaração antecipada de classe 484-485
declaração de parâmetro 194-195
declaração externa 809-810
dcc laração(ões) 70-71
declarações dos parâmetros 194-195
declarando uma função membro const static 478
declarar um tipo de retorno para um construtor 417,420
declarar uma função membro const 457-459
declarar uma referência 440
decoração do nome 230-231
decrementar 126-127
decrementar um ponteiro 338-339
decriptador 913-9 14
#define 411-412, 858-859, 861-862
#define NDEBUG 478-479, 86 1-862
#define P1 3.14159 857
defined 859-860
defined 859-860
definição 127-128
definição de classe 401-402
definição de estrutura 396-397, 808-809. 82 1-822
definição de função 193-196,211-212
definição de função como um protótipo de função 194-195. 197-198
definição de função mime 408-409
definição de gabarito 232-234, 68 1-682
definição de gabarito de classe 684-685
definição de getchar como macro 858-859
definição de macro 860-861

definição simples da class Time 40 1-402
deitel@deitel .com 52-53
deleção 482-483
deletando memória alocada dinamicamente 47 8-479
deletando um item de uma árvore binária 787-78 8
deletar um registro de um arquivo 746-747
delete 471,473-475, 485, 533-534, 714-7 17, 762-763, 877-878
delete [] (desalocação dinâmica de array) 474-475
delete operator 62 1-622 delimitador (com valor default '\n') 648-649
delimitador default 650-651
Demonstração de desempilhamento da pilha 709-7 10
Demonstrando a classe adaptadora priorityqueue da Biblioteca Padrão 955
Demonstrando a classe adaptadora stack da Biblioteca Padrão 951- 952
Demonstrando a classe bitset e o Crivo de Eratóstenes 987-988
Demonstrando a classe Circie 587 Demonstrando a classe Cylinder 589-590
Demonstrando a classe Point 55 Demonstrando a função membro flags 663-664
Demonstrando a função membro width 654-65 5
Demonstrando a função substr 896-897
Demonstrando a ordem em que construtores e destruidores são chamados 42 1-422
Demonstrando alguns algoritmos matemáticos da Biblioteca Padrão 965-966
Demonstrando as funções de manipulação de elementos do gabarito de classe vector da Biblioteca Padrão 934-935

Demonstrando as funções equal,
mismatch e
iexicographicai_compare da
Biblioteca Padrão 957-958 Demonstrando as funções erase e
replace 90 1-902
Demonstrando as funções fiji,
filin. generate e
generate_n da Biblioteca Padrão
955-956
Demonstrando as funções **f** ind de string 898-899

Demonstrando as funções insert de string 903-904
Demonstrando as funções remove.
removeif. **remove copy** e
remove_copy_if da Biblioteca
Padrão 960-961 Demonstrando as funções **replace.**
replace_if, replace_copy e
repiacecopyif da Biblioteca
Padrão 963 Demonstrando as operações sobre
conjuntos (set) da Biblioteca Padrão 977
Demonstrando as palavras-chave operadores l0l7-1018
Demonstrando atribuição e concatenação de string 891-892
Demonstrando autoytr 714-7 15 Demonstrando copy_backward.
merge. unique e reverse 972- 973
Demonstrando dynamic cast 10 14- 1015
Demonstrando gabarito de classe set da Biblioteca Padrão 946
Demonstrando herança de interface com a hierarquia de classes Shape
104-105, 624-627
Demonstrando herança múltipla 592 Demonstrando mnplace_merge.
unique_copy e reversecopy
974-975
Demonstrando iteradores de *siream* **de** entrada e saída 923-924
Demonstrando leitura a partir de um objeto istringstream 909-9 10
Demonstrando lowerbound.
upper_bound e equal_range
979-981
Demonstrando new disparando badalloc em caso de erro 713
Demonstrando new retornando O em caso de erro 712
Demonstrando o gabarito de classe deque da Biblioteca Padrão 941- 942
Demonstrando o gabarito de classe iist da Biblioteca Padrão 937-938
Demonstrando o gabarito de classe map da Biblioteca Padrão 949-950
Demonstrando o gabarito de classe multimap da Biblioteca Padrão
947-948
Demonstrando o gabarito de classe multiset da Biblioteca Padrão
943-944
Demonstrando o gabarito de classe **Stack 684-686**
Demonstrando o gabarito de classe vector da Biblioteca Padrão 931- 932

D



Demonstrando o operador
constcast 1 (X)7- 1008
Demonstrando o operador
reinterpretcast 1008-1009 Demonstrando o operador
staticcast 1006-1007
Demonstrando o qualificador de tipo const 28 1-282
Demonstrando o tipo de dados primitivo bool 1004
Demonstrando o uso de naxnespaces
1010-1011
Demonstrando os algoritmos miri e max 984-985
Demonstrando os gabaritos de classe adaptadora queue da Biblioteca Padrão *953-954*
Demonstrando os operadores . * e
1025
Demonstrando polimorfismo com a hierarquia de classes Employee
612-613
Demonstrando setnewhandler
7 13-714
Demonstrando swap. iter swap e swap_ranges 971
Demonstrando typeid 1013-1014 Demonstrando um array de ponteiros
para funções 353-354
Demonstrando um construtor
explicit 1020-1021
Demonstrando um membro de dados mutable 1024-1025
Demonstrando um objeto função binária
990-991
dependente de máquina 55-56, 322-324, 338-339, 479-480
depuração 203-204, 478-479, 699-700
depurador 857
depurar 5 8-59, 64-65
deque 919-920, 926-927, 929-931, 936, 937, 939-940, 950-953
<deque> 200-201, 921-922, 941-942
dequeue 778
derreferenciando um ponteiro O 322-
324

derreferenciando um ponteiro 322-326, 329

derreferenciando um ponteiro void *
340-341
derivar uma classe de outra 405-406
derreferenciar um iterador 923-925, 927

derreferenciar um iterador const 925-
926

derreferenciar um iterador posicionado fora de seu contêiner 925-926
derreferenciar um não-ponteiro 322-324 desaloca memória alocada com new
76 1-762
desalocar memória 714-715

desconectando um *stream* de entrada de um *stream* de saída 667-668
desempenho 56-57
desempenho de classificação e pesquisa em árvore binária 806
desempenho de pesquisa 806
desempenho na pesquisa em listas 806
desempilhamento da pilha 698-700,
708-711
desempilhando a pilha de chamadas de função 709-7 10
desenhando uma forma 608-609
desenvolvimento de aplicativos rápido
(RAD) 43 1-432
desenvolvimento de classe 524-525
desenvolvimento de software orientado a componentes 481-482
desigualdade 1016-1017
desligar 697-698
deslocamento 630-632, 739-740
deslocamento à direita (») 8 14-815
deslocamento a partir do início de um arquivo 735-736
deslocamento em bytes 739-740
deslocamento para um ponteiro 341-342
deslocar um intervalo de números 201-
202
destruído automaticamente 213-214
destruidor 404-405, 420-423, 434-435,
920-921
destruidor da classe base 62 1-622
destruidor de classe base virtual
62 1-622
destruidor de classe derivada 621-622 destruidor de classe derivada chamado
antes do destruidor de sua classe base
577-579

destruidor do objeto disparado 711-712 destruidor não recebe parâmetros nem
retorna valor 420-421
destruidor virtual 62 1-622 destruidores não podem receber
argumentos 404-405
destruidores não podem ser sobrecarregados 404-405
destruindo objetos dinamicamente 302-
304
destruir objetos 485
desviar se negativo 799-800
desvio incondicional 878-879
desvio incondicional goto 878-879
detalhes secretos" da implementação
1023-1024
detalhes de implementação, ocultos
396-397
detectar um erro 697-698
diagnósticos que ajudam na depuração de programa 200-201
diagrama de atividades 237-238, 298- 299, 366-367

diagrama de casos de uso 159-160
diagrama de classes 16 1-162, 234-235, 298-299, 366, 432-435, 437
diagrama de classes da UML 432-434
diagrama de colaborações 365
diagrama de mapa de estados 236-23 8, 298-299, 366-367
diagrama de objetos 301-302
diagrama de seqüência 301-304, 365- 367, 432-434, 440
diagramas de estado 236
diagramas de objetos 164
diálogo 7 1-72
diâmetro 103-105
Dicas de desempenho 60-61

Dicas de portabilidade 60-61, 65-66
Dicas de teste e depuração 60-61
dicionário 757-758
dicionário computadorizado 757
differencetype 922-923
dígito 70-71, 354-355
direção de busca 735-736
diretiva #error do pré-processador
860-861
diretiva do pré-processador #define
680-681, 857
diretiva do pré-processador #endif
859-860,41 1-412
diretiva do pré-processador #ifdef
859-860
diretiva do pré-processador # fndef
859-860,411-412
diretiva do pré-processador #include
197-201, 408-409, 856
diretiva do pré-processador #line
860-861
diretiva do pré-processador #undef
858-859, 86 1-862
diretivas condicionais do pré- processador 85 8-859
diretivas do pré-processador 61,63,67- 68, 70-71, 411-412
diretório do sistema de arquivos 760
disco 52-54, 61, 63, 797-798
disco magnético 727
disco óptico 727
disparando novamente uma exceção
707-708
disparando uma exceção 703-704
disparar exceções derivadas de exceções padrão 7 16-717
disparar exceções não-derivadas de exceções padrão 7 16-717
disparar exceções padrão 716-717
disparar novamente uma exceção 706-
707

disparar objetos const 706
disparar um int 703-704
dispositivo de entrada 52-53

dispositivo de memória secundária 61, 63, 727
dispositivo de saída 5 *3-54*
distribuidor 378-379
DivideByZeroException 702-703
divides 989-990
dividir para conquistar 190, 192-193
divisão 53-54, 74-76, 120-121
divisão de inteiros 74-75, 118-119
divisão de ponto flutuante 118-119
divisão por zero 63, 115, 697-698, 700-
701, 875-876
divisão por zero é indefinida 480-481
documentação de classes derivadas 584-
585

documentar um programa 66-67
documento de especificações da UML

1.3 84-85

dois maiores valores 179-180
dois níveis de refinamento 116
dois-pontos (:) 211-212, 460-461, 575- 579, 594-595, 878-879
dois-pontos 299-300
dois-pontos (:) no cabeçalho de uma definição de classe 57 1-573
DOS 868, 875-876

double 135-136, 191-193, 198-199,
875-876

downcast 1007-1008, 1014-1016
download gratuito 1054-1055
downloads gratuitos de edição de demonstração 1(154-1055
Duas maneiras de declarar e usar funções que não recebem argumentos
22 1-222

dump 383-385

dynamic_cast 568-569, 7 16-717, 1013-10 16

E

E maiúsculo 657-65 8
EIS de alto nível 639
EIS de arquivos 640-641
EIS na memória 907-908
EIS não-formatada 639-640, 650-651
EBCDIC (Extended Binarv Coded Decimal Interchange Code) 360-361
economizando memória 209-2 10
editando um arquivo 61,63
editar 61,63
editor 6 1-62
editor de ligação 6 1-62, 872-873
editor de texto 354-355, 733-734
editor emacs 61,63
efeitos colaterais 210-211, 219, 223-
224, 680-681
elemento de ordem zero 262-263
elemento de sorte 199-201
elemento de um array 262
elemento fora do intervalo 53 1-532

elemento particionador 985 elevando um inteiro a uma potência
inteira recursivamente 220-221 elevar a uma potencia 173-174, 192-193 Elevar uma variável ao cubo usando
chamada por referência com um
argumento ponteiro 325-326 Elevar uma variável ao cubo usando
chamada por valor 325-326 #elif 859-860
eliminação de duplicatas 760, 7 86-787,
793-794

eliminação do comando goto 153-154
eliminação do goto 101-103
elipse 96-97, 101-103
else 105-106
e-mail 63-64
embaralhar 345-346

embaralhar cartas 850-85 1
empacotando código como uma função
192- 193
empilhamento de estruturas de controle
102-104, 150-152

empilhando 102-106, 155-156
empty 920-921, 95 3-955
encadeamento 72-73, 643-646
encadeamento de atribuições 544-545
encadeamento de put 645-646
Encadeando chamadas a funções membro 470-473
encadeando operadores += *544-545*
encadeando operadores de inserção em *stream* 643-644
encapsula 82-83
encapsulamento 405-406, 428-429, 445, 462-465, 573
end 920-921, 923-928, 933-934, 985- 986

endereço de memória 320-321, 644-645
endereço de um campo de bits 824-825
endereço (&) de uma estrutura 808-809
endi 72-73, 119-120,642-643
endLine 655-656
enfllleirar 48 1-482
enqueue 778
entrada com *streams* 640-641, 645-646
entrada e saída de *strings* 805-806
entrada padrão 7 1-72, 868
entrada via teclado 117-118
entrada/saída *(EIS)* 190-191, 639
entrada/saída com *stream,s* 67-68
entrada/saída de array 480-481
entrada/saída de objetos 750-751
entrada/saída formatadas 639-640, 738-

739
enum 206-208
enumeração 206-208. 857
enumeração na classe ios 656-657
Enviando dados de saída para strings na memória 907-908

enviando para a saída tens de dados de tipos primitivos 64 1-642
Enviando para a saída o valor de uma expressão 643-644
enviando para a saída um valor em ponto flutuante 657-658

Enviando um *string* para a saída usando
duas inserções em *stream* 642-643
Enviando um *string* para a saída usando
inserção em *stream* 64 1-642
enviar mensagens usando uma
referência 437
EOF 138-139, 647-651. 825
equação da linha reta 76-77
equal 929-930, 957-960
equal_range 945-946, 978-984
equalto 989-990
equivalente decimal de um caractere
ASCII 805-806
equivalente em palavras do valor de um
cheque 392-393
erase 90 1-902, 92 1-922. 937
erro de compilação 68-69
erro de estouro 479-480
erro de estouro aritmético 716-717
erro de formato 665-666
erro de lógica 77-78. 112-113
erro de lógica fatal 115
erro de sintaxe 68-69
erro de subscrito fora do intervalo
válido 703-704
erro de *underflow* aritmético 716-717
erro detectado em um construtor 710-

711

erro do editor de ligação 1009-1010
erro durante a compilação 68-69
erro durante a execução 63
erro fatal 63, 115, 385-386
erro fatal durante a execução 322-324,
573-574
erro não-fatal 63, 197-198,699-700 erro por um 113-114, 130-131, 263-

