EIGHTH EDITION

An Introduction to Programming with C++

Diane Zak

AN INTRODUCTION TO PROGRAMMING WITH C++

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

EIGHTH EDITION

AN INTRODUCTION TO PROGRAMMING WITH C++

DIANE ZAK

CENGAGE Learning®

Australia • Brazil • Mexico • Singapore • United Kingdom • United States

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

This is an electronic version of the print textbook. Due to electronic rights restrictions, some third party content may be suppressed. Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. The publisher reserves the right to remove content from this title at any time if subsequent rights restrictions require it. For valuable information on pricing, previous editions, changes to current editions, and alternate formats, please visit www.cengage.com/highered to search by ISBN#, author, title, or keyword for materials in your areas of interest.

Important Notice: Media content referenced within the product description or the product text may not be available in the eBook version.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

CENGAGE Learning®

An Introduction to Programming with C++, Eighth Edition
Diane Zak

Product Director: Kathleen McMahon

Product Team Manager: Kristin McNary

Senior Product Manager: Jim Gish

Senior Content Developer: Alyssa Pratt

Product Assistant: Abigail Pufpaff

Marketing Manager: Eric LaScola

Senior Production Director: Wendy Troeger

Production Director: Patty Stephan

Senior Content Project Manager: Jennifer K. Feltri-George

Managing Art Director: Jack Pendleton

Cover image(s): © Rudchenko Liliia/Shutterstock.com

Unless otherwise noted all screenshots are courtesy of Microsoft Corporation

Open Clip art source: OpenClipArt

© 2016 Cengage Learning

WCN: 02-200-203

ALL RIGHTS RESERVED. No part of this work covered by the copyright herein may be reproduced, transmitted, stored or used in any form or by any means graphic, electronic, or mechanical, including but not limited to photocopying, recording, scanning, digitizing, taping, Web distribution, information networks, or information storage and retrieval systems, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without the prior written permission of the publisher.

For product information and technology assistance, contact us at
Cengage Learning Customer & Sales Support, 1-800-354-9706

For permission to use material from this text or product, submit all requests online at **www.cengage.com/permissions**
Further permissions questions can be emailed to
permissionrequest@cengage.com

Library of Congress Control Number: 2015940474
ISBN: 978-1-285-86011-4

Cengage Learning
20 Channel Center Street
Boston, MA 02210
USA

Cengage Learning is a leading provider of customized learning solutions with employees residing in nearly 40 different countries and sales in more than 125 countries around the world. Find your local representative at **www.cengage.com**

Cengage Learning products are represented in Canada by Nelson Education, Ltd.

For your course and learning solutions, visit **www.cengage.com**

Purchase any of our products at your local college store or at our preferred online store **www.cengagebrain.com**

Notice to the Reader
Publisher does not warrant or guarantee any of the products described herein or perform any independent analysis in connection with any of the product information contained herein. Publisher does not assume, and expressly disclaims, any obligation to obtain and include information other than that provided to it by the manufacturer. The reader is expressly warned to consider and adopt all safety precautions that might be indicated by the activities described herein and to avoid all potential hazards. By following the instructions contained herein, the reader willingly assumes all risks in connection with such instructions. The publisher makes no representations or warranties of any kind, including but not limited to, the warranties of fitness for particular purpose or merchantability, nor are any such representations implied with respect to the material set forth herein, and the publisher takes no responsibility with respect to such material. The publisher shall not be liable for any special, consequential, or exemplary damages resulting, in whole or part, from the readers' use of, or reliance upon, this material.

Printed in the United States of America
Print Number: 01 Print Year: 2016

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Brief Contents

	Preface	xiv
	Read This Before You Begin	xviii
CHAPTER 1	An Introduction to Programming	1
CHAPTER 2	Beginning the Problem-Solving Process	23
CHAPTER 3	Variables and Constants	51
CHAPTER 4	Completing the Problem-Solving Process	75
CHAPTER 5	The Selection Structure	113
CHAPTER 6	More on the Selection Structure	157
CHAPTER 7	The Repetition Structure	201
CHAPTER 8	More on the Repetition Structure	247
CHAPTER 9	Value-Returning Functions	279
CHAPTER 10	Void Functions	329
CHAPTER 11	One-Dimensional Arrays	369
CHAPTER 12	Two-Dimensional Arrays	425
CHAPTER 13	Strings	461
CHAPTER 14	Sequential Access Files	511
CHAPTER 15	Classes and Objects	551
APPENDIX A	C++ Keywords	593
APPENDIX B	ASCII Codes	595
APPENDIX C	Common Syntax Errors	597
APPENDIX D	How To Boxes	599
	Index	603

The Answers.pdf and data files can be found online at CengageBrain.com.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Contents

Preface xiv
Read This Before You Begin xviii

CHAPTER 1 An Introduction to Programming 1

Programming a Computer 2
The Programmer's Job 2
Employment Opportunities 2
A Brief History of Programming Languages 3
Machine Languages 3
Assembly Languages 3
High-Level Languages 4
Control Structures 5
The Sequence Structure 5
The Selection Structure 6
The Repetition Structure 8
LAB 1-1 Stop and Analyze 10
LAB 1-2 Plan and Create 10
LAB 1-3 Modify 11
LAB 1-4 What's Missing? 11
Chapter Summary 12
Key Terms 12
Review Questions 13
Exercises 14

CHAPTER 2 Beginning the Problem-Solving Process 23

Problem Solving 24
Solving Everyday Problems 24
Creating Computer Solutions to Problems 25
Step 1—Analyze the Problem 26
Hints for Analyzing Problems 26
Step 2—Plan the Algorithm 28
Step 3—Desk-Check the Algorithm 31
The Gas Mileage Problem 34
LAB 2-1 Stop and Analyze 36
LAB 2-2 Plan and Create 36
LAB 2-3 Modify 39
LAB 2-4 What's Missing? 39
LAB 2-5 Desk-Check 40
LAB 2-6 Debug 40

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Chapter Summary . . . 42
Key Terms . . . 43
Review Questions . . . 43
Exercises . . . 45

CHAPTER 3 Variables and Constants . . . 51

Beginning Step 4 in the Problem-Solving Process . . . 52
Internal Memory . . . 52
Selecting a Name for a Memory Location . . . 53
Revisiting the Addison O'Reilly Problem from Chapter 2 . . . 54
Selecting a Data Type for a Memory Location . . . 55
How Data Is Stored in Internal Memory . . . 57
Selecting an Initial Value for a Memory Location . . . 60
Declaring a Memory Location . . . 62
LAB 3-1 Stop and Analyze . . . 64
LAB 3-2 Plan and Create . . . 65
LAB 3-3 Modify . . . 67
LAB 3-4 What's Missing? . . . 67
LAB 3-5 Desk-Check . . . 67
LAB 3-6 Debug . . . 68
Chapter Summary . . . 68
Key Terms . . . 69
Review Questions . . . 70
Exercises . . . 71

CHAPTER 4 Completing the Problem-Solving Process . . . 75

Finishing Step 4 in the Problem-Solving Process . . . 76
Getting Data from the Keyboard . . . 76
Displaying Messages on the Computer Screen . . . 78
Arithmetic Operators in C++ . . . 81
Type Conversions in Arithmetic Expressions . . . 82
The `static_cast` Operator . . . 83
Assignment Statements . . . 84
Arithmetic Assignment Operators . . . 87
Step 5—Desk-Check the Program . . . 88
Step 6—Evaluate and Modify the Program . . . 90
LAB 4-1 Stop and Analyze . . . 94
LAB 4-2 Plan and Create . . . 95
LAB 4-3 Modify . . . 98
LAB 4-4 What's Missing? . . . 98
LAB 4-5 Desk-Check . . . 99
LAB 4-6 Debug . . . 99
Chapter Summary . . . 99
Key Terms . . . 101
Review Questions . . . 102
Exercises . . . 104

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

CHAPTER 5 The Selection Structure 113

Making Decisions 114
Flowcharting a Selection Structure 116
Coding Selection Structures in C++ 118
Comparison Operators 120
Swapping Numeric Values 121
Displaying the Area or Circumference 124
Logical Operators 126
Using the Truth Tables 129
A Different Version of the Area or Circumference Program 130
Summary of Operators 132
Converting a Character to Uppercase or Lowercase 133
Formatting Numeric Output 134
LAB 5-1 Stop and Analyze 137
LAB 5-2 Plan and Create 138
LAB 5-3 Modify 143
LAB 5-4 What's Missing? 143
LAB 5-5 Desk-Check 143
LAB 5-6 Debug 144
Chapter Summary 144
Key Terms 145
Review Questions 146
Exercises 147

CHAPTER 6 More on the Selection Structure 157

Nested Selection Structures 158
Flowcharting a Nested Selection Structure 161
Coding a Nested Selection Structure 163
Logic Errors in Selection Structures 165
First Logic Error: Using a Compound Condition Rather Than a Nested Selection Structure 167
Second Logic Error: Reversing the Outer and Nested Decisions 169
Third Logic Error: Using an Unnecessary Nested Selection Structure 169
Fourth Logic Error: Including an Unnecessary Comparison in a Condition 170
Multiple-Alternative Selection Structures 173
The `switch` Statement 174
LAB 6-1 Stop and Analyze 177
LAB 6-2 Plan and Create 178
LAB 6-3 Modify 182
LAB 6-4 What's Missing? 182
LAB 6-5 Desk-Check 183
LAB 6-6 Debug 183
Chapter Summary 184
Key Terms 184
Review Questions 185
Exercises 187

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

CHAPTER 7 The Repetition Structure 201

Repeating Program Instructions 202
Using a Pretest Loop to Solve a Real-World Problem 204
Flowcharting a Pretest Loop 206
The `while` Statement 208
Using Counters and Accumulators 211
The Stock Price Program 213
Counter-Controlled Pretest Loops 217
The `for` Statement 219
The Total Payroll Program 220
The Tip Program 223
Another Version of the Commission Program 224
The Even Integers Program 226
LAB 7-1 Stop and Analyze 229
LAB 7-2 Plan and Create 230
LAB 7-3 Modify 234
LAB 7-4 What's Missing? 234
LAB 7-5 Desk-Check 234
LAB 7-6 Debug 234
Chapter Summary 235
Key Terms 236
Review Questions 236
Exercises 239

CHAPTER 8 More on the Repetition Structure 247

Posttest Loops 248
Flowcharting a Posttest Loop 250
The `do while` Statement 252
Nested Repetition Structures 254
The Clock Program 255
The Car Depreciation Program 258
LAB 8-1 Stop and Analyze 262
LAB 8-2 Plan and Create 263
LAB 8-3 Modify 267
LAB 8-4 What's Missing? 267
LAB 8-5 Desk-Check 267
LAB 8-6 Debug 268
Chapter Summary 268
Key Terms 268
Review Questions 269
Exercises 270

CHAPTER 9 Value-Returning Functions 279

Functions 280
Value-Returning Functions 281
The pow Function 281

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The `sqrt` Function . . . 282
The Hypotenuse Program . . . 283
The `rand`, `srand`, and `time` Functions . . . 285
The Guessing Game Program . . . 289
Creating Program-Defined Value-Returning Functions . . . 291
Calling a Function . . . 293
The Savings Account Program . . . 295
Function Prototypes . . . 298
Completing the Savings Account Program . . . 298
The Scope and Lifetime of a Variable . . . 304
LAB 9-1 Stop and Analyze . . . 305
LAB 9-2 Plan and Create . . . 307
LAB 9-3 Modify . . . 314
LAB 9-4 What's Missing? . . . 314
LAB 9-5 Desk-Check . . . 315
LAB 9-6 Debug . . . 316
Chapter Summary . . . 316
Key Terms . . . 317
Review Questions . . . 318
Exercises . . . 320

CHAPTER 10 Void Functions . . . 329

Functions . . . 330
Creating Program-Defined Void Functions . . . 331
Passing Variables to a Function . . . 337
Reviewing Passing Variables *by Value* . . . 338
Passing Variables *by Reference* . . . 341
LAB 10-1 Stop and Analyze . . . 345
LAB 10-2 Plan and Create . . . 347
LAB 10-3 Modify . . . 356
LAB 10-4 What's Missing? . . . 356
LAB 10-5 Desk-Check . . . 356
LAB 10-6 Debug . . . 357
Chapter Summary . . . 357
Key Terms . . . 358
Review Questions . . . 358
Exercises . . . 360

CHAPTER 11 One-Dimensional Arrays . . . 369

Arrays . . . 370
One-Dimensional Arrays . . . 370
Declaring and Initializing a One-Dimensional Array . . . 371
Entering Data into a One-Dimensional Array . . . 373
Displaying the Contents of a One-Dimensional Array . . . 375
The Calories Program . . . 376
Passing a One-Dimensional Array to a Function . . . 382
Calculating a Total and an Average . . . 384

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The Social Media Program—Searching an Array . . . 385
The Currency Converter Program—Accessing an Individual Element . . . 388
The Highest Number Program—Finding the Highest Value . . . 391
Parallel One-Dimensional Arrays . . . 396
Sorting the Data Stored in a One-Dimensional Array . . . 399
LAB 11-1 Stop and Analyze . . . 406
LAB 11-2 Plan and Create . . . 407
LAB 11-3 Modify . . . 411
LAB 11-4 What's Missing? . . . 411
LAB 11-5 Desk-Check . . . 411
LAB 11-6 Debug . . . 412
Chapter Summary . . . 413
Key Terms . . . 413
Review Questions . . . 414
Exercises . . . 416

CHAPTER 12 Two-Dimensional Arrays . . . 425

Using Two-Dimensional Arrays . . . 426
Declaring and Initializing a Two-Dimensional Array . . . 427
Entering Data into a Two-Dimensional Array . . . 428
Displaying the Contents of a Two-Dimensional Array . . . 431
The Chapton Company Program . . . 432
Accumulating the Values Stored in a Two-Dimensional Array . . . 434
Searching a Two-Dimensional Array . . . 436
Passing a Two-Dimensional Array to a Function . . . 443
LAB 12-1 Stop and Analyze . . . 444
LAB 12-2 Plan and Create . . . 446
LAB 12-3 Modify . . . 449
LAB 12-4 What's Missing? . . . 449
LAB 12-5 Desk-Check . . . 449
LAB 12-6 Debug . . . 450
Chapter Summary . . . 450
Key Terms . . . 450
Review Questions . . . 451
Exercises . . . 452

CHAPTER 13 Strings . . . 461

The `string` Data Type . . . 462
Getting String Input from the Keyboard . . . 462
The Primrose Auction House Program . . . 464
The `ignore` Function . . . 468
Determining the Number of Characters in a `string` Variable . . . 471
Accessing the Characters in a `string` Variable . . . 473
Searching the Contents of a `string` Variable . . . 477
Removing Characters from a `string` Variable . . . 481
Replacing Characters in a `string` Variable . . . 484
Inserting Characters Within a `string` Variable . . . 485

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Duplicating a Character Within a `string` Variable . . . 487
Concatenating Strings . . . 489
LAB 13-1 Stop and Analyze . . . 491
LAB 13-2 Plan and Create . . . 492
LAB 13-3 Modify . . . 497
LAB 13-4 What's Missing? . . . 497
LAB 13-5 Desk-Check . . . 498
LAB 13-6 Debug . . . 498
Chapter Summary . . . 499
Key Terms . . . 500
Review Questions . . . 501
Exercises . . . 504

CHAPTER 14 Sequential Access Files . . . 511

File Types . . . 512
Creating File Objects . . . 512
Opening a Sequential Access File . . . 514
Determining Whether a File Was Opened Successfully . . . 516
Writing Data to a Sequential Access File . . . 518
Reading Information from a Sequential Access File . . . 519
Testing for the End of a Sequential Access File . . . 522
Closing a Sequential Access File . . . 522
The eBook Collection Program . . . 523
LAB 14-1 Stop and Analyze . . . 528
LAB 14-2 Plan and Create . . . 530
LAB 14-3 Modify . . . 539
LAB 14-4 What's Missing? . . . 540
LAB 14-5 Desk-Check . . . 540
LAB 14-6 Debug . . . 541
Chapter Summary . . . 541
Key Terms . . . 542
Review Questions . . . 543
Exercises . . . 544

CHAPTER 15 Classes and Objects . . . 551

Object-Oriented Terminology . . . 552
Defining a Class in C++ . . . 553
Instantiating an Object and Referring to a Public Member . . . 556
Example 1—A Class That Contains a Private Data Member and Public Member Methods . . . 557
Header Files . . . 561
Example 2—A Class That Contains a Parameterized Constructor . . . 562
Example 3—Reusing a Class . . . 565
Example 4—A Class That Contains Overloaded Methods . . . 566
LAB 15-1 Stop and Analyze . . . 569
LAB 15-2 Plan and Create . . . 571
LAB 15-3 Modify . . . 576

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

LAB 15-4 What's Missing? 576
LAB 15-5 Desk-Check 576
LAB 15-6 Debug 578
Chapter Summary 578
Key Terms 579
Review Questions 580
Exercises 581

APPENDIX A C++ Keywords 593

APPENDIX B ASCII Codes 595

APPENDIX C Common Syntax Errors 597

APPENDIX D How To Boxes 599

Index 603

The Answers.pdf and data files can be found online at CengageBrain.com.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Preface

An Introduction to Programming with C++, Eighth Edition uses the C++ programming language to teach programming concepts. This book is designed for a beginning programming course. Although the book provides instructions for using several specific C++ compilers (such as Microsoft® Visual C++®, Dev-C++, and Code::Blocks), it can be used with most C++ compilers, often with little or no modification.

Organization and Coverage

An Introduction to Programming with C++, Eighth Edition contains 15 chapters and several appendices. In the chapters, students with no previous programming experience learn how to plan and create well-structured programs. They also learn how to write programs using the sequence, selection, and repetition structures, as well as how to create and manipulate functions, sequential access files, arrays, strings, classes, and objects.

Approach

An Introduction to Programming with C++, Eighth Edition is distinguished from other textbooks because of its unique approach, which motivates students by demonstrating why they need to learn the concepts and skills presented. Each chapter begins with an introduction to one or more programming concepts. The concepts are illustrated with code examples and sample programs. The sample programs allow the student to observe how the current concept can be used before they are introduced to the next concept. The concepts are taught using standard C++ commands. Following the concept portion in each chapter (except Chapter 1) are six labs: Stop and Analyze, Plan and Create, Modify, What's Missing?, Desk-Check, and Debug. Each lab teaches students how to apply the chapter concepts; however, each does so in a different way.

Features

An Introduction to Programming with C++, Eighth Edition is an exceptional textbook because it also includes the following features:

READ THIS BEFORE YOU BEGIN This section is consistent with Cengage Learning's unequaled commitment to helping instructors introduce technology into the classroom. Technical considerations and assumptions about hardware, software, and default settings are listed in one place to help instructors save time and eliminate unnecessary aggravation.

LABS Each chapter (except Chapter 1) contains six labs that teach students how to apply the concepts taught in the chapter to real-world problems. In the first lab, which is the Stop and Analyze lab, students are expected to stop and analyze an existing program. Students plan and create a program in the Plan and Create lab, which is the second lab. The third lab is the Modify lab. This lab requires students to

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

modify an existing program. In the fourth lab, which is the new What's Missing? lab, students are asked to find one or more missing instructions in a program. However, before they can accomplish this task, they must put the existing instructions in the proper order. The fifth lab is the Desk-Check lab, in which students follow the logic of a program by desk-checking it. The sixth lab is the Debug lab. This lab gives students an opportunity to find and correct the errors in an existing program. Answers to the labs are provided in the Answers.pdf file available at CengageBrain.com. Providing the answers allows students to determine whether they have mastered the material covered in the chapter.

HOW TO BOXES The How To boxes in each chapter summarize important concepts and provide a quick reference for students. The How To boxes that introduce new statements, operators, stream manipulators, or functions contain the syntax and examples of using the syntax.

STANDARD C++ SYNTAX Like the previous edition of the book, this edition uses the standard C++ syntax in the examples, sample programs, and exercises in each chapter.

PSEUDOCODE AND FLOWCHARTS Both planning tools are shown for many of the programs within the chapters.

TIP These notes provide additional information about the current concept. Examples include alternative ways of writing statements, warnings about common mistakes made when using a particular command, and reminders of related concepts learned in previous chapters.

MINI-QUIZZES Mini-Quizzes are strategically placed to test students' knowledge at various points in each chapter. Answers to the quiz questions are provided in the Answers.pdf file, allowing students to determine whether they have mastered the material covered thus far before continuing with the chapter.

WANT MORE INFO? FILES These notes direct students to files that accompany each chapter in the book. The files contain additional examples and further explanations of the concepts covered in the chapter. The files are in PDF format and are available online at CengageBrain.com. Search for the ISBN associated with your book (from the back cover of your book) using the search box at the top of the page. This will take you to the product page where free companion resources can be found.

SUMMARY A Summary section follows the labs in each chapter. The Summary section recaps the programming concepts and commands covered in the chapter.

KEY TERMS Following the Summary section in each chapter is a listing of the key terms introduced throughout the chapter, along with their definitions.

REVIEW QUESTIONS Review Questions follow the Key Terms section in each chapter. The Review Questions test the students' understanding of what they learned in the chapter.

PAPER AND PENCIL EXERCISES The Review Questions are followed by Pencil and Paper Exercises, which are designated as TRY THIS, MODIFY THIS, INTRODUCTORY, INTERMEDIATE, ADVANCED, and SWAT THE BUGS. The answers to the TRY THIS Exercises are provided at the end of the chapter. The ADVANCED Exercises provide practice in applying cumulative programming knowledge or allow students to explore alternative solutions to programming tasks. The SWAT THE BUGS Exercises provide an opportunity for students to detect and correct errors in one or more lines of code.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

COMPUTER EXERCISES The Computer Exercises provide students with additional practice of the skills and concepts they learned in the chapter. The Computer Exercises are designated as TRY THIS, MODIFY THIS, INTRODUCTORY, INTERMEDIATE, ADVANCED, and SWAT THE BUGS. The answers to the TRY THIS Exercises are provided at the end of the chapter. The ADVANCED Exercises provide practice in applying cumulative programming knowledge or allow students to explore alternative solutions to programming tasks. The SWAT THE BUGS Exercises provide an opportunity for students to detect and correct errors in an existing program.

New to this Edition!

ANSWERS.PDF FILE The answers to the Mini-Quizzes and Labs are now contained in the Answers.pdf file (rather than in Appendix A); this file is available to students at CengageBrain.com.

NEW EXAMPLES, PROGRAMS, LABS, QUESTIONS, AND EXERCISES The chapters contain new code examples, sample programs, Labs, Review Questions, and Exercises.

WHAT'S MISSING? LAB The chapters contain a new Lab called What's Missing?. In the What's Missing? Lab, students must determine the one or more missing instructions in a program. However, before they can do this, they must first put the existing instructions in the proper order.

VIDEOS These notes direct students to videos that accompany each chapter in the book. Many of the videos have been revised from the previous edition. The videos explain and/or demonstrate one or more of the chapter's concepts. The videos are available online at CengageBrain.com. Search for the ISBN associated with your book (from the back cover of your book) using the search box at the top of the page. This will take you to the product page where free companion resources can be found.

INSTALLATION VIDEOS These videos, which have been revised from the previous edition, show students how to install various C++ compilers (such as Microsoft Visual C++, Dev-C++, and Code::Blocks). The videos are named Ch04-Installation *developmentTool*, where *developmentTool* is the name of the C++ development tool covered in the video. The videos are available online at CengageBrain.com. Search for the ISBN associated with your book (from the back cover of your book) using the search box at the top of the page. This will take you to the product page where free companion resources can be found.

STEP-BY-STEP INSTRUCTIONS This book is accompanied by files that contain step-by-step instructions for completing Labs 4-2, 4-3, 4-4, 4-6, 5-2, 5-3, and 5-6 using various C++ compilers. The files, which have been revised from the previous edition, are named Ch04-Lab4-*X developmentTool*.pdf and Ch05-Lab5-*X developmentTool*.pdf, where *X* represents the lab number, and *developmentTool* is the name of the C++ development tool covered in the file. The files are in PDF format and are available online at *www.cengagebrain.com*. Search for the ISBN associated with your book (from the back cover of your book) using the search box at the top of the page. This will take you to the product page where free companion resources can be found.

APPENDICES Appendices B, C, D, and E are now Appendices A, B, C, and D. The information in Appendix A from the previous edition is now contained in the Answers.pdf file.

POW FUNCTION The pow function is now covered along with the built-in value-returning functions in Chapter 9 (rather than in Chapter 8).

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Instructor Resources

The following resources are available on the Instructor Companion Site (sso.cengage.com) to instructors who have adopted this book. Search for this title by ISBN, title, author, or keyword. From the Product Overview page, select the Instructor's Companion Site link to access your complementary resources.

INSTRUCTOR'S MANUAL The Instructor's Manual follows the text chapter by chapter to assist you in planning and organizing an effective, engaging course. The manual includes learning objectives, chapter overviews, ideas for classroom activities, and additional resources. A sample course **Syllabus** is also available.

TEST BANK Cengage Learning Testing Powered by Cognero is a flexible, online system that allows you to:

- author, edit, and manage test bank content from multiple Cengage Learning solutions
- create multiple test versions in an instant
- deliver tests from your LMS, your classroom or wherever you want

POWERPOINT PRESENTATIONS This book comes with Microsoft PowerPoint slides for each chapter. These are included as a teaching aid for classroom presentation, to make available to students on the network for chapter review, or to be printed for classroom distribution. Instructors are encouraged to customize the slides to fit their course needs, and may add slides to cover additional topics using the complete **Figure Files** from the text, also available on the Instructor Companion Site.