264
erro síncrono 697-698
erros 61,63
Erros comuns de programação 60-61
erros de ligação 230-231
erros relacionados a catch 711-712
escala 269-270
escalar 278-279. 334-335
escalonável 269-270
escopo 131-132,210-211,404-405,
582-583, 697-698, 872-873, 1009-
1010

escopo aninhado 703-704 escopo de arquivo 211-212, 406-407,
474-475. 69 1-692. 882-883 escopo de bloco 211-212 escopo de classe 211-212,404-407 escopo de função 211-212,406-407 escopo de gabarito de classe 686-688

escopo de protótipo de função 211-212
escopo de um switch 706
escopo de uma constante simbólica ou
macro 858-859
escopo de uma função 873-874
escopo de urna variável 208-209
escopo global 421-422, 1012-1013
escrevendo dados aleatoriamente em um
arquivo de acesso aleatório 742-743
esgotar a memória 216-217. 573-574
espaçamento interno 660-661
espaçamento vertical 103-104. 129-130
espaço (◦) 70-71
espaço de objetos 1000-1001, 1056
espaço em branco 645-649, 651-652,
860-861
espaço em disco 711-714.732-733
espaço vazio 740-741
espaços 36 1-362
espaços para preenchimento 660-661
especificação de exceção 708-709
especificação de exceção vazia 708-709
especificações da UML 1.3 162-163
especificações de ligação 883-884
especificador de acesso a membro 401-
402, 465-466
especificador de acesso a membro
public 401-402
especificadores de classe de
armazenamento 208-209
especificadores de ligação static
1012-1013
especificidades 610-611
espiral 216-2 17
estação de trabalho 54-55
estado consistente 404.417,420,424-
425

estado de erro de um *,stream* 665-666
estado de formato 663-664
estado de ftrmato anterior 663-664
estado inicial 236
estados 236. 298-299

estados de erro 645-646, 666-667
estendem a linguagem de programação base 480-481
estouro 875-876
estouro aritmético 697-698
estratégia de tratamento de exceções
706
estrutura 262, 320, 33 1-332, 396-397, 808, 857
estrutura de auto-referência 397-398,
808-809
estrutura de controle 100-106. 416-417,
878-879
estrutura de controle aninhado 152-153,
878-879
estrutura de controle de entrada única e saída única 102-104
estrutura de dados encadeada 397-398

estrutura de dados linear 762-763, 780- 781
estrutura de dados não-linear, bidimensional 780-781
estrutura de dados subjacente 954-955 estrutura de dados último a entrar,
primeiro a sair (LIFO) 479-480, 774- 775, 9 19-920, 950-95 1
estrutura de entrada única e saída única
104-105. 150-152
estrutura de repetição 101-103, 109-
110,115,796-797
estrutura de repetição do/while 102-
104, 142-144, 152-153
estrutura de repetição for 129-130
estrutura de seleção 101-104
estrutura de seleção dupla 102-104, 121-122, 137
estrutura de seleção múltipla 102-104, 137

estrutura de seleção simples 137
estrutura de seleção simples if 102-
105, 107-108
estrutura for 130-131, 134-135, 152- 155

estrutura for aninhada 295
estrutura for externa 295
estrutura funcional de um programa 68-69
estrutura hierárquica 566-567
estrutura hierárquica semelhante a urna árvore 566-567
estrutura if 77-78, 80-81, 102-105. 107-108, 137, 146-147, 154-156
estrutura if/else 102-109, 121-122, 137, 146-147
estrutura if/else aninhada 106-108 estrutura se/senão em pseudocódigo
105- 106
estrutura seqüencial 101-103
estrutura switch 102-104, 137, 140- 143, 154-155, 564-565, 608, 706,
750-751
estrutura switch com break 140-141 estrutura while 102-104, 109-111.
115, 118-119, 121-122, 128-129,
142-143, 152-156
estruturas de dados 262, 760, 917-918
estruturas de dados dinâmicas 262, 320, 760

estruturas de dados pré-empacotadas
760

estruturas normalmente são passadas através de chamada por valor 399-

400

estruturas profundamente aninhadas
153-154

Estudo de caso: uma classe Date 547-

Estudo de caso: uma classe String
535-536

esvaziando um *stream* 651-652
esvaziar o buifer de saída 72-73, 642- 643
esvaziar um buffer 667-668
esvaziar um *stream* 874
etapa de particionarnento 387-388
etapa de recursão 214, 218-219. 387- 388
etiqueta de estrutura 396-397
"é um" 565-566, 570-572, 577-579. 584-585, 592, 594-595
"é um" é herança 564-5 65
evento 236
evento inesperado 875-876
evitar perda de memória 7 16-717
evitar que arquivos de cabeçalho sejam incluídos mais de uma vez 411-412
evitar que o objeto de uma classe seja atribuido a outro 534
evitar que objetos de classe sejam copiados 534
evitar repetir código 192-193, 419-420 Evitar um problema de precedência entre o operador de inserção em
síream e o operador condicional 646-
647

evitar uma chamada de função 222-223
examinar dados 404
examinar qualquer tentativa de
modificar dados 424-425
exceção bad alloc 711-7 14, 716-
717, 761-762, 936
exceção de classe base 7 16-717
exceção de divisão por zero 700-701
exceção de ponto flutuante 875-876
exceção inesperada 698-699
exceção invalid argument 936
exceção length error 936
exceção não-capturada 698-699
exceção não-listada em especificação de
exceção 708-709
exceção out_of_bounds 936
exceção out of_range 716-717.
892-893, 936, 986-987
exceções ocorrendo em um tratador de
exceções 706
exception 716-717
execução condicional de atividades 302-
304

execução condicional de diretivas do pré-processador 856
execução seqüencial 101-103
executar 61-63
executar um programa 61, 63
Exemplo de uso de arrays bidimensionais 295-296
Exemplos

repetição controlada por contador
127- 128
Repetição controlada por contador com a estrutura for 129-130

Demonstrando o gabarito de classe Stack 684-686
Demonstrando o gabarito de classe vector da Biblioteca Padrão
93 1-932
Demonstrando o operador constcast 1007-1008

Demonstrando o operador reinterpretcast 1008- 1009

Demonstrando o operador staticcast 1006-1007
Demonstrando o qualificador de tipo const 28 1-282
Demonstrando o tipo de dado primitivo bool 1004
Demonstrando o uso de
namespaces 1010-1011
Demonstrando os algoritmos mm e max 984-985
Demonstrando os gabaritos de classes adaptadoras queue da Biblioteca Padrão 953-954
Demonstrando os operadores * e
-> 1025
Demonstrando polimorfismo com a hierarquia de classes
Empioyee 6 12-613
Demonstrando
setnewhand].er 713-714 Demonstrando wap, iter_swap e
swap_ranges 97 1
Demonstrando typeid 10 13- 1014

Demonstrando um array de ponteiros para funções 353-354
Demonstrando um explicit constructor 1020-1021
Demonstrando um membro de dados mutabie 1024-1025
Demonstrando um objeto função binária 990-991
Deturpação de nomes para possibilitar a ligação segura quanto ao tipo 23 1-232
Disparando novamente uma exceção 707-708
Duas maneiras de declarar e usar funções que não recebem argumentos 221-222
Elevar uma variável ao cubo usando chamada por referência com um ponteiro como argumento 325-326
Elevar uma variável ao cubo usando chamada por valor 325-
326

Encadeando chamadas de funções membro 470-47 1
Enviando o valor de uma
expressão para a saída 643-644

Enviando um string para a saída usando duas inserções em stream 642-643
Enviando um string para a saída usando inserção em stream 641-
642

Evitando um problema de precedência entre o operador de inserção em stream e o operador condicional 646-647
Exemplo de uso de arrays bidimensionais 295-296
Exibindo valores em ponto flutuante nos formatos default do sistema, científico e fixo
662-663
Friends podem acessar membros priva te da classe 465-466
Função maximum definida pelo programador 196-197
Funções não-friendJnãomembro não podem acessar membros private 467-468
Gerando números de Fibonacci recursivamente 217-218
Gerando valores para serem colocados nos elementos de um array 265-267
Implementação do tipo de dados abstrato Time como uma classe
402-403
Implementando uma classe proxy
482-483
Imprimindo as características de um string 897-899
Imprimindo em múltiplas linhas com um único comando, usando cout 69-70
Imprimindo em uma mesma linha com comandos separados usando cout 69-70
Imprimindo o valor de uma union nos tipos de dados dos dois membros 88 1-882
Imprimindo um inteiro com espaçamento interno e forçando o sinal mais 660-661
Imprimindo um inteiro sem sinal em binário 815-816
Imprimindo um string, um caractere de cada vez, usando um ponteiro não-constante para dados constantes 330-331
Inicializando arrays
multidimensionais 294
Inicializando e usando uma variável constante corretamente
267-268
Inicializando os elementos de um array com uma declaração 265-
267

Inicializando os elementos de um array com zeros 264-265
Inteiros deslocados e ajustados a uma escala produzidos por 1 + rand() % 6 201-202
Lançando um dado de seis faces 6000 vezes 202-203

Leitura de caracteres com a função membro getline 650
Lendo e imprimindo um arquivo seqüencial 734-735
Lendo um arquivo de acesso aleatório seqüencialmente 744-
746

Manipulando uma lista encadeada
763-764
O operador sizeof quando aplicado a um nome de array retorna o número de bytes no array 336-337
Operador de extração de stream
retornando falso quando
encontra fim de arquivo 647-
648

Operadores de inserção em stream e extração de stream definidos pelo usuário 520-52 1
Ordem na qual construtores e
destruidores de classe base e
classe derivada são chamados
579-580
Ordenando um array com buhble sort 283-284
Os operadores ponteiros & e *
322-324
Passando arrays e elementos individuais de arrays para funções 279-28 1
Pesquisa binária em um array ordenado 290-291
Pesquisa linear em um array 288-
289

Pré-incrementando e pós- incrementando 126
Processando uma fila 778
Programa de análise de dados de uma pesquisa 285
Programa de classificação de uso geral usando ponteiros para
funções 350-35 1
Programa de consulta de crédito
735-736
Programa de contas bancárias 747-
748
Programa de embaralhamento e distribuição de cartas 348
Programa de impressão de texto
66-67
Programa de lançamento de dados usando arrays em vez de
switch 273-274

Programa de média da turma com repetição controlada por contador 111-112
Programa de média da turma com repetição controlada por sentinela 116-118
Programa para simular o jogo de craps 206-208
Retornando uma referência para um membro de dados
priva te 429-430
Separando a interface da implementação da classe Time
409-410
Simulação de embaralhamento e distribuição de cartas de alto desempenho 811-812
Sobrescrevendo membros de uma classe base em uma classe derivada *573-574*
Somatório com for 134-135 Tentando chamar
polimorficamente uma função herdada por herança múltipla
1026- 1027
Tentando modificar dados através de um ponteiro não-constante para dados constantes 330-33 1
Tentando modificar um ponteiro constante para dados constantes
333-334
Tentando modificar um ponteiro constante para dados nãoconstantes 332-333
Tentando utilizar uma referência não-inicializada 227-228
Tentativa errada de acessar membros private de uma classe 412
Tentativa errada de inicializar uma constante de um tipo de dado primitivo através de atribuição
457-459
Tornando aleatório o programa de lançamento de dado 204-205
Tratando arrays de caracteres como strings 275-276
Um exemplo de chamada por referência 225-226
Um exemplo de escopos 212-213 Um exemplo simples de tratamento de exceção com divisão por zero 700-701
Um objeto const precisa ser inicializado 268-269

Um programa de adição 70-71 Um programa de análise de pesquisa entre alunos 270-271 Um programa que imprime histogramas 272-273
Um programa simples com pilha 775-776

Um programa simples com pilha usando composição 777
Uma classe Array com sobrecarga de operadores 525-526

Uma classe de inteiros enormes 559-560
Uma classe de números complexos 556-557
Uma classe String com sobrecarga de operadores 536-537

Usando a estrutura do/while 143-144
Usando a função membro f iii e o manipulador setfill 660-661

Usando a função swap para permutar strings 896-897
Usando argumentos de linha de comando 87 1-872
Usando argumentos default 228-229

Usando as funções exit e atexit 874
Usando as funções membro get. put e eof 647-649
Usando atof 830-83 1
Usando atoi 830-831
Usando atol 83 1-832
Usando campos de bits para armazenar um baralho de cartas 822-823
Usando classes base virtual 1028-1029
Usando funções da Biblioteca Padrão para fazer um heapsort 98 1-982, 984
Usando funções gabarito 682-683
Usando funções set e get 425-426
Usando goto 879-8 80
Usando inicializadores de objetos membro 461-462
Usando isdigit, isalpha. isalnume **isxdigit** 826
Usando islower. isupper. tolower e toupper 826-827
Usando isspace. iscntrl, ispunct, isprint e isgraph 828-829
Usando listas de argumentos de tamanhos variáveis 869-870
Usando membro de dados static para manter uma contagem do número de objetos de uma classe 475-476
Usando memchr 84 1-842
Usando memcmp 841-842
Usando memcpy 839-840
Usando memmove 840-841
Usando memset 842-843

Usando o comando break em uma estrutura for 145-146
Usando o comando continue em uma estrutura for 145-146
Usando o indicador ias: showbase 661-662 Usando o indicador ios: :uppercase 663-664 Usando o manipulador de stream endl 642-643
Usando o operador sizeof para determinar tamanhos de tipos de dados padrão 338
Usando o operador unário de resolução de escopo 229-230
Usando o ponteiro this 468-469 Usando os manipuladores de stream **hex, oct, dec e** setbase 652-653
Usando os operadores de deslocamento sobre bits 819-820

Usando os operadores de igualdade e relacionais 79-80

Usando os operadores sobre bits
AND, OR inclusivo, OR
exclusivo e complemento 817-
818

Usando quatro métodos de
referenciar elementos de array
342-343
Usando strcat e strncat
358-359
Usando strchr 834-835
Usando strcmp e strncmp
359-360
Usando strcpy e strncpy 358
Usando strcspn 835-836
Usando strerror 843-844
Usando strlen 362-363
Usando strpbrk 836-837
Usando strrchr 836-837
Usando strspn 837-838
Usando strstr 838-839
Usando strtod 832-833
Usando strtok 36 1-362
Usando strtol 833-834
Usando strtoul 834-835
Usando tratamento de sinais 876-
877

Usando um Construtor com argumentos default 418
Usando um função mime para calcular o volume de um cubo
223-224
Usando um gabarito de função
233-234
Usando um inicializador de membro para inicializar uma constante de um tipo de dados primitivo 457-45 8

Usando um iterador para enviar um string para a saída 906-
907

Usando um objeto
ostringstreaxn alocado dinamicamente 908-909
Usando uma classe Time com
objetos const e funções
membro const 453-454, *456-*
457
Usando uma função utilitária 414-
415