SOLUTION FILES Solutions to the Labs, Review Questions, Pencil and Paper Exercises, and Computer Exercises are available. The Solution Files also contain the sample programs that appear in the figures throughout the book.

DATA FILES Data Files are required to complete many Labs and Computer Exercises in this book. They are available on the Instructor Companion Site as well as on CengageBrain.com.

Acknowledgments

Writing a book is a team effort rather than an individual one. I would like to take this opportunity to thank my team, especially Alyssa Pratt (Senior Content Developer), Jennifer K. Feltri-George (Senior Content Project Manager), Marisa Taylor (Senior Project Manager), and Nicole Ashton, Serge Palladino, Chris Scriver (Quality Assurance). Thank you for your support, enthusiasm, patience, and hard work; it made a difficult task much easier. Last, but certainly not least, I want to thank Fred D'Angelo, Pima Community College East Campus; Charles Nelson, Rock Valley College; and Mark Shellman, Gaston College for their invaluable ideas and comments.

Diane Zak

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Read This Before You Begin

Technical Information

Data Files

You will need data files to complete the Labs and Computer Exercises in this book. Your instructor may provide the data files to you. You may obtain the files electronically at CengageBrain.com. Search for the ISBN associated with your book (from the back cover of your text) using the search box at the top of the page. This will take you to the product page where free companion resources can be found.

Each chapter in this book has its own set of data files, which are stored in a separate folder within the Cpp8 folder. The files for Chapter 4 are stored in the Cpp8\Chap04 folder. Similarly, the files for Chapter 5 are stored in the Cpp8\Chap05 folder. Throughout this book, you will be instructed to open files from or save files to these folders.

You can use a computer in your school lab or your own computer to complete the Labs and Computer Exercises in this book.

Using Your Own Computer

To use your own computer to complete the Labs and Computer Exercises in this book, you will need a C++ compiler. This book is accompanied by videos that show students how to install various C++ compilers (such as Microsoft Visual C++, Dev-C++, and Code::Blocks). The videos are named Ch04-Installation *development Tool*, where *development Tool* is the name of the C++ development tool covered in the video. You may obtain the files electronically at CengageBrain.com. Search for the ISBN associated with your book (from the back cover of your book) using the search box at the top of the page. This will take you to the product page where free companion resources can be found.

The book was written and Quality Assurance tested using Microsoft Visual C++ in Visual Studio Ultimate 2015. It also was tested using Code::Blocks and Dev-C++. However, the book can be used with most C++ compilers, often with little or no modification. At the time of this writing, you can download a free copy of the Community Edition of Visual Studio 2015, which contains the Visual C++ compiler, at https://www.visualstudio.com/en-us/downloads/visual-studio-2015-downloads-vs.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

CHAPTER 1

An Introduction to Programming

After studying Chapter 1, you should be able to:

- Define the terminology used in programming
- Explain the tasks performed by a programmer
- Understand the employment opportunities for programmers and software engineers
- Explain the history of programming languages
- Explain the sequence, selection, and repetition structures
- Write simple algorithms using the sequence, selection, and repetition structures

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Programming a Computer

In essence, the word **programming** means *giving a mechanism the directions to accomplish a task*. If you are like most people, you've already programmed several mechanisms, such as your digital video recorder (DVR), cell phone, or coffee maker. Like these devices, a computer also is a mechanism that can be programmed.

The directions (typically called instructions) given to a computer are called **computer programs** or, more simply, **programs**. The people who write programs are called **programmers**. Programmers use a variety of special languages, called **programming languages**, to communicate with the computer. Some popular programming languages are C++, Visual Basic, C#, Java, and Python. In this book, you will use the C++ programming language.

The Programmer's Job

Ch01-Programmers

When a company has a problem that requires a computer solution, typically it is a programmer who comes to the rescue. The programmer might be an employee of the company; or he or she might be a freelance programmer, which is a programmer who works on temporary contracts rather than for a long-term employer.

To begin the process of developing a program, the programmer meets with the user, who is the person (or persons) responsible for describing the problem. In many cases, this person or persons also will eventually use the solution. Depending on the complexity of the problem, multiple programmers may be involved, and they may need to meet with the user several times. Programming teams often contain subject matter experts, who may or may not be programmers. For example, an accountant might be part of a team working on a program that requires accounting expertise. The purpose of the initial meetings with the user is to determine the exact problem and to agree on a solution.

After the programmer and user agree on the solution, the programmer begins converting the solution into a computer program. During the conversion phase, the programmer meets periodically with the user to determine whether the program fulfills the user's needs and to refine any details of the solution. When the user is satisfied that the program does what he or she wants it to do, the programmer rigorously tests the program with sample data before releasing it to the user, who will test it further to verify that it correctly solves the problem. In many cases, the programmer also provides the user with a manual that explains how to use the program. As this process indicates, the creation of a good computer solution to a problem—in other words, the creation of a good program—requires a great deal of interaction between the programmer and the user.

Employment Opportunities

Ch01-Programmer Qualities

When searching for a job in computer programming, you will encounter ads for "computer programmers" as well as for "computer software engineers." Although job titles and descriptions vary, computer software engineers typically are responsible for designing an appropriate solution to a user's problem, while computer programmers are responsible for translating the solution into a language that the computer can understand—a process called **coding**. Software engineering is a higher-level position that requires the ability to envision solutions. Using a construction analogy, software engineers are the architects, while programmers are the carpenters.

Keep in mind that, depending on the employer and the size and complexity of the user's problem, the design and coding tasks may be performed by the same employee, no matter what his or her job title is. In other words, it's not unusual for a software engineer to code his or her solution, just as it's not unusual for a programmer to have designed the solution he or she is coding.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Programmers and software engineers need to have strong problem-solving and analytical skills, as well as the ability to communicate effectively with team members, end users, and other non-technical personnel. Typically, computer software engineers are expected to have at least a bachelor's degree in software engineering, computer science, or mathematics, along with practical work experience, especially in the industry in which they are employed. Computer programmers usually need at least an associate's degree in computer science, mathematics, or information systems, as well as proficiency in one or more programming languages.

Computer programmers and software engineers are employed by companies in almost every industry, such as telecommunications companies, software publishers, financial institutions, insurance carriers, educational institutions, and government agencies. The Bureau of Labor Statistics predicts that employment of computer software engineers will increase by 22 percent from 2012 to 2022. The employment of computer programmers, on the other hand, will increase by 8 percent over the same period. In addition, consulting opportunities for freelance programmers and software engineers are expected to increase as companies look for ways to reduce their payroll expenses.

There is a great deal of competition for programming and software engineering jobs, so jobseekers will need to keep up to date with the latest programming languages and technologies. A competitive edge may be gained by obtaining vendor-specific or language-specific certifications, as well as knowledge of a prospective employer's business. More information about computer programmers and computer software engineers can be found on the Bureau of Labor Statistics Web site at *www.bls.gov*.

A Brief History of Programming Languages

Just as human beings communicate with each other through the use of languages such as English, Spanish, Hindi, and Chinese, programmers use a variety of programming languages to communicate with the computer. In the next sections, you will follow the progression of programming languages from machine languages to assembly languages, and then to high-level languages.

Ch01-History

Machine Languages

Within a computer, all data is represented by microscopic electronic switches that can be either off or on. The off switch is designated by a *0*, and the on switch is designated by a *1*. Because computers can understand only these on and off switches, the first programmers had to write the program instructions using nothing but combinations of *0*s and *1*s; for example, a program might contain the instruction `00101 10001 10000`. Instructions written in *0*s and *1*s are called **machine language** or **machine code**. The machine languages (each type of machine has its own language) represent the only way to communicate directly with the computer. As you can imagine, programming in machine language is very tedious and error-prone and requires highly trained programmers.

Assembly Languages

Slightly more advanced programming languages are called assembly languages. The **assembly languages** simplify the programmer's job by allowing the programmer to use mnemonics in place of the *0*s and *1*s in the program. **Mnemonics** are memory aids—in this case, alphabetic abbreviations for instructions. For example, most assembly languages use the mnemonic `ADD` to represent an add operation and the mnemonic `MUL` to represent a multiply operation. An example of an instruction written in an assembly language is `ADD bx, ax`.

Programs written in an assembly language require an **assembler**, which also is a program, to convert the assembly instructions into machine code—the *0*s and *1*s the computer can

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

understand. Although it is much easier to write programs in assembly language than in machine language, programming in assembly language still is tedious and requires highly trained programmers. Programs written in assembly language are machine specific and usually must be rewritten in a different assembly language to run on different computers.

High-Level Languages

High-level languages represent the next major development in programming languages. **High-level languages** are a vast improvement over machine and assembly languages because they allow the programmer to use instructions that more closely resemble the English language. An example of an instruction written in a high-level language is `grossPay = hours * rate`. In addition, high-level languages are more machine independent than are machine and assembly languages. As a result, programs written in a high-level language can be used on many different types of computers.

Programs written in a high-level language usually require a compiler, which also is a program, to convert the English-like instructions into the *0*s and *1*s the computer can understand. Some high-level languages also offer an additional program called an interpreter. Unlike a **compiler**, which translates all of a program's high-level instructions before running the program, an **interpreter** translates the instructions line by line as the program is running.

Like their predecessors, the first high-level languages were used to create procedure-oriented programs. When writing a **procedure-oriented program**, the programmer concentrates on the major tasks that the program needs to perform. A payroll program, for example, typically performs several major tasks, such as inputting the employee data, calculating the gross pay, calculating the taxes, calculating the net pay, and outputting a paycheck. The programmer must instruct the computer every step of the way, from the start of the task to its completion. In a procedure-oriented program, the programmer determines and controls the order in which the computer processes the instructions. In other words, the programmer must determine not only the proper instructions to give the computer but the correct sequence of those instructions as well. Examples of high-level languages used to create procedure-oriented programs include COBOL (Common Business-Oriented Language), BASIC (Beginner's All-Purpose Symbolic Instruction Code), and C.

Most objects in an object-oriented program are designed to perform multiple tasks. These tasks are programmed using the same techniques used in procedure-oriented programming.

More advanced high-level languages can be used to create object-oriented programs in addition to procedure-oriented ones. Different from a procedure-oriented program, which focuses on the individual tasks the program must perform, an **object-oriented program** requires the programmer to focus on the objects that the program can use to accomplish its goal. The objects can take on many different forms. For example, programs written for the Windows environment typically use objects such as check boxes, list boxes, and buttons. A payroll program, on the other hand, might utilize objects found in the real world, such as a time card object, an employee object, or a check object.

Because each object in an object-oriented program is viewed as an independent unit, an object can be used in more than one program, usually with little or no modification. A check object used in a payroll program, for example, also can be used in a sales revenue program (which receives checks from customers) and an accounts payable program (which issues checks to creditors). The ability to use an object for more than one purpose enables code reuse, which saves programming time and money—an advantage that contributes to the popularity of

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

object-oriented programming. Examples of high-level languages that can be used to create both procedure-oriented and object-oriented programs include C++, Visual Basic, Java, and C#. In this book, you will learn how to use the C++ programming language to create procedure-oriented and object-oriented programs.

Mini-Quiz 1-1

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

1. Instructions written in *0*s and *1*s are called _____ language.
2. When writing a(n) _____ program, the programmer concentrates on the major tasks needed to accomplish a goal.
   a. procedure-oriented
   b. object-oriented
3. When writing a(n) _____ program, the programmer breaks up a problem into interacting objects.
   a. procedure-oriented
   b. object-oriented
4. Most high-level languages use a(n) _____ to translate the instructions into a language that the computer can understand.

Control Structures

All computer programs, no matter how simple or how complex, are written using one or more of three basic structures: sequence, selection, and repetition. These structures are called **control structures** or **logic structures** because they control the flow of a program's logic. In other words, they control the order in which the computer executes the program's instructions. You will use the sequence structure in every program you write. In most programs, you also will use the selection and repetition structures. This chapter gives you an introduction to the three control structures. More detailed information about each structure, as well as how to implement these structures using the C++ language, is provided in subsequent chapters.

The Sequence Structure

You use the sequence structure each time you follow a set of step-by-step instructions, in order, from beginning to end. The instructions might be a recipe for making chocolate chip cookies. Or, they might be the MapQuest directions to your favorite restaurant. They could also be the instructions for assembling a robot, which are shown in Figure 1-1. The instructions shown in the figure are called an **algorithm**, which is a set of step-by-step instructions for accomplishing a task.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

1. get one head, one body, one right arm, one left arm, one right leg, and one left leg
2. attach head to top of body
3. attach right leg to bottom-right portion of body
4. attach left leg to bottom-left portion of body
5. attach right arm to right side of body
6. attach left arm to left side of body

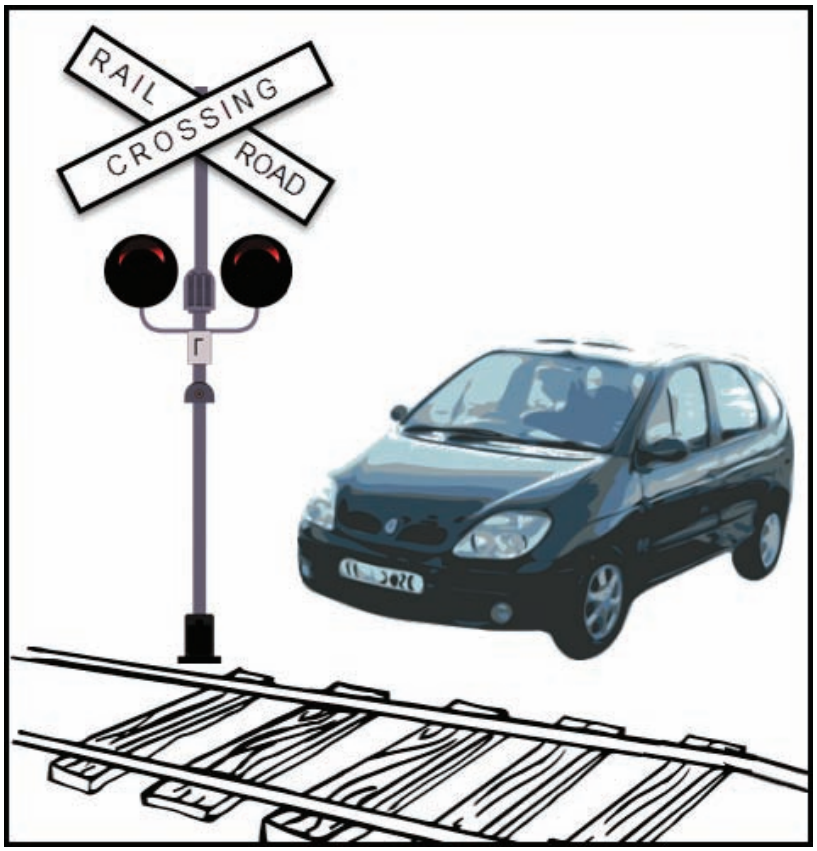
Figure 1-1 An example of the sequence structure

In a computer program, the **sequence structure** directs the computer to process the program instructions, one after another, in the order listed in the program. You will find the sequence structure in every program.

The Selection Structure

The **selection structure**, also called the **decision structure**, indicates that a decision (based on some condition) needs to be made, followed by an appropriate action derived from that decision. You use the selection structure every time you drive your car and approach a railroad crossing. Your decision, as well as the appropriate action, is based on whether the crossing signals (flashing lights and ringing bells) are on or off, as indicated in Figure 1-2.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.




```
1. approach the railroad crossing
2. if (the crossing signals are on)
        stop your car before crossing the railroad tracks
        wait for the train to go by and the crossing signals to turn off
   end if
3. proceed with caution over the railroad tracks
```

condition

instructions indented in the true path

Figure 1-2 An example of the selection structure

The portion within the parentheses in the figure is called the condition and specifies the decision that you must make. The condition in a selection structure must result in either a true or a false answer. In this case, either the crossing signals are on (true) or they are not on (false).

Notice that the two instructions within the selection structure are indented. Indenting in this manner indicates that the instructions should be followed only when the crossing signals are on—in other words, only when the condition results in an answer of true. The instructions to be followed when a selection structure's condition evaluates to true are referred to as the structure's true path. The *end if* instruction in Figure 1-2 denotes the end of the selection structure.

Figure 1-3 shows how the selection structure can be used in a game program. In this game, our superhero gets one shot at the villain. He needs to raise his right arm before taking the shot. If he hits the villain, he should say "Got Him" and then lower his right arm. If he doesn't hit the villain, he should say "Missed Him" before lowering his right arm.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

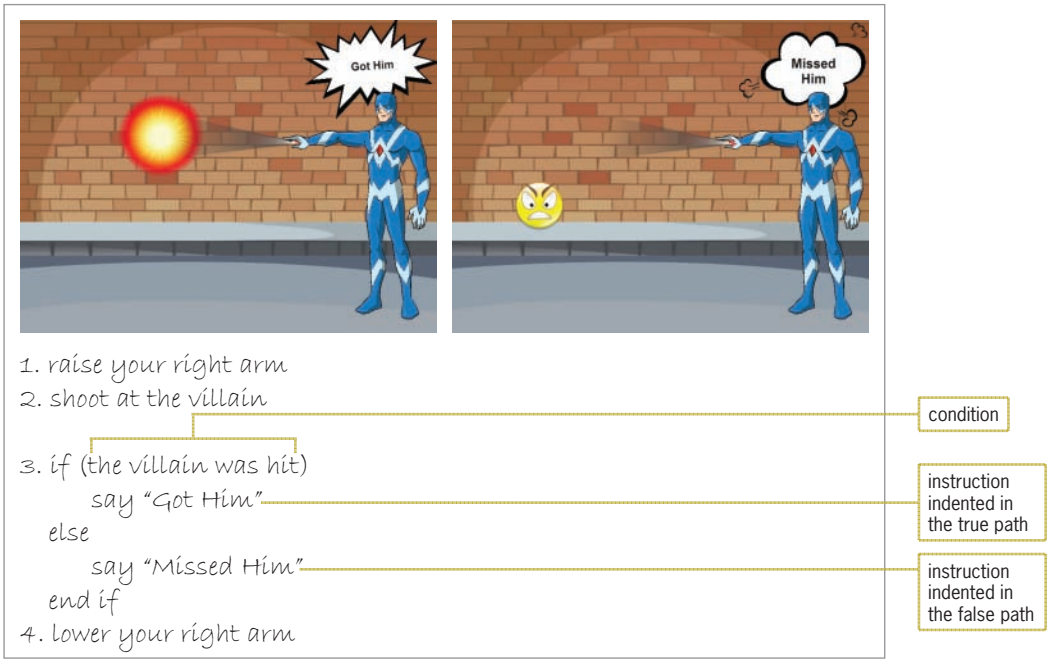


Figure 1-3 Another example of the selection structure
Image by Diane Zak; created with Reallusion CrazyTalk Animator

Unlike the selection structure from Figure 1-2, which requires two specific actions to be taken only when the structure's condition evaluates to true, the selection structure in Figure 1-3 requires our superhero to take one action when the condition evaluates to true but a different action when it evaluates to false. In other words, the selection structure in Figure 1-3 has both a true path and a false path. The *else* instruction marks the beginning of the false path.

Notice that the instruction in each path is indented. Indenting in this manner clearly indicates the instruction to be followed when the condition evaluates to true (the villain was hit), as well as the one to be followed when the condition evaluates to false (the villain was not hit). Although both paths in Figure 1-3's selection structure contain only one instruction, each can contain many instructions.

When used in a computer program, the selection structure alerts the computer that a decision needs to be made, and it provides the appropriate action(s) to take based on the result of that decision.

The Repetition Structure

The last of the three control structures is the **repetition structure**, which indicates that one or more instructions need to be repeated until some condition is met. You will find the repetition structure in many recipes; some examples are shown in Figure 1-4. Notice that the condition can be phrased in several different ways.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

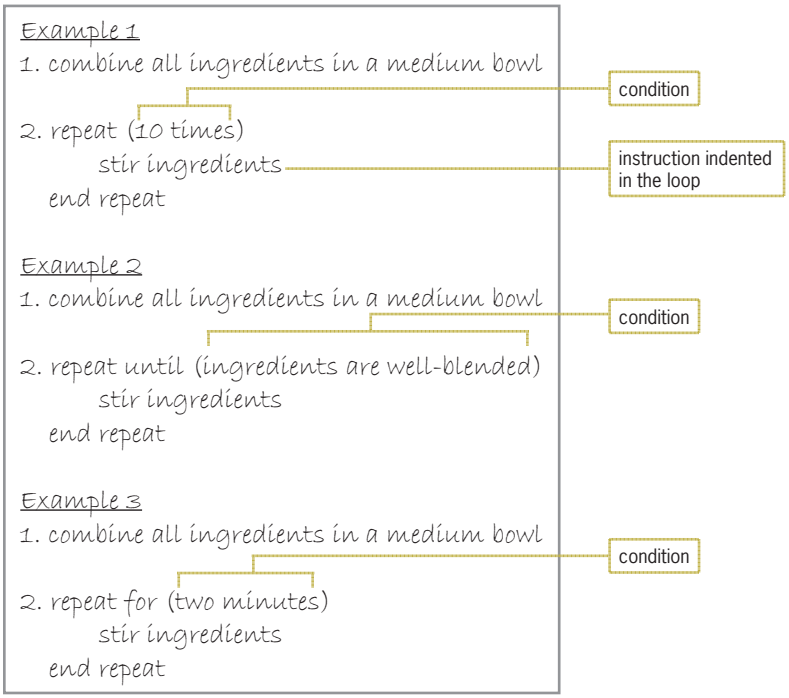


Figure 1-4 Examples of the repetition structure

When used in a program, the repetition structure directs the computer to repeat one or more instructions until some condition is met, at which time the computer should stop repeating the instructions. The repetition structure is also referred to as a **loop** or an **iteration**.

Mini-Quiz 1-2

1. What are the three basic control structures?
2. Which control structure is contained in all programs?
3. The step-by-step instructions that accomplish a task are called a(n) _____.
4. Which structure tells the computer to repeat one or more instructions in a program?
5. Which structure ends when its condition has been met?
6. The _____ structure, also called the decision structure, instructs the computer to evaluate a condition and then follow one of two paths based on the result of the evaluation.

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The answers to the labs are contained in the Answers.pdf file.

LAB 1-1 Stop and Analyze

Study the algorithm shown in Figure 1-5, and then answer the questions.

```
repeat for (each customer buying a TV)
      enter the original price of the TV
      calculate the discount by multiplying the original price by 15%
      calculate the total due by subtracting the discount from the original price
      print a bill showing the original price, discount, and total due
end repeat
```

Figure 1-5 Algorithm for Lab 1-1

QUESTIONS

1. Which control structures are used in the algorithm?
2. What will the algorithm print when the price of the TV is $2,100?
3. How would you modify the algorithm so that it can be used for only the first 10 customers buying a TV?
4. How would you modify the algorithm so that it allows the user to also enter the discount rate and then uses that rate to calculate the discount?

LAB 1-2 Plan and Create

The 10 salespeople at Harkins Company are paid a 10% bonus when they sell more than $10,000 in product; otherwise, they receive a 5% bonus. Create an appropriate algorithm using only the instructions shown in Figure 1-6.

```
display the salesperson's name and bonus
else
end if
end repeat
enter the salesperson's name and sales
if (the sales are greater than 10,000)
calculate the bonus by multiplying the sales by 5%
calculate the bonus by multiplying the sales by 10%
repeat (10 times)
```

Figure 1-6 Instructions for Lab 1-2

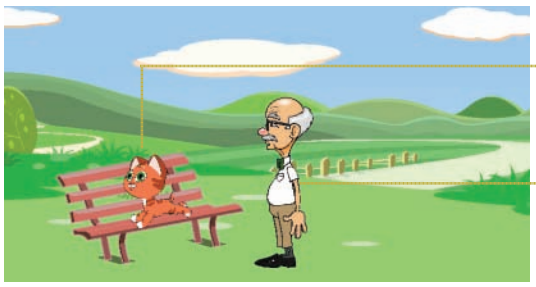
Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

LAB 1-3 Modify

Modify the algorithm shown in Figure 1-5 so that it gives a 25% discount to customers who are also employees of the store; all other customers receive a 15% discount.

LAB 1-4 What's Missing?

Harold wants to sit down on the park bench, but his cat Ginger may or may not be already seated there. Put the instructions shown in Figure 1-7 in the proper order, and then determine the one or more missing instructions.




```
end if
end repeat
end repeat
gently shove Ginger off the bench
if (Ginger is on the bench)
repeat (2 times)
sit down on the bench
turn left 90 degrees
walk forward one complete step
```

Figure 1-7 Instructions for Lab 1-4
Image by Diane Zak; created with Reallusion CrazyTalk Animator

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Chapter Summary

Programs are the step-by-step instructions that tell a computer how to perform a task. Programmers, the people who write computer programs, use various programming languages to communicate with the computer. The first programming languages were machine languages, also called machine code. The assembly languages came next, followed by the high-level languages. The first high-level languages were used to create procedure-oriented programs. More advanced high-level languages are used to create object-oriented programs, as well as procedure-oriented ones.

An algorithm is the set of step-by-step instructions that accomplish a task. The algorithms for all computer programs contain one or more of the following three control structures: sequence, selection, and repetition. The control structures, also called logic structures, are so named because they control the flow of a program's logic.