Usando uma referência
inicializada 226-227
Usando uma union anônima 882- 883

Exemplos
Justificação à esquerda e
justificação à direita 659-660
exemplos com a estrutura for 132-133
exemplos de herança *565-566*
exemplos do *ObjectSpace STL Tool Kit* 1000. 1055-1056
exemplos e exercícios de recursividade no texto 220-221
exercício definitivo sobre sobrecarga de operadores 555-556
exercícios com ponteiros 385-386
<exception> 199-201, 706-709, 7 16-7 17
Exibindo valores em ponto flutuante nos formatos default do sistema

científico e fixo 662-663
exit42l-422, 699-700, 7 13-714, 732- 733, 874
EXITFAILURE 874
EXITSUCCESS 874
exp 192-193
expandir uma macro 857-858
expansão de macro 858-859
expiicit 1020-1023
“explosão” exponencial de chamadas
2 19-220
expoente 562
exponenciação 77-78, 135-136
expressão 104-105. 191-193
expressão algébrica 75-76
expressão aritmética infixa 79 1-792
expressão com ponteiros 338-339, 341- 342

expressão condicional 105-107, 646- 647, 703-704
expressão controladora 140-141
expressão de coerção 859-860
expressão pós-fixa 792-793, 803-805
expressões intensivas em operadores
535-536
Extended Binary Coded Decimal Interchange Cade (EBCDIC) 360- 361
extensão de sinal 142-143, 814-815

extensibilidade 608,610-611,697,928- 929
extensibilidade da STL 920-921
extensibilidade de C++ 522-523
extensões de nomes de arquivos 61, 63
extern “C” 883-884
extern 208-211, 872-873
extração de *stream* 79-80
extrair características comuns 595-597

F

fabs 192-193
“faça seu ponto” 206-208
faiibit 645-646,650-651,732-733

fala 639
“falando de objetos” 8 1-82, 93-94, 396
falando para um computador 53-54
falha de new 697-698, 711-712
falhas não-recuperáveis 665-666
faise77-78, 102-106, 109-110, 219-
220, 1004
FAQ sobre bibliotecas de C++ 105 1-
1052
`FAQs 1051-1053`
FAQs sobre a STL 1000-1001, 1055-
1056
fase de análise 366
fase de análise orientada a objetos 159-
160
fase de compilação 68-69
fase de edição 63
fase de inicialização 114-115
fase de processamento 114-115
fase de término 114-115

`fases 114-115`
fator de escala 20 1-202, 205-206
faturação prima 1000
fatorial 182-183, 214-217

fatorial de um inteiro não-negativo 182-
183

"faxina" de término 420-42 1
"fazer AND com" 816-817
"fim de entrada de dados" 114-115
"formato universal" 398-399
fazendo coerção 340-341
Fazendo coerção de ponteiros para classe base para ponteiros para classe `derivada 568-569`
fazendo coerção para baixo de um ponteiro 567-568

fazendo referência a um objeto de classe base com um ponteiro de classe base 582-583
fazendo referência a um objeto de classe base com um ponteiro de classe derivada 582-5 83
fazendo referência a um objeto de classe derivada com um ponteiro de classe base 582-583

fazendo referência a um objeto de classe derivada com um ponteiro de classe derivada 582-583
fazendo *spool* 778
fazendo *upcasting* de um ponteiro 568-
569

fechar um *stream* 874
ferramenta de depuração 478-479, 861- 862

ferramenta de desenvolvimento de
programas 104-105, 124-125
ferramentas para C++ 1050
FIFO 481-482, 778, 919-920, 940-941,
952-953
fila 481-482, 760-763, 773-774, 778
fila com duas extremidades 940-94!
fila com prioridades 917-918
fila cresce infinitamente 793-794
fila de caixa em um supermercado 778,
793-794
fila de espera 48 1-482
fila em uma rede de computadores 778
filho 780-78 1
filho da direita 780-781
filho da esquerda 780-78 1
fiji 928-929, 955-956
filln 928-929, 955-95 6
fim de arquivo 139-141, 356-357, 647-
648, 665-666, 733-734
fim de linha 72-73
fim de main 68-69
fim de um *stream* `735-736`
fim de um *string* `760-761`
fim de uma fila 760, 778
fim de uma lista 805-806
fim de uma seqüência 968, 970
finalidade do programa 66-67
find900-90l, 902-903, 927-930, 942-
943, 945-946, 968, 970
findeach 929-930
findend 929-930
findfirstnotof 900-901
findfirstof 900-901, 929-930
firidif 929-930, 968-970
findiastnotof 900-90!
firidlastof 900-901
fins 201-202
first 945-947
fiag 656-657
fioat 70-71, 118-119, 199-200,875-
876

<fioat h> 200-201
floatfieid 662-663, 665-666
floor 192-193
flush 642-643
fluxo de controle 81-82, 109-110,698-
699

fluxo de controle de uma chamada a função virtual 631

função operador de coerção 535-536 função operador de coerção
sobrecarregada 535-536
função operador de extração de *stream* operator» sobrecarregada 543-
544
função operator« sobrecarregada
571-573, 588-589
função palindrome 1000
função pow 77-78, 135-137, 192-193
função predicado 413-4 14, 770, 939- 940, 959-960, 962-965, 967-970,
973-974, 976-978, 981,983-985
função predicado binária 939-940, 959- 960, 967-970, 973-974, 977-978,
981, 983-986
função predicado isFull 413-414
função predicado unária 939-940, 962- 965
função qualityPoints 254-255
função que chama a si mesma 214
função que não recebe argumentos 22!-
222
função raise 875-876
função rand 199-202, 311
função recursiva 214, 2 16-217, 763-764
função rollDice 206-208
função set4l3-4l4, 424-429, 462-465
função smallest 248-249
função sobrecarregada 229-231. 680- 683
função sobrescrita 610-611
função sobrescrita chama versão da classe base 575-577
função sqrt da biblioteca de
matemática 198-199
função square 199-200
função static 534-535
função swap 334-337
função time 205-206
função toupper 329-330
ftinção tripleByRefererice 259- 260
função tripleCallByValue 259- 260
função utilitária 413-414, 489-490, 502- 504
função utilitária recursiva 785-786
função virtual 583-584, 608-6 12,
620-622, 624-625, 629-632, 880-881,
921-922, 955, 1014-1017, 1027-1028 função virtual "impura" 622, 624-
625
função virtual chamada de uma referência à classe base 620-621
função virtual em uma classe base
620-621
função virtual pura 609-612, 615, 6 19-620, 622, 624-625, 629-630
funções "pré-empacotadas" 190
funções com listas de parâmetros vazias
221-222

funções como blocos de construção
192- 193
funções contêiner da STL 92 1-922
funções da biblioteca de matemática
135-136, 190-193
funções da biblioteca de tratamento de caracteres 825
funções da biblioteca padrão ANSI
105 1-1052
funções de entrada/saída da biblioteca
200-201
funções de memória da biblioteca de tratamento de *strings* 839-840

funções de pesquisa da biblioteca de tratamento de *strings* 834-835
funções de union não podem ser virtual 880-88 1
funções devem ser pequenas 194-196
funções f ind 898-899, 900-901
funções f ind de string 898-899
funções friend para melhorar o desempenho 462-465
funções insert 903-904
funções insert de string 902-904
funções membro 83-84
funções membro de uma classe 404-405
funções membro normalmente public
402-403
funções membro normalmente públicas
402-403
funções membro que não recebem argumentos 405-406
funções não-friendlnão-membro não podem acessar membros private
467-468
funções não-virtual em uma classe abstrata 622, 624-625
funções operador SI 9-5 20
funções operador de atribuição 533-534
funções padronizadas 192-193
funções para manipular dados nos
contêineres da biblioteca padrão 199-
201
funções personalizadas 193-194
funções public 404-405
<functional> 200-201, 989-990, 992

G

gabarito 680, 760, 763-764, 774-775,
857-858
gabarito autoytr 714-715
gabarito de classe 680, 683-684, 763- 764, 785-786
gabarito de classe adaptadora queue
953-954
gabarito de classe autoytr 714-715 gabarito de classe List 770, 774-775, 777, 778
gabarito de classe lista encadeada 805-
806

gabarito de classe pilha 774-775, 805-
806
gabarito de classe Stack 684-685,
689-690, 777
gabarito de classe Tree 785-786
gabarito de classe vector 930-931
gabarito de função 23 3-234
gabarito de função max 260
gabarito de função mm 260
gabarito de função printArray 680-
681
gabaritos de função 230-23 3, 680-684
gabaritos *efriends* 690-691
gabaritos e herança 690-691
gcount 650-65 1
generalidades 610-611
generate 928-929, 955-958
generate_n 928-929. 955-958
geração de números aleatórios 164-165,
273-274
gerador de jogos de palavras cru7adas
394
gerador de números aleatórios 488-490
gerando números de Fibonacci
recursivamente 217-218
Gerando valores para serem colocados
nos elementos de um array 265-267
gerenciador da tela 611-612, 621-622
gerenciamento dinâmico de objetos
484-485
gerente 185-186
get4l2-413, 423-424
getc 858-859
getline 356-357, 650, 890-891
gosub 804-805
gráfico de barras 184-185, 272-273
gráficos de tartaruga 312-313
gráficos orientados a objetos 1051-1052
grafo 184-185

gratuito 1054-1055
greater 989-990
greaterequal 989-990
Green Eggs Report 1052-1053
guillemets **(« »)** 302-304
H
handle432-436, 504-505, 571-573
handie de objeto 407 -408, 478
handie de referência 437
handie implícito 406-407
handie para ponteiro 406-407
handie para um nome 406-407, 478
handie para um objeto 406-407
handier captura todas 705
handier catch 702-703, 705, 706,
710-7 12
handier catch para um grupo de
exceções 705
handies 433-434
hardware 50-53
heap 954-955, 98 1-984

`heapsort` 954-95 5, 981, 983-984
help-site.com 1050
herança 82-83, 163, 40 1-402, 406-408,
411-412, 441-442, 452, 462-465,
564-566, 57 1-573, 585, 595-597,
621-622, 680, 803, 955
herança de classe base virtual 1028-

1029

herança de implementação 622, 624-

625

herança de interface 622, 624-625 herança de uma classe base abstrata
611-612
herança diamante 1026
herança múltipla 82-83, 564-567, 588- 589, 59 1-595, 639-640, 1026
herança para explorar as características comuns entre classes 366
herança private 564-565, 567-568, 57 1-573, 577-578, 774-775
herança private como uma forma de composição 577-578
herança protected 564-565, 567- 568, 577-579
herança public 564-565, 567-568, 582-585, 587. 589-590, 615, 619-
620, 616, 708-709, 1005-1006 herança simples 564-565. 594-595,

1026

herança virtual 1028-1029
herda membros de dados e funções membro de uma classe base 564
Hewlett-Packard 917-918
hex 651-652, 665-666
hexadecimal 184-185, 641-642, 661- 664, 833-834
hierarquia 577-579, 611-612
hierarquia de categorias de iteradores
926-927
hierarquia de classes 577-579,611-612, 62 1-622, 635-636, 803
hierarquia de classes de *EIS* em *stream*
729-730
hierarquia de classes de exceção 716-

717

hierarquia de classes Employee 612-

613

hierarquia de classes Shape 104-105,
624-627
hierarquia de dados 727-7 29
hierarquia de exceções da biblioteca padrão 716-7 17
hierarquia de formas 606
hierarquia de herança 566-567, 573- 574, 589-590, 605-606, 608-609,
611-612, 1016-1017
hierarquia de herança polimórfica 1016-

1017

hierarquia de promoção para tipos de dados primitivos 199-200

hierarquia padrão exception 716-

717

hierarquia Shape 566-567
hierarquias no mundo real 566-567
hipotenusa 185-186
histograma 184-185, 272-273, 286-288
homepage do *Microsoft Visual C++*
1054- 1055
homepage do *Visual* **C++** 1054-1055
hora de calendário 205-206
hora em que o arquivo-fonte foi
compilado 86 1-862
horário normal 178-179

IBM 54-55
IBM Coi-poration 57-58
identificador(es) 70-7 1, 102-104,211- 212
identificadores globais 1009-1010
identificadores para nomes de variáveis
208-209
identificar as classes 160
ifstreain640-64l, 728-731, 733- 735, 744-746, 87 1-872
#if 859-860
#include 856
#include “nomeDeArquivo” 856
#include <cstring> 873-874
#include <iomanip> 119-120,
657-658
#include <iostream> 66-67
ignorando espaço em branco 651-652,
657-658
ignorar o caractere de retorno *(enter)*
485-486
ignore 522-523, 650-651
igual a 77-7 8
imagem executável 61, 63
imagem gráfica 639
implementação 301, 409-410
implementação de uma classe 404-405, 408-409, 412
implementação de uma função 615.
6 19-620
Implementação do tipo de dado abstrato Time como uma classe 402-403
Implementando uma classe *proxy* 482-
483

implicitamente virtual 608-609 imprecisão de números em ponto
flutuante 135-136
impressora 63, 639, 778 imprimindo caractere diferente de

espaço 825
imprimindo caractere diferente de espaço, dígito e letra 827-828
imprimindo características de um string 897-899
imprimindo datas 39 1-392

imprimindo em múltiplas linhas com um único comando, usando cout

69-70

imprimindo em uma linha com comandos separados usando cout

69-70
imprimindo o valor de uma union nos dois tipos de dados do membro 881-

882

imprimindo recursivamente do fim para o início dados udos do teclado 220-

221

imprimindo um inteiro com espaçamento intemo e forçando o sinal mais 660-661
imprimindo um inteiro sem sinal em binário 8 15-816
imprimindo um *string,* um caractere de cada vez, usando um ponteiro nãoconstante para dados constantes 330-

331

imprimindo uma árvore 795
imprimindo uma árvore binária em um formato de árvore bidimensional
787-788
imprimir um array do fim para o início recursivamente 220-221
imprimir um array recursivamente 220-
221, 317-318
imprimir um *string* do fim para o início recursivamente 220-221, 317-318
imprimir um *string* lido do teclado do
fim para o início recursivamente 220-