The sequence structure directs the computer to process the program instructions, one after another, in the order listed in the program. The selection structure, also called the decision structure, directs the computer to evaluate a condition and then select an appropriate action based on the result of that evaluation. The repetition structure directs the computer to repeat one or more program instructions until some condition is met. The sequence structure is used in all programs. Most programs also contain both the selection and repetition structures.

Key Terms

Algorithm—the set of step-by-step instructions that accomplish a task

Assembler—a program that converts assembly instructions into machine code

Assembly languages—programming languages that use mnemonics, such as `ADD`

Coding—the process of translating a solution into a language that the computer can understand

Compiler—a program that converts high-level instructions into a language that the computer can understand; unlike an interpreter, a compiler converts all of a program's instructions before running the program

Computer programs—the directions given to computers; also called programs

Control structures—the structures that control the flow of a program's logic; also called logic structures; sequence, selection, and repetition

Decision structure—another term for the selection structure

High-level languages—programming languages whose instructions more closely resemble the English language

Interpreter—a program that converts high-level instructions into a language that the computer can understand; unlike a compiler, an interpreter converts a program's instructions line by line as the program is running

Iteration—another term for the repetition structure

Logic structures—another term for control structures

Loop—another term for the repetition structure

Machine code—another term for machine language

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Machine language—computer instructions written in *0*s and *1*s; also called machine code

Mnemonics—the alphabetic abbreviations used to represent instructions in assembly languages

Object-oriented program—a program designed by focusing on the objects that the program could use to accomplish its goal

Procedure-oriented program—a program designed by focusing on the individual tasks to be performed

Programmers—the people who write computer programs

Programming—giving a mechanism the directions to accomplish a task

Programming languages—languages used to communicate with a computer

Programs—the directions given to computers; also called computer programs

Repetition structure—the control structure that directs the computer to repeat one or more instructions until some condition is met, at which time the computer should stop repeating the instructions; also called a loop or an iteration

Selection structure—the control structure that directs the computer to make a decision and then take the appropriate action based on the result of that decision; also called the decision structure

Sequence structure—the control structure that directs the computer to process each instruction in the order listed in the program

Review Questions

1. Which of the following is not a programming control structure?
   a. repetition
   b. selection
   c. sequence
   d. sorting

2. Which of the following control structures is used in every program?
   a. repetition
   b. selection
   c. sequence
   d. switching

3. The set of instructions for adding together two numbers is an example of the _____ structure.
   a. control
   b. repetition
   c. selection
   d. sequence

4. The set of step-by-step instructions that solve a problem is called _____ .
   a. an algorithm
   b. a list
   c. a plan
   d. a sequential structure

5. The instruction “Brush your hair 5 times” is an example of which structure?
   a. control
   b. repetition
   c. selection
   d. sequence

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

6. The instruction "If it's dark, turn the light on" is an example of which structure?
   a. control
   b. repetition
   c. selection
   d. sequence

7. Which control structure would an algorithm use to determine whether a credit card holder is over his credit limit?
   a. repetition
   b. selection
   c. both repetition and selection

8. Which control structure would an algorithm use to calculate a 5% commission for each of a company's salespeople?
   a. repetition
   b. selection
   c. both repetition and selection

9. A company pays a 3% annual bonus to employees who have been with the company more than 5 years; other employees receive a 1% bonus. Which control structure(s) would an algorithm use to calculate every employee's bonus?
   a. repetition
   b. selection
   c. both repetition and selection

10. Which control structure would an algorithm use to determine whether a customer is entitled to a senior discount?
    a. repetition
    b. selection
    c. both repetition and selection

Exercises

Pencil and Paper

TRY THIS

1. Harold is five steps away from his cat Ginger, who is an unknown distance away from a chair, as illustrated in Figure 1-8. Using only the instructions listed in the figure, create an algorithm that directs Harold to step over Ginger and sit in the chair. You may use an instruction more than once. In the `repeat (x times)` instruction, replace `x` with the appropriate number of times you want the loop instruction(s) repeated. Be sure to indent the instructions appropriately. (The answers to TRY THIS Exercises are located at the end of the chapter.)

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

```
end repeat
repeat (x times)
repeat until (you are directly in front of the chair)
sit down in the chair
step over Ginger
turn left 90 degrees
walk forward one complete step
```

Figure 1-8
Image by Diane Zak; created with Reallusion CrazyTalk Animator

TRY THIS

2. A store gives a 15% discount to customers who are at least 55 years old, and a 10% discount to all other customers. Using only the instructions shown in Figure 1-9, write an algorithm that displays the amount of money a customer owes. Be sure to indent the instructions appropriately. (The answers to TRY THIS Exercises are located at the end of the chapter.)

```
calculate the amount due by subtracting the discount from the sales amount
calculate the discount by multiplying the sales amount by 10%
calculate the discount by multiplying the sales amount by 15%
display the amount due
else
end if
enter the customer's age and the sales amount
if (the customer's age is less than 55)
```

Figure 1-9

MODIFY THIS

3. Modify the algorithm shown earlier in Figure 1-5 so that it gives a 25% discount if the customer buying a TV is an employee of the store; all other customers buying a TV should receive a 15% discount.

INTRODUCTORY

4. Using only the instructions shown in Figure 1-10, create an algorithm that displays the average of four numbers. Be sure to indent the instructions appropriately.

```
calculate the average by dividing the sum by 4
calculate the sum by adding the number to the sum
display the average
end repeat
enter a number
repeat (4 times)
```

Figure 1-10

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

INTRODUCTORY

5. You have just purchased a new personal computer system. Before putting the system components together, you read the instruction booklet that came with the system. The booklet contains a list of the components that you should have received. The booklet advises you to verify that you received all of the components by matching those that you received with those on the list. If a component was received, you should cross its name off the list; otherwise, you should draw a circle around the component name in the list. Using only the instructions listed in Figure 1-11, create an algorithm that shows the steps you should take to verify that you received the correct components. Be sure to indent the instructions appropriately.

```
circle the component name on the list
cross the component name off the list
else
end if
end repeat
if (the component was received)
read the component name from the list
repeat for (each component name on the list)
search for the component
```

Figure 1-11

INTERMEDIATE

6. Using only the instructions shown in Figure 1-12, write two versions of an algorithm that displays the total amount each customer owes for concert tickets. Be sure to indent the instructions appropriately.

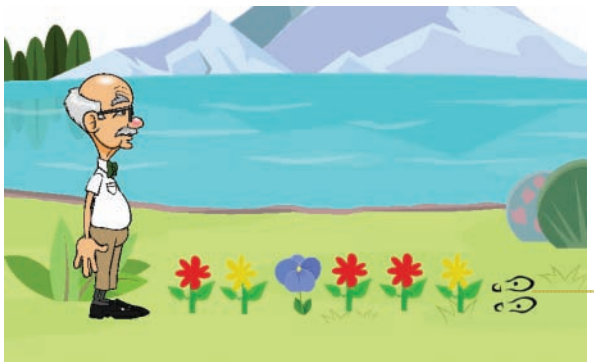
```
calculate the amount owed by multiplying the number of tickets by $35
display the amount owed
display the message "You can purchase up to 4 tickets only."
else
end if
end repeat
enter the number of tickets
if (the number of tickets is greater than 4)
if (the number of tickets is less than or equal to 4)
repeat for (each customer)
```

Figure 1-12

INTERMEDIATE

7. Harold is standing in front of a flowerbed that contains six flowers, as illustrated in Figure 1-13. Create an algorithm that directs Harold to pick the flowers as he walks to the other side of the flowerbed. He should pick all red flowers with his right hand. Flowers that are not red should be picked with his left hand. Use only the instructions shown in the figure; however, an instruction can be used more than once. Be sure to create an algorithm that will work for any combination of colored flowers.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.




```
else
end if
end repeat
if (the flower is red)
pick the flower with your left hand
pick the flower with your right hand
repeat (x times)
walk forward one complete step
```

Figure 1-13
Image by Diane Zak; created with Reallusion CrazyTalk Animator

ADVANCED

8. The algorithm shown in Figure 1-14 should calculate and print the gross pay for five workers; however, some of the instructions are missing from the algorithm. Complete the algorithm. If an employee works more than 40 hours, he or she should receive time and one-half of his or her pay rate for the hours worked over 40.

```
_____
      enter the employee's name, hours worked, and pay rate
      _____
            calculate gross pay = hours worked times pay rate
      else
            calculate regular pay = pay rate times 40
            calculate overtime hours = hours worked minus 40
            calculate overtime pay = _____
            calculate gross pay = _____
      end if
      print the employee's name and gross pay
end repeat
```

Figure 1-14

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

ADVANCED

9. All employees at Kranston Sports Inc. are paid based on an annual salary rather than an hourly wage. However, some employees are paid weekly, while others are paid every other week (biweekly). Employees paid weekly receive 52 paychecks; employees paid biweekly receive 26 paychecks. The algorithm shown in Figure 1-15 should calculate and display the gross pay for each employee; however, some of the instructions are missing from the algorithm. Complete the algorithm.

```
repeat (for each employee)
    enter the employee's payment schedule and annual salary
    if (the employee's payment schedule is weekly)
        calculate gross pay = _____
    _____
    calculate gross pay = _____
    _____
    _____
_____
```

Figure 1-15

ADVANCED

10. Create an algorithm that tells someone how to evaluate the following expression: 7 * 5 – 20 / 2 + 4 * 2. The / operator means division, and the * operator means multiplication. (As you may remember from your math courses, division and multiplication are performed before addition and subtraction.)

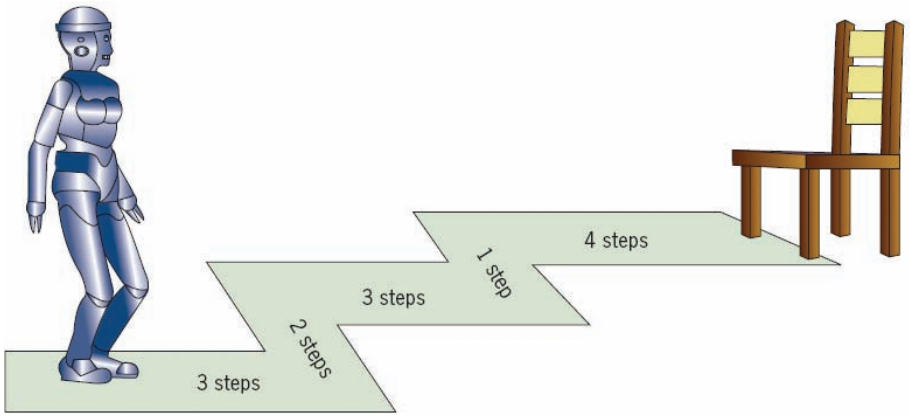
ADVANCED

11. Store A is having a BoGoHo (Buy One, Get One Half Off) sale. Store B is not having a sale, but sometimes its prices are much lower than at Store A. Write an algorithm that determines whether it's cheaper to buy two of the item at Store A or Store B. The algorithm should display either the message "Buy at store A" or the message "Buy at store B". If the prices would be the same at both stores, display the "Buy at store A" message.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

12. The algorithm in Figure 1-16 should get Robin the Robot seated in the chair, but it does not work correctly. Correct the algorithm.

SWAT THE BUGS



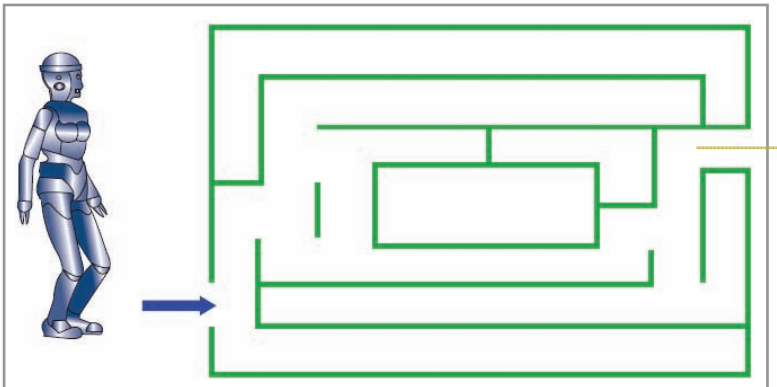

```
1. repeat (3 times)
       walk forward one complete step
   end repeat
2. turn left 90 degrees
3. repeat (2 times)
       walk forward one complete step
   end repeat
4. turn right 90 degrees
5. repeat (2 times)
       walk forward one complete step
   end repeat
6. turn right 90 degrees
7. walk forward one complete step
8. turn right 90 degrees
9. repeat (4 times)
       walk forward one complete step
   end repeat
10. turn right 90 degrees
11. sit down
```

Figure 1-16

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

SWAT THE BUGS

13. The algorithm in Figure 1-17 does not get Robin the Robot through the maze. Correct the algorithm.




```
1. walk into the maze
2. turn left 90 degrees
3. repeat until (you are directly in front of a wall)
       walk forward one complete step
   end repeat
4. turn right 90 degrees
5. repeat until( you are directly in front of a wall)
       walk forward one complete step
   end repeat
6. turn right 90 degrees
7. repeat until (you are directly in front of a wall)
       walk forward one complete step
   end repeat
8. turn right 90 degrees
9. repeat until (you are directly in front of a wall)
       walk forward one complete step
   end repeat
10. turn right 90 degrees
11. repeat until (you are directly in front of a wall)
       walk forward one complete step
   end repeat
12. turn left 90 degrees
13. repeat until (you are directly in front of a wall)
       turn right 90 degrees
   end repeat
14. repeat until (you are out of the maze)
       walk forward one complete step
   end repeat
```

Figure 1-17

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Answers to TRY THIS Exercises

See Figure 1-18.

```
1. repeat (5 times)
       walk forward one complete step
   end repeat
2. step over Ginger
3. repeat until (you are directly in front of the chair)
       walk forward one complete step
   end repeat
4. repeat (2 times)
       turn left 90 degrees
   end repeat
5. sit down in the chair
```

Figure 1-18

See Figure 1-19.

```
1. enter the customer's age and the sales amount
2. if (the customer's age is less than 55)
       calculate the discount by multiplying the sales amount by 10%
   else
       calculate the discount by multiplying the sales amount by 15%
   end if
3. calculate the amount due by subtracting the discount from the sales amount
4. display the amount due
```

Figure 1-19

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

CHAPTER 2

Beginning the Problem-Solving Process

After studying Chapter 2, you should be able to:

- Explain the problem-solving process used to create a computer program
- Analyze a problem
- Complete an IPO chart
- Plan an algorithm using pseudocode and flowcharts
- Desk-check an algorithm

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Problem Solving

This chapter introduces you to the process that programmers follow when solving problems that require a computer solution. Although you may not realize it, you use a similar process to solve hundreds of small problems every day. Because most of these problems occur so often, you typically solve them almost automatically, without giving much thought to the process your brain goes through to arrive at the solutions. Unfortunately, problems that are either complex or unfamiliar usually cannot be solved so easily; most require extensive analysis and planning. Understanding the thought process involved in solving simple and familiar problems will make solving complex or unfamiliar ones easier.

In this chapter, you will explore the thought process that you follow when solving common problems. You will also learn how to use a similar process to create a computer solution to a problem—in other words, how to create a computer program. The computer solutions you create in this chapter will contain the sequence control structure only, in which each instruction is processed in order from beginning to end. Computer solutions requiring the selection structure are covered in Chapters 5 and 6, and those requiring the repetition structure are covered in Chapters 7 and 8.

Solving Everyday Problems

Ch02-Bill Paying

The first step in solving a problem is to analyze the problem. Next, you plan, review, implement, evaluate, and modify (if necessary) the solution. Consider, for example, how you solve the problem of paying a bill that you received in the mail. First, your mind analyzes the problem to identify its important components. One very important component of any problem is the goal of solving the problem. In this case, the goal is to pay the bill. Other important components of a problem are the things that you can use to accomplish the goal. In this case, you will use the bill itself, as well as the preaddressed envelope that came with the bill. You will also use a bank check, pen, return address label, and postage stamp.

After analyzing the problem, your mind plans an algorithm. Recall from Chapter 1 that an algorithm is the set of step-by-step instructions that describe how to accomplish a task. In other words, an algorithm is a solution to a problem. The current problem's algorithm, for example, describes how to use the bill, preaddressed envelope, bank check, pen, return address label, and postage stamp to pay the bill. Figure 2-1 shows a summary of the analysis and planning steps for the bill-paying problem.

result of the analysis step

result of the planning step

Items used to accomplish the goal	Algorithm	Goal
bill preaddressed envelope bank check pen return address label postage stamp	1. use the pen to fill in the bank check's date, payee, numerical amount, and written amount 2. use the pen to sign the bank check 3. use the pen to write the customer account number on the bank check 4. put the return address label on the preaddressed envelope 5. put the postage stamp on the preaddressed envelope 6. put the bill and bank check in the preaddressed envelope 7. seal the preaddressed envelope 8. mail the preaddressed envelope	pay the bill

Figure 2-1 Summary of the analysis and planning steps for the bill-paying problem

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

After planning the algorithm, you review it (in your mind) to verify that it will work as intended. When you are satisfied that the algorithm is correct, you implement it by following each of its instructions in the order indicated. After implementing the algorithm, you evaluate it, and, if necessary, you modify it. In this case, for example, you may decide to include the selection structure shown in instruction 6 in Figure 2-2.

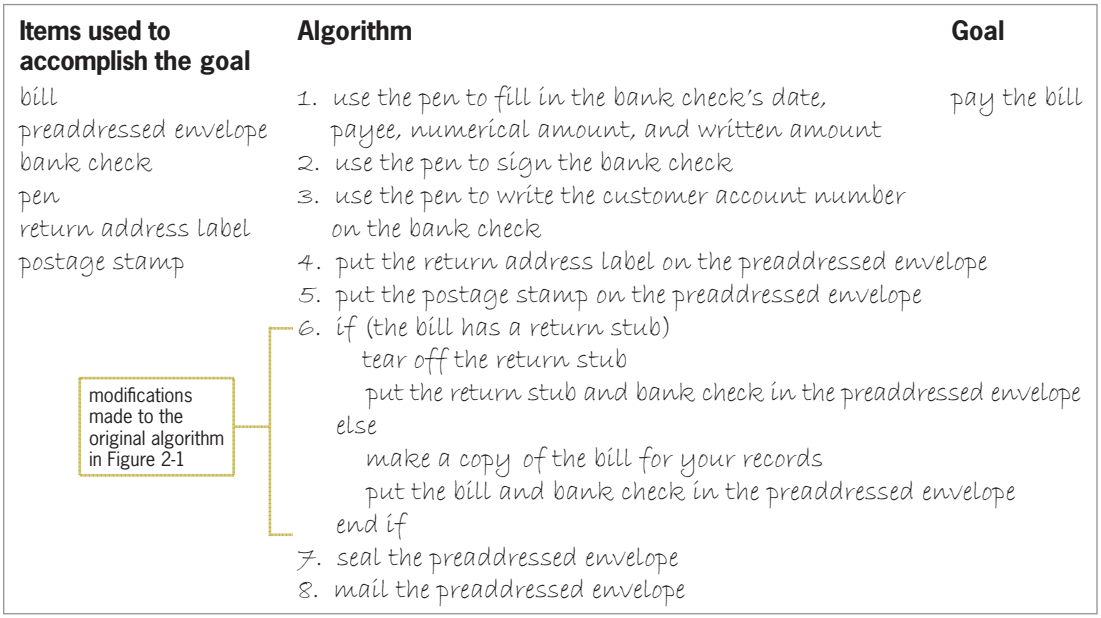


Figure 2-2 Modified algorithm for the bill-paying problem

Creating Computer Solutions to Problems

In the previous section, you learned how you create a solution to a familiar problem. A similar problem-solving process is used to create a computer program, which is simply a solution that is implemented with a computer. Figure 2-3 shows the steps that computer programmers follow when solving problems that require a computer solution. This chapter covers the first three steps; the last three steps are covered in Chapters 3 and 4.

HOW TO Create a Computer Solution to a Problem

1. Analyze the problem
2. Plan the algorithm
3. Desk-check the algorithm
4. Code the algorithm into a program
5. Desk-check the program
6. Evaluate and modify (if necessary) the program

Figure 2-3 How to create a computer solution to a problem

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Step 1—Analyze the Problem

You cannot solve a problem unless you understand it, and you cannot understand a problem unless you analyze it—in other words, unless you identify its important components. The two most important components of any problem are the problem's output and its input. The **output** is the goal of solving the problem, and the **input** is the item or items needed to achieve the goal. When analyzing a problem, you always search first for the output and then for the input. Let's begin by analyzing the problem specification shown in Figure 2-4.

> Addison O'Reilly wants a program that calculates and displays the cost of a 4K Ultra HD TV, which is finally on sale at one of the stores in her area. The program should calculate the cost by multiplying the sale price by the state sales tax rate and then adding the result to the sale price.

Figure 2-4 Problem specification for Addison O'Reilly

When searching for the output, ask yourself the following question: *What does the user want to see displayed on the screen, printed on paper, or stored in a file?* The answer to this question is typically stated as nouns and adjectives in the problem specification. The problem specification in Figure 2-4 indicates that Addison (the program's user) wants to see the cost of the TV displayed on the screen; therefore, the output is the cost. In this context, the word *cost* is a noun.

After determining the output, you then determine the input, which is also usually stated as nouns and adjectives in the problem specification. Here, look for an answer to the following question: *What information will the computer need to know to display, print, or store the output items?* It helps to think about the information that *you* would need to solve the problem manually because the computer will need to know the same information. In this case, to determine the cost, both you and the computer need to know the sale price and the sales tax rate; these items, therefore, are the input. In this context, *sale*, *sales*, and *tax* are adjectives, while *price* and *rate* are nouns. This completes the analysis step for the Addison O'Reilly problem. Some programmers use an **IPO chart** to organize and summarize the results of the analysis step, as shown in Figure 2-5. **IPO** is an acronym for Input, Processing, and Output.

Input	Processing	Output
sale price sales tax rate	Processing items: Algorithm:	cost

Figure 2-5 Partially completed IPO chart showing the input and output items

Hints for Analyzing Problems

Unfortunately, analyzing real-world problems will not be as easy as analyzing the problems found in a textbook. The analysis step is the most difficult of the problem-solving steps, and it requires a lot of time, patience, and effort. If you are having trouble analyzing a problem, try reading the problem specification several times, as it is easy to miss information during the first reading. If the problem still is unclear to you, do not be shy about asking the user for more information. Remember, the more you understand a problem, the easier it will be for you to write a correct and efficient solution.

When reading a problem specification, it helps to use a pencil to lightly cross out the information that you feel is unimportant to the solution, as shown in Figure 2-6. Doing this reduces the amount of information you need to consider in your analysis. If you are not sure whether an item of information is important, ask yourself this question: *If I didn't know this information, could*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

I still solve the problem? If your answer is *Yes*, then the information is superfluous, and you can ignore it. If you later find that the information is important, you can always erase the pencil line.

~~Addison O'Reilly wants a program that~~ calculates and displays the cost ~~of a 4K Ultra HD TV, which is finally on sale at one of the stores in her area. The program should~~ calculate the cost by multiplying the sale price by the state sales tax rate and then adding the result to the sale price.

Figure 2-6 Problem specification with unimportant information crossed out

Some problem specifications, like the one shown in Figure 2-7, are difficult to analyze because they contain incomplete information. In this case, it is clear that the output is the weekly gross pay and the input is the hourly pay and the number of hours worked during the week. However, most companies pay a premium (such as time and one-half) for the hours worked over 40. You cannot tell whether the premium applies to the additional five hours that Cintia worked because the problem specification does not contain enough information. Before you can solve this problem, you will need to ask the payroll manager about the company's overtime policy.

Cintia Johanson earns $11.50 per hour. Last week, she worked 45 hours. Create a program that calculates and displays her weekly gross pay.

Figure 2-7 Problem specification that does not contain enough information

As a programmer, it is important to distinguish between information that truly is missing and information that simply is not stated explicitly in the problem specification—that is, information that is implied. For example, consider the problem specification shown in Figure 2-8. To solve the problem, you need to calculate the area of a rectangle; you do this by multiplying the rectangle's length by its width. Therefore, the area is the output, and the length and width are the input. Notice, however, that the words *length* and *width* do not appear in the problem specification. Although both items are not stated explicitly, neither is considered missing information. This is because the formula for calculating the area of a rectangle is common knowledge. (The formula can also be found in any math book or on the Internet.) With practice, you will also be able to "fill in the gaps" in a problem specification.

For more examples of analyzing problems, see the Analyzing Problems section in the Ch02WantMore.pdf file.

Gordon Matthew wants a program that calculates and displays the area of any rectangle.

Figure 2-8 Problem specification in which the input is not explicitly stated

Mini-Quiz 2-1

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Identify the output and input in each of the following problem specifications.

1. Alycia Thompkins is expecting her salary to increase by a specific percentage next year. She wants a program that she can use to display her raise and new salary amounts.
2. Professor Carlos wants a program that displays the average of two test scores: a midterm and a final.
3. The manager of a local restaurant wants a program that displays the suggested amounts to tip a waiter, using tip percentages of 10%, 15%, and 20%. The tip should be calculated on the customer's entire bill.
4. If James Monet saves $2 per day, how much will he save in one year?

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Step 2—Plan the Algorithm

The second step in the problem-solving process is to plan the algorithm that will transform the problem's input into its output. You record the algorithm in the Processing column of the IPO chart. Each instruction in the algorithm will describe an action that the computer needs to take in order to derive the output from the input. Therefore, each instruction should start with a verb.

Most algorithms begin with an instruction to enter the input items into the computer. Next, you usually record instructions to process the input items to achieve the problem's output. The processing typically involves performing one or more calculations using the input items. Most algorithms end with an instruction to display, print, or store the output items. *Display*, *print*, and *store* refer to the computer screen, the printer, and a file on a disk, respectively.

Figure 2-9 shows the problem specification and IPO chart for the Addison O'Reilly problem. Notice that each instruction in the algorithm starts with a verb: *enter*, *calculate*, and *display*. The algorithm begins by entering the input items. It then uses the input items to calculate the output item. An algorithm should state both *what* is to be calculated and *how* to calculate it. In this case, the cost is calculated by multiplying the sale price by the sales tax rate and then adding the result to the sale price. The last instruction in the algorithm displays the output item. To avoid confusion, it is important that the algorithm is consistent when referring to the input and output items. For example, if the input item is listed as *sales tax rate*, then the algorithm should refer to the item as *sales tax rate* rather than a different name, such as *tax rate* or *rate*.

Problem specification

Addison O'Reilly wants a program that calculates and displays the cost of a 4K Ultra HD TV, which is finally on sale at one of the stores in her area. The program should calculate the cost by multiplying the sale price by the state sales tax rate and then adding the result to the sale price.

Input	Processing	Output
sale price sales tax rate	Processing items: none	cost

Algorithm:
1. enter the sale price and sales tax rate
2. calculate the cost by multiplying the sale price by the sales tax rate and then adding the result to the sale price
3. display the cost

each instruction begins with a verb

Figure 2-9 Problem specification and IPO chart for the Addison O'Reilly problem

The algorithm in Figure 2-9 is composed of short English statements, referred to as **pseudocode**, which means *false code*. It's called false code because, although it resembles programming language instructions, pseudocode cannot be understood by a computer. Programmers use pseudocode to help them while they are planning an algorithm. It allows them to jot down their ideas using a human-readable language without having to worry about the syntax (rules) of the programming language itself. Pseudocode is not standardized; every programmer has his or her own version, but you will find some similarities among the various versions. For instance, one programmer may write the statement to calculate a rectangle's area as "calculate area by multiplying length by width", while another programmer might use the statement "area = length * width".

Besides using pseudocode, programmers also use flowcharts when planning algorithms. A **flowchart** uses standardized symbols to visually depict an algorithm. You can draw the flowchart symbols by hand, or you can use the drawing or shapes feature in a word processor. You can also use a flowcharting program, such as SmartDraw or Microsoft Visio.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Figure 2-10 shows the algorithm from Figure 2-9 in flowchart form. The flowchart contains three different symbols: an oval, a parallelogram, and a rectangle. The symbols are connected with lines, called **flowlines**. The oval symbol is called the **start/stop symbol** and is used to indicate the beginning and end of the flowchart. Between the start and stop ovals are two parallelograms, called input/output symbols. You use the **input/output symbol** to represent input tasks (such as getting information from the user) and output tasks (such as displaying, printing, or storing information). The first parallelogram in Figure 2-10 represents an input task, while the last parallelogram represents an output task. The rectangle in a flowchart is called the **process symbol** and is used to represent tasks such as calculations.

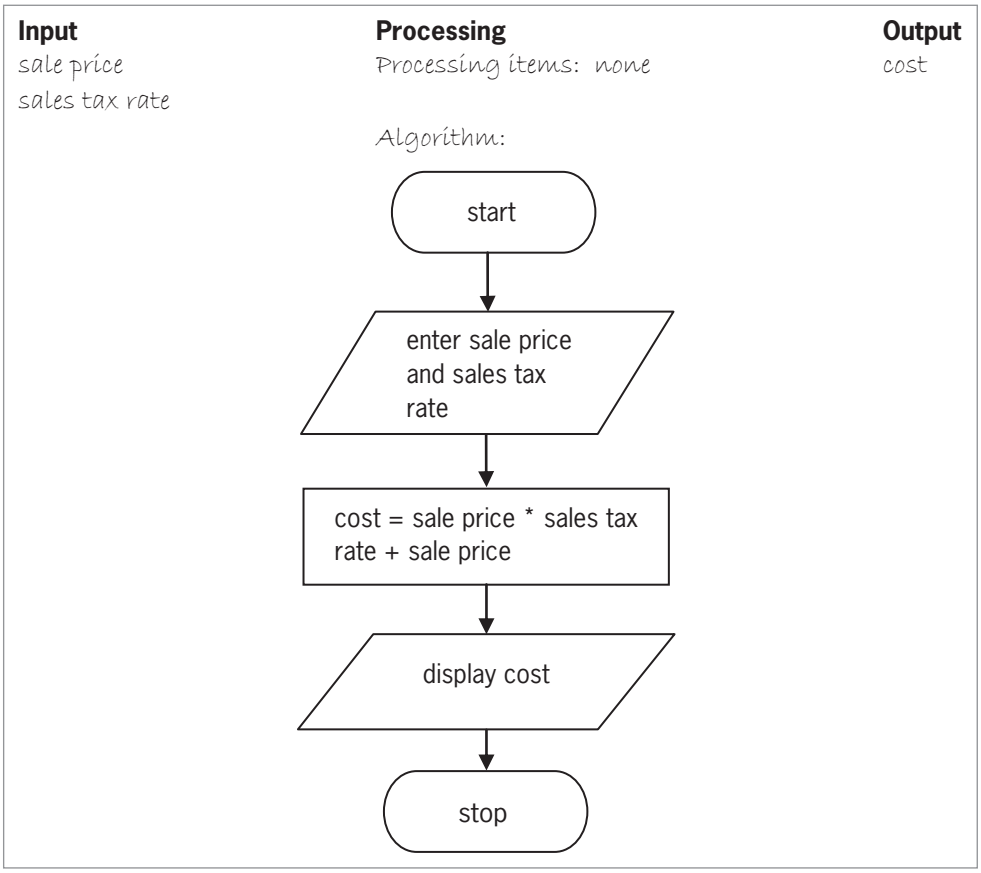


Figure 2-10 Figure 2-9's algorithm in flowchart form

When planning an algorithm, you do not need to create both pseudocode and a flowchart; you need to use only one of these planning tools. The tool you use is really a matter of personal preference. For simple algorithms, pseudocode works just fine. However, when an algorithm becomes more complex, its logic may be easier to see in a flowchart. As the old adage goes, a picture is sometimes worth a thousand words.

Even a very simple problem can have more than one solution. Figure 2-11 shows a different solution to the Addison O'Reilly problem. In this solution, the sales tax is calculated in a separate instruction rather than in the instruction that calculates the cost. The sales tax item is neither an input item (because it's not provided by the user) nor an output item (because it won't be displayed, printed, or stored in a file). Instead, the sales tax is a special item, commonly referred to as a processing item. A **processing item** represents an intermediate value that the algorithm uses when processing the input into the output. In this case, the algorithm uses the two input

For more examples of planning algorithms, see the Planning Algorithms section in the Ch02WantMore.pdf file.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

items (sale price and sales tax rate) to calculate the sales tax (an intermediate value). It then uses this intermediate value, along with the sale price, to compute the cost.

Problem specification

Addison O'Reilly wants a program that calculates and displays the cost of a 4K Ultra HD TV, which is finally on sale at one of the stores in her area. The program should calculate the cost by multiplying the sale price by the state sales tax rate and then adding the result to the sale price.

Input	Processing	Output
sale price sales tax rate	Processing items: sales tax	cost

Algorithm (pseudocode):

1. enter the sale price and sales tax rate
2. calculate the sales tax by multiplying the sale price by the sales tax rate
3. calculate the cost by adding the sales tax to the sale price
4. display the cost

Algorithm (flowchart):

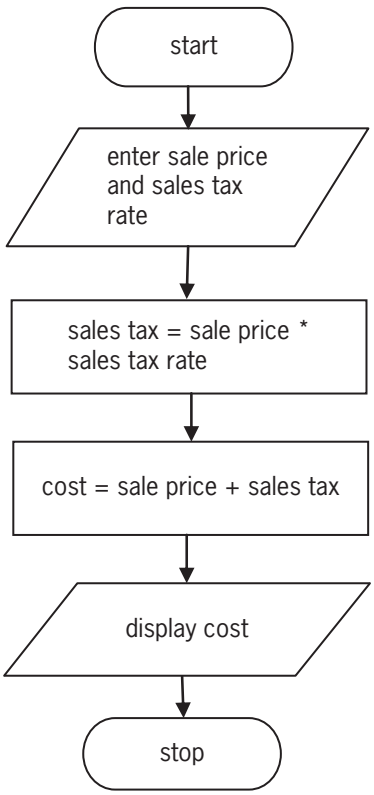


Figure 2-11 A different solution to the Addison O'Reilly problem

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The algorithms shown in Figures 2-9 through 2-11 produce the same result and simply represent two different ways of solving the same problem.

Mini-Quiz 2-2

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

1. The parallelogram in a flowchart is called the _____ symbol.
2. In a flowchart, calculation tasks are placed in a processing symbol, which has a(n) _____ shape.
3. Alycia Thompkins is expecting her salary to increase by a specific percentage next year. She wants a program that she can use to display her raise and new salary amounts. The output is the raise and new salary. The input is the current salary and raise percentage. Complete an appropriate IPO chart, using pseudocode in the Algorithm section.
4. Professor Carlos wants a program that displays the average of two test scores: a midterm and a final. The input is the midterm score and final score. The output is the average score. Complete an appropriate IPO chart, using a flowchart in the Algorithm section. The algorithm should use a processing item for the sum of both scores.

Step 3—Desk-Check the Algorithm

Ch02-Addison

After analyzing a problem and planning its algorithm, you then desk-check the algorithm. The term **desk-checking** refers to the fact that the programmer reviews the algorithm while seated at his or her desk rather than in front of the computer. Desk-checking is also called **hand-tracing** because the programmer uses a pencil and paper to follow each of the algorithm's instructions by hand. You desk-check an algorithm to verify that it is not missing any instructions and that the existing instructions are correct and in the proper order.

Before you begin the desk-check, you first choose a set of sample data for the input values, which you then use to manually compute the expected output value. For the Addison O'Reilly algorithm, you will use input values of $2300 and .05 (the decimal equivalent of 5%) as the sale price and sales tax rate, respectively. A manual calculation of the cost results in $2415, as shown in Figure 2-12.

```
$ 2300   (sale price)
*   .05  (sales tax rate)
  115    (sales tax)
+ 2300   (sale price)
$ 2415   (cost)
```

Figure 2-12 Manual cost calculation for the first desk-check

You now use the sample input values to desk-check the algorithm, which should result in the expected output value of $2415. It is helpful to use a desk-check table when desk-checking an algorithm. The table should contain one column for each input item listed in the IPO chart, as well as one column for each output item and one column for each processing item (if any). You can perform the desk-check using either the algorithm's pseudocode or its flowchart.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Figure 2-13 shows one solution for the Addison O'Reilly problem along with a partially completed desk-check table. (The flowchart for this solution is shown earlier in Figure 2-11.) Notice that the desk-check table contains four columns: two for the input items, one for the processing item, and one for the output item.

Input	Processing	Output
sale price	Processing items:	cost
sales tax rate	sales tax	

Algorithm (pseudocode):
1. enter the sale price and sales tax rate
2. calculate the sales tax by multiplying the sale price by the sales tax rate
3. calculate the cost by adding the sales tax to the sale price
4. display the cost

sale price	sales tax rate	sales tax	cost

Figure 2-13 Addison O'Reilly solution and partially completed desk-check table

The first instruction in the algorithm is to enter the two input values. You record the results of this instruction by writing 2300 and .05 in the sale price and sales tax rate columns, respectively, in the desk-check table. See Figure 2-14.

sale price	sales tax rate	sales tax	cost
2300	.05		

Figure 2-14 Input values entered in the desk-check table

The second instruction calculates the sales tax by multiplying the sale price by the sales tax rate. The desk-check table shows that the sale price is 2300 and the sales tax rate is .05. When making the calculation, always use the table to determine the values of the sale price and sales tax rate. Doing this helps to verify the accuracy of the algorithm. If, for example, the table did not show any amount in the sales tax rate column, you would know that your algorithm missed an instruction; in this case, it neglected to enter the sales tax rate. When you multiply the sale price (2300) by the sales tax rate (.05), you get 115. You record the number 115 in the sales tax column, as shown in Figure 2-15.

sale price	sales tax rate	sales tax	cost
2300	.05	115	

Figure 2-15 Processing item's value entered in the desk-check table

The third instruction calculates the cost by adding the sales tax (115) to the sale price (2300). When you add 115 to 2300, you get 2415. You record the number 2415 in the cost column, as shown in Figure 2-16.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

sale price	sales tax rate	sales tax	cost
2300	.05	115	2415

Figure 2-16 Output value entered in the desk-check table

The last instruction in the algorithm displays the cost. In this case, the number 2415 will be displayed because that is what appears in the cost column. Notice that this amount agrees with the manual calculation shown in Figure 2-12; therefore, the algorithm appears to be correct. The only way to know for sure, however, is to test the algorithm a few more times with different input values. For the second desk-check, you will test the algorithm using $5200 and .03 as the sale price and sales tax rate, respectively. The cost should be $5356, as shown in Figure 2-17.

```
$ 5200   (sale price)
*   .03  (sales tax rate)
    156  (sales tax)
+ 5200   (sale price)
$ 5356   (cost)
```

Figure 2-17 Manual cost calculation for the second desk-check

Recall that the first instruction in the algorithm is to enter the sale price and sales tax rate. Therefore, you write 5200 and .03 in the appropriate columns in the desk-check table, as shown in Figure 2-18. Although it's not required, some programmers find it helpful to lightly cross out the previous value in a column before recording a new value. Doing this helps keep track of the column's current value.

sale price	sales tax rate	sales tax	cost
~~2300~~	~~.05~~	115	2415
5200	.03		

Figure 2-18 Second set of input values entered in the desk-check table

The second instruction calculates the sales tax by multiplying the value in the sale price column (5200) by the value in the sales tax rate column (.03). You record the result (156) in the sales tax column. See Figure 2-19.

sale price	sales tax rate	sales tax	cost
~~2300~~	~~.05~~	~~115~~	2415
5200	.03	156	

Figure 2-19 Value of the second desk-check's processing item entered in the desk-check table

The third instruction calculates the cost by adding the value in the sales tax column (156) to the value in the sale price column (5200). You record the result (5356) in the cost column, as shown in Figure 2-20. The last instruction in the algorithm displays the cost. In this case, the number 5356 will be displayed, which agrees with the manual calculation shown in Figure 2-17.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

sale price	sales tax rate	sales tax	cost
~~2300~~	~~.05~~	~~115~~	~~2415~~
5200	.03	156	5356

Figure 2-20 Value of the second desk-check's output item entered in the desk-check table

For more examples of desk-checking algorithms, see the Desk-Checking Algorithms section in the Ch02WantMore.pdf file.

To be sure an algorithm works correctly, you should desk-check it several times using both valid and invalid data. **Valid data** is data that the algorithm is expecting the user to enter. For example, the algorithm that you just finished desk-checking expects the user to provide positive numbers for the input values. **Invalid data** is data that the algorithm is not expecting the user to enter, such as a negative number for the sale price. You should test an algorithm with invalid data because users sometimes make mistakes when entering data. In later chapters in this book, you will learn how to write algorithms that correctly handle input errors. For now, however, you can assume that the user will always enter valid data.

The Gas Mileage Problem

Ch02-Gas Mileage

The gas mileage problem will help reinforce what you learned in this chapter. Figure 2-21 shows the problem specification.

When Sheila Jones began her trip from Vermont to Oregon, she filled her car's tank with gas and reset its trip meter to zero. After traveling 324 miles, Sheila stopped at a gas station to refuel; the gas tank required 17 gallons. Sheila wants a program that calculates and displays her car's gas mileage at any time during the trip. The gas mileage is the number of miles her car can be driven per gallon of gas.

Figure 2-21 Problem specification for the gas mileage problem

First, analyze the problem, looking for nouns and adjectives that represent both the output and the input. The output should answer the question: *What does the user want to see displayed on the screen, printed on paper, or stored in a file?* The input should answer the question: *What information will the computer need to know to display, print, or store the output items?* In the gas mileage problem, the output is the miles per gallon, and the input is the miles driven and gallons used.

Next, plan the algorithm. Recall that most algorithms begin with an instruction to enter the input items into the computer, followed by instructions that process the input items and then display, print, or store the output items. Figure 2-22 shows the completed IPO chart for the gas mileage problem.

Input	Processing	Output
miles driven gallons used	Processing items: none Algorithm: 1. enter the miles driven and gallons used 2. calculate the miles per gallon by dividing the miles driven by the gallons used 3. display the miles per gallon	miles per gallon

each instruction begins with a verb

Figure 2-22 IPO chart for the gas mileage problem

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

After planning the algorithm, you then desk-check it. You will desk-check the algorithm twice, first using 324 and 17 as the miles driven and gallons used, respectively, and then using 400 and 15. Figure 2-23 shows the completed desk-check table for the gas mileage problem. (The miles per gallon are rounded to two decimal places.)

miles driven	gallons used	miles per gallon
~~324~~	~~17~~	~~19.06~~
400	15	26.67

Figure 2-23 Desk-check table for the gas mileage problem

Mini-Quiz 2-3

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

1. Desk-check the algorithm shown in Figure 2-24 twice. First, use a current salary of $32,600 and a raise percentage of .05 (the decimal equivalent of 5%). Then use $54,700 and .02.
2. Desk-check the algorithm shown in Figure 2-25 twice. First, use 75 and 83 as the midterm score and final score, respectively. Then use 98 and 93.

Input	Processing	Output
current salary raise percentage	Processing items: none Algorithm: 1. enter the current salary and raise percentage 2. calculate the raise by multiplying the current salary by the raise percentage 3. calculate the new salary by adding the raise to the current salary 4. display the raise and new salary	raise new salary

Figure 2-24 IPO chart for Question 1 in Mini-Quiz 2-3

Input	Processing	Output
midterm score final score	Processing items: sum Algorithm: 1. enter the midterm score and final score 2. calculate the sum by adding together the midterm score and final score 3. calculate the average score by dividing the sum by 2 4. display the average score	average score

Figure 2-25 IPO chart for Question 2 in Mini-Quiz 2-3

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The answers to the labs are contained in the Answers.pdf file.

LAB 2-1 Stop and Analyze

Study the IPO chart shown in Figure 2-26, and then answer the questions.

Input	Processing	Output
quantity sold item cost item selling price	Processing items: price and cost difference Algorithm: 1. enter the quantity sold, item cost, and item selling price 2. calculate the price and cost difference by subtracting the item cost from the item selling price 3. calculate the profit by multiplying the price and cost difference by the quantity sold 4. display the profit	profit

quantity sold	item cost	item selling price	price and cost difference	profit

Figure 2-26 IPO chart and partially completed desk-check table for Lab 2-1

QUESTIONS

1. Complete the desk-check table using two sets of input values. First, use 100, 5, and 8 as the quantity sold, item cost, and item selling price, respectively. Then use 650, 2.50, and 3.75. What will the algorithm display using the first set of input values? What will the algorithm display using the second set of input values?
2. How would you modify the IPO chart to also display the difference between the item cost and item selling price?
3. How would you modify the IPO chart and desk-check table to eliminate the use of a processing item?

LAB 2-2 Plan and Create

In this lab, you will plan and create an algorithm for the manager of Jericho Bakery. The problem specification is shown in Figure 2-27.

Jericho Bakery sells a variety of doughnuts and muffins for $0.50 each. The manager of the bakery wants a program that calculates and displays the total number of items (doughnuts and muffins) purchased by a customer, as well as the total cost of the order.

Figure 2-27 Problem specification for Lab 2-2

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

First, analyze the problem, looking first for the output and then for the input. Recall that the output and input are typically stated as nouns and adjectives in the problem specification. Asking the question *What does the user want to see displayed on the screen, printed on paper, or stored in a file?* will help you determine the output. In this case, the manager wants to see the total number of items ordered and the total cost of the order displayed on the computer screen. The question *What information will the computer need to know to display, print, or store the output items?* will help you determine the input. In this case, the input is the number of doughnuts purchased, the number of muffins purchased, and the price of one item (doughnut or muffin). Figure 2-28 shows the input and output items entered in an IPO chart.

Input	Processing	Output
number of doughnuts ordered number of muffins ordered item price	Processing items: none Algorithm:	total number of items ordered total cost

Figure 2-28 Partially completed IPO chart for Lab 2-2

After determining a problem's output and input, you then plan its algorithm. Recall that most algorithms begin by entering the input items into the computer. The first instruction in the current problem's algorithm, for example, will be *enter the number of doughnuts ordered, number of muffins ordered, and item price.* Notice that the instruction refers to the input items using the same names listed in the Input column of the IPO chart.

After the instruction to enter the input items, you usually record instructions to process those items, typically including the items in one or more calculations. In this case, you will add together the number of doughnuts ordered and the number of muffins ordered, giving the total number of items ordered. You then will calculate the total cost by multiplying the total number of items ordered by the item price.

Recall that most algorithms end with an instruction to display, print, or store the output items. The last instruction in this algorithm will display the total number of items ordered and total cost on the screen. Figure 2-29 shows the completed IPO chart. Notice that each instruction in the algorithm begins with a verb.

Input	Processing	Output
number of doughnuts ordered number of muffins ordered item price	Processing items: none	total number of items ordered total cost

Algorithm:
1. enter the number of doughnuts ordered, number of muffins ordered, and item price
2. calculate the total number of items ordered by adding together the number of doughnuts ordered and number of muffins ordered
3. calculate the total cost by multiplying the total number of items ordered by the item price
4. display the total number of items ordered and total cost

each instruction begins with a verb

Figure 2-29 Completed IPO chart for Lab 2-2

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

After completing the IPO chart, you then move on to the third step in the problem-solving process, which is to desk-check the algorithm. You begin by choosing a set of sample data for the input values. You then use the values to manually compute the expected output. You will desk-check the current algorithm twice: first using 4, 2, and 0.50 as the number of doughnuts ordered, number of muffins ordered, and item price, respectively, and then using 0, 12, and 0.45. The manual calculations for both desk-checks are shown in Figure 2-30.

First desk-check		**Second desk-check**	
4	(number of doughnuts ordered)	0	(number of doughnuts ordered)
+ 2	(number of muffins ordered)	+ 12	(number of muffins ordered)
6	(total number of items ordered)	12	(total number of items ordered)
* 0.50	(item price)	* 0.45	(item price)
$ 3.00	(total cost)	$ 5.40	(total cost)

Figure 2-30 Manual calculations for the two desk-checks

Next, you create a desk-check table that contains one column for each input, processing, and output item. You then begin desk-checking the algorithm. The first instruction is to enter the input values. Figure 2-31 shows these values entered in the desk-check table.

number of doughnuts ordered	number of muffins ordered	item price	total number of items ordered	total cost
4	2	0.50		

Figure 2-31 First set of input values entered in the desk-check table

The second and third instructions calculate the total number of items ordered and total cost. Figure 2-32 shows these values entered in the desk-check table.

number of doughnuts ordered	number of muffins ordered	item price	total number of items ordered	total cost
4	2	0.50	6	3.00

Figure 2-32 Calculated values entered in the desk-check table

The last instruction in the algorithm displays the two output values. According to the desk-check table in Figure 2-32, the total number of items ordered and total cost are 6 and 3.00, respectively; both amounts agree with the manual calculations shown earlier in Figure 2-30.

Now use the second set of input values to desk-check the algorithm: 0, 12, and 0.45. Figure 2-33 shows the result of the second desk-check. Notice that the amounts in the total number of items ordered column (12) and total cost column (5.40) agree with the manual calculations shown earlier in Figure 2-30.

number of doughnuts ordered	number of muffins ordered	item price	total number of items ordered	total cost
~~4~~	~~2~~	~~0.50~~	~~6~~	~~3.00~~
0	12	0.45	12	5.40

Figure 2-33 Desk-check table showing the result of the second desk-check

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

LAB 2-3 Modify

Jericho Bakery has raised the price of its muffins from $0.50 to $0.55. Make the appropriate modifications to the IPO chart shown earlier in Figure 2-29. Desk-check the algorithm twice. For the first desk-check, use 4, 2, 0.50, and 0.55 as the number of doughnuts ordered, number of muffins ordered, doughnut price, and muffin price, respectively. For the second desk-check, use 0, 12, 0.60, and 0.70.

LAB 2-4 What's Missing?

A local club sells boxes of three types of cookies: shortbread, pecan sandies, and chocolate mint. The club leader wants a program that displays the percentage that each of the cookie types contributes to the total cookie sales. For example, if the club sells 50 boxes of shortbread cookies, 25 boxes of pecan sandies cookies, and 45 boxes of chocolate mint cookies, the shortbread cookies account for approximately 41.7% of the cookie sales. The pecan sandies and chocolate mint cookies account for approximately 20.8% and 37.5%, respectively, of the cookie sales. Figure 2-34 contains a list of items and instructions that you can use for this lab. First, enter the appropriate input, processing (if any), and output items in the IPO chart. Determine whether any items are missing from the list. Next, put the instructions shown in the figure in the proper order, and then determine the one or more missing instructions.