221

imprimir uma linha de texto 66-67 imprimir uma lista do fim para o início
793-794
imprimir uma lista do fim para o início recursivamente 793-794
imprimir uma lista encadeada do fim para o início 763-764
imprimir uma lista encadeada do fim para o início recursivamente 220-221
includes 976
incluir um arquivo de cabeçalho 408-
409

incluir um arquivo de cabeçalho múltiplas vezes 411-412
inclusão circular 43 8-439
incrementar o contador de instruções
800-801
incrementar um iterador 927-928
incrementar um ponteiro 338-339
incremento de uma variável de controle
13 1-133
incremento ou decremento de uma variável de controle 126 127
indentação 68-69, 80-82, 103-108. 129- 130

independente de implementação 412,
445-446

indicador 798-799
indicador de fim de arquivo 647-649,
732-734, 87 1-872
indicador de justificação 660-661
indicador dec 657-658, 66 1-662, 665- 666
indicador fixed *657-658,* 662-663,
665-666
indicador hex 657-65 8
indicador internal 657-660, 665- 666
indicador ios: : Left 348, 657-658,
665-666
indicador ios: : scientific 657- 658, 662-663, 665-666
indicador ios: skipws 657-658 indicador ios: : uppercase 657- 658, 663-664
indicador left *657-058,* 658-059, 660- 661. 665-666
indicador oct 657-658, 66 1-662, 665- 666
indicador right 657-661, 665-666 indicador scientific 657-658, 662- 663, 665-666
indicador showbase 657-658, 661- 662
indicador showpos 657-658, 660-661,
664-665
indicador skipws 657-658
indicador uppercase 657-658, 661- 664
indicadores antes de serem desligados
665-666
indicadores de erro 650-651
indicadores de formato 65 1-652, 656- 657, 663-665
indicadores de justificação internal
660-661
indireção 320-321
inft)rmação de grafo 272-273
informação sobre tipo 750-751
informação sobre tipo durante a execução (RTTI) 199-201, 564, 568- 569, 1004, 1013-1014
inicialização da largura 654-655 inicialização de array automático 276-
277
inicialização de array static 275-277
inicialização de membros 499-500
Inicialização e uso corretos de uma variável constante 267-268
inicializador = O (em classes virtuais puras) 609-610
inicializador 265-267, 417, 420, 577- 579
inicializador de classe base 577-579 inicializador de membro 453-454, 456- 459, 533-534, 543-544, 57 1-573,
575-579, 58 1-582

inicializador de membro para um
membro de dados const 459-460
inicializador de objeto membro 461-465
inicializadores default 4 19-420
Inicializando arrays multidimensionais
294
"inicializando duplamente" objetos membros inicializados 462-465
inicializando objetos de classe 413-414, 4 16-417, 446-447
Inicializando os elementos de um array com uma declaração 265-267
Inicializando os elementos de uni array com zeros 264-265
inicializando um ponteiro declarado const 332-333
inicializar com um comando de atribuição 457-459
inicializar com zero 110-111
inicializar em um estado consistente
4 17,420
inicializar o valor de um membro de dados private 413-414,446-447
inicializar os membros da classe base de uma classe derivada 5 77-579
inicializar ponteiro com O (nulo) 770
inicializar um ponteiro 32 1-322
inicializar uma constante de um tipo de dado primitivo 457-45 8
inicializar uma estrutura 809-8 10
inicializar variável static em escopo de arquivo 475-476
início de um arquivo 735
início de um *stream* 73 5-736
início de uma fila 48 1-482, 760, 778
mime 222-223, 405-406, 5 19-520, 534-535, 545. 858-859, 989-990
inneryroduct 929-930, 984-985
inOrderTraversal 786-787
inplace_merge 974-975
input 798-799
inserção 482-483, 760
inserção em lista encadeada com recursividade 220-221
inserção em *stream* 67-68, 69-70
inserção no fim de um vector 929-

930
inserção recursiva em árvore binária
220-22 1
inserir a posição de memória na pilha
799-800
insert 902-904, 936, 945-946, 948- 949
instanciado 83-84, 396-397
instrução 6 1-63
instrução assistida por computador
(CAI) 254-255
instrução branch **zero** da SML 800- 801
instrução de desvio incondicional 803 instrução desviar se zero 799-802

instrução halt em SML 803
instrução ilegal 875-876
int *& 224-225, 226-227
int * 326-327
int * const 326-327
int6l-68, 70-71, 198-199
int& 226-227
integridade da classe base preservada
por herança 583-584
integridade de uma estrutura de dados
interna 481-482
inteiro 67-68, 70-71. 18 1-182
inteiro binário 18 1-182
inteiro criptografado 182-183
inteiro ímpar 182-183
inteiro par 182-183
inteiro sem sinal em binário 815-8 16
inteiros aleatórios no intervalo de 1 a 6
20 1-202
inteiros começando com O (octal) 661-
662
inteiros começando com **Ox** ou OX
(hexadecimal) 66 1-662
inteiros deslocados e ajustados a uma
escala 202-203
inteiros deslocados e ajustados a uma
escala produzidos por 1 + rand O
% 6 201-202
inteiros hexadecimais 322-324
interações entre objetos 365
intercalar dois objetos lista ordenados
790-79 1
intercambiabilidade de arrays e
ponteiros 343-344
interface 396-397, 404-405, 408-4 lO, 6 11-612
interface de uma classe 40 1-402, 404-
405
interface permanece a mesma 428-429
interface protected 564-565
interface public 404-405, 408-409, 412-413,570-572,587,611-612
interface que pode ser herdada 609-610, 622, 624-625
interfaces 83-84
*International Standards Organiation
(ISO)* 50-51, 56, 481-482
lnternet 5 1-52, 63-66
Internetworking Protocol (1P) 63-64
interpretador de Simple 805-806
interrupção 875-876
intervalo 968, 970
intervalo arbitrário de subscritos 480-
481
intervalos aleatórios 793-794
Intranets 57-61, 63-64
introdução à STL na Web 1000. 1055-
1056
invalidargument 7 16-717

invocando uma função membro nãoconst sobre um objeto const 452-
453

invocar uma função 190-191, 193-194
ios 1026
ios: adjustfield 660-663, *665-*

666
ios: :app73O-732
ios: :ate73l-732
ios: badbit 665-666
ios: beg 735-736
ios: :binary 73 1-732. 744-746
ios: : cur 735-736
ios: dec *657-658,* 66 1-662, 665-666
ios: end *735-736*
ios: : eofbit 665-666
ios: : failbit 665-666
ios: :fixed 119-120, 137, 657-658,
662-663. 665-666
ios: floatfield 662-663, 665-

666

ios: goodbit 661-662, 665-666
ios: :hex657-658, 661-662
ios: : in 73 1-735, 746-747
ios: : internal 657-658, 665-666
ios: : oct 657-658, 661-662, 665-666
ios: : out 731-732, 746-747
ios: : right 348, 657-658, 665-666
ios: : showbase 657-662
ios: :showpoint 119-120, 137,
657-658, 664-665
ios: showpos 657-658, 660-661,
664-665
ios: trunc 73 1-732
iostreain 728-729
IP 63-64
isalnum 825
isalpha 825
iscntrl 825. 827-828
isdigit 825-827
isgraph 825, 827-828
islower 329, 825-827
ISO 50-51
isprint 825, 827-828
ispunct 825, 827-828
isspace 825, 827-828
istreajuiterator 923-925
istringstream 907-910
isupper 825-827
ISV 621-622
isxdigit 825
<iomanip . h> 200-20 1
<iomanip> 200-20 1
<iostream.h> 200-201
<iterator> 200-201, 974-976
iterswap 928-929, 971
iteração 2 19-220
iterador “após o fim” 967 -968
iterador482-483, 680-68 1, 906-908
iterador apontando para o primeiro
elemento após o fim do contêiner
923-924

iterador bidirecional 925-927, 930-931, 937, 943-944, 946, 947-948, 973-
976, 985
iterador de acesso aleatório 925-927, 930-93 1, 933-934, 940-941, 943-944,
959-960, 967-970, 976, 981-986 iterador de entrada 925-927, 959-960,
962-963, 965-968, 973-974, 976-978,
985-986
iterador de saída 925-927, 956-957, 965-966, 976-978, 985-986
iterador do *stream* de entrada 923-924
iterador de *stream* de saída 923-924

iterador end 927-928
iterador para a frente 925-926, 930-931, 964-965, 97 1-972, 974-975, 985-986
iterador para o próximo elemento de um contêiner 923-924
iterador typedef 926-927
iteradores de comparação 927
iterativo 214-215
iterator 917-918, 920-924, 927,
945-946

J

Jacobson, Ivar 84
Jacopini, G. 101-103, 154-155
japonês 480-48 1
Java 5 1-52, 57-60

Java Como Programar: Terceira Edição
57-58
jogando 199-201
jogando cartas 345-346
jogo da forca 914-915
jogo de "adivinhar o número" 255-256
jogo de azar 206-208
jogo de *craps* 206-209
jogo de dados 206-208
jogos de cartas 345-346
juros compostos 134-135, 183-186, 188
juros compostos usando for 134-135
juros sobre depósito 186, 188
justificação à direita 137, 348, 657-660
justificação à esquerda 105-106, 137, 348, 658-660
justificação à esquerda e justificação à
direita 659-660

K

Kernighan and Ritchie C 56 *KIS ("mantenha-o simples" - Keep It Simple)* 64-65
Koenig, A. 697

L

L sufixo de inteiro 875-876
1 sufixo de inteiro 875-876
L sufixo de ponto flutuante 875-876
1 sufixo de ponto flutuante 875-876
Lobo ratory for Computer Science 63-64
laço 109-111, 115

laço aninhado dentro de um laço 121-
122

laço controlado por contador 113-114,
120- 122
laço controlado por sentinela 117-118,
382-383, 796-79]
laço de contagem 128-129
laço de retardo 132-133
laço de simulação 30 1-302
laço infinito 109-110, 118-119, 131132, 143-144, 181-182. 216-217
lado esquerdo de uma atribuição 149-
152, 262-263, 428-429, 53 1-532
lados de um quadrado 185-186
lados de um triângulo 182-183
lados de um triângulo retângulo 182-

183

lançamento de moeda 20 1-202, 254- 255
lançando dois dados 206-208, 311
lançando um dado 202-203
Lançando um dado de seis faces 6000 vezes 202-203
“lapidar classes valiosas” 83-84
lapidando classes valiosas 405-406
largura de campo 137, 263-264, 641- 642,651-652,654-655
largura de um campo de bits 822-823 largura de um intervalo de números aleatórios 205-206
largura inicializada implicitamente com
O 654-65 5
Lee, Meng 917-918
legibilidade 66-67, 103-104, 121-122, 193- 194
leiaute não-contíguo de uma fila duplamente encadeada na memória
940-941
Leis de DeMorgan 185-186 leitura a partir de um *string* na memória

201
leitura de arquivo 640-641 Leitura de caracteres com a função
membro getline 650
leitura não-destrutiva 73-74 lendo de strings na memória 907-

908

Lendo e imprimindo um arquivo
seqüencial 734-735
Lendo seqüencialmente um arquivo de
acesso aleatório 744-746
length 898-899
length_error 716-717, 890-891
ler dados 63
ler dados de um arquivo
seqüencialmente 733-734
ler uma linha de texto 650
ler uma linha de texto inteira para um
array 356-357
less 989-990
lessequal 989-990

162-

loglO 192-193
log2n 7 87-788
logaritmo 192-193
logaritmo natural 192-193
logic_error 7 16-717
lógica de desvio 608-609
lógica de switch 142-143, 608, 632- 633

loja de vendas pelo correio 184-185
long 142-143, 198-199, *875-876*
long double 199-200, 875-876
long int 142-143, 199-200, 216-217
losango - símbolo de decisão 102-105, 109-110. 132-133
losango 96-97, 101-103, 105-106, 163, 237-238
losango cheio 162-163
lote 5 3-54

less< double >946-947
less< int > 943-945, 947-948
letra 354-355
letra minúscula 825-827
letras maiúsculas (A-Z) 826-827
letras maiúsculas 70-71, 96-97, 200-201, 329, 825
letras minúsculas 70-71, 96-97, 109-110, 200-201, 329, 727-729
lexicographical compare 957-960
liberar memória 714-715, 761-762
licença 583-584
licenciamento 431-432
LIFO 479-480, 683-684, 686-688, 774-775, 9 19-920, 950-951
ligação 61, 63, 208-209, 874, 1012-1013
ligação externa 873-874
ligação interna 873-874
ligação segura quanto ao tipo 230-231, 883-884
limerick 389-390
limite de alinhamento dependente de máquina 761-762
limite de crédito em uma conta de cartão de crédito 177-178
limite de palavra 809-810
limite de palavra dupla 809-810
limite de uma unidade de armazenamento 24-825
limite de unidade de memória 824-825
limites de array 272-273
limites de tamanho de ponto flutuante 200-201
limites de tamanho integrais 200-201
limites de tipos de dados numéricos 199-201
<lirnits . h> 200-201
<limits> 199-201
<list> 200-201, 92 1-922
linguagem C++ 65-66
linguagem de máquina 55-56, 209-210, 800-801
linguagem de programação 55-56
linguagem de programação C++ 59-60
linguagem extensível 216-217, 272-273, 402-403, 480-481
linguagem híbrida 56-57, 466-467
linguagem Logo 312-313
linguagem natural de um computador 55-56
linguagem portável 65-66
linguagem sem tipos 56
linguagem simbólica 55-56
linguagem Simple 796
linguagens de alto nível 55-56
linguagens de programação procedurais 83-84, 396-397
linguagens orientadas a objetos 59-60
linha 76-77
linha cheia com uma ponta de seta 237-238
linha com uma ponta de seta cheia 301-302
linha de comando 868
linha de comando do UNIX 61, 63, 868-869
linha de comunicação com um arquivo 73 1-732, 734-735
linha de fluxo 101-105
linha de texto 650
linha de vida 30 1-304
linha em branco 71-72, 116
linha reta 76-77
linhas 292-293
linhas de comunicação dedicadas 63-64
link 61-63, 408-409, 760-761, 780-781
link com ponteiro 762-763
links 164
links do editor de ligação 61, 63
list 917-920, 922-923, 926-927, 929-931 List< STACKTYPE > 777
lista 760-761
lista circular duplamente encadeada 773-774
lista circular simplesmente encadeada 773-774
lista de argumentos de tamanho variável 869-870
lista de argumentos separados por vírgulas 191-193
lista de classes base separadas por vírgulas 594-595
lista de disparo 708-709
lista de inicializadores 265-267, 355-356
lista de inicializadores de array 265-267
lista de inicializadores de membros 460-461,486-487,594-595
lista de inicializadores separados por vírgulas 809-8 10
lista de parâmetros 194-195
lista de parâmetros de função vazia 222-223
lista de parâmetros void 194-195
lista duplamente encadeada 773-774, 9 19-920, 937
lista encadeada 482-483, 760, 762-765, 767-768, 773-774
lista encadeada de objetos de classe base 564-565
lista indexada 806
lista separada por vírgulas 80-81, 131-132, 191-195, 320-321, 456-457
lista simplesmente encadeada 773-774
literal 7 1-72
livros sobre C++ 1050
<locale> 199-201
log 192-193
Lovelacc. Lady Ada 58-59
lowerbound978-981, 983-984
Ivalue ("valor da esquerda") 126-127, 150-152, 226-227, 262-263, 321-324, 428-429, 53 1-532, 534-535, 545-546, 927, 942-943
Ivalues como *rvalues* 150-152