Input	Processing	Output
	Processing items: Algorithm:	

Items

chocolate mint contribution
chocolate mint sold
pecan sandies contribution
pecan sandies sold
shortbread contribution
shortbread sold

Instructions

calculate chocolate mint contribution by dividing chocolate mint sold by total sold, and then multiplying the result by 100
calculate pecan sandies contribution by dividing pecan sandies sold by total sold, and then multiplying the result by 100
calculate shortbread contribution by dividing shortbread sold by total sold, and then multiplying the result by 100
display shortbread contribution, pecan sandies contribution, and chocolate mint contribution
enter shortbread sold, pecan sandies sold, and chocolate mint sold

Figure 2-34 Items and instructions for Lab 2-4

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

LAB 2-5 Desk-Check

The algorithm in Figure 2-35 displays three suggested amounts to tip a waiter. Desk-check the algorithm twice. First, use $102.50 and $5.80 as the restaurant bill and sales tax, respectively. Then, use $56.78 and $2.18.

Input	Processing	Output
restaurant bill sales tax	Processing items: bill before sales tax	10% tip 15% tip 20% tip

Algorithm:
1. enter the restaurant bill
2. calculate the bill before sales tax by subtracting the sales tax from the restaurant bill
3. calculate the 10% tip by multiplying the bill before sales tax by 10%
4. calculate the 15% tip by multiplying the bill before sales tax by 15%
5. calculate the 20% tip by multiplying the bill before sales tax by 20%
6. display the 10% tip, 15% tip, and 20% tip

Figure 2-35 IPO chart for Lab 2-5

LAB 2-6 Debug

The algorithm in Figure 2-36 should calculate and display the average of three numbers, but it is not working correctly. In this lab, you will find and correct the errors in the algorithm.

Input	Processing	Output
first number second number third number	Processing items: sum	average

Algorithm:
1. enter the first number, second number, and third number
2. calculate the average by dividing the sum by 3
3. display the average number

Figure 2-36 IPO chart for Lab 2-6

You locate the errors in an algorithm by desk-checking it. First, choose a set of sample data for the input values. In this case, you will use the numbers 25, 63, and 14. Now use the values to manually compute the expected output—in this case, the average. The average of the three numbers is 34. Next, create a desk-check table that contains a column for each input, processing, and output item. This desk-check table will contain five columns. Finally, walk through each of the instructions in the algorithm, recording the appropriate values in the desk-check table. The first instruction in the algorithm in Figure 2-36 is to enter the three input values. Figure 2-37 shows these values entered in the desk-check table.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

first number	second number	third number	sum	average
25	63	14		

Figure 2-37 Three input values entered in the desk-check table

The next instruction calculates the average by dividing the sum by 3. Notice that the sum column in the desk-check table does not contain a value. This fact alerts you that the algorithm is missing an instruction. In this case, it is missing the instruction to calculate the sum of the three numbers. The missing instruction is shaded in Figure 2-38.

Input	**Processing**	**Output**
first number second number third number	Processing items: sum Algorithm: 1. enter the first number, second number, and third number 2. calculate the sum by adding together the first number, second number, and third number 3. calculate the average by dividing the sum by 3 4. display the average number	average

Figure 2-38 Missing instruction added to the IPO chart for Lab 2-6

The additional instruction calculates the sum by adding together the first number, second number, and third number. According to the desk-check table shown earlier in Figure 2-37, those values are 25, 63, and 14, respectively. The sum of those values is 102. Figure 2-39 shows the sum entered in the desk-check table.

first number	second number	third number	sum	average
25	63	14	102	

Figure 2-39 Sum entered in the desk-check table

The next instruction calculates the average by dividing the sum by 3. According to the desk-check table, the sum is 102. Dividing 102 by 3 results in 34. Figure 2-40 shows the average entered in the desk-check table.

first number	second number	third number	sum	average
25	63	14	102	34

Figure 2-40 Average entered in the desk-check table

The last instruction in the algorithm displays the average number. Notice that the desk-check table does not contain a column with the heading "average number." Recall that it is important to be consistent when referring to the input, output, and processing items in the IPO chart. In this case, the last instruction in the algorithm should be *display the average* rather than *display the average number.* According to the desk-check table, the average column contains the number 34, which is correct. Figure 2-41 shows the corrected algorithm. The changes made to the original algorithm (shown earlier in Figure 2-36) are shaded in the figure.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Input	Processing	Output
first number second number third number	Processing items: sum Algorithm: 1. enter the first number, second number, and third number 2. calculate the sum by adding together the first number, second number, and third number 3. calculate the average by dividing the sum by 3 4. display the average	average

Figure 2-41 Corrected algorithm for Lab 2-6

On your own, desk-check the corrected algorithm shown in Figure 2-41 using the numbers 33, 56, and 70.

Chapter Summary

The process you follow when creating solutions to everyday problems is similar to the process used to create a computer program, which is also a solution to a problem. This problem-solving process typically involves analyzing the problem and then planning, reviewing, implementing, evaluating, and modifying (if necessary) the solution.

Programmers use tools such as IPO (Input, Processing, Output) charts, pseudocode, and flowcharts to help them analyze problems and develop algorithms.

The first step in the problem-solving process is to analyze the problem. During the analysis step, the programmer first determines the output, which is the goal or purpose of solving the problem. The programmer then determines the input, which is the information needed to reach the goal.

The second step in the problem-solving process is to plan the algorithm. During the planning step, programmers write the instructions that will transform the input into the output. Most algorithms begin by entering some data (the input items), then processing that data (usually by performing some calculations), and then displaying some data (the output items).

The third step in the problem-solving process is to desk-check the algorithm to determine whether it will work as intended. First, choose a set of sample data for the input values. Then use the values to manually compute the expected output. Next, create a desk-check table that contains a column for each input, processing, and output item. Finally, walk through each of the instructions in the algorithm, recording the appropriate values in the desk-check table.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Key Terms

Desk-checking—the process of manually walking through each of the instructions in an algorithm; also called hand-tracing

Flowchart—a tool that programmers use to help them plan (or depict) an algorithm; consists of standardized symbols connected by flowlines

Flowlines—the lines that connect the symbols in a flowchart

Hand-tracing—another term for desk-checking

Input—the items a program needs in order to achieve the output

Input/output symbol—the parallelogram in a flowchart; used to represent input and output tasks

Invalid data—data that the algorithm is not expecting the user to enter

IPO—an acronym for Input, Processing, and Output

IPO chart—a chart that some programmers use to organize and summarize the results of a problem analysis

Output—the goal of solving a problem; the items the user wants to display, print, or store

Process symbol—the rectangle symbol in a flowchart; used to represent tasks such as calculations

Processing item—an intermediate value (neither input nor output) that an algorithm uses when processing the input into the output

Pseudocode—a tool that programmers use to help them plan an algorithm; consists of short English statements; means *false code*

Start/stop symbol—the oval symbol in a flowchart; used to mark the beginning and end of the flowchart

Valid data—data that the algorithm is expecting the user to enter

Review Questions

1. Which of the following is the first step in the problem-solving process?
   a. Plan the algorithm
   b. Analyze the problem
   c. Desk-check the algorithm
   d. Code the algorithm into a program

2. Programmers refer to the goal of solving a problem as the _____ .
   a. input
   b. output
   c. processing
   d. purpose

3. Programmers refer to the items needed to reach a problem's goal as the _____ .
   a. input
   b. output
   c. processing
   d. purpose

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

4. A problem's _____ will answer the question *What does the user want to see displayed on the screen, printed on the printer, or stored in a file?*

   a. input
   b. output
   c. processing
   d. purpose

5. A problem's _____ will answer the question *What information will the computer need to know to display, print, or store the output items?*

   a. input
   b. output
   c. processing
   d. purpose

6. The calculation instructions in an algorithm should state _____ .

   a. only *what* is to be calculated
   b. only *how* to calculate something
   c. both *what* is to be calculated and *how* to calculate it
   d. both *what* is to be calculated and *why* it is calculated

7. Most algorithms follow the format of _____ .

   a. entering the input items; then displaying, printing, or storing the input items; and then processing the output items
   b. entering the input items; then processing the output items; and then displaying, printing, or storing the output items
   c. entering the input items; then processing the input items; and then displaying, printing, or storing the output items
   d. entering the output items; then processing the output items; and then displaying, printing, or storing the output items

8. The short English statements that represent an algorithm are called _____ .

   a. flow diagrams
   b. IPO charts
   c. pseudocharts
   d. pseudocode

9. The oval in a flowchart is called the _____ symbol.

   a. calculation
   b. input/output
   c. process
   d. start/stop

10. A desk-check table should contain _____ .

   a. one column for each input item
   b. one column for each output item
   c. one column for each processing item
   d. all of the above

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Exercises

Pencil and Paper

1. The principal of a local school wants a program that displays the average number of students per teacher at the school. The principal will enter the number of students enrolled and the number of teachers employed. Complete an IPO chart for this problem. Plan the algorithm using a flowchart. Also complete a desk-check table for your algorithm. For the first desk-check, use 1200 and 60 as the number of students and number of teachers, respectively. Then use 2500 and 100. (The answers to TRY THIS Exercises are located at the end of the chapter.) TRY THIS

2. Balloon Emporium sells both latex and Mylar balloons. The store owner wants a program that allows him to enter the price of a latex balloon, the price of a Mylar balloon, the number of latex balloons purchased, the number of Mylar balloons purchased, and the sales tax rate. The program should calculate and display the total cost of the purchase. Complete an IPO chart for this problem. Plan the algorithm using pseudocode. Desk-check the algorithm using $1.50 as the latex balloon price, $2.50 as the Mylar balloon price, 5 as the number of latex balloons purchased, 10 as the number of Mylar balloons purchased, and .04 as the sales tax rate. Then desk-check it using $1.25, $3.75, 10, 4, and .06. (The answers to TRY THIS Exercises are located at the end of the chapter.) TRY THIS

3. Modify the IPO chart shown earlier in Figure 2-22 so that it also displays the cost per mile driven. Desk-check the algorithm using 324, 17, and $3.10 as the miles driven, gallons used, and cost per gallon of gas. Then desk-check it using 450, 20, and $2.75. MODIFY THIS

4. All of the employees at Merks Sales are paid based on an annual salary rather than an hourly wage. However, some employees are paid weekly while others are paid every other week (biweekly). Weekly employees receive 52 paychecks; biweekly employees receive 26 paychecks. The payroll manager wants a program that displays two amounts: an employee's weekly gross pay and his or her biweekly gross pay. Complete an IPO chart for this problem. Desk-check the algorithm using $56,700 as the salary. Then desk-check it using $32,660. INTRODUCTORY

5. Norbert Catering is famous for its roast beef sandwiches. The store's owner wants a program that he can use to estimate the number of pounds of roast beef a customer should purchase, given the desired number of sandwiches and the amount of meat per sandwich. Typically, one sandwich requires two to three ounces of meat, but some customers prefer four or five ounces per sandwich. Complete an IPO chart for this problem. Desk-check the algorithm using 50 as the number of sandwiches and 4 ounces as the amount of meat per sandwich. Then desk-check it using 224 and 2 ounces. INTRODUCTORY

6. An airplane has both first-class and coach seats. The first-class tickets cost more than the coach tickets. The airline wants a program that calculates and displays the total amount of money the passengers paid for a specific flight. Complete an IPO chart for this problem. Desk-check the algorithm using 9, 52, $125, and $90 as the number of first-class tickets sold, the number of coach tickets sold, the price of a first-class ticket, and the price of a coach ticket, respectively. Then desk-check it using your own set of data. INTRODUCTORY

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

INTERMEDIATE

7. The annual property tax in Richardson County is $1.75 for each $100 of a property's assessed value. The county clerk wants an application that will display the property tax after he enters the property's assessed value. Complete an IPO chart for this problem. Desk-check it twice, using your own data.

INTERMEDIATE

8. Each time Tania visits the dentist, her dental insurance requires her to pay a $20 co-pay and 15% of the remaining charge. She wants a program that displays the total amount she needs to pay, as well as the total amount the insurance should pay. Complete an IPO chart for this problem. You can assume that the dental charge will always be at least $20. Desk-check the algorithm using $110 as the dental charge. Then desk-check it using your own data.

INTERMEDIATE

9. Treyson Liu has just purchased his first home. His local flooring store is having a great sale, and he would like to replace the tile floor in his rectangular den with carpeting. He wants a program that calculates and displays the cost of the carpeting. Complete an IPO chart for this problem. Desk-check the algorithm using 25 feet as the length, 30 feet as the width, and $5.65 as the cost per square foot of carpeting, respectively. Then desk-check it using your own set of data.

INTERMEDIATE

10. Carlos receives 24 paychecks each year. Each paycheck, he contributes a specific percentage of his gross pay to his retirement plan at work. His employer also contributes to his retirement plan, but at a different rate. Carlos wants a program that will calculate and display the total annual contribution made to his retirement plan by him and his employer. Complete an IPO chart for this problem. Desk-check the algorithm using $1465 as the gross pay, 4% as Carlos's contribution rate, and 2% as his employer's contribution rate. Then desk-check it using your own set of data.

INTERMEDIATE

11. The manager of a local restaurant wants a program that displays the total cost for running a party in the restaurant's banquet room. The restaurant charges a base fee for renting the room. It also charges a fee per guest. Complete an IPO chart for this problem. Desk-check the algorithm twice, using your own sets of data.

ADVANCED

12. Sam wants a program that displays the number of seconds it takes for a baseball to travel a specified distance at a specified speed. The distance will be given in feet, and the speed will be given in miles per hour. Complete an IPO chart for this problem. Desk-check the algorithm using 60.5 feet as the distance and 105 miles per hour as the speed. Then desk-check it using 54 feet and 89 miles per hour. In each desk-check, round the answer to four decimal places.

ADVANCED

13. The payroll clerk at Nosaki Company wants a program that calculates and displays an employee's gross pay, federal withholding tax (FWT), Social Security and Medicare (FICA) tax, state tax, and net pay. The clerk will enter the hours worked (which is never over 40), hourly pay rate, FWT rate, FICA tax rate, and state income tax rate. Complete an IPO chart for this problem. Desk-check the algorithm using 30, $10, .2, .08, and .04 as the hours worked, pay rate, FWT rate, FICA rate, and state tax rate, respectively. Then desk-check it using your own set of data.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

14. The algorithm shown in Figure 2-42 should calculate and display the total amount due, but it is not working correctly. Correct the algorithm, and then desk-check it using 20, $0.25, and .045 (the decimal equivalent of 4.5%) as the number of folders purchased, folder price, and sales tax rate, respectively.

SWAT THE BUGS

Input	Processing	Output
number purchased	Processing items:	total due
folder price	subtotal	
sales tax rate	sales tax	

Algorithm:
1. enter the number purchased and folder price
2. calculate the subtotal by multiplying the number purchased by the folder price
3. calculate the sales tax by dividing the subtotal by the sales tax rate
4. calculate the total due by adding the sales tax to the subtotal
5. display the total due

Figure 2-42

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

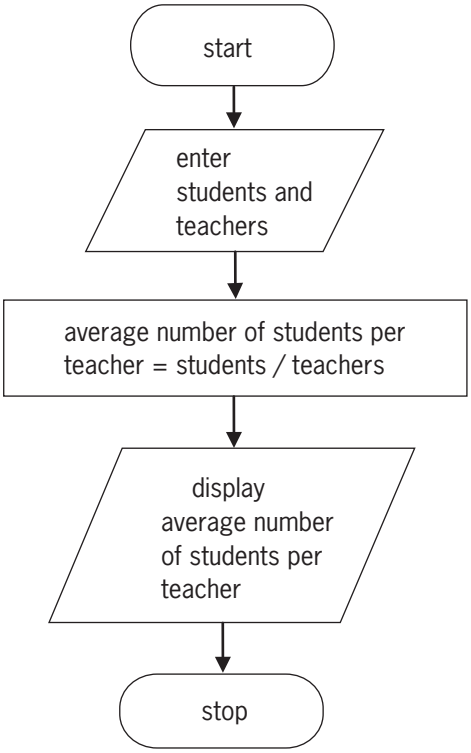
Answers to TRY THIS Exercises

1. See Figure 2-43.

Input
students
teachers

Processing
Processing items: none

Algorithm:



Output
average number of students per teacher

students	teachers	average number of students per teacher
~~1200~~	~~60~~	~~20~~
2500	100	25

Figure 2-43

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

2. See Figure 2-44.

Input	Processing	Output
latex price	Processing items:	total cost
Mylar price	total latex cost	
latex purchased	total Mylar cost	
Mylar purchased	subtotal	
sales tax rate	sales tax	

Algorithm:

1. enter the latex price, Mylar price, latex purchased, Mylar purchased , and sales tax rate
2. calculate the total latex cost by multiplying the latex purchased by the latex price
3. calculate the total Mylar cost by multiplying the Mylar purchased by the Mylar price
4. calculate the subtotal by adding together the total latex cost and total Mylar cost
5. calculate the sales tax by multiplying the subtotal by the sales tax rate
6. calculate the total cost by adding the sales tax to the subtotal
7. display the total cost

latex price	Mylar price	latex purchased	Mylar purchased	sales tax rate
~~1.50~~	~~2.50~~	~~5~~	~~10~~	~~.04~~
1.25	3.75	10	4	.06

total latex cost	total Mylar cost	subtotal	sales tax	total cost
~~7.50~~	~~25~~	~~32.50~~	~~1.30~~	~~33.80~~
12.50	15	27.50	1.65	29.15

Figure 2-44

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

CHAPTER 3

Variables and Constants

After studying Chapter 3, you should be able to:

- Distinguish among a variable, a named constant, and a literal constant
- Explain how data is stored in memory
- Select an appropriate name, data type, and initial value for a memory location
- Declare a memory location in C++

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Ch03-Chapter Preview

Beginning Step 4 in the Problem-Solving Process

Chapter 2 covered the first three steps in the problem-solving process: analyze the problem, plan the algorithm, and then desk-check the algorithm. When the programmer is satisfied that the algorithm is correct, he or she moves on to the fourth step, which is to code the algorithm into a program. Coding the algorithm refers to the process of translating the algorithm into a language that the computer can understand; in this book, you will use the C++ programming language.

Programmers use the information in the IPO chart, which they created in the analysis and planning steps, as a guide when coding the algorithm. The programmer begins by assigning a descriptive name to each unique input, processing, and output item. The programmer also assigns to each item a data type and (optionally) an initial value. The name, data type, and initial value are used to store the input, processing, and output items in the computer's internal memory while the program is running.

Internal Memory

The internal memory of a computer is composed of memory locations, with each memory location having a unique numeric address. It may be helpful to picture memory locations as shoe boxes, similar to the ones illustrated in Figure 3-1. As you know, shoe boxes come in different types and sizes. There are small boxes for children's sandals, larger boxes for adult sneakers, and even larger boxes for boots. The type and size of the footwear determine the appropriate type and size of the box. Like shoe boxes, memory locations also come in different types and sizes. Here, too, the type and size of the item you want to store determine the appropriate type and size of the memory location. The item stored in a memory location can be a number, such as the small number .0005 or the much larger number 1,500,892.35. The item can also be **text**, which is a group of characters treated as one unit and not used in a calculation. Examples of text include a name, an address, or a phone number. The item can also be a Boolean value (true or false) or a C++ instruction. Unlike the shoe boxes in the figure, however, each memory location inside a computer can hold only one item of data at a time.

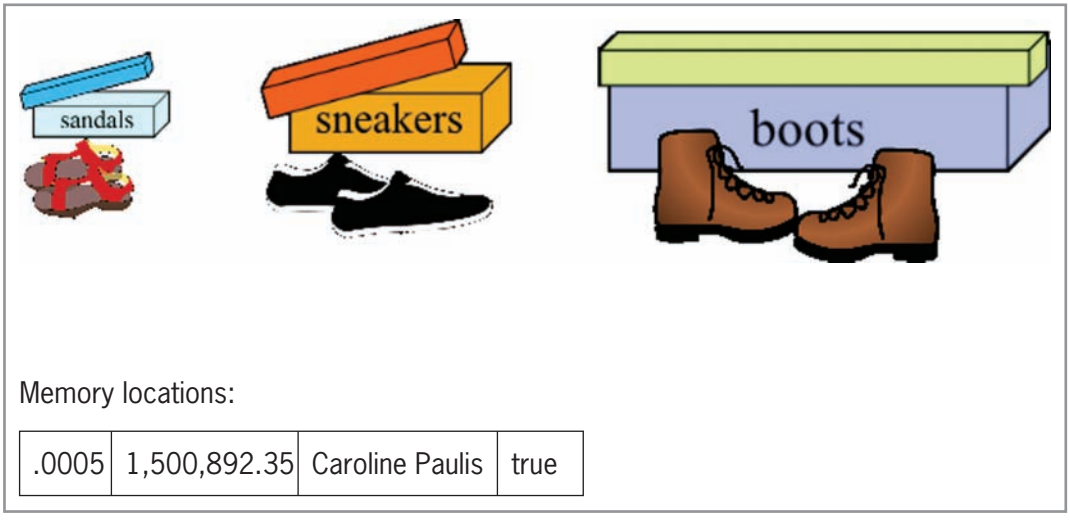


Image by Diane Zak; created with Reallusion CrazyTalk Animator

Figure 3-1 Illustration of shoe boxes and memory locations

Some of the memory locations inside the computer are automatically filled with data while you use your computer. For example, when you enter the number 5 at your keyboard, the computer saves the number 5 in a memory location for you. Likewise, when you start an application, each

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

program instruction is placed in a memory location, where it awaits processing. Other memory locations are reserved by programmers for use in a program. Such memory locations are used to store the values of the input, processing, and output items as the program is running. Reserving a memory location is also referred to as declaring the memory location.

A programmer declares a memory location using a C++ instruction that assigns a name, a data type, and (optionally) an initial value to the location. The name allows the programmer to refer to the memory location using one or more descriptive words, rather than a cryptic numeric address, in code. Descriptive words are easier to remember and serve to self-document your code. The data type indicates the type of data—for example, numeric or textual—the memory location will store.

There are two types of memory locations that a programmer can declare: variables and named constants. A **variable** is a memory location whose value can change (vary) during **runtime**, which is when a program is running. Most of the memory locations declared in a program are variables. A **named constant**, on the other hand, is a memory location whose value cannot be changed during runtime. In a program that inputs the radius of any circle and then calculates and outputs the circle's area, a programmer would declare variables to store the values of the radius and area; doing this allows those values to vary while the program is running. However, he or she would declare a named constant to store the value of pi (π), which is used in the formula for calculating the area of a circle. (The formula is πr^2.) A named constant is appropriate in this case because the value of pi (3.141593 when rounded to six decimal places) will always be the same.

Selecting a Name for a Memory Location

Every memory location that a programmer declares must be assigned a name. The name, also called the identifier, should describe the contents of the variable. A good memory location name is one that is meaningful right after you finish a program and also years later when you (or perhaps a coworker) need to modify the program.

A memory location's name must follow several specific rules in C++. It must begin with a letter and contain only letters, numbers, and the underscore character. No punctuation marks, spaces, or other special characters (such as $ or %) are allowed in the name. In addition, the name cannot be a **keyword**, which is a word that has a special meaning in the programming language you are using. Keywords are also referred to as reserved words. Appendix A in this book contains a list of the C++ keywords, which must be entered using lowercase letters.

Refer to the Tip that appears next to Figure 3-2 for an exception to beginning a memory location's name with a letter.

Memory location names are case sensitive in C++. This means that in addition to using the exact spelling when referring to a specific memory location in a program, you must also use the exact case. For example, if you declare a memory location named `discount` at the beginning of a program, you must use the name `discount`, rather than `Discount` or `DISCOUNT`, to refer to that memory location throughout the program.

Many C++ programmers use uppercase letters when naming named constants and use lowercase letters when naming variables. This practice allows them to easily distinguish between the named constants and variables in a program. If a named constant's name contains more than one word, an underscore character can be used to separate the words, like this: `TAX_RATE`. However, if a variable's name contains two or more words, most C++ programmers enter the name using **camel case**, which means they capitalize the first letter in the second and subsequent words in the name, like this: `grossPay`. Camel case refers to the fact that the uppercase letters appear as "humps" in the name because they are taller than the lowercase letters. The rules for naming memory locations in C++ are shown in Figure 3-2, along with examples of valid and invalid names.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Technically, a memory location's name in C++ can begin with an underscore. However, this usually is done only for the names of memory locations declared within a class. You will learn about classes later in this book.

HOW TO Name a Memory Location in C++

1. The name must begin with a letter.
2. The name can contain only letters, numbers, and the underscore character. No punctuation marks, spaces, or other special characters are allowed in the name.
3. The name cannot be a keyword. Appendix A contains a list of keywords in C++.
4. Names in C++ are case sensitive.

Valid names

```
grossPay, interest, TAX_RATE, PI
```

Invalid names	Reason
`2018Sales`	the name must begin with a letter
`end Balance`	the name cannot contain a space
`first.name`	the name cannot contain punctuation
`int`	the name cannot be a keyword
`RATE%`	the name cannot contain a special character

Figure 3-2 How to name a memory location in C++

Revisiting the Addison O'Reilly Problem from Chapter 2

Figure 3-3 shows one of the problem specifications, IPO charts, and desk-check tables from Chapter 2.

Problem specification

Addison O'Reilly wants a program that calculates and displays the cost of a 4K Ultra HD TV, which is finally on sale at one of the stores in her area. The program should calculate the cost by multiplying the sale price by the state sales tax rate and then adding the result to the sale price.

Input	Processing	Output
sale price sales tax rate	Processing items: sales tax Algorithm: 1. enter the sale price and sales tax rate 2. calculate the sales tax by multiplying the sale price by the sales tax rate 3. calculate the cost by adding the sales tax to the sale price 4. display the cost	cost

sale price	sales tax rate	sales tax	cost
~~2300~~	~~.05~~	~~115~~	~~2415~~
5200	.03	156	5356

Figure 3-3 Problem specification, IPO chart, and desk-check table from Chapter 2

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Four memory locations will be needed to store the four input, processing, and output items contained in the IPO chart. The memory locations will be variables because each item's value should be allowed to vary during runtime. Figure 3-4 lists possible names (identifiers) for the variables.

IPO chart item	Variable name
sale price	salePrice
sales tax rate	taxRate
sales tax	salesTax
cost	cost

Figure 3-4 Names of the variables for the Addison O'Reilly problem

Mini-Quiz 3-1

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

1. How many items can a memory location store at a time?
2. Which of the following can be used in a C++ program to refer to the `salesTax` variable?
   a. `salesTax`
   b. `salestax`
   c. `Salestax`
   d. Any of the above can be used.
3. Which of the following is a valid name for a memory location?
   a. `Income&Expense`
   b. `4thQtrSales`
   c. `quarter#3`
   d. `TAX_RATE`
4. What are the two types of memory locations that a programmer can declare?

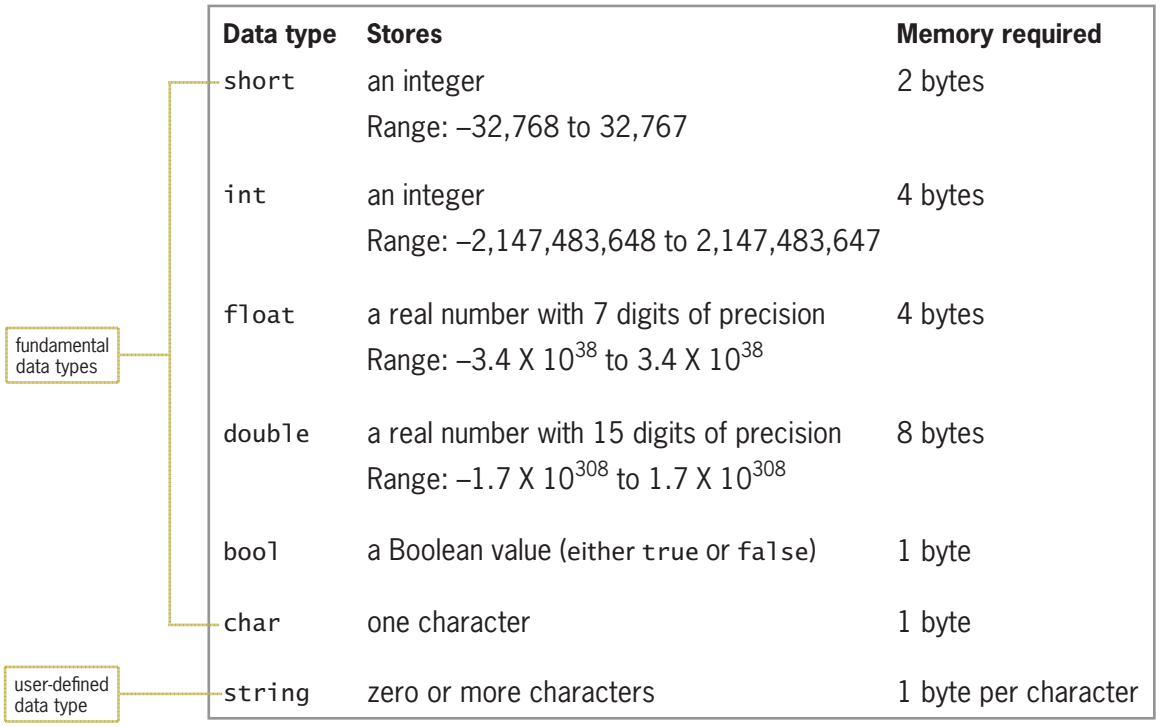
Selecting a Data Type for a Memory Location

The item that a memory location will accept for storage is determined by the location's data type, which the programmer assigns to the location when he or she declares it in a program. The most commonly used data types in C++ are listed in Figure 3-5, along with the values each type can store and the amount of memory needed to store a value.

The memory requirements and values for the different data types are implementation dependent. However, the ones listed in Figure 3-5 are typical for personal computers.

Except for the `string` data type, the data types listed in Figure 3-5 belong to a group of data types called fundamental data types. The **fundamental data types** are the basic data types built into the C++ language and often are referred to as primitive data types or built-in data types. The `string` data type, on the other hand, was added to the C++ language through the use of a class and is referred to as a **user-defined data type**. A class is simply a group of instructions that the computer uses to create an object. In this case, the `string` class (user-defined data type) creates a `string` variable, which is considered an object. You will learn more about classes and objects in subsequent chapters in this book.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.



Data type	Stores	Memory required
`short`	an integer Range: –32,768 to 32,767	2 bytes
`int`	an integer Range: –2,147,483,648 to 2,147,483,647	4 bytes
`float`	a real number with 7 digits of precision Range: -3.4×10^{38} to 3.4×10^{38}	4 bytes
`double`	a real number with 15 digits of precision Range: -1.7×10^{308} to 1.7×10^{308}	8 bytes
`bool`	a Boolean value (either `true` or `false`)	1 byte
`char`	one character	1 byte
`string`	zero or more characters	1 byte per character

Figure 3-5 Most commonly used data types in C++

As Figure 3-5 indicates, `bool` memory locations can store either the Boolean value `true` or the Boolean value `false`. The Boolean values are named in honor of the English mathematician George Boole (1815–1864), who invented Boolean algebra. You could use a `bool` variable in a program to keep track of whether a customer's bill is either paid (`true`) or not paid (`false`).

Memory locations assigned the `char` data type can store one character only. A **character** is a letter, a symbol, or a number that will not be used in a calculation. Some programmers pronounce `char` as "care" because it is short for *character*, while others pronounce `char` as in the first syllable of the word *charcoal*. A `string` memory location, on the other hand, can store zero or more characters.

Memory locations assigned either the `short` or `int` data type can store integers only. An **integer** is a whole number, which is a number that does not contain a decimal place. Examples of integers include the numbers 0, 45, and –678. The differences between the `short` and `int` data types are in the range of numbers each type can store and the amount of memory needed to store the number.

Memory locations assigned either the `float` or `double` data type can store **real numbers**, which are numbers that contain a decimal place. Examples of real numbers include the numbers 75.67, –3.45, and 783.5689. The differences between the `float` and `double` data types are in the range of numbers each type can store, the precision with which the number is stored, and the amount of memory needed to store the number.

In most of the programs you create in this book, you will use the `int` data type for memory locations that will store integers, and use the `double` data type for memory locations that will store numbers with a decimal place. The `double` data type was chosen over the `float` data type because it stores real numbers more precisely, using 15 digits of precision rather than only seven

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

digits. At this point, however, it is important to caution you about real numbers. Even with 15 digits of precision, not all real numbers can be represented exactly within the computer's internal memory. As a result, some calculations may not result in accuracy to the penny. You will learn more about using real numbers in calculations in Chapter 4. Figure 3-6 shows the data type selected for each variable in the Addison O'Reilly problem.

IPO chart item	Variable name	Data type
sale price	`salePrice`	`double`
sales tax rate	`taxRate`	`double`
sales tax	`salesTax`	`double`
cost	`cost`	`double`

Figure 3-6 Data type assigned to each variable for the Addison O'Reilly problem

How Data Is Stored in Internal Memory

Knowing how data is stored in the computer's internal memory will help you understand the importance of a memory location's data type. Numbers are represented in internal memory using the binary (or *base 2*) number system. The **binary number system** uses only the two digits 0 and 1. Although the binary number system may not be as familiar to you as the **decimal number system**, which uses the 10 digits 0 through 9, it is just as easy to understand. First, we'll review the decimal (or *base 10*) number system that you learned about in elementary school.

Ch03-Number System

As Figure 3-7 indicates, the position of each digit in the decimal number system is associated with the system's base number, 10, raised to a power. Starting with the rightmost position, the positions represent the number 10 raised to a power of 0, 1, 2, 3, and so on. In the decimal number 110, the 0 is in the 10^0 position, the middle 1 is in the 10^1 position, and the leftmost 1 is in the 10^2 position. Keep in mind that in all numbering systems, the result of raising the base number to the 0^{th} power is 1, and the result of raising it to the 1^{st} power is the base number itself. A base number raised to the 2^{nd} power indicates that the base number should be squared—in other words, multiplied by itself. As a result, the decimal number 110 means zero 1s (10^0), one 10 (10^1), and one 100 (10^2). The decimal number 3475 means five 1s (10^0), seven 10s (10^1), four 100s (10^2), and three 1000s (10^3). Similarly, the decimal number 21509 means nine 1s (10^0), zero 10s (10^1), five 100s (10^2), one 1000 (10^3), and two 10000s (10^4).

HOW TO Use the Decimal (*Base 10*) Number System

Decimal number	10^7	10^6	10^5	10^4	10^3	10^2	10^1	10^0
110						1	1	0
3475					3	4	7	5
21509				2	1	5	0	9

Figure 3-7 How to use the decimal (*base 10*) number system

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Compare the decimal number system illustrated in Figure 3-7 with the binary number system illustrated in Figure 3-8. Like the decimal number system, the position of each digit in the binary number system is also associated with the system's base number raised to a power. However, in the binary number system, the base number is 2 rather than 10. Starting with the rightmost position, the positions represent 2 raised to a power of 0, 1, 2, 3, and so on. In the binary number 110, the 0 is in the 2^0 position, the middle 1 is in the 2^1 position, and the leftmost 1 is in the 2^2 position. Therefore, the binary number 110 means zero 1s (2^0), one 2 (2^1), and one 4 (2^2). The decimal equivalent of the binary number 110 is 6, which is calculated by adding together 0 + 2 + 4 (zero 1s + one 2 + one 4). In other words, the decimal number 6 is stored in a memory location using the binary number 110. The binary number 11010 means zero 1s (2^0), one 2 (2^1), zero 4s (2^2), one 8 (2^3), and one 16 (2^4). The decimal equivalent of the binary number 11010 is 26, which is calculated by adding together 0 + 2 + 0 + 8 + 16. The decimal equivalent of the last binary number shown in Figure 3-8 is 9 (one 1 + zero 2s + zero 4s + one 8).

HOW TO Use the Binary (*Base 2*) Number System

Binary number	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Decimal equivalent
110						1	1	0	6
11010				1	1	0	1	0	26
1001					1	0	0	1	9

Figure 3-8 How to use the binary (*base 2*) number system

Unlike numeric data, character data (which is data assigned to memory locations that can store characters) is represented in internal memory using ASCII codes. **ASCII** (pronounced *ASK-ee*) stands for American Standard Code for Information Interchange. The ASCII coding scheme assigns a specific numeric code to each character on your keyboard. Figure 3-9 shows a partial listing of the ASCII codes along with their binary representations. The full ASCII chart is contained in Appendix B in this book.

As Figure 3-9 indicates, the uppercase letter A is assigned the ASCII code 65, which is stored in internal memory using the eight bits ("binary digits") 01000001 (one 64 and one 1). Notice that the lowercase version of each letter is assigned a different ASCII code than the letter's uppercase version. The lowercase letter a, for example, is assigned the ASCII code 97, which is stored in internal memory using the eight bits 01100001. This fact indicates that the computer does not consider both cases of a letter to be equivalent. In other words, the uppercase letter A is not the same as the lowercase letter a. This concept will become important when you compare characters in later chapters.

At this point, you may be wondering why the numeric characters on your keyboard are assigned ASCII codes. For example, shouldn't a 9 be stored using the binary number system, as you learned earlier? The answer is that the computer uses the binary number system to store the *number* 9, but it uses the ASCII coding scheme to store the *character* 9. But how does the computer know whether the 9 is a number or a character? The answer to this question is simple: by the memory location's data type.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Character	ASCII	Binary	Character	ASCII	Binary	Character	ASCII	Binary
0	48	00110000	K	75	01001011	g	103	01100111
1	49	00110001	L	76	01001100	h	104	01101000
2	50	00110010	M	77	01001101	i	105	01101001
3	51	00110011	N	78	01001110	j	106	01101010
4	52	00110100	O	79	01001111	k	107	01101011
5	53	00110101	P	80	01010000	l	108	01101100
6	54	00110110	Q	81	01010001	m	109	01101101
7	55	00110111	R	82	01010010	n	110	01101110
8	56	00111000	S	83	01010011	o	111	01101111
9	57	00111001	T	84	01010100	p	112	01110000
:	58	00111010	U	85	01010101	q	113	01110001
;	59	00111011	V	86	01010110	r	114	01110010
A	65	01000001	W	87	01010111	s	115	01110011
B	66	01000010	X	88	01011000	t	116	01110100
C	67	01000011	Y	89	01011001	u	117	01110101
D	68	01000100	Z	90	01011010	v	118	01110110
E	69	01000101	a	97	01100001	w	119	01110111
F	70	01000110	b	98	01100010	x	120	01111000
G	71	01000111	c	99	01100011	y	121	01111001
H	72	01001000	d	100	01100100	z	122	01111010
I	73	01001001	e	101	01100101			
J	74	01001010	f	102	01100110			

Figure 3-9 Partial ASCII chart

Here is an example of the importance of a memory location's data type: Consider a program that displays the message "Enter your pet's age:" on the computer screen. The program stores your response in a variable named `age`. When you press the 9 key on your keyboard in response to the message, the computer uses the data type of the `age` variable to determine whether to store the 9 as a number (using the binary number system) or as a character (using the ASCII coding scheme). If the variable's data type is `int`, the 9 is stored as the binary number 1001 (one 1 + one 8). If the variable's data type is `char`, on the other hand, the 9 is stored as a character using the ASCII code 57, which is represented in internal memory as 00111001 (one 1 + one 8 + one 16 + one 32).

The memory location's data type also determines how the computer interprets a memory location's existing data. If a program instruction needs to access the value stored in a memory location—perhaps to display the value on the screen—the computer uses the memory location's data type to determine the value's data type. To illustrate this point, assume that a memory location named `inputItem` contains the eight bits 01000001. If the memory location's data type is `char`, the computer displays the uppercase letter A on the screen. This is because the computer interprets the 01000001 as the ASCII code 65, which is equivalent to the uppercase letter A. However, if the memory location's data type is `int`, the computer displays the number 65 on the screen because the 01000001 is interpreted as the binary representation of the decimal number 65. In summary, the data type of a memory location is important because it determines how the data is stored when first entered into the memory location. It also determines how the data is interpreted when the memory location is used in an instruction later in the program.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 3-2

1. The `int` data type is one of the fundamental data types in C++.
   a. True
   b. False
2. In the binary number system, the decimal number 27 is represented as _____.
   a. 10111
   b. 11011
   c. 10011
   d. none of the above
3. What is the ASCII code for the uppercase letter H, and how is it represented in the computer's internal memory?
4. Which data type can store a real number?
   a. `double`
   b. `int`
   c. `float`
   d. both a and c

Selecting an Initial Value for a Memory Location

In addition to assigning a name and data type to each variable and named constant used in a program, you should also assign an initial value to each. Assigning an initial (or beginning) value to a memory location is referred to as **initializing**. With the exception of a `bool` memory location, which is initialized using either the C++ keyword `true` or the C++ keyword `false`, you typically initialize a memory location by assigning a literal constant to it. Unlike variables and named constants, literal constants are not memory locations. Rather, a **literal constant** is an item of data that can appear in a program instruction and be stored in a memory location.

Although initializing variables is optional in most programming languages, including C++, it is considered a good programming practice to do so and is highly recommended.

The data type of a literal constant should match the data type of the memory location to which it is assigned. Integers should be assigned to memory locations having the `short` or `int` data type. Memory locations having the `float` or `double` data type should be initialized using real numbers. Integers and real numbers are called **numeric literal constants**; examples include the numbers 146, 0.0, and –2.5. Numeric literal constants can contain numbers, the plus sign, the minus sign, and the decimal point. They can also contain either the lowercase letter e or the uppercase letter E, both of which are used to represent exponential (or e) notation. Scientific programs use e notation to represent very small and very large numbers. Numeric literal constants cannot contain a space, a comma, or a special character, such as the dollar sign ($) or percent sign (%). A numeric literal constant with no decimal place is considered an `int` data type in C++, whereas a numeric literal constant with a decimal place is considered a `double` data type.

Programmers use character literal constants to initialize `char` memory locations. A **character literal constant** is one character enclosed in single quotation marks, such as the letter 'X', the dollar sign '$', and a space ' ' (two single quotation marks with a space character between). A `string` memory location is initialized using a **string literal constant**, which is zero or more characters enclosed in double quotation marks. The word "Hello", the message "Enter your

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

pet's age:", and the **empty string** "" (two double quotation marks with no space between) are examples of string literal constants.

When a program instructs the computer to assign a value to a memory location, the computer first compares the value's data type with the memory location's data type. The comparison is made to verify that the value is appropriate for the memory location. If the value's data type does not match the memory location's data type, the computer uses a process called **implicit type conversion** to convert the value to fit the memory location. For example, if a program initializes a `double` variable named `price` to the integer 9, the computer converts the integer to a real number before storing the value in the variable. The computer does this by appending a decimal point and the number 0 to the end of the integer, like this: 9.0. The computer then stores the real number 9.0 in the `price` variable. When a value is converted from one data type to another data type that can store larger numbers, the value is said to be **promoted**. In this case, the `int` value 9 is promoted to the `double` value 9.0. (As shown earlier in Figure 3-5, the `double` data type can store larger numbers than can the `int` data type.) In most cases, the implicit promotion of values does not adversely affect a program's output.

Ch03-Type Conversion

However, now consider a program that declares an `int` named constant called `MIN_WAGE`. If you use a real number—such as 9.25—to initialize the named constant, the computer first converts the real number to an integer by truncating (dropping off) the decimal portion of the number; it then stores the result in the memory location. As a result, the computer will store the number 9 rather than the number 9.25 in the `MIN_WAGE` memory location. When a value is converted from one data type to another data type that can store only smaller numbers, the value is said to be **demoted**. In this case, the `double` value 9.25 is demoted to the `int` value 9. The implicit demotion of values can adversely affect a program's output. Therefore, it's important to initialize memory locations using values that have the same data type as the memory location.

If a memory location is a named constant, the problem specification and IPO chart will provide the appropriate initial value to use, and that value will remain the same during runtime. (Recall that the contents of a named constant cannot change while the program is running.) The initial value for a variable, on the other hand, is not stated in a problem specification or IPO chart because the user supplies the value while the program is running. Therefore, you usually use the values shown in Figure 3-10.

Recall that a numeric literal constant with a decimal place is treated as a `double` number in C++.

HOW TO Initialize Variables

Data type	Typical initial value
`short`	0
`int`	0
`float`	0.0
`double`	0.0
`string`	"" (empty string)
`char`	' ' (a space)
`bool`	`true` or `false`

Figure 3-10 How to initialize variables

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Figure 3-11 shows the initial values for the variables in the Addison O'Reilly problem.

IPO chart item	Variable name	Data type	Initial value
sale price	salePrice	double	0.0
sales tax rate	taxRate	double	0.0
sales tax	salesTax	double	0.0
cost	cost	double	0.0

Figure 3-11 Initial values for the variables in the Addison O'Reilly problem

Declaring a Memory Location

Now that you know how to select an appropriate name, data type, and initial value for a memory location, you can learn how to declare variables and named constants in a C++ program. We'll begin with variables.

You declare a variable using a **statement**, which is a C++ instruction that causes the computer to perform some action after being executed (processed) by the computer. A statement that declares a variable causes the computer to set aside a memory location with the name, data type, and initial value you provide. A variable declaration statement is one of many different types of statements in C++.

The syntax and examples of a variable declaration statement are shown in Figure 3-12. The term **syntax** refers to the rules of a programming language. One rule in C++ is that all statements must end with a semicolon. Another rule is that the programmer must provide a data type and name for the variable being declared. He or she can also provide an initial value for the variable.

Using the shoe box analogy from the beginning of the chapter, initializing a variable is similar to removing the current contents of a box before using it.

Items that the programmer provides are italicized in a statement's syntax, as shown in Figure 3-12. Items appearing in square brackets—in this case, the = symbol and *initialValue*—are optional. In other words, the C++ language does not require variables to be initialized. However, initializing variables is highly recommended. If you do not provide an initial value, the variable may contain a meaningless value. Programmers refer to the meaningless value as *garbage* because it is the remains of what was last stored in the memory location that the variable now occupies. Items in boldface in a syntax are required. In a variable declaration statement, the semicolon is required; the = symbol is required only when the programmer is providing an initial value for the variable.

HOW TO Declare a Variable in C++

Syntax

dataType variableName [**=** *initialValue*]**;**

Examples

```
int quantity = 0;
double salesTax = 0.0;
bool insured = true;
char grade = ' ';
string city = "";
```

Figure 3-12 How to declare a variable in C++

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

After a variable is declared, you can use its name to refer to it later in the program, such as in a statement that displays the variable's value or uses the value in a calculation. You will learn how to write such statements in Chapter 4. Figure 3-13 shows the declaration statements you would use to declare the four variables in the Addison O'Reilly problem.

IPO chart item	Variable name	Data type	Initial value	C++ statement
sale price	salePrice	double	0.0	double salePrice = 0.0;
sales tax rate	taxRate	double	0.0	double taxRate = 0.0;
sales tax	salesTax	double	0.0	double salesTax = 0.0;
cost	cost	double	0.0	double cost = 0.0;

Figure 3-13 Variable declaration statements for the Addison O'Reilly problem

Now we'll look at how you declare a named constant. Figure 3-14 shows the C++ syntax and includes examples of declaring several named constants. The **const** keyword indicates that the memory location is a named constant, which means its value cannot be changed during runtime. If a program statement attempts to change the value stored in a named constant, the C++ compiler will display an error message. As you learned in Chapter 1, a compiler converts the instructions written in a high-level language (such as C++) into the *0*s and *1*s the computer can understand.

HOW TO Declare a Named Constant in C++

Syntax

const *dataType constantName* **=** *value*;

Examples

```
const double PI = 3.141593;
const int MIN_AGE = 65;
const bool PAID = true;
const char YES = 'Y';
const string BANK = "Harrison Trust and Savings";
```

Figure 3-14 How to declare a named constant in C++

As you can with variables, you can use a named constant in another statement that appears after its declaration statement. For example, after entering the `const double PI = 3.141593;` statement in a program, you can use `PI` in a statement that calculates the area of a circle; the computer will use the value stored in the named constant (3.141593) to calculate the area.

Using named constants in a program has several advantages. First, named constants make a program more self-documenting and easier to modify because they allow the use of meaningful words (such as `PI`) in place of values that are less clear (3.141593). Second, unlike the value stored in a variable, the value stored in a named constant cannot be inadvertently changed during runtime. Third, typing `PI` rather than 3.141593 in a statement is easier and less prone to typing errors. If you do mistype `PI` in a statement that calculates a circle's area—for example, if you type `Pi` rather than `PI`—the C++ compiler will display an error message. Mistyping 3.141593 in the area-calculation statement, however, will not trigger an error message and will result in an incorrect answer. Finally, if a named constant's value needs to be changed in the future, you will need to modify only the declaration statement, rather than all of the statements that use the value.

For more examples of declaring variables and named constants, see the Declaring Memory Locations section in the Ch03WantMore.pdf file.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 3-3

1. Which of the following is a character literal constant?
   a. `'$'`
   b. `'56'`
   c. `'No'`
   d. all of the above
2. Which of the following is a string literal constant?
   a. `"$"`
   b. `"56"`
   c. `"No"`
   d. all of the above
3. If you assign the number 3.49 to an `int` variable, what will the computer store in the variable?
4. Write a C++ statement that declares and initializes an `int` variable named `population`.
5. Write a C++ statement that declares the `INTEREST_RATE` named constant. The constant should have the `double` data type and contain the number 0.05.
6. If you earn $10.25 per hour and worked 20 hours, how much less would you be paid if your hourly rate were inadvertently assigned to an `int` variable in a program?

LAB 3-1 Stop and Analyze

The answers to the labs are contained in the Answers.pdf file.

Study the IPO chart shown in Figure 3-15, and then answer the questions.

Input	Processing	Output
quantity sold item cost ($5.45) item selling price	Processing items: price and cost difference Algorithm: 1. enter the quantity sold and item selling price 2. calculate the price and cost difference by subtracting the item cost from the item selling price 3. calculate the profit by multiplying the price and cost difference by the quantity sold 4. display the profit	profit

Figure 3-15 IPO chart for Lab 3-1

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

QUESTIONS

1. How many memory locations will the problem require?
2. How many of the memory locations will be variables, and how many will be named constants? Why did you choose one type over the other?
3. How would you write the appropriate declaration statements? Use the `int` data type for the quantity sold, and the `double` data type for the remaining input, processing, and output items.

LAB 3-2 Plan and Create

In this lab, you will plan and create an algorithm that displays a 10% commission on a sales amount. The problem specification is shown in Figure 3-16.

Problem specification

Boughton Inc. wants a program that calculates and displays the amount of a salesperson's commission. The commission is calculated by multiplying the salesperson's sales amount by 10%.