M

Macintosh 654-655, 732-733
macro 199-200, 680-681. 868-869, 870-871
macros definidas no cabeçalho <cstdarg> 868-869
magnitude 660-661
magnitude justificada à direita 657-658
main 66-73, 193-194, 42 1-422, 874
maior elemento de uma coleção 556-557
maior inteiro não maior do que 192-193
maior que 77-78
maior que ou igual a 77-78
makeheap 981, 983-984
makefile 874
malloc 471, 473-475. 877-878
Manhattan 186. 188
manipulação de bits 813-814
manipulação de campos de bits 824-825
manipulação de caracteres 190-191
manipulação de ponteiro perigosa 629-630
manipulação de ponteiros 629-630, 760
manipulação de *strings* 190-191. 801-802
manipulação de *strings* de caracteres 824-825
manipuladorboolalpha 1005-1006
manipulador de *stream* 72-73. 119-120, 137, 65 1-652, 655-657, 660-661, 735
manipulador de *srream* dec 65 1-652
manipulador de *stream* não parametrizado 119-120

multiprogramação *54-55*
multiset 9 19-922, 926-927, 942- 943, 945-947
multiset< **iflt,** less< int»
944-945
mu/titasking 58-59
multithreading 698-699
Musser, David 917-918
mutable 208-209, 460-461, 1023- 1024

mutuamente exclusivo 660-66 1

N

nainespace 1009-1010, 1012-1013
nainespace aninhado 1012-1013
namespace global 1012-1013
namespace. 1010-1011
namespaces predefinidos 1010-1012
não pode ser alocado espaço 878-879
não-igual 77-78
NDEBUG 861-862
nega te 990-991
new 471,473-475, 485, 533-534, 761-763, 877-878, 936
<new.h>713
<new>7l1-713
new chama o construtor 471, 473-474
new disparando bad alloc em caso de falha 712
new disparando uma exceção se a memória disponível é insuficiente 575-577
new falha 710-711
new falha na alocação de memória 699-700

new retornando O em caso de falha 711-713

new teve sucesso 543-545
newhandler 699-700
nível de “hits e bytes” 813-814
nível de indentação 105-106
nível mais alto de precedência 75-76
nodo 762-763
nodo de rede 77X
nodo esquerdo 785-7 86
nodo filho 794-795
nodo folha 793-794
nodo mais à direita de uma subárvore 794-795
nodo pai 780-781, 793-794
nodo raiz 780-781, 785-786
nodo rai7 da subárvore direita 780-781
nodo raiz da subárvore esquerda 780-781
nodo substituto 794-795
nome com subscrito usado como um *rvalue* 53 1-532
nome de arquivo 730-731, 733-734, 861-862
nome de array 340-342

nome de array como um ponteiro 324-325

nome de array como um ponteiro constante 340-342
nome de array como um ponteiro constante para o início do array 331-

nome de atributo 236
nome de estrutura 808, 808-809
nome de função 190-193, 210-211,349-350, 873-874
nome de parâmetro em um tratador catch 703-704
nome de um arquivo-fonte 86 1-862
nome de um array 263-264, 324-325
nome de um elemento de array com subscrito 278-279
nome de uma atividade 237-23 8
nome de uma função 191-193
nome de uma variável 73-74, 208-209
nome de uma variável de controle 126-127
nome de variável 796
nome de variável de controle 13 1-132
nome de variável global 873-874
nomes de função adulterados 230-231
nomes de operações 299-300
nomes de parâmetros em protótipos de funções para documentação 197-198
nomes de variáveis em protótipos de função 279-281
nomes significativos 194-196
not 1016-1017
NOTlógico(!) 146-147, 185-186, 1016- 1017
noteq 1016-1017
notequalto 990-991
notação científica 119-120,641-642
notação com colchetes duplos para arrays bidimensionais 545-546
notação de array 341-342
notação de ponteiro 34 1-342
notação de ponto fixo 657-658
notação fixa 641-642
notação hexadecimal 641-642
notação infixa 79 1-792
notação ponteiro/deslocamento 341-342
notação ponteiro/subscrito 341-342
notação pós-fixa 79 1-792
nothrow 713
nothrowt 713
nova linha (**'** \n') 68-69, 72-73, 80-8 1, 354-355, 642-646, 827-828
nova linha 68-69
novo bloco de memória 878-879
novo estilo de coerções 1005-1006
novos manipuladores de *stream* 655-656
novos operadores de coerção 1006-1007
nthelement 985
NULL 32 1-322, 832-835
nulo (O) 357-358. 760-761

número binário 825
número correto de argumentos 194-195
número da posição 262
número de argumentos 194-195
número de elementos em um array 337-338
número de linha 796, 798-800, 86 1-862
número de ponto flutuante 113-114, 116, 118-119, 120-121
número de ponto flutuante em formato científico 662-663
número do sinal 875-876
número hexadecimal (base 16) 644-645, 65 1-652, 657-658, 66 1-662, 832-833
número ímpar 186, 188
número não-especificado de argumentos 868-869
número octal 641-642, *66* 1-662, 825, 833-834
número perfeito 254-255
número primo 987-988
número real 116
números aleatórios 203-204, 273-274

números complexos 448-449. 556-557
números de Fibonacci 2 19-220
números decimais 184-185, 661-662, 825

números inteiros 70-7 1
números mágicos 269-270
números pseudo-aleatórios 203-204

O

o arquivo-fonte de data é compilado
86 1-862
o encontro de caminhos de decisão 238
o menor dentre vários inteiros 183-184
O operador sizeof quando aplicado a um nome de array retorna o número de bytes no array 336-337
o resultado do deslocamento de um valor com sinal para a direita é
dependente de máquina 820-82 1
Object Constraint Language (OCL) 164
Object Management Group (OMG) 84
objeto *5* 1-52, 56-57, 59-60, 83-84, 155- 156, 396-397
objeto autoptr fica fora de escopo
7 14-7 15
objeto automático 699-700, 710-711
objeto automático local 423-424
objeto cliente 298-299
objeto const 268-269, 398-399
objeto const deve ser inicializado
268-269
objeto da classe derivada é um objeto da classe base 57 1-573, 582-583
objeto de classe derivada 705
objeto de entrada em *stream* (cm) 70-
72

objeto de saída padrão (cout) 67-68. 639-640, 728-729

objeto disparado 703-705

objeto do *stream* de entrada padrão 728-
729

objeto exceção 702-703
objeto exceção de divisão por iero 702-
703
objeto fila 793-794
objeto função 943-944, 947-948, 989- 992

objeto função binária 990-991
objeto função de comparação 943-944,
947-948

objeto função de comparação less
943-944. 954-955

objeto função less< int **>** 943-945 objeto função less< T **>** 947-948,
954-955

objeto função pode encapsular dados
992

objeto grande 225-226
objeto hospedeiro 460-461
objeto interage 396
objeto iterador482-483, 918-919
objeto local automático 421-422
objeto local static 42 1-424
objeto que envia 298-299
objeto que recebe 298-299
objeto sai do escopo 420-421
objeto sem nome 88 1-882
objeto servidor 298-299
objeto Stririg temporário 544-545
objeto temporário 535-536
objetos 82-83
objetos automáticos em um bloco try
706

objetos componentes 437
objetos contêm somente dados 406-407
objetos criados dinamicamente 435-436
objetos da classe derivada tratados como objetos da classe base 582-583
objetos de superclasse duplicados 1026-
1027
Observações de engenharia de software
60-61
obtém o valor de 79-80
obter o valor de um membro de dados private 4 12-413
obter ponteiro 735
oct 65 1-652
octal 184-185, 66 1-662
ocultação 462-465
ocultação de informações 83-84,211- 212, 334-335, 396-397, 404, 413-
414.479-480

ocultando a implementação 405-406 ocultando detalhes de implementação
190-191

ocultar a implementação 479-480, 484- 485
ocultar dados private dos clientes
408-409

ocultar detalhes de implementação 479-
480

ocultar nomes em escopos mais
externos 211-212
ocultar uma representação de dados
interna 481-482
ofstream728-735, 742-746, 87 1-872
Oito Rainhas 3 16-317
Oito Rainhas com recursividade 220-
221, 317-318
Oito Rainhas: abordagens de “força
bruta” 316-317
“olho de boi” 237-238
OMG 84
OOAD 84
OOD 8 1-82
OOP 82-84
opção de compilador 65-66
opções combinadas por ou 656-65 7
opções de menu 746-747
operação com *stream* falhou 665-666
operação de enfileirar 481-482
operação de retirada da fila 481-482
operação de sincronização de um
istream e um *o,streo,fl* 667-668
operações 16 1-162, 234-235, 298-299, 30 1-302, 363-364

operações com iterador de acesso aleatório 927-928
operações com iteradores bidirecionais
927-928

operações com iteradores para a frente
927

operações de classes 366-367
operações de entradalsaída 101-103
operações de iterador 927
operações implementadas como funções
234-235
operações permitidas sobre dados 480-
481

operador **&&** 148-149
operador [] sobrecarregado 531-532
operador **+** 985
operador **+=** 893-894
operador **+** sobrecarregado 550-55 1
operador **<** 985-986
operador « 1004-1005
operador « sobrecarregado 519-520, 545, 64 1-642
operador 67-68, 124-125
operador- 922-923
operadorAND sobre bits (&) 814-819, 851
operador AND sobre bits com atribuição
(&=) 820-821
operador auto_ptr sobrecarregado *
716-717

operador autoytr sobrecarregado -
>715-716
operador binário **+** 74-75
operador binário 72-75, 148-149

operador binário de resolução de escopo
(: :) 228-229. 404-405, 475-476, 594-595, 1009-1010, 1026 operador condicional (?:) 105-106 operador condicional ternário (?:) 219,
5 18-5 19

operador const cast 1007-1008.
1023- 1024

operador de acesso a membro (.) 397- 398

operador de adição com atribuição **(+=)**
124- 125
operador de adição com atribuição **(+)** sobrecarregado 550-551
operador de atribuição **(=) 72-73, 81-82,** 126-127, 430-431. 516-517, 922-923
operador de atribuição **(=**
sobrecarregado 530-531, 533-534,
543-545. 582-583
operador de atribuição e deslocamento à direita com extensão do sinal 820-
821

operador de chamada de função O 545-
546, 630-632, 715-7 16
operador de chamada de função O sobrecarregado 545-546
operador de coerção 116, 118-119, 198- 199, 340-341, 535-536, 1004-1006,
1017-10 18

operador de coerção dynaiuic_cast
1015-1016

operador de complemento (-) 814-815 operador de complemento sobre bits (-)
816-8 17, 819-821
operador de concatenação **+** sobrecarregado 545
operador de concatenação de ***string*** sobrecarregado 544-545
operador de concatenação
sobrecarregado 544-545
operador de decremento unário (-) 125 operador de derreferência de ponteiro
(*) 397-398, 630-632 operador de derreferenciação (*) 322-
324

operador de desigualdade sobrecarregado 530-531, 534
operador de deslocamento à direita (»)
516, 640-641, 814-815. 820-821, 851
operador de deslocamento à direita sobre bits (»)516
operador de deslocamento à esquerda
(«)5l6, 640-641, 814-815, 819-
821. 850-851
operador de deslocamento à esquerda com atribuição 820-821
operador de deslocamento à esquerda sobre bits («) 516, 8 19-820
operador de deslocamento sobre bits
8 19-820

operador de endereço (&) 321-322, 324- 325, 334-335, 5 16-517, 1026
operador de extração de *síream»*
(“obter de”) 70-72, 230-231, 516,
520-521, 532-533, 640-64!, 645-648,
733-734, 750-751, 890-89!
operador de extração de *stream* » sobrecarregado 643-646
operador de extração de *streun?* em uma condição de continuação de laço 647-
648

operador de extração de *stream* retornando *false* no fim de arquivo
647-648

operador de extração de *síream* sobrecarregado 530-532
operador de igualdade **(** e ! **=)** 146- 147, 524-525
operador de igualdade **=** 922-923 operador de igualdade **(=)**
sobrecarregado *531-532,* 534, 545
operador de igualdade 79-80
operador de incremento 125, 546-547
operador de incremento sobrecarregado
547-548

operador de incremento unário (++) 125
operador de indireção (*) 322-325
operador de inserção em *stream* **«**
(“enviar para”) 72-73, 230-231, 516,
520-521, 532-533, 594-595, 640-642,
733-734

operador de inserção em *stream* **«** sobrecarregado 643-644
operador de inserção em *siream* sobrecarregado 530-53 1, 549-55 1, 570-572, 594-595, 750-751
operador de membro de estrutura (.)
880-881
operador de negação lógica **()** 146-149 operador de negação sobrecarregado
545

operador de pós-decremento 125
operador de pós-incremento 125
operador de pós-incremento
sobrecarregado 549-551
operador de pré-decremento 125
operador de pré-incremento 125
operador de pré-incremento
sobrecarregado 549-551
operador de resolução de escopo (: :)
397-398, 475-476. 573, 577-578.

594-595. 686-688, 690-691. 1009-
1010. 1012-1013, 1026
operador de saída sobrecarregado 750-
751

operador de seleção de membro (.)
397-398. 406-407. 469-470, 608-609,
615-616, 618. 620-621. 716-717
operador de seleção de membro seta (-
>) 406-407

operador de subscrito sobrecarregado
53 1-532, 534-535, 545-546, 556-557
operador delete 621-622, 761-762
operador dynamic cast 1014-1015
operador menor do que sobrecarregado
545

operador menor que < 922-923
operador módulo (%) 74-75, 96-97, 181- 182, 201-202, 205-206
operador OR exclusivo sobre bits ()
814-817, 819
operador OR exclusivo sobre bits com atribuição **(=)** 820-821
operador OR inclusivo sobre bits (1)
814-817, 819
operador OR inclusivo sobre bits com atribuição **(1=)** 820-821
operador parênteses (O) 75-76, 119- 120

operador ponto (.) 397-398, 406-407,
469-470. 608-609, 615-616, 618-621,
716-7 17
operador relaciona! 79-80
operador seta (->) 397-398, 469-470
operador sizeof 336-338, 757-758. 809-8 10
operador subscrito [] 892-893
operador subscrito [] usado com *strings* 890-891
operador subscrito 942-943
operador subscrito de array ([1) 531- 532

operador subscrito de um mapa 949-
950
operador unário 119-120, 148-149, 32!- 322, 518-519
operador vírgula **(,)** 131-132, 219
operador(es) re!acional(is) 77-78, 130- 131

operadores aritméticos 74-75
operadores aritméticos binários 119-120
operadores binários sobrecarregados
518-5 19
operadores condicionais 106-107, 646- 647
operadores de atribuição abreviados
124- 125
operadores de atribuição da classe base não-herdados pela classe derivada
577-579
operadores de atribuição sobre bits 820-
821. 986-987
operadores de conversão 535-536, 1005- 1006
operadores de decremento 125, 546-547
operadores de igualdade 77-78, 104-105
operadores de inserção em *strealn* e extração de *stream* definidos pelo usuário 520-521
operadores de inserção/extração de *strea,n* 643-644

operadores de ponteiros 322-324
operadores lógicos 146-147
operadores multiplicativos .***1.*** % 119- 120

operadores ponteiros para membros de classe 1004

operadores que devem ser não-membros
522-523
operadores que podem ser
sobrecarregados 517-518
operadores relacionais >, <. >=. e <=
146-147

operadores sobre bits 813-815, 820-821
operadores sobrecarregados 525-526
operadores unários mais (+) e menos
(-) 119-120
operadores unários sobrecarregados
518-5 19
operando 72-75, 105-106, 3t2, 791-799
operando da direita 67-68
operandos int promovidos para
float 118-119
operator **void*** 735
operator' 545
**operator = ** 534, 920-921
operator O) 545-546. 556-557, 989-
990, 992
operator [] 545-546, 703-704
operator+ 516-5 17
operator++ 546-547, 55 1-552
operator++ (int) 546-547
operator+= 544-545
operator< 545. 920-921
operator« 522-523, 531-532, 543-
544, 562. 573
operator<= 920-921
operator= 533-534, 544-545, 920-
92!
operator== 534, 545, 920-921, 959-
960
operator> 920-921
operator>= 545, 920-921
operator»52l-523, 531-532. 543-
544
or 1016-1017
OR exclusivo sobre bits 1016-1017
OR inclusivo sobre bits 1016-1017
ORlógico(l 1)146-147,185-186,819,
1016-10 17
OR lógico sobre bits 986-987
oreq 1016-1017
ordem de avaliação de operadores 96
ordem de avalização 75-76, 219
ordem de classificação 969-970, 973-
974

ordem de construção 577-579
ordem de construção de objetos membro 577-579
ordem de término vinda do sistema operacional 875-876
ordem inversa dos bits em um inteiro sem sinal 851

ordem na qual as ações devem ser executadas 100-101. 110-111
Ordem na qual construtores e destruidores de classe base e classe derivadas são chamados 579-580
ordem na qual construtores e destruidores são chamados 421-423,
434-435
ordem na qual destruidores são chamados 42 1-422
ordem na qual inicializadores de membros são listados 577-579
ordem na qual operadores são aplicados a seus operandos 219
ordem na qual tratadores de exceção são listados 705
ordem último a entrar, primeiro a sair
(LIFO) 683-684, 686-688
ordenando 482-483, 729-730, 760, 968, 970

ordenando arrays 282-283

ordenando *strings* 201
ordenando um array com *bubble sort* 283-284
orientação a objetos 8 1-82, 396
orientado a cliente 405-406
orientado a objetos 83-84
orientado para ações 83-84
os operadores ponteiros *&* e * 322-324
ostream 562, 640-642, 647-648, 650-651, 728-729, 73 5-736, 740-746, 750-751
ostreainiterator 923-925
ostreamiterator< int >935-936
ostringstrearn 907-909
otimização 874-875
otimização de compilador 874-8 75
otimizações sobre constantes 452-453
otimizando o compilador Simple 803-805
outCredit 744-746
outros conjuntos de caracteres 890
overflowerror 716-717
overhead 545
overhead de chamada de função 222-223, 545
overhead de chamada por valor 399-400
overhead de cópia 398-399
overhead de função 85 8-859
overhead de função virtual 955
overhead de uma chamada de função 857-858
overhead de uma chamada de função extra 534-53 5
overhead durante a execução 629-630
oxímoro 267-268

P

pacote (de aplicativos) gráfico 635-636 pacote 778

pacotes 63-64
padrão ANSI C++ 1052-1053
padrão ANSL[ISO C++ 65-66, 1000- 1001, 1055-1056
padrão *de facto* 48 1-482
padrão de tabuleiro de xadrez 96-97, 181-182
pagamento bruto 178-179
pair 945-946, 959-960
palavra 381, 809-810
palavra(s)-chave 102-104. 109-110
palavra-chave operador de desigualdade 1016-1017
palavra-chave template 232-233. 680-68 1
palavras e frases descritivas 234-235
palavras-chave operadores 1004, 1016- 1018

palavras-chave operadores lógicos 1016- 10 17
palavras-chave operadores sobre bits 1016- 1017
palavras-chave operadores sobre bits com atribuição 1016-1017
palavras-chave operator 5 16-517
palavras-chave reservadas 109-110
pal mndromo 790-79 1
papéis 164
papel 53-54
par chave/valor 946-949

par de chaves {} 80-81
par de iteradores 927-928
par de parênteses mais interno 75-76
paralelogramo 565-566
parâmetro 191-195, 209-210
parâmetro de função como uma variável local 194-196
parâmetro de referência constante 225-226
parâmetro de tamanho de gabarito não- tipo 689-690
parâmetro de tipo 232-235, 68 1-682, 684-685, 689-690
parâmetro de tipo formal 232-233. 686- 688
parâmetro de tipo formal em uma definição de gabarito de função 681-682
parâmetro não-tipo 689-690
parâmetro por referência 223-225
parâmetros de tipo formais de uma definição de gabarito 680-681
parâmetros em funções 194-196
parênteses aninhados 75-77
parênteses para forçar a ordem de avaliação 81-82
parênteses redundantes 77-78, 147-148
parte da classe base de um objeto de classe derivada 615-616, 618, 620
parte **else** do **if** 108-109
partes fracionárias 118-119
partialsort 985-986
partial_sort_copy 985-986
partialsum 929-930, 985
partition 928-929, 985
Pascal 50-52, 58-59, 1050-1051
Pascal, Blaise 5 8-59
passagem por uma expressão da esquerda para a direita 791-792
passando além do fim de um array 340-341
Passando arrays e elementos individuais de arrays para funções 279-281
passando arrays para funções 278-279, 286-288
passando objetos grandes 225-226 passando opções para um programa 87 1-872
passando um array inteiro 279-281 passando um elemento de array 279-281
passando um nome de arquivo para um programa 87 1-872
passando um objeto através de chamada por valor 430-431
passando valores para funções 534-535 passar argumentos de tamanho grande como referências const 399-400
passar o tamanho de um array 336-337 passar o tamanho de um array como um argumento 278-279
passar uma estrutura 810-811 passar uma estrutura através de chamada por referência 810-811
passara através de chamada por valor 334-335
Passeio do cavalo 313-315
Passeio do cavalo; abordagens de força bruta 315-316
Passeio do cavalo; teste de passeio fechado 317
patrimônio em software 83-84, 405-406 “peças padronizadas, intercambiáveis” 83-84
“pensando em objetos” 81-82. 396
Pensando em objetos 5 1-52, 66-67, 396
pequeno círculo 101-103. 105-106. 109- 110
percorrer a subárvore direita 786-787
percorrer a subárvore esquerda 786-787
percorrer para a frente e para trás 773-

percorrer um contêiner 760 percorrer uma árvore binária 78 1-782,
787-788

percorrer uma árvore binária em ordem de nível 787-788. 795
percorrer uma lista 772-773 percurso de uma árvore binária na pósordem, com recursividade 220-22 l
percurso de uma árvore binária na préordem, com recursividade 220-221

percurso em ordem 781-782, 795
percurso em ordem de uma árvore
binária com recursividade 220-22 1
percurso em pós-ordem 78 1-782, 793-
795
percurso em pré-ordem 78 1-782
percurso na ordem inversa 906-907
percurso, travessia 906-907
perda de dados 665-666
perda de memória *544-545, 575-577,*
714-717, 761-762, 905-906, 918-919
perda de recurso 699-700, 711-7 12
permutação 985
permutador lexicográfico 985
permutando strings 896-897
permutar 284-285
permutar dois strings 896-897
personalizar software 583-584
pesquisa 270-273, 284-285
pesquisa binária 288-291
Pesquisa binária em um array ordenado
290-291
pesquisa binária recursiva 220-221,
288-289, 317-318
pesquisa de opinião 284-285
pesquisa em árvore binária 794-795
pesquisa linear 288-291
pesquisa linear em um array 288-289
pesquisa linear recursiva 220-221, 288-
289, 317-318
pesquisando (Ou procurando) 482-483, 760, 968. 970
pesquisando em arrays 288-289
pesquisando em blocos de memória
838-839
pesquisando em *„strings* 356-357, 829-
830
pesquisando uma lista encadeada
recursivamente 220-221
pesquisando uma lista recursivamente
793-794
pesquisar uma lista encadeada 763-764, 806
PHOAKS = People Helping One
Another Know Stuff 1051-1052
Pi 96-97, 185-186
P1 857, 857-858
pig Lalin 390-391
pilha 479-480, 482-483, 683-684, 686-
688, 708-709, 760-763, 773-776
pilha de chamada de funções 33 1-332,
708-709
pilha de chamadas 33 1-332
pilhas implementadas com arrays 479-
480
pilhas usadas por compiladores 791-792
pipe **(1)** 868-869
planta arquitetônica 396-397
plataforma de hardware 56
Plauger, P.J. 56-57
plus 990-991
pointer 922-923

polimorfismo 142-143, 452, 564-565,
583-584, 589-590, 608-609, 609-6 12,
620-622, 629-630, 803, 955
polimorfismo e referências 629-630 polimorfismo em linguagens não-
orientadas a objetos 629-630
polimorfismo substituindo a lógica de switch 564-565
polinomial 77-78
polinômio de segundo grau 77-78
ponta de seta 30 1-304
ponteiro 225-226, 320, 338-339
ponteiro chamado this 468-469
ponteiro const 5 24-525
ponteiro constante 340-34 1
ponteiro constante para dados constantes 329, 332-334
ponteiro constante para dados nãoconstantes 329, 33 1-333
ponteiro constante para uma constante inteira 332-333
ponteiro da *vtable* de um objeto 630-
632
ponteiro de classe base para um objeto da classe derivada 6 10-612, 62 1-622, 629
ponteiro de posição em arquivo de saída
740-743
ponteiro de posição em um arquivo 735,
744-746
ponteiro de *vtable* 632-633
ponteiro implícito 462-465
ponteiro indefinido 533-534, 544-545
ponteiro não-constante para dados constantes 329-332
ponteiro não-constante para dados não constantes 329
ponteiro nulo (O) 732-733. 760-761, 794-795, 878-879
ponteiro para a frente 773-774
ponteiro para classe base (ou tipo de
referência) 568-569, 582-584, 610-
613, 629-630, 705, 762-763, 1007-
1008, 1014-1015
ponteiro para classe derivada 567-568, 1007-1008, l014-1015
ponteiro para float 320-32 1
ponteiro para função 349-350, 629-632, 713-7 14, 990-992, 1025
ponteiro para o fim 805-806
ponteiro para trás 773-774
ponteiro para um objeto 404-405
ponteiro para um objeto de classe derivada 610-611
ponteiro para uma estrutura 331-332
ponteiro para uma função 349-35 2, 874
ponteiro para void (void *) 340-341
ponteiro para void 340-341
ponteiro *put* 735-736
ponteiro this 462-465, 468-470, 478, 519-520, 534

ponteiro this usado explicitamente
468-469
ponteiros declarados const 332-333
ponteiros e arrays 340-341
ponteiros e referências para classes base abstratas 611-612
ponteiros e subscritos de arrays 340-341 ponteiros para memória alocada dinamicamente 469-470, 534
ponteiros podem ser usados com subscritos 341-342
ponto de disparo 700-701, 704-705
ponto de entrada 150-152
ponto de entrada de uma estrutura de controle 150-152
ponto de entrada único 150-152
ponto de saída 150-152
ponto de saída único 150-152
ponto decimal 116. 119-120, 135-136, 64 1-642, 658-659
ponto decimal em um número de ponto flutuante 191-193
ponto flutuante 653-654, 657-658, 662- 663
ponto-e-vírgula (;) 67-68, 80-81, 108- 109, 194-196, 397-398, 417, 420.
706, 856
ponto-e-vírgula no fim de um protótipo de função 198-199
ponto-e-vírgula que terminam a definição de uma estrutura 808-809
pop 686-688, 774-775, 950-955 pop_back 930-931, 940-941, 95l-
952, 954-955
popfront 937, 940-943, 952-953
pop_heap 982-984
porção controladora 159-160, 164
porção do mundo 159-160
portabilidade 65-66
portável 56

pós-decremento 126-127, 546-547
posição corrente em um *stream* 735-736
posição de memória 73-74. 110-111
posição de memória do Simpletron 805-
806
posição na memória 73-74
posição temporária 800, 802-803
posicionamento em relação à posição atual em um *tream* 690-691
pós-incrementar um iterador 927 pós-incremento 125-127. 546-547, 550- 551
postOrderTraversal 786-787
potência 192-193
power 172-173
Powersoft 1054-1055
precedência 75-78, 8 1-82. 126-127. 131-132, 148-149, 219, 341-342
precedência de operadores 75-76, 821- 822
precedência do operador condicional
106- 107

precedência e associatividade de
operadores 148-149
precedência não-alterada por sobrecarga
518-5 19
precisão 119-120,641-642,651-653
precisão default 119-120
precision 652-653
pré-decremento 126-127, 546-547
preencher com caracteres especificados
64 1-642
preenchimento 654-655, 660-661, 824-
825

preenchimento em uma estrutura 824-
825
preenchimentos de buifer 667-668 Pré-incrementando e pós-
incrementando 126
pré-incremento 126-127, 546-547, 550- 551

preOrderTraversal 786-787
pré-processador 61-63, 197-198, 856
pré-processador C++ 61, 63, 67-68
pré-processar 61, 63
pressionando as teclas em um teclado
639

prevyermutation 985-986
prevenção de erro 608-609
primeira passagem do compilador Simple 798-801, 803
primeiro a entrar, primeiro a sair (FIFO) 481-482, 778, 919-920, 940-941.
952-953
primeiro argumento implícito 468-469
primeiro refinamento 114-115, 12 1-122,
346-347
primo 254-25 5
principal 135-136, 186, 188
princípio do menor privilégio 209-2 10,
282-283, 326-329, 335-336, 345-346,
408-409, 4 13-414, 452-453, 735,
872-874, 927
priorityqueue 9 19-922, 926-927,
950-951, 954-955
private 40 1-403, 413-4 14
privilégio 332-33 3
privilégios de acesso 329, 332-333
probabilidade 20 1-202
problema da inclusão circular 494-495
problema da média da turma 110-111, 116
problema de Peter Minuit 137, 186, 188

problema de precedência 646-647
problema do **else** sem correspondente
180-181
problema dos resultados do exame 122-
123

problema genérico de média da turma
113-114. 116
problemas de ambigüidade 592, 594- 595
procedimento 100-101

procedimento de recuperação 697-698
procedimento puro 406-407
processador de textos 354-355
processamento de acesso instantâneo
745-746
processamento de arquivo 639-641 processamento de arquivo de dados formatado 727
processamento de dados “brutos” 727
processamento de dados comercial 756
processamento de exceções polimórfico
711-712
processamento de interrupção 697-698
processamento de textos 39 1-392
processamento de transações 947-948
processamento em lotes 53-54
processamentos de listas 763-764
processando *strings* 320
processando strings em *streams* 907-
908
Processando uma fila 778
processo de análise e projeto orientados a objetos (OOAD) 84
processo de compilação para um programa Simple 80 1-802
processo de projeto 366-367
processo de refinamento 114-115
procurando *strings* e caracteres em um string 898-899
produtividade 63
produto de inteiros ímpares 183-184
programa 52-53
programa com pilha usando composição
777
programa de análise de dados 285, 287- 288
programa de análise de pesquisa entre alunos 270-271
programa de análise dos dados de pesquisa 285, 287-288
programa de classificação de uso geral usando ponteiros para funções 350-
351

programa de computador 52-53 Programa de consulta de crédito 735-
736

programa de contas a receber 756
Programa de contas bancárias 747-748
programa de conversão de medidas 393-
394
programa de *craps* 259-260 Programa de embaralhamento e
distribuição de cartas 348
programa de folha de pagamento 55-56 programa de impressão de histograma
272-273
Programa de impressão de texto 66-67 programa de lançamento de dados 273-
274