Figure 3-16 Problem specification for Lab 3-2

First, analyze the problem, looking for the output first and then for the input. Recall that the output answers the question *What does the user want to see displayed on the screen, printed on paper, or stored in a file?*, and the input answers the question *What information will the computer need to know to display, print, or store the output items?* In this case, the user wants to see the commission amount displayed on the screen. To do this, the computer will need to know the commission rate and the sales amount. The sales amount will be entered by the user, whereas the problem specification indicates that the value to use for the commission rate is 10%. Figure 3-17 shows the input and output items entered in an IPO chart.

Input	Processing	Output
commission rate (10%) sales amount	Processing items: none Algorithm:	commission

Figure 3-17 Partially completed IPO chart showing the input and output items

After determining a problem's output and input, you then plan its algorithm. Recall that most algorithms begin with an instruction to enter the input items into the computer, followed by instructions that process the input items, typically including the items in one or more calculations. Most algorithms end with one or more instructions that display, print, or store the output items. Figure 3-18 shows the completed IPO chart for the commission problem.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Input	Processing	Output
commission rate (10%) sales amount	Processing items: none Algorithm: 1. enter the sales amount 2. calculate the commission by multiplying the sales amount by the commission rate 3. display the commission	commission

Figure 3-18 Completed IPO chart for Lab 3-2

After completing the IPO chart, you then move on to the third step in the problem-solving process, which is to desk-check the algorithm. You begin by choosing a set of sample data for the input values. You then use the values to manually compute the expected output. You will desk-check the current algorithm twice: first using $1328.50 as the sales amount and then using $267.90. For the first desk-check, the commission should be $132.85; for the second desk-check, it should be $26.79. The manual calculations for both desk-checks are shown in Figure 3-19.

First desk-check calculation	Second desk-check calculation
1328.50 (sales amount)	267.90 (sales amount)
* .1 (commission rate in decimal form)	* .1 (commission rate in decimal form)
132.85 (commission)	26.79 (commission)

Figure 3-19 Manual commission calculation for the two desk-checks

Next, you create a desk-check table that contains one column for each input, processing, and output item. You then begin desk-checking the algorithm. Figure 3-20 shows the completed desk-check table. Notice that the amounts in the commission column agree with the results of the manual calculations shown in Figure 3-19.

commission rate	sales amount	commission
~~.1~~	~~1328.50~~	~~132.85~~
.1	267.90	26.79

Figure 3-20 Completed desk-check table for Lab 3-2

After desk-checking an algorithm to ensure that it works correctly, you can begin coding it. You begin by declaring memory locations that will store the values of the input, processing (if any), and output items. The commission problem will need three memory locations to store the commission rate, sales amount, and commission. The sales amount and commission should be stored in variables because the user should be allowed to change the sales amount, which then will change the commission, while the program is running. The commission rate, however, will be stored in a named constant because its value should not change during runtime. The variables and named constant will store real numbers, so you will use the `double` data type for each one. Figure 3-21 shows the input, processing, and output items from the IPO chart, along with the corresponding C++ statements.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

IPO chart information	C++ instructions
Input	
commission rate (10%)	`const double COMM_RATE = 0.1;`
sales amount	`double sales = 0.0;`
Processing	
none	
Output	
commission	`double commission = 0.0;`

Figure 3-21 IPO chart information and C++ instructions for Lab 3-2

LAB 3-3 Modify

Modify the IPO chart shown earlier in Figure 3-18 so that it allows the user to enter the commission rate. Then make the appropriate modifications to Figure 3-21.

LAB 3-4 What's Missing?

Professor Merrita wants a program that calculates and displays the volume of a cylinder, given the cylinder's radius (r) and height (h). The formula for calculating the volume is $\pi r^2 h$. Figure 3-22 contains a list of items and C++ instructions that you can use for this lab. Enter the appropriate input, processing (if any), and output items in a chart similar to the one shown in Figure 3-21. Also enter the corresponding C++ instructions. Determine whether any items or C++ instructions are missing from the list.

Items	C++ statements
height	`double height = 0.0;`
pi (3.14)	`double radius = 0.0;`
radius	`double volume = 0.0;`
volume	

Figure 3-22 Items and statements for Lab 3-4

LAB 3-5 Desk-Check

Create an appropriate algorithm for Lab 3-4, and then desk-check it twice. Use 9 and 6 as the height and radius for the first desk-check, then use 17 and 15.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

LAB 3-6 Debug

Correct the C++ instructions shown in Figure 3-23. The memory locations will store real numbers.

IPO chart information	C++ instructions
Input	
first number	`first = 0.0;`
second number	`second = 0.0;`
third number	`third = 0.0;`
Processing	
sum	`Double sum = 0.0`
Output	
average	`average = 0.0;`

Figure 3-23 IPO chart information and C++ instructions for Lab 3-6

Chapter Summary

The fourth step in the problem-solving process is to code the algorithm, which means to translate it into a language that the computer can understand. You begin by declaring a memory location for each unique input, processing, and output item listed in the IPO chart. The memory locations will store the values of those items while the program is running.

Numeric data is stored in the computer's internal memory using the binary number system. Character data is stored using the ASCII coding scheme.

A memory location can store only one value at a time.

A memory location's data type determines how a value is stored in the location, as well as how the value is interpreted when retrieved.

Programmers can declare two types of memory locations: variables and named constants. You declare a memory location using a statement that assigns a name, a data type, and an initial value to the memory location. The initial value is required when declaring named constants but is optional when declaring variables. However, it is highly recommended that variables be initialized to ensure they don't contain garbage.

In most cases, memory locations are initialized using a literal constant. The exception to this is a `bool` memory location, which is initialized using a C++ keyword (either `true` or `false`).

The data type of a literal constant assigned to a memory location should be the same as the memory location's data type. If the data types do not match, the computer uses implicit type conversion to either promote or demote the value to fit the memory location. Promoting a value does not usually affect a program's output; however, demoting a value may cause a program's output to be incorrect.

The C++ programming language has a set of rules, called syntax, which you must follow when using the language. One rule is that all statements in C++ must end with a semicolon.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Key Terms

ASCII—a coding scheme used to represent character data; stands for American Standard Code for Information Interchange

Binary number system—a system that uses only the two digits 0 and 1; used to represent numeric data in the computer's internal memory

Camel case—a naming convention that capitalizes only the first letter in the second and subsequent words in a memory location's name

Character—a letter, a symbol, or a number that will not be used in a calculation

Character literal constant—one character enclosed in single quotation marks

const—the keyword used to declare a named constant in C++

Decimal number system—a system that represents numbers using the digits 0 through 9

Demoted—refers to the conversion of a number from one data type to another data type that can store only smaller numbers

Empty string—two quotation marks with no space between, like this ""

Fundamental data types—the basic data types built into the C++ language; also called primitive data types or built-in data types

Implicit type conversion—the process the computer follows when converting a numeric value to fit a memory location that has a different data type

Initializing—assigning a beginning value to a memory location

Integer—a whole number, which is a number without a decimal place

Keyword—a word that has a special meaning in the programming language you are using

Literal constant—an item of data that can appear in a program instruction and be stored in a memory location

Named constant—a memory location whose value cannot be changed while a program is running

Numeric literal constants—numbers

Promoted—refers to the conversion of a number from one data type to another data type that can store larger numbers

Real numbers—numbers that contain a decimal place

Runtime—occurs while a program is running

Statement—a C++ instruction that causes the computer to perform some action after being executed (processed) by the computer; all statements in C++ must end with a semicolon

String literal constant—zero or more characters enclosed in double quotation marks

Syntax—the rules you must follow when using a programming language; every programming language has its own syntax

Text—a group of characters treated as one unit and not used in a calculation

User-defined data type—a data type added to the C++ language through the use of a class; an example is the `string` data type

Variable—a memory location whose value can change (vary) while a program is running

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Review Questions

1. The rules you must follow when using a programming language are called its _____ .

   a. guidelines
   b. procedures
   c. regulations
   d. syntax

2. Which of the following declares a variable that can store an integer?

   a. `int quantity = 0;`
   b. `integer quantity = 0;`
   c. `quantity = 0;`
   d. `Int quantity = 0;`

3. A C++ statement must end with a _____ .

   a. colon
   b. comma
   c. period
   d. semicolon

4. The declaration statement for a named constant requires _____ .

   a. a data type
   b. a name
   c. a value
   d. all of the above

5. Which of the following creates a variable that can store real numbers?

   a. `double totalDue = '0.0';`
   b. `double totalDue = 0.0;`
   c. `double totalDue = "0.0";`
   d. `totalDue = 0.0;`

6. Which of the following is a valid name for a variable?

   a. `amount-sold`
   b. `amountSold`
   c. `1stQtrAmountSold`
   d. both b and c

7. Which of the following declares a `char` named constant called `TOP_GRADE`?

   a. `const char TOP_GRADE = 'A';`
   b. `const char TOP_GRADE = "A";`
   c. `const char TOP_GRADE;`
   d. both a and c

8. If a memory location's data type is `int`, how will it store the number 54?

   a. 01110110
   b. 00110110
   c. 00110111
   d. none of the above

9. If memory location's data type is `char`, how will it store the character 6?

   a. 01110110
   b. 00110111
   c. 00110110
   d. none of the above

10. If you use a real number to initialize an `int` variable, the real number will be _____ before it is stored in the variable.

    a. demoted
    b. promoted
    c. reduced
    d. upgraded

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Exercises

Pencil and Paper

TRY THIS

1. Complete the C++ instructions column in Figure 3-24. Use the `double` data type for the input and output items. (The answers to TRY THIS Exercises are located at the end of the chapter.)

IPO chart information	C++ instructions
Input	
food	
rent	
utilities	
car payment	
Processing	
none	
Output	
total expenses	

Figure 3-24

TRY THIS

2. Complete the C++ instructions column in Figure 3-25. The numbers of latex and Mylar balloons purchased will be integers. All of the remaining items will be real numbers. Use the `int` and `double` data types. (The answers to TRY THIS Exercises are located at the end of the chapter.)

IPO chart information	C++ instructions
Input	
latex price	
Mylar price	
latex purchased	
Mylar purchased	
sales tax rate (6%)	
Processing	
total latex cost	
total Mylar cost	
subtotal	
sales tax	
Output	
total cost	

Figure 3-25

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

MODIFY THIS

3. Complete TRY THIS Exercise 2, and then modify the IPO chart information and C++ instructions to indicate that the prices of the latex and Mylar balloons will always be $2.25 and $3.75, respectively.

INTRODUCTORY

4. Aaron Lakely is going to the grocery store to buy some bananas and apples, both of which are sold by the pound. He wants a program that calculates and displays the total cost of his order, including a 3% sales tax. First, create an IPO chart for this problem, and then desk-check the algorithm twice. For the first desk-check, use 2 and 3.5 as the number of pounds of bananas and apples, respectively. And use $0.99 and $1.89 as the price per pound of bananas and apples, respectively. For the second desk-check, use your own set of data. After desk-checking the algorithm, list the input, processing, and output items in a chart similar to the one shown in Figure 3-25, and then enter the appropriate C++ declaration statements.

INTRODUCTORY

5. Archie wants a program that calculates and displays a team's final score in a football game, given the numbers of the team's field goals, touchdowns, one-point conversions, two-point conversions, and safeties. First, create an IPO chart for this problem, and then desk-check the algorithm twice. For the first desk-check, use 3, 2, 2, 0, and 1 as the numbers of field goals, touchdowns, one-point conversions, two-point conversions, and safeties. For the second desk-check, use your own set of data. After desk-checking the algorithm, list the input, processing, and output items in a chart similar to the one shown in Figure 3-25, and then enter the appropriate C++ declaration statements.

INTERMEDIATE

6. Builders Inc. wants a program that allows its salesclerks to enter the diameter of a circle and the price of railing material per foot. The program should calculate and display the total price of the railing material. Use 3.1416 as the value of pi. First, create an IPO chart for this problem, and then desk-check the algorithm twice. For the first desk-check, use 35 feet as the diameter and $2 as the price per foot. For the second desk-check, use 15.5 and $3.50. After desk-checking the algorithm, list the input, processing, and output items in a chart similar to the one shown in Figure 3-25, and then enter the appropriate C++ declaration statements.

INTERMEDIATE

7. Michael wants a program that calculates and displays the percentage of the total points he contributed to his basketball team's final score. Michael will provide the number of two-point baskets, the number of three-point baskets, and the number of free throw points his team made. He will also provide the number of two-point baskets, number of three-point baskets, and number of free throw points he made. First, create an IPO chart for this problem, and then desk-check the algorithm twice. For the first desk-check, use 25, 14, 10, 11, 4, and 3. The first three values represent the team's two-point baskets, three-point baskets, and free throw points. The last three values represent Michael's two-point baskets, three-point baskets, and free throw points. For the second desk-check, use your own set of data. When recording the percentage in the desk-check table, you can round it to one decimal place—for example, 36.3. After desk-checking the algorithm, list the input, processing, and output items in a chart similar to the one shown in Figure 3-25, and then enter the appropriate C++ declaration statements.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

8. Gabrielle receives 52 paychecks each year. She always deposits a specific percentage of her gross pay into her savings account. She also receives a bonus check, which is always more than $250, at the end of the year. She always deposits $150 of her bonus into her savings account. Gabrielle wants a program that calculates and displays the total amount she deposited during the year. Complete an IPO chart for this problem. Desk-check the algorithm twice, using your own sets of data. After desk-checking the algorithm, list the input, processing, and output items in a chart similar to the one shown in Figure 3-25, and then enter the appropriate C++ declaration statements. INTERMEDIATE

9. Cranston Berries sells three types of berries: strawberries, blueberries, and raspberries. Sales have been booming this year and are expected to increase next year. The sales manager wants a program that allows him to enter the projected increase (expressed as a decimal number) in berry sales for the following year. He will also enter the current year's sales for each type of berry. The program should display the projected sales for each berry type. Complete an IPO chart for this problem. Desk-check the algorithm twice, using your own sets of data. After desk-checking the algorithm, list the input, processing, and output items in a chart similar to the one shown in Figure 3-25, and then enter the appropriate C++ declaration statements. INTERMEDIATE

10. Juan wants a program that calculates and displays the number of miles per gallon he drove his car on a recent trip. When he started the trip, the car's gas tank was full and its odometer read 5500. Before reaching his final destination, Juan stopped at two different gas stations to purchase gas. At the first stop, he purchased 15.5 gallons of gas; at that time, the odometer read 5860. At the second stop, he purchased 18.7 gallons of gas; at that time, the odometer read 6280. First, create an IPO chart for this problem. Although specific values are provided for the odometer readings and gallons of gas, do not use named constants for those values. After creating the IPO chart, desk-check the algorithm twice. For the first desk-check, use the values provided in this exercise. For the second desk-check, use your own set of data. After desk-checking the algorithm, list the input, processing, and output items in a chart similar to the one shown in Figure 3-25, and then enter the appropriate C++ declaration statements. ADVANCED

11. Correct the C++ instructions shown in Figure 3-26. SWAT THE BUGS

IPO chart information	C++ instructions
Input	
original price	`double original = 0.0;`
discount rate (10%)	`double DISC_RATE = 10%;`
Processing	
discount	`int discount = 0;`
Output	
new price	`double new price = 0.0;`

Figure 3-26

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Answers to TRY THIS Exercises

1. See Figure 3-27.

IPO chart information	C++ instructions
Input	
food	`double food = 0.0;`
rent	`double rent = 0.0;`
utilities	`double utilities = 0.0;`
car payment	`double car = 0.0;`
Processing	
none	
Output	
total expenses	`double totalExpenses = 0.0;`

Figure 3-27

2. See Figures 3-28.

IPO chart information	C++ instructions
Input	
latex price	`double latexPrice = 0.0;`
Mylar price	`double mylarPrice = 0.0;`
latex purchased	`int latexPurchased = 0;`
Mylar purchased	`int mylarPurchased = 0;`
sales tax rate (6%)	`const double TAX_RATE = .06;`
Processing	
total latex cost	`double totalLatexCost = 0.0;`
total Mylar cost	`double totalMylarCost = 0.0;`
subtotal	`double subtotal = 0.0;`
sales tax	`double salesTax = 0.0;`
Output	
total cost	`double totalCost = 0.0;`

Figure 3-28

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

CHAPTER 4

Completing the Problem-Solving Process

After studying Chapter 4, you should be able to:

- Get numeric and character data from the keyboard
- Display information on the computer screen
- Write arithmetic expressions
- Type cast a value
- Write an assignment statement
- Code the algorithm into a program
- Desk-check a program
- Evaluate and modify a program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Finishing Step 4 in the Problem-Solving Process

Ch04-Preview

The fourth step in the problem-solving process is to code the algorithm into a program. As you learned in Chapter 3, the programmer begins the fourth step by declaring a memory location for each unique input, processing, and output item listed in the problem's IPO chart. The memory locations will store the values of those items while the program is running. Recall that each memory location must be assigned a name and data type. If the memory location is a named constant, it must also be assigned a value. Assigning an initial value to a variable is optional but highly recommended to ensure that the variable does not contain garbage.

Figure 4-1 shows the problem specification, IPO chart information, and variable declaration statements for the Addison O'Reilly problem from Chapter 3. Recall that the `double` data type was selected for these variables because it allows each to store a real number with the greatest precision.

Problem specification

Addison O'Reilly wants a program that calculates and displays the cost of a 4K Ultra HD TV, which is finally on sale at one of the stores in her area. The program should calculate the cost by multiplying the sale price by the state sales tax rate and then adding the result to the sale price.

IPO chart information	**C++ instructions**
Input	
sale price	`double salePrice = 0.0;`
sales tax rate	`double taxRate = 0.0;`
Processing	
sales tax	`double salesTax = 0.0;`
Output	
cost	`double cost = 0.0;`

Algorithm:

1. enter the sale price and sales tax rate
2. calculate the sales tax by multiplying the sale price by the sales tax rate
3. calculate the cost by adding the sales tax to the sale price
4. display the cost

Figure 4-1 Problem specification, IPO chart information, and variable declaration statements

After declaring the necessary memory locations, the programmer begins coding the algorithm. The first instruction in the algorithm shown in Figure 4-1 is to enter the two input items. You will have the user enter the items at the keyboard.

Getting Data from the Keyboard

Ch04-cin

In C++, you use objects to perform standard input and output operations, such as getting a program's input items and displaying its output items. The objects are called **stream objects** because they handle streams. A **stream** is defined in C++ as a sequence of characters.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

In this section, you will learn about the standard input stream object, `cin` (pronounced *see in*). The **`cin`** object tells the computer to pause program execution while the user enters one or more characters at the keyboard; the object temporarily stores the characters as they are typed.

The `cin` object is not a physical object that can be seen or touched. Rather, it is an object created through the use of a class, and it resides in a special area of the computer's internal memory. As you learned in Chapter 3, a class is a group of instructions that the computer uses to create an object.

The `cin` object is typically used with the **extraction operator** (**`>>`**), which extracts (removes) characters from the object and sends them "in" to the data area of the computer's internal memory. The `cin` object and extraction operator allow the user to communicate with the computer while a program is running.

Figure 4-2 illustrates the relationship among the keyboard, `cin` object, extraction operator, and internal memory. As the figure indicates, the characters you type at the keyboard are sent first to the `cin` object, where they remain until the extraction operator removes them, sending them to the data area of the computer's internal memory.

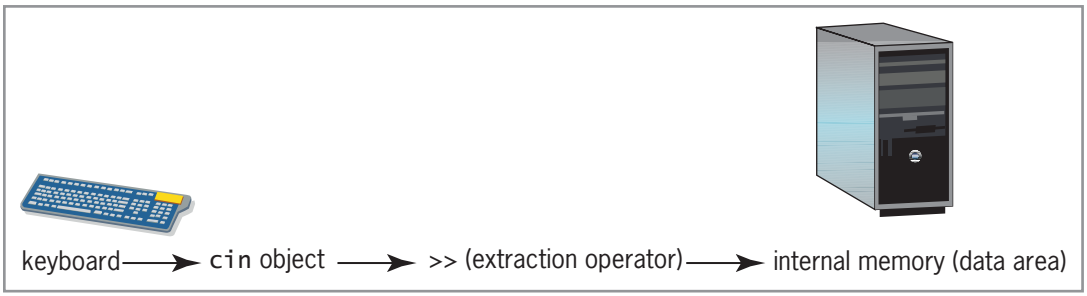


Figure 4-2 Relationship among the keyboard, `cin` object, extraction operator, and internal memory

The extraction operator stops removing characters from the `cin` object when it encounters a **white-space character**, which can be a newline, tab, or blank. You enter a newline character when you press the Enter key on your keyboard. You enter a tab character when you press the Tab key, and you enter a blank character when you press the Spacebar.

Because many strings entered at the keyboard contain one or more blank characters (for example, the string *San Francisco, CA*), the extraction operator is used mainly to get numeric and character data, but not string data. Recall from Chapter 3 that numeric data is a number that will be used in a calculation, while character data is a letter, a symbol, or a number that will *not* be used in a calculation. String data is zero or more characters treated as one unit. This chapter covers inputting numeric and character data only; inputting string data is covered in Chapter 13.

Figure 4-3 shows the syntax and examples of statements that use `cin` and the extraction operator to get numeric and character input from the keyboard. (For clarity, the variable declaration statements are included in the examples.) You can tell that the syntax and examples are statements because a semicolon appears at the end of each. Recall that a statement is a C++ instruction that causes the computer to perform some action after being executed (processed) by the computer. The `cin` portion of the `cin >> price;` statement, for example, tells the computer to pause program execution to allow the user to enter the price at the keyboard. The `cin` object temporarily stores the price as the user types it. When the user presses the Enter key, the extraction operator in the statement removes the price from the `cin` object and sends it to the computer's internal memory, where it is stored in the `price` variable. Similarly, the `cin >> middleInitial;` statement waits for the user to enter a character and ultimately stores the user's response in the `middleInitial` variable.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

HOW TO Use `cin` and >> to Get Numeric or Character Data

Syntax

```
cin >> variableName;
```

semicolon

extraction operator

Example 1

```
double price = 0.0;
cin >> price;
```

Example 2

```
char middleInitial = ' ';
cin >> middleInitial;
```

Figure 4-3 How to use `cin` and >> to get numeric or character data

The input items in the Addison O'Reilly problem are numeric, so you can use the `cin` object and extraction operator to get the items from the user at the keyboard. The appropriate statements are shaded in Figure 4-4.

IPO chart information	**C++ instructions**
Input	
sale price	`double salePrice = 0.0;`
sales tax rate	`double taxRate = 0.0;`
Processing	
sales tax	`double salesTax = 0.0;`
Output	
cost	`double cost = 0.0;`
Algorithm:	
1. enter the sale price and sales tax rate	`cin >> salePrice;`
	`cin >> taxRate;`
2. calculate the sales tax by multiplying the sale price by the sales tax rate	
3. calculate the cost by adding the sales tax to the sale price	
4. display the cost	

Figure 4-4 Input statements for the Addison O'Reilly problem

When the `cin >> salePrice;` statement is processed by the computer while the program is running, a blank window containing a blinking cursor will appear on the computer screen. The blinking cursor indicates that the computer is waiting for the user to enter *something*, but it does not indicate what that *something* is. Should the user enter an age, a price, or a middle initial? You indicate the type of information to enter by displaying a message, called a **prompt**, on the computer screen.

Ch04-cout

Displaying Messages on the Computer Screen

As you learned earlier, you use objects to perform standard input and output operations in C++. In this section, you will learn about the standard output stream object, `cout` (pronounced *see out*). The **`cout`** object is used with the **insertion operator** (<<) to send information "out" to

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

the user via the computer screen. The information can be any combination of literal constants, named constants, or variables. The `cout` object and insertion operator allow the computer to communicate with the user while a program is running.

Figure 4-5 shows the syntax and examples of statements that use `cout` and the insertion operator. Notice the required semicolon that appears at the end of the syntax and examples. Also notice that you can include more than one insertion operator in a statement. The four `cout` statements in the figure tell the computer to display different types of messages on the computer screen. The first two messages prompt the user to enter specific items of data: a price and a middle initial. The third message simply alerts the user that the program has ended. The message displayed by the last `cout` statement contains the string "Bonus: $" and the contents of the `bonusAmt` variable. If the variable contains the number 540.75, the statement will display the message "Bonus: $540.75" on the computer screen. The `endl` in the last `cout` statement is one of the stream manipulators in C++. A **stream manipulator** allows you to manipulate (or manage) the characters in either the input or output stream. When used with the `cout` object, the **`endl`** stream manipulator advances the cursor to the next line on the computer screen, which is equivalent to pressing the Enter key. When typing `endl` (which stands for *end of line*) in a statement, be sure to type the lowercase letter *l* rather than the number *1*.

Like the `cin` object, the `cout` object is created in the computer's internal memory through the use of a class.

HOW TO Use the `cout` Object

Syntax

```
cout << item1 [<< item2 << itemN];
```

(insertion operator; semicolon)

Examples

```
cout << "Enter the price: ";
cout << "What is your middle initial? ";
cout << "End of program";
cout << "Bonus: $" << bonusAmt << endl;
```

(stream manipulator)

Note: The last `cout` statement is equivalent to the following three lines of code:

```
cout << "Bonus: $";
cout << bonusAmt;
cout << endl;
```

Figure 4-5 How to use the `cout` object

The Addison O'Reilly program will use the `cout` object and insertion operator to display a meaningful prompt for each of the input items. Each prompt should be entered above its corresponding `cin` statement so that it will appear *before* the user is expected to enter the information. The two prompts are shaded in Figure 4-6. Keep in mind that the prompts merely display a message on the computer screen. They don't allow the user to actually enter the data being requested; for that, you need the `cin` object and extraction operator.

Also shaded in Figure 4-6 is the statement that displays the cost of the TV on the computer screen; the statement corresponds to the last instruction in the algorithm. Although it is customary to code an algorithm's instructions in the order they appear in the algorithm, the statement to display the cost is included now simply because this section covers displaying messages on the computer screen.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

IPO chart information	C++ instructions
Input	
sale price	`double salePrice = 0.0;`
sales tax rate	`double taxRate = 0.0;`
Processing	
sales tax	`double salesTax = 0.0;`
Output	
cost	`double cost = 0.0;`
Algorithm:	
1. enter the sale price and sales tax rate	`cout << "Enter the sale price: ";`
	`cin >> salePrice;`
	`cout << "Enter the sales tax rate: ";`
	`cin >> taxRate;`
2. calculate the sales tax by multiplying the sale price by the sales tax rate	
3. calculate the cost by adding the sales tax to the sale price	
4. display the cost	`cout << "Cost: $" << cost << endl;`

Figure 4-6 Prompts and output statement for the Addison O'Reilly problem

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 4-1

1. Which of the following stores the value entered at the keyboard in a variable named `population`?
   a. `cin >> population;`
   b. `cin << population;`
   c. `cout >> population;`
   d. `population << cout;`
2. Which of the following displays the contents of the `quantity` variable on the computer screen?
   a. `cin >> quantity;`
   b. `cin << quantity;`
   c. `cout >> quantity;`
   d. `cout << quantity;`
3. Which of the following is considered a white-space character in C++?
   a. a blank
   b. a tab
   c. a newline
   d. all of the above
4. The insertion operator looks like this: _____ .

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Arithmetic Operators in C++

Instructions 2 and 3 in the algorithm shown in Figure 4-6 involve arithmetic calculations. You direct the computer to perform a calculation by writing an arithmetic expression that contains one or more arithmetic operators. Figure 4-7 lists the standard arithmetic operators available in C++, along with their precedence numbers. The precedence numbers indicate the order in which the computer performs the operation in an expression. Operations with a precedence number of 1 are performed before operations with a precedence number of 2, and so on. However, you can use parentheses to override the order of precedence because operations within parentheses are always performed before operations outside of parentheses.

C++ also provides arithmetic assignment operators, which you will learn about later in this chapter.

Operator	Operation	Precedence number
( )	override normal precedence rules	1
–	negation (reverses the sign of a number)	2
*, /, %	multiplication, division, and modulus arithmetic	3
+, –	addition and subtraction	4

Ch04-Arithmetic Operators

Figure 4-7 Standard arithmetic operators and their order of precedence

Although the negation and subtraction operators use the same symbol (a hyphen), there is a difference between both operators: The negation operator is unary, whereas the subtraction operator is binary. Unary and binary refer to the number of operands required by the operator. Unary operators require one operand, whereas binary operators require two operands. The expression `-7` uses the unary negation operator to turn the positive number 7 into a negative number. The expression `9 - 4`, on the other hand, uses the binary subtraction operator to subtract the number 4 from the number 9.

One of the arithmetic operators listed in Figure 4-7, the modulus (or remainder) operator (`%`), might be less familiar to you. The **modulus operator** is used to divide two integers and returns the remainder of the division; the remainder is always an integer. For example, the expression `211 % 4` (read 211 mod 4) equals 3, which is the remainder after dividing 211 by 4. A common use for the modulus operator is to determine whether a number is even or odd. If you divide a number by 2 and the remainder is 0, the number is even; if the remainder is 1, however, the number is odd.

The modulus operator is used to divide integers only, and the result (remainder) is always an integer.

Some of the operators listed in Figure 4-7, like the addition and subtraction operators, have the same precedence number. When an expression contains more than one operator having the same priority, those operators are evaluated from left to right. In the expression `13 + 8 / 4 - 2 * 6`, the division is performed first, followed by the multiplication, addition, and subtraction. The result of the expression is the number 3, as shown in Figure 4-8. You can use parentheses to change the order in which the operators in an expression are evaluated, like this: `13 + 8 / (4 - 2) * 6`. The expression containing the parentheses evaluates to 37 rather than to 3, as shown in Figure 4-8. This is because the parentheses tell the computer to perform the subtraction operation first.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Original expression	`13 + 8 / 4 - 2 * 6`
The division is performed first	`13 + 2 - 2 * 6`
The multiplication is performed next	`13 + 2 - 12`
The addition is performed next	`15 - 12`
The subtraction is performed last	`3`
Original expression	`13 + 8 / (4 - 2) * 6`
The subtraction is performed first	`13 + 8 / 2 * 6`
The division is performed next	`13 + 4 * 6`
The multiplication is performed next	`13 + 24`
The addition is performed last	`37`

Figure 4-8 Expressions containing more than one operator having the same precedence

Type Conversions in Arithmetic Expressions

Ch04-Type Conversions

In Chapter 3, you learned about implicit type conversions in statements that declare memory locations. Recall that, if necessary, the computer will either promote or demote the value in a declaration statement to match the memory location's data type. The computer also makes implicit type conversions when performing an arithmetic operation with two values having different data types. In those cases, the value with the lower-ranking data type is always promoted, temporarily, to the higher-ranking data type. A data type ranks higher than another data type if it can store larger numbers. The value returns to its original data type after the arithmetic operation is performed.

Figure 4-9 shows examples of expressions that require implicit type conversions. As you study the examples, keep in mind that a number with a decimal place is considered a `double` number in C++. The figure also explains how each expression is evaluated by the computer. In Examples 2 and 4, `num` is an `int` variable that contains the number 10. When a variable name appears in an expression, the computer uses the value stored in the variable when evaluating the expression.

When both operands in an expression are integers, the result is an integer. When both are `double` numbers, the result is a `double` number. When one operand is an integer and the other is a `double` number, the result is a `double` number.

Example 1 `75.5 * 2`

The integer 2 is implicitly promoted to the **`double`** number 2.0 before being multiplied by the **`double`** number 75.5. The result is the **`double`** number 151.0.

Example 2 `3 * (1.5 + num)`

1. The value stored in the **`num`** variable (the integer 10) is implicitly promoted to the **`double`** number 10.0 before it is added to the **`double`** number 1.5. The result is the **`double`** number 11.5.
2. The integer 3 is implicitly promoted to the **`double`** number 3.0 before being multiplied by the **`double`** number 11.5 (the result of Step 1). The result is the **`double`** number 34.5.

Example 3 `15 / 0.4`

The integer 15 is implicitly promoted to the **`double`** number 15.0 before it is divided by the **`double`** number 0.4. The result is the **`double`** number 37.5.

Example 4 `num / 4.0`

The value stored in the **`num`** variable (the integer 10) is implicitly promoted to the **`double`** number 10.0 before being divided by the **`double`** number 4.0. The result is the **`double`** number 2.5.

Figure 4-9 Examples of expressions that require implicit type conversions

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

At this point, it is important to highlight what happens when you divide one integer by another integer in C++ because the result may not be what you expect. When both the dividend and divisor are integers, the quotient is always an integer in C++. For example, the result of the expression `24 / 10` is the integer 2 rather than the real number 2.4. So how do you get the quotient as a real number? You do so by converting at least one of the integers involved in the division operation to a real number. If the integer is a numeric literal constant, you can convert it to a real number by adding .0 to it, like this: `24.0 / 10`. When the computer evaluates the `24.0 / 10` expression, it will implicitly convert the integer 10 to the `double` number 10.0 before dividing it into the `double` number 24.0; the result will be the `double` number 2.4. You can also use either the expression `24 / 10.0` or the expression `24.0 / 10.0`; both expressions evaluate to 2.4. Similarly, if the `num` variable contains the integer 10, the result of the expression `24.0 / num` is also 2.4. This is because the integer stored in the `num` variable will be implicitly promoted to the `double` data type before the division is performed.

But what if neither of the integers involved in the division operation is a literal constant? For example, what if both the dividend and divisor are `int` variables? Now how do you get the quotient as a real number? In that case, you need to explicitly convert the value stored in at least one of the `int` variables in the expression to either the `double` or `float` data type. (However, recall that the programs in this book will use the `double` data type for real numbers.) You can use the `static_cast` operator to perform the conversion.

The `static_cast` Operator

C++ provides the **`static_cast` operator** for explicitly converting data from one data type to another. This type of conversion is called an **explicit type conversion** or a **type cast**. In the operator's syntax, which is shown in Figure 4-10, *data* can be a literal constant, a named constant, or a variable, and *dataType* is the data type to which you want the data converted. When the computer processes the operator, it first makes a temporary copy of the data's value in memory. It then converts only the copied value to the specified data type; the `static_cast` operator does not affect the original value.

HOW TO Use the `static_cast` Operator

Syntax
`static_cast<dataType>(data)`

Example 1 `static_cast<double>(numA) / static_cast<double>(numB)`

1. The `numA` integer (18) is *explicitly* promoted to the `double` number 18.0.
2. The `numB` integer (4) is *explicitly* promoted to the `double` number 4.0.
3. Step 1's result (18.0) is divided by Step 2's result (4.0), giving the `double` number 4.5.

Example 2 `static_cast<double>(numA) / numB`

1. The `numA` integer (18) is *explicitly* promoted to the `double` number 18.0.
2. The `numB` integer (4) is *implicitly* promoted to the `double` number 4.0.
3. Step 1's result (18.0) is divided by Step 2's result (4.0), giving the `double` number 4.5.

Figure 4-10 How to use the `static_cast` operator *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Example 3 `numA / static_cast<double>(numB)`

1. The numB integer (4) is *explicitly* promoted to the double number 4.0.
2. The numA integer (18) is *implicitly* promoted to the double number 18.0.
3. Step 2's result (18.0) is divided by Step 1's result (4.0), giving the double number 4.5.

Example 4 `7.35 * static_cast<double>(numA)`

1. The numA integer (18) is *explicitly* promoted to the double number 18.0.
2. Step 1's result (18.0) is multiplied by the double number 7.35, giving the double number 132.3.

Note: The `static_cast` operator is not required in Example 4 because the computer will *implicitly* convert the contents of the numA variable to the double data type before performing the multiplication operation.

Example 5 `const float PRICE = static_cast<float>(35.98);`

The double number 35.98 is *explicitly* demoted to the `float` data type before being stored in the PRICE named constant.

Figure 4-10 How to use the `static_cast` operator

Study closely the examples shown in Figure 4-10. In the examples, numA and numB are `int` variables that contain the numbers 18 and 4, respectively. When processing the expression in Example 1, the computer makes a copy of the integer 18 and a copy of the integer 4 in its internal memory. It then converts only the copied integers to the double numbers 18.0 and 4.0, respectively. The values stored in the numA and numB variables are not converted; they are still integers. After performing the division, which results in a quotient of 4.5, the computer removes the copied values from its internal memory. A similar process is followed when the computer processes the expressions shown in Examples 2 and 3, either of which can also be used to divide the numA integer by the numB integer, returning a double number as the quotient.

For more examples of type conversions in arithmetic expressions, see the Type Conversions section in the Ch04WantMore.pdf file.

The expression in Example 4 uses the `static_cast` operator to explicitly promote the integer stored in the numA variable to the double data type before it is multiplied by the double number 7.35. Although the same answer would be achieved with implicit type conversion, the type casting makes the programmer's intent clear to anyone reading the program. The statement in Example 5 uses the `static_cast` operator to explicitly demote the double number 35.98 to the `float` data type. (Recall that the double data type ranks higher than the `float` data type because it can store larger numbers and also store numbers with greater precision.)

In most cases, the result of an arithmetic expression is assigned to a variable in a program. You do this using an assignment statement.

Assignment Statements

You can use an **assignment statement** to assign a value to a variable while a program is running. It cannot, however, be used to assign a value to a named constant because the contents of a named constant cannot be changed during runtime. When a value is assigned to a variable, it replaces the existing value in the memory location; this is because a variable can store only one value at any time.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Ch04-Assignment Statement

Figure 4-11 shows both the syntax and examples of an assignment statement in C++. (For clarity, the variable declaration statements are included in the examples.) The = symbol in an assignment statement is called the **assignment operator**. When the computer processes an assignment statement, it first evaluates the expression that appears on the right side of the assignment operator and then stores the result in the variable whose name appears on the left side of the assignment operator. The expression can include one or more items, and the items can be literal constants, named constants, variables, or operators such as arithmetic operators or the `static_cast` operator.

As with declaration statements, the data type of the expression in an assignment statement must match the data type of the variable to which the expression is assigned; otherwise, the computer implicitly converts the value to fit the memory location. However, recall from Chapter 3 that implicit type conversions—more specifically, those that demote the value—do not always give you the expected results. Therefore, it is considered a good programming practice to use a type cast, if necessary, to explicitly convert the value of the expression to fit the memory location. For example, the `static_cast` operator in Example 3 in Figure 4-11 explicitly converts the integer stored in the `numA` variable to the `double` data type.

HOW TO Write an Assignment Statement

Syntax
variableName = expression;

Example 1
```
int population = 0;
population = 5600;
```
The assignment statement assigns the integer 5600 to the `population` variable.

Example 2
```
int females = 5;
int males = 7;
int total = 0;
total = females + males;
```
The assignment statement assigns the integer 12 to the `total` variable.

Example 3
```
int numA = 18;
int numB = 4;
double quotient = 0.0;
quotient = static_cast<double>(numA) / numB;
```
The assignment statement assigns the `double` number 4.5 to the `quotient` variable.

Example 4
```
char middleInitial = ' ';
middleInitial = 'P';
```
The assignment statement assigns the letter P to the `middleInitial` variable.

Example 5
```
string state = "";
state = "NJ";
```
The assignment statement assigns the string "NJ" to the `state` variable.

Example 6
```
const double TAX_RATE = 0.045;
double sale = 50.0;
double tax = 0.0;
tax = sale * TAX_RATE;
```
The assignment statement assigns the `double` number 2.25 to the `tax` variable.

When writing assignment statements that contain a calculation, remember to "compute on the right and assign to the left."

Figure 4-11 How to write an assignment statement *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

For more examples of assignment statements, see the Assignment Statements section in the Ch04WantMore.pdf file.

Example 7

```
double price = 100.0;
price = price * 0.95;
```

The assignment statement assigns the `double` number 95.0 to the `price` variable.

Figure 4-11 How to write an assignment statement

It is easy to confuse an assignment statement with a variable declaration statement in C++. For example, the assignment statement `quantity = 75;` looks very similar to the variable declaration statement `int quantity = 75;`. The noticeable difference is the data type that appears at the beginning of the declaration statement. However, keep in mind that a variable declaration statement creates (and optionally initializes) a *new* variable. An assignment statement, on the other hand, assigns a value to an *existing* variable.

Shaded in Figure 4-12 are the appropriate calculation statements for the Addison O'Reilly problem. Because all of the items in both calculation statements have the same data type, neither statement requires any implicit or explicit type conversions.

For more examples of coding algorithms, see the Coding Algorithms section in the Ch04WantMore.pdf file.

IPO chart information	**C++ instructions**
Input	
sale price	`double salePrice = 0.0;`
sales tax rate	`double taxRate = 0.0;`
Processing	
sales tax	`double salesTax = 0.0;`
Output	
cost	`double cost = 0.0;`
Algorithm:	

1. enter the sale price and sales tax rate	`cout << "Enter the sale price: ";`
	`cin >> salePrice;`
	`cout << "Enter the sales tax rate: ";`
	`cin >> taxRate;`

2. calculate the sales tax by multiplying the sale price by the sales tax rate	`salesTax = salePrice * taxRate;`
3. calculate the cost by adding the sales tax to the sale price	`cost = salePrice + salesTax;`
4. display the cost	`cout << "Cost: $" << cost << endl;`

Figure 4-12 Calculation statements for the Addison O'Reilly problem

You have finished coding the algorithm, which is Step 4 in the problem-solving process. At this point, it is important to caution you about a problem you might encounter when using real numbers in calculations. As mentioned in Chapter 3, not all real numbers can be represented exactly within the computer's internal memory. As a result, the answer to some calculations may not be accurate to the penny. For example, the expression `7.0 / 3.0` yields a quotient of 2.333333... (with the number 3 repeating indefinitely). The number 2.333333... can be stored only as an approximation in the computer's internal memory. Because many real numbers cannot be stored precisely, some programmers do not use them in monetary calculations where accuracy to the penny is required. Instead, some

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

programmers use integers, while others use special classes designed to perform precise calculations using real numbers. These special classes can be purchased from third-party vendors, such as Rogue Wave Software. You can learn more about the problem of using real numbers in monetary calculations by completing Computer Exercise 16 at the end of this chapter. The exercise also allows you to explore the use of integers in calculations. (Your instructor may require you to use integers in monetary calculations; however, for simplicity, this book will use real numbers.)

Arithmetic Assignment Operators

In addition to the standard arithmetic operators listed earlier in Figure 4-7, C++ also provides several **arithmetic assignment operators**, which can be used to abbreviate assignment statements that contain an arithmetic operator. However, the assignment statement must have the following format, in which *variableName* is the name of the same variable: *variableName = variableName arithmeticOperator value;*. For example, you can use the multiplication assignment operator (*=) to abbreviate the statement `price = price * 1.05;` as follows: `price *= 1.05;`. Both statements tell the computer to multiply the contents of the `price` variable by 1.05 and then store the result in the `price` variable. Arithmetic assignment operators are also called compound assignment operators or shortcut operators.

In the assignment statement's syntax, *value* is usually either a constant (literal or named) or the name of a different variable.

Figure 4-13 shows the syntax of a C++ statement that uses an arithmetic assignment operator. The figure also lists the most commonly used arithmetic assignment operators; each consists of an arithmetic operator followed immediately by the assignment operator (=). The arithmetic assignment operators do not contain a space; in other words, the addition assignment operator is +=, not + =. Including a space in an arithmetic assignment operator is a common syntax error. Also included in the figure are examples of using the operators. To abbreviate an assignment statement, you simply remove the variable name that appears on the left side of the assignment operator (=), and then put the assignment operator immediately after the arithmetic operator.

Ch04-Arithmetic Assignment Operators

HOW TO Use an Arithmetic Assignment Operator

Syntax

variableName arithmeticAssignmentOperator value;

Operator	Purpose
+=	addition assignment
-=	subtraction assignment
*=	multiplication assignment
/=	division assignment
%=	modulus assignment

Example 1

Original statement:
```
rate = rate + .05;
```
Abbreviated statement:
```
rate += .05;
```

Example 2

Original statement:
```
price = price - discount;
```
Abbreviated statement:
```
price -= discount;
```

Figure 4-13 How to use an arithmetic assignment operator

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 4-2

1. Write a C++ assignment statement that multiplies the value stored in an `int` variable named `quantity` by the number 2.5, and then assigns the result to a `double` variable named `totalDue`. Use implicit type conversion.
2. Rewrite the answer to Question 1 using a type cast (explicit type conversion).
3. In C++, the expression `9 / 2 + 1.5` will evaluate to _____, when it should evaluate to _____. Why does the expression evaluate incorrectly?
4. Rewrite the expression from Question 3 so that it will evaluate correctly.
5. Rewrite the `ordered = ordered - 7;` statement using an arithmetic assignment operator.

Step 5—Desk-Check the Program

Ch04-Addison Desk-Check

The fifth step in the problem-solving process is to desk-check the program to make sure that each instruction in the algorithm was translated correctly. You should desk-check the program using the same sample data used to desk-check the algorithm, and the results of both desk-checks should be the same. For your convenience when comparing the results of both desk-checks later in this section, Figure 4-14 shows the desk-check table that you completed for the Addison O'Reilly algorithm in Chapter 2.

sale price	sales tax rate	sales tax	cost
~~2300~~	~~.05~~	~~115~~	~~2415~~
5200	.03	156	5356

Figure 4-14 Algorithm's desk-check table from Chapter 2

When desk-checking a program, you first place the names of the declared memory locations (variables and named constants) in a new desk-check table, along with each memory location's initial value. Figure 4-15 shows the result of desk-checking the variable declaration statements shown earlier in Figure 4-12.

salePrice	taxRate	salesTax	cost
0.0	0.0	0.0	0.0

Figure 4-15 Variable names and initial values entered in the program's desk-check table

Next, you desk-check the remaining C++ instructions in order, recording in the desk-check table any changes made to the contents of the variables. In the Addison O'Reilly program, the first instruction following the declaration statements is the `cout << "Enter the sale price: ";` statement. The statement displays a prompt on the computer screen, but it does not make any changes to the contents of the program's variables; therefore, no entry is necessary in the desk-check table.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The next statement, `cin >> salePrice;`, allows the user to enter the sale price, and it stores the user's response in the `salePrice` variable. If the user enters the number 2300, the statement stores the number 2300.0 in the variable because the variable has the `double` data type. Therefore, you record 2300.0 in the `salePrice` column in the desk-check table. (As you learned in Chapter 2, some programmers find it helpful to lightly cross out the previous value in a column before recording a new value; however, this is not a requirement.)

The next statement in the program is a `cout` statement that merely prompts the user to enter the sales tax rate. The `cin >> taxRate;` statement that follows it waits for the user's response and then stores the response in the `taxRate` variable. If the user enters the number .05, you record .05 in the desk-check table's `taxRate` column. Figure 4-16 shows the input values recorded in the program's desk-check table.

salePrice	taxRate	salesTax	cost
~~0.0~~	~~0.0~~	~~0.0~~	0.0
2300.0	.05		

Figure 4-16 Input values entered in the program's desk-check table

The `salesTax = salePrice * taxRate;` statement in the program multiplies the contents of the `salePrice` variable (2300.0) by the contents of the `taxRate` variable (.05) and then stores the result (115.0) in the `salesTax` variable. As you learned earlier, the expression that appears on the right side of the assignment operator is always evaluated first, and then the result is stored in the variable whose name appears on the left side of the assignment operator. As a result of this statement, you record 115.0 in the `salesTax` column in the desk-check table, as shown in Figure 4-17.

Remember that assignment statements "compute on the right and assign to the left."

salePrice	taxRate	salesTax	cost
~~0.0~~	~~0.0~~	~~0.0~~	0.0
2300.0	.05	115.0	

Figure 4-17 Sales tax amount entered in the desk-check table

The next statement, `cost = salePrice + salesTax;`, adds the contents of the `salePrice` variable (2300.0) to the contents of the `salesTax` variable (115.0) and then stores the result (2415.0) in the `cost` variable. In the desk-check table, you record the number 2415.0 in the `cost` column, as shown in Figure 4-18.

salePrice	taxRate	salesTax	cost
~~0.0~~	~~0.0~~	~~0.0~~	~~0.0~~
2300.0	.05	115.0	2415.0

Figure 4-18 Cost amount entered in the desk-check table

The last statement in the program displays a message along with the contents of the `cost` variable. You have completed desk-checking the program using the first set of test data. If you compare the second row of values in Figure 4-18 with the first row of values shown earlier in

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Figure 4-14, you will notice that the results obtained when desk-checking the program are the same as the results obtained when desk-checking the algorithm.

For this program, you will perform one more desk-check. However, you should always perform several desk-checks (using different data) to make sure that a program works correctly. For the second desk-check, you will use 5200 and .03 as the sale price and sales tax rate, respectively. Each time you desk-check a program, remember to complete all of the program's statements, beginning with the first statement and ending with the last statement. In this case, the first statement declares and initializes the `salePrice` variable, and the last statement displays the cost on the computer screen. The completed desk-check table is shown in Figure 4-19. Here again, if you compare the fourth row of values in Figure 4-19 with the second row of values in Figure 4-14, you will notice that the program's results are the same as the algorithm's results.

salePrice	taxRate	salesTax	cost
~~0.0~~	~~0.0~~	~~0.0~~	~~0.0~~
~~2300.0~~	~~.05~~	~~115.0~~	~~2415.0~~
~~0.0~~	~~0.0~~	~~0.0~~	~~0.0~~
5200.0	.03	156.0	5356.0

Figure 4-19 Program's desk-check table showing the results of the second desk-check

Step 6—Evaluate and Modify the Program

The final step in the problem-solving process is to evaluate and modify (if necessary) the program. You evaluate a program by entering your C++ instructions into the computer, along with other instructions that you will learn about later in this section, and then using the computer to run (execute) the program. While the program is running, you enter the same sample data used when desk-checking the program. If the results obtained when running the program differ from those shown in the program's desk-check table, it indicates that the program contains errors, referred to as **bugs**. The bugs must be located and removed from the program before the program is released to the user. The programmer's job is not finished until the program runs without errors and produces the expected results.

The process of locating and correcting the bugs in a program is called **debugging**. Program bugs are typically caused by either syntax errors or logic errors. A **syntax error** occurs when you break one of the programming language's rules. Most syntax errors are a result of typing errors that occur when entering instructions, such as typing `cut` (instead of `cout`) or neglecting to enter a semicolon at the end of a statement. In most cases, syntax errors are easy to both locate and correct because they trigger an error message from the C++ compiler. The error message indicates the general vicinity of the error and includes a brief description of the error. Appendix C contains a list of common syntax errors.

Another type of error, called a runtime error, can occur while a program is running. Entering a letter when a number is expected will cause a runtime error.

Unlike syntax errors, logic errors are much more difficult to find because they do not trigger an error message from the compiler. A **logic error** can occur for a variety of reasons, such as forgetting to enter an instruction or entering the instructions in the wrong order. Some logic errors occur as a result of calculation statements that are correct syntactically but incorrect mathematically. For example, consider the statement `average = midterm + final / 2;`, which is supposed to calculate the average of two numeric test scores. The statement