Programa de lançamento de dados usando arrays em vez de switch
273-274

Programa de média da turma com repetição controlada por contador
111-112
Programa de média da turma com repetição controlada por sentinela
116-118
programa de processamento de arquivo
756

programa de processamento de crédito 740-741
programa de processamento de transações 745-746
programa deteste 615-616, 618, 620
programa editor 61. 63. 100-101
programa em pseudocódigo 100-101
programa estruturado 150-152
programa gerador de palavras a partir de números de telefone 757
programa interpretador 56
programa para jogar 199-201
programa para jogar pôquer 378-379
Programa para simular o jogo de *craps* 206-207
programa pré-processador 61, 63
programação baseada em objetos (OBP) 452
programação defensiva 131-132
programação em linguagem de máquina 380-381
programação estruturada 50-53, 56-60, 65-66, 81-82. 100-103, 146-147, 396, 479-480, 87-879
programação genérica 917-919. 921-922, 1000-1001, 1055-1056
programação orientada a objetos (OOP) 51-53, 56-57. 65-66. 82-83. 155-156, 192-193, 363-364,401-402.452, 564-565, 565-567, 583-584. 589-590, 608-609
programação polimórfica 630-632
programação sem goto 101-103
programador 52-53
programador de computador 52-53
programas com múltiplos arquivos-fonte 872-874
programas grandes 190
programas tradutores 55-56
Programmer's Source 1054-1055
projeto 84
projeto de programa 104-105
projeto Green 57-58
Projeto Mac 63-64
Projeto Mac do MIT 63-64
projeto orientado a objetos (OOD) 81-82, 155-156, 159-160, 186, 188, 298-299, 363-3 64, 396, 432-434, 465-466
projetos 874
proliferação de classes 584-585
promoção 118-119
prompt 7 1-72, 116-118, 868-869
prompt de linha de comando 868-869

remove copyif 928-929, 960-961, 963
removeif 928-929, 960-961
removendo o atributo "const" 1008- 1009

"removidos pela máscara" 816-817
rend907-908, 921-922, 933-934
rendimentos 178-179
repetibilidade da função rand 203-204
repetição 102-104, *154-156*
repetição controlada por contador 110-112, 126-128, 219-220
repetição controlada por sentinela 113-116
repetição definida 110-111
repetição indefinida 114-115
repetindo código 192-193
replace 901-902, 928-929, 963
replace_copy 928-929, 963-965
replace_copy_if 928-929, 963-965

replaceif 928-929, 963
representação de dados 480-481
representação de membros de dados
428-429
representação de um baralho em um
array bidimensional 345-346
representação gráfica de um algoritmo
101- 103
representação gráfica de uma árvore
binária 780-781
representação interna de um string
542-543
representação numérica de caracteres
139- 140
requisitos 64-65, 84, 159-160
requisitos do sistema 159-160
reset 986-987
resize 898-899
resolução de sobrecarga 683-684
restaurar o estado de um *stream* para
bom 665-666
resto após divisão inteira 74-75
restrição 164
Restrições 164
retângulo 302-304
retângulo com cantos arredondados 236
reticências 870-871
reticências *(.* **.)** em um protótipo de
função 868-869
retirada de lista encadeada com
recursividade 220-221
retirar da fila 48 1-482
retirar um nodo de uma lista 772-773
Retornando uma referência a um
membro de dados private 429-

430

retornar para a fase de edição 63
retornar um resultado 196-197
retornar um resultado inteiro 194-195
retornar um valor 67-68

retornar um valor de um construtor 417,
420

retorno do carro (\r) 68-69, 825,
827-828
retumando valores a partir de funções
534-535
reutilização 155-156, 192-193, 336-337, 564, 683-684
reutilização de software 56-57, 60-61,
68-69, 155-156, 192-193, 405-406, 564,583-585,592,621-622,680-
681, 683-685
reutilizado 83-84
reutilizando componentes 60-61
reutilizar 405-406, 583-584
“reutilizar, reutilizar, reutilizar.” 83-84,
405-406, 760, 917-918
reverse 928-929, 937, 97 1-975
reversecopy 928-929, 974-976
reverseiterator 92 1-923, 927,
933-934
rfind 900-901
Richards, Martin 56
Ritchie, D. 56
Rogue Wave 60-61
rotação de um disco ou de uma fita 639
rotate 928-929, 985-986
rotatecopy 928-929, 985-986
rótulo 211-212
rótulo case 137, 140-141, 21 1-212
rótulo de ação 237-23 8
rótulo especificado em um comando

goto 878-879
rótulos em uma estrutura switch 211-
212

RTTI 564, 568-569, 1004, 1013-1014
Rumbaugh, James 84
runtimeerror 716-717
rvalue (“valor à direita”) 150-152, 226- 227, 53 1-532, 534-535, 927

S

saída antecipada de um laço 144-145
saída com buifer 640-641
saída com *stream* 640-641
saída de arquivo 640-641
saída de caracteres 64 1-642
saída de inteiros 641-642
saída de letras maiúsculas 641-642
saída de ponteiros 64 1-642
saída de tipos de dados padrão 641-642
saída de valores em ponto flutuante
64 1-642
saída de variáveis char * 644-645
saída na tela 66-67, 803
saída não-formatada 641-642
saída padrão 868
saída para um *string* na memória 199-
201
saída sem buifer 640-641
saídas acumuladas 72-73

sair 237-238
sair de um laço 186, 188
sair de um programa 709-7 10
sair de uma estrutura profundamente aninhada 87 8-879
sair de uma função 68-69
“sair fora” em qualquer extremidade de um array 524-525
salta o resto de uma estrutura switch
144- 145
salta o resto do código no laço 145-146
Scott, Kendall 84-85
search 929-930
searchn 929-930
seção “administrativa” do computador
53-54
seção de “almoxarifado” do computador
53-54
seção de “expedição” do computador
53-54

seção de “recebimento” do computador

52-53
seção de inicialização do for 134-135
Seção especial
Construindo seu próprio
computador 380-381
Seção especial - Construindo seu
próprio compilador 796
“seção industrial” do computador 53-54
second 945-946
seek get 735
seek put 735
seekp 735, 742-746
segunda passagem do compilador
Simple 803
segundo refinamento 114-115, 121-122,
346-347
segurança 430-431

seleção 102-104, 152-155
seleção dupla 154-155
seleção simples 154-155
selecionar um *substring* 545-546
semente 204-205
"sneakernet" 54-55
seno 192-193
seno trigonométrico 192-193
sensível a maiúsculas e minúsculas 70-71, 109-110
separação em funções 52-53
Separando a interface e a implementação da classe Time 409-410

separando um programa em funções 192-193, 22 1-222
separando unidades léxicas em *strings* 356-357
separar a interface da implementação 408-409
separar as palavras de uma sentença 793-794

sequência 102-104, 151-152, *154-155,* 971-974, 985-986
seqüência da saída 923-924
seqüência de entrada 923-924
seqüência de escape 68-70, 179-180, 642-643
seqüência de escape com barra invertida (\)68-69
seqüência de escape de alarme ('a' 68-69, 827-828
seqüência de escape de tabulação 179-180
seqüência de inteiros 183-184
seqüência de mensagens 366
seqüência de números aleatórios 203-204
seqüências de bits 727-729
seqüências de caracteres 739-740
série de Fibonaeci 217-218
servidor de arquivos 54-55
set 9 19-920, 926-927, 942-943, 946, 985-986
<set> 200-201, 92 1-922, 943-944, 946

setdifference 976-978 setintersection 976, 978-979, 981
setnewhandj.er 699-700, 713- 714

setsyrnmetricdifference 976, 978-979, 981
setterminate 704-705, 7Ut-7U9
setunexpected 709-710, 716-717
setunion 976, 978-979, 981
set< double, less< double>> 946-947
seta 96-97, 105-106, 109-110
setiosflags 119-120, 137, 348
setiosflags (ios: left) 137
setjump 699-700
setprecision 119-120,652-653
setprecisiori(2) 135-136
setw 137,263-264, 348, 356-357, 522-523, 654-655, 658-659
Shakespeare, William 390-391
short 142-143, 198-199

short int 142-143
showpoint 657-658, 664-665
SIGABRT 875-876
SIGFPE 875-876
SIGILL 875-876
SIGINT 875-876
signal para capturar eventos
1? inesperados 875-876
SIGSEGV 875-876
SIGTERM 875-876
Silicon Graphics Standard Templare
Library Programmer's Guide 1000-
1001, 1055-1056
símbolo de ação 101-103

símbolo de acrescentar na saída (»)
868-869
símbolo de composição 162-163
símbolo de decisão 102-105
símbolo de redirecionamento da saída>
868-869
símbolo de redirecionamento de entrada
<868
símbolo elipse 10 1-103
símbolo pequeno círculo lO 1-105
símbolo pequeno losango 238
símbolo retângulo 101-106, 109-110, 132-133, 302-304, 565-566
Simpletron Machine Language (SML) 388-389, 760-761, 796, 797-798,
803-805
Simula 59-60
Simulação
A tartaruga e a lebre 379-3 80
simulação 299-300, 365, 382-383, 485- 486
simulação de embaralhamento e
distribuição de cartas 345-346, 348,
808,811-812
simulação de embaralhamento e distribuição de cartas de alto desempenho 811-812
simulação de supermercado 793-794
simulador 159-160, 234-235
simulador de computador 382-383
simulador de elevador 396
simulador de vóo 635-636
simulador do Simpletron 388-389, 760- 761. 797-798, 803, 805-806
simulador em software 155-156
sin 192-193
sinais padrão 875-876
sinal 875-876
sinal de atenção interativo 875-876
sinal de percentual (%) (operador módulo) 74-75
sinal interativo 876-877
sinal justificado *à* esquerda 657-658
sinal mais (+) indicando visibilidade pública 43 2-434
sinal mais 660-661
sinal menos **(-)** indicando visibilidade privada 432-434
sinking sort 28 3-284
sinônimo 322-325
sintaxe de inicial ização de membros
485
sintaxe de inicializador de membro 456-
457
sistema acionado por menu 353-354
sistema de contas a receber 729-730
sistema de folha de pagamento 635-636
sistema de gerência de banco de dados
(DBMS) 727-729
sistema de numeração binário 814-815,
833-834

sistema de numeração com base 10
192-193, 657-658
sistema de numeração com base 16657- 658
sistema de numeração com base 8 657-
658
sistema de numeração decimal (base
10) 657-658, 832-834
sistema de numeração hexadecimal 825 sistema de numeração octal (base 8)
65 1-652, 657-658, 832-833 sistema de ponto de venda 739-740 sistema de reserva de companhias

aéreas 312, 739-740
sistema de software em camadas 611-
612
sistema de um banco 739-740 sistema operacional 53-56, 356-357,
732-733
sistema operacional orientado a objetos
611-612
sistemas baseados no Microsoft's Windows 54-55
sistemas compatíveis com o IBM PC
654-655
sistemas de processamento de transações 739-740
sistemas operacionais de memória virtual 58-59
Site Cygnus na Web 1054-1055
site da Web 50-51
site da World Wide Web 65-66
site na Web da *Quadralay Corpo ration*
105 1-1052
sites na Web relacionados com a STL
1000, 1055-1056
size 920-92 1, 932-933, 953-954, 973- 974, 986-987, 989-990
sizet 336-337, 740-741
sizetype 922-923
sizeof 336-338, 406-407, 468-469, 742-746, 76 1-762, 834-835, 859-860
sizeof nomeDeArray 336-337
Smalltalk 56-57, 565-566
SML 381, 797-798
sobrecarga de destruidores 420-421
sobrecarga de função 229-230, 401- 402, 573, 639, 868-869
sobrecarga de operador unário 523-5 24 sobrecarga de operadores 72-73, 230- 231, 400-401, 516, 639, 814-815
sobrecarregado 643-644
sobrecarregando « e » 230-231
sobrecarregando 72-73, 229-230, 401- 402, 516, 680
sobrecarregando funções gabarito 683-
684
sobrecarregando o + 51 8-5 l9
sobrecarregando o + 518-5 19
sobrecarregando o operador binário +=
523-524

sobrecarregando o operador de chamada de função O *545-546, 556-557*
sobrecarregando o operador de inserção em *stream* 750-75 1
sobrecarregando o operador de pós- incremento 551-552, *546-547*
sobrecarregando operadores 230-231 sobrecarregando operadores binários
523-524
sobrecarregando operadores de inserção em *stream* e extração de *stream* 520-
521

sobrecarregando operadores de prédecremento e pós-incremento 547-
548

sobrecarregando um operador de atribuição 518-519
sobrecarregar o operador « 400-40!
sobrecarregar o operador de adição (+)
5 16-5 17

sobrecarregar o operador de igualdade
400-40!
sobrecarregar o operador unário 523-
524

sobrecarregar uma função membro 406-
407

sobrecarregar uma função membro const com um versão não-const
452-453
sobrecarregou um operador como uma função não-membro, não-friend
519-520

sobrescrevendo 575-577
Sobrescrevendo membros de classe base em uma classe derivada 573-574
sobrescrevendo uma função 573, 575- 577
sobrescrever 608-609, 615, 619-620 sobrescrever uma função virtual pura 622, 624-625
software 50-53
software comercial 5 8-59
software confiável 878-879
software de alta qualidade depurado 564
software de layout de página 354-355
software embalado em plástico 583-584
software para a STL 1000-1001. 1056
solicitação de término enviada para o programa 875-876
solução iterativa 2 19-220
solução recursiva 2 19-220
soma dos elementos em um array 269-270

soma recursiva de dois inteiros 220-221 somar os elementos de um array recursivamente 220-221
somatório com for 134-135
sort 937, 939-940, 968, 970, 985-986
splice 937, 939-940
.spool de impressão 778
spool para disco 778

sqrt 191-193
srand 203-206
srand (time (0)) 205-206
stable_partition 928-929, 985
stablesort 985-986
stack 684-685. 9 17-920, 926-927, 950-953
Stack< double > 686-688, 777
stack<jnt> 686-688
Stack<T> 686-688
Standard Template Librarv Online Reference Home Page 1000-1001, 1055-1056
Standard Template Librarv Programmer's Guide 1000-1001, 1055- 1056
Standard Template Library with BorlandC++ 1001-1002, 1056
static 208-211,227-228,873-874, 881-883
staticcast (coerção com verificação de tipo durante a compilação) 127-128, 148-149, 264-265
staticcast 118-119, 139-140, 1005-1008
staticcast< DerivedCJ.asse *> 1007-1008
std narnespace 1010-1012
std: :bad_exception 716-717 std: :cin70-72
std: :cout66-68 std: :endl72-73
Stepanov, Alexander 917-918, 10001001, 1055-1056
STL 629-630, 917-918
strcat 357-359, 544-545, 873-874
strcmp 357-360, 873-874
strcmp para comparar *strings* de caracteres 545
strcpy 357-358, 543-545
strcspn 834-836
stream de bytes 639
stream de caracteres 67-68
stream de entrada 645-646, 648-649
stream de entrada padrão (cm) 639- 640

stream de entrada padrão 85 8-859 *stream* de erro padrão sem buffer 639-640

tream de saída 935-936 *stream* padrão de erro com buffer 639-640

stream padrão de erros (cerr) 63
strerror 842-843
<sstream> 199-201, 907-908
<stack> 200-201, 921-922, 951-952
<stdexcept> 199-201, 716-717
<stdio . **h**> 200-201
<stdlib . h> 200-201
<string . h> 200-201

<string> 199-201, 890
string 343-344, 354-355, 544-545. 808
string462-465, 890-891, 919-920
string concatenado 544-545
string constante 354-355
string de caracteres 67-68, 263-264,
273-274
string de caracteres interno 543-544
string de saída 641-642
string é um ponteiro constante 355-356
string literal 7 1-72, 274-275, 354-356
string nulo 740-741
string sendo separado em "unidades
léxicas" 36 1-362
string terminado por nulo 274-275, 345-
346, 644-645
string vazio 897, 899
string: constiterator 906-
907

string: :npos 900-901
strings baseados em ponteiro 354-355
strings char * no estilo de C 904-905
strings como objetos completos 263-
264, 354-355
strings como objetos de classe
completos 452
strings no estilo de C 904-905
strlen 358, 362-363
strncat 3 57-359
strncmp 357-360
strncpy 357-358
Stroustrup, B. 56-57. 59-60, 680, *691,*
1052- 1053
strpbrk 834-837
strrchr 834-837
strspn 834-838
strstr 834-838
strtod 829-830, 832-833
strtok 357-358, 361-362
strtol 830-834
strtoul 830-831, 833-834
struct 396-397, 727-729, 740-741, 760. 808. 880-88!
subárvore direita 785-7 87, 794-795
subárvore esquerda 780-781, 785-787,
794-795
subclasse 565-566
subconj unto de objeto 565-566
subobjeto 1027-1028
subproblema 214
subscrito 262-263
subscrito através de um vector 936
subscrito de array fora dos limites 697-

698
subscrito de linha 293-294
subscrito fora do intervalo válido 936
subscritos de arrays 332-333. 342-343,
703-704
subscritos de colunas 292-293 substantivos 59-60, 160, 161, 186, 188,
234-235

tipo do ponteiro this 468-469
tipo parametrizado *694-695*
tipo size **t** 336-337
tipos de argumentos corretos 194-195
tipos de dados primitivos 480-48 1
tipos de exceções da STL 936
tipos de retorno corretos 194-195
tipos definidos pelo usuário 83-84
<time . h> 200-20 1
tolerância a falhas 697-699
tolower 825-827
top 951-953, 955
topo 114-115
topo de uma pilha 760, 774-775
Torres de Hanói 256-25 8
Torres de Hanói com recursividade 220-221
total 110-111, 114-115, 121-122,209-210
total acumulado 114-115
toupper 825-827
trabalhador 190-191
trabalhador tarefeiro 185-186
trabalhadores comissionados 185-186
trabalho 53-54
tradução 55-56
traduzir 61,63
transação 756
transferência de controle 101-103
transform928-929, 965-966, 968, 970
transição 23o-237
transição de estado 236-23 7
Transmission Control Protocol 63-64
transmitir com segurança 182-183
transporte (vai um) 55 1-552
trapezóide 565-566
tratador 702-703
tratador de exceção 697-698, 700-701, 704-705
tratador de exceção com void * 705
tratador de falha de new 713
tratador de interrupção 356-357
tratador de new 713-714
tratador de sinais 876-877
tratador void * 705
tratadores de exceção procurados em ordem 704-705
tratamento de erro 697
tratamento de exceções 199-201, 697
tratamento de sinais 876-877
Tratando arrays de caracteres como *.strings* 275-276
travessia recursiva de um labirinto 220-221
Tree<int> 784-785
triângulo reto 182-183, 185-186
Triplas de Pitágoras 185-186
true 77-78, 102-110, 128-129, 215-216. 534

trunca a parte fracionária de um double 198-199
truncar 74-75, 118-119.462-465,730-731
tutoriais para a STL 1000, 1055-1056
tutorial de C++ 1050-1051
typedef 810-811,890,907-908,921-922, 944-945, 948-949
typedefs em contêineres de primeira classe 922-923
typeid568-569,716-717, 1013-1015
<typeinfo> 199-201, 1014-1015
typename 232-233, 680-681

U

UAL 53-54

Um ambiente típico de C++ 6 1-62
Um exemplo de chamada por referência
225-226
Um exemplo de escopos 212-213
Um exemplo simples de tratamento de exceção com divisão por zero 700-
701
Um objeto const deve ser inicializado
268-269
um para dois 162-163
um para um 162-163
Um programa de adição 70-71
Um programa de análise de pesquisa entre alunos 270-27 1
Um programa que imprime histogramas
272-273
Um programa simples com pilha de dados 775-776
Um programa simples de pilha de dados usando composição 777
Uma classe Array com sobrecarga de operadores *525-526*
Uma classe de números complexos 556-
557
Uma classe *huge integer* 559-560
Uma classe String com sobrecarga de operador 536-537
UML 8 1-82, 84
UML Distilled
Second Edition 84-85
UML Partners 84-85
underflowerror 716-717
unexpected 708-709. 716-717
unidade central de processamento
(CPU) 5 3-54
unidade de aritmética e lógica (UAL)
53-54
unidade de compilação 1012-1013
unidade de disco 639
unidade de memória 53-54, 824-825
unidade de memória endereçável 824-
825
unidade de memória secundária 53-54 unidade de processamento 52-53

unidade Ixica *(token)* 357-358, 361- 362, 800-803
unidade lógica 52-5 3
unidades “isoladas’ 54-55
Unzfied Modeling Language (UML) 81- 82, 84
union 879-881
union anônima 88 1-882
union sem construtor 880-881
urlique 928-929, 937, 97 1-975
uniquecopy 928-929, 974-976
United States Department of Defense
(DOD) **58-59**
University of Illinois at UrbanaChampaign 63-64
UNIX 54-56, 59-61, 63, 139-140, 654- 655, 698-699. 732-733, 87 1-872,
874-876
unsetf(ios: :skipws) 657-658
unsigned 199-200, 204-205, 875-876
unsigned char 199-200
unsigned int 204-205. 336-337
unsigned long 199-200. 216-217, 833-834, 875-876
unsigned long int 199-200, 216- 217
unsigned short 199-200
unsigned short int 199-200
upcasting 1026-1027
upperbound 942-943, 945-946,
978-981
usa um 585
Usando a estrutura do/while 143-144
Usando a função membro f iii e o manipulador setfill 660-661
Usando a função swap para permutar dois strings 896-897
Usando argumentos de linha de comando 87 1-872
Usando argumentos default 228-229
usando arrays em vez de switch 273-
274

Usando as funções **exit e atexit**
874
Usando atof 830-83 1
Usando atoi 830-831
Usando atol 83 1-832
Usando campos de bits para armazenar um baralho 822-823
Usando classes base virtual 1028- 1029
Usando funções da biblioteca padrão para executar um *heapsort* 981-982, 984
Usando funções gabarito 682-683
Usando funções membro get. put e eof 647-649
Usando funções *set* e *get* 425-426
Usando goto 879-880
Usando inicializadores de objetos membro 461-462

unidade de saída 5 3-54

Usando isdigit, isalpha.
isainuin e isxdigit 826 Usando islower, isupper,
tolower e toupper 826-827 Usando isspace, iscntrl,
ispunct, isprint e isgraph
828-829
Usando listas de argumentos de tamanho variável 869-870
Usando memchr 84 1-842
Usando memcmp 841-842
Usando memcpy 839-840
Usando mernmove 840-841
Usando memset 842-843
Usando o comando break em uma estrutura for 145-146
Usando o comando continue em uma estrutura for 145-146
Usando o indicador ios: showbase
66 1-662
Usando o indicador
ios: : uppercase 663-664
Usando o manipulador de *siream* endi
642-643
Usando o operador de resolução de escopo unário 229-230
Usando o operador sizeof para determinar tamanhos de tipos de dados padrão 338
Usando o ponteiro this 468-469 Usando operadores de igualdade e relacionais 79-80
Usando os manipuladores de *stream* hex, oct, dec e setbase 652- 653

Usando os operadores AND sobre bits, OR inclusivo sobre hits, OR
exclusivo sobre bits e complemento
sobre bits 817-818
Usando os operadores de deslocamento
sobre bits 819-820
usando *pipe* 868-869
Usando quatro métodos para referenciar
elementos de arrays 342-343
Usando strcat e strncat 358-359
Usando strchr 834-835
Usando strcmp e strncmp 359-360
Usando strcpy e strncpy 358
Usando strcspn 835-836
Usando strerror 843-844
Usando strlen 362-363
Usando strpbrk 836-837
Usando strrchr 836-837
Usando strspn 837-838
Usando strstr 838-839
Usando strtod 832-833
Usando strtok 36 1-362
Usando strtol 833-834
Usando strtoul 834-835
usando subscritos 940-941
usando subscritos com um ponteiro e
um deslocamento 342-343

usando subscritos duplos para arrays bidimensionais 556-557
Usando tratamento de sinais 876-877 Usando um construtor com argumentos default 418
Usando um função mime para calcular o volume de um cubo 223-

224

Usando um gabarito de função 23 3-234 Usando um inicializador de membro
para inicializar uma constante de um tipo de dado primitivo 457-458
Usando um iterador para enviar um string para a saída 906-907
Usando um membro de dados static para manter uma contagem do número de objetos de uma classe
475-476

Usando um objeto ostringstream alocado dinamicamente 908-909
Usando uma classe Time com objetos const e funções membro const
453-454, 456-457
Usando uma função utilitária 4 14-415 Usando uma referência inicializada
226-227

Usando uma union anônima 882-883 usar explicitamente o rótulo private

412

USENIX C++ Conference 680 `using 79-80, 87-88`

`usingnamespace 1010-1012`
uso de buffer 667-668
uso explícito de `sizeof 471,` 473-474
uso explícito do ponteiro this 468-

469

usos indevidos de sobrecarga 517-5 18
utilitário make 874
<utiiity> 200-201

V

`vaarg 868-87 1`
`vaend 868-87 1`
`vaiist 868-87 1`
`va_start 868-87 1`
`vaiarray 919-920`
validação 907-908
validação de dados 413-414
validar uma chamada de função 197-
198
valor 7 1-72
valor"lixo" 112-113
valor absoluto 192-193
valor `booi faise 147-148`
valor `bool true 147-148`
valor da direita 150-152
valor da esquerda 150-152, 322-324
valor de deslocamento 205-206
valor de indicador 114-115
valor de ponto fixo 137

valor de ponto flutuante em notação científica 663-664
valor de sentinela 114-115, 117-118, 139- 140
valor de uma variável 73-74, 208-209 valor default para um argumento 228

229

valor diferente de zero tratado como `true` 149-150

valor do sinal 114-115
valor fantasma 114-115
valor final de uma variável de controle
126-127, 132-133
valor indefinido 112-113,573
valor inicial de um atributo 236
valor inicial de uma variável de controle
126-127, 129-130
valor maximuxn 197-198
valor médio 286-28 8
valor posicional 181-182
valor temporário 118-119, 198-199
valores de face de cartas 345-346
valores de nodos duplicados 780-781
valores de ponteiros como inteiros
hexadecimais 322-324
valores dos naipes de cartas 345-346
valores em ponto flutuante 119-120
valores mapeados 942-943
vaiuetype 92 1-923, 948-949
variáveis de classe static 474-475,
530-531

variável 70-71, 83-84, 100-101, 191193, 396-397
variável automática 209-210, 227-228,
774-775

variável com escopo de classe está oculta 406-407

variável com escopo de função 406-407
variável constante 267-270
variável contadora 112-113
variável de coerção visível no depurador

857

variável de controle 129-130
variável global 210-214, 228-229, 278- 279, 872-873, 1012-1013
variável local 191-193, 209-211, 213- 214, 33 1-332, 882-883
variável local automática 211-212
variável local static 211-212,214, 275-276, 957-958

variável não-inicializada 112-113
variável ponteiro 714-715
variável ponteiro static 210-211

varrendo imagens 53-54
VAX VMS 139-140, 732-733
`vector 9` 19-920, 926-927, 929-935,
950-955

`<vector> 200-201, 921-922, 930-93 1` vendedor de software independente
`(ISV) 56-57, 408-409, 583-584. 621-` 622

verbos 186, 188, 298-299, 366-367,
396-397

verbos em uma especificação de sistema

83-84

verificação de argumentos desativada
222-223
verificação de erro 190-19 1, 697
verificação de intervalo 4 13-414, *524-*
525, 892-893, 929-930
verificação de intervalo de subscrito
480-481
verificação de limites 272-273
verificação de limites de array 272-273
verificação de sintaxe 804-805
verificação de tipo 680-681, 857-859
verificação de validade 424-425, 428-

429

verificador de ortografia 852-85 3 verificar recursivamente se um *string* é
um palíndromo 220-221, 317-318 versão const de operator [] 534-

535

vi 61, 63
vida de um objeto 533-534
vídeo 639
vídeo digital 639
vídeo 1/O 639-640
vinculação dinâmica 608-609, 615-616,

vinculação estática 594-595, 608-609.
6 15-616, 618, 620-621, 629
vinculação tardia 62 1-622
violação de acesso 356-357, 9 18-919
violação de acesso à memória 918-919
violação de segmentação 875-876
violar o encapsulamento 564-565
virtual pura 61 1-613
virtual pura em classe base 622,
624-625
visibilidade private 43 2-434
visibilidade public 43 2-435
visualizando a recursão recursivamente
220-221
VMS 87 1-872
void * 340-341, 647-649, 705, 839-

840

void 221-222
volatile 874 875, 1007-1008
voltas para o primeiro 55 1-552
volume de um cubo 223-224
voz digital 639
vtable 629-630, 63 2-633

W

walkthrough do código 485
wchart 890
what 702-703, 712

width 654-655
WindowsNT 698-699
Wirth, Nicklaus 58-59
World Wide Web 52-5 3, 64-65
write 650-651. 740-743
www. ansi org 65-66
www.cygnus.com/misc/ wp/ 65- 66

www. deitei, com 50-53 www.omg.org 84-85

X

Xerox's Pato Alto Research Center (PARC) 56-57
xor 1016-1017
xor 164
xor_eq 1016-1017

Z

ZDNet University 1050-1051
zerar bits 665-666
O inicial 66 1-662
Ox inicial e OX inicial 66 1-662
zeros após a vírgula 119-120, 191-193,
657-658
zeros e uns 727

618, 620-622, 629-630, 632-633

Fim da obra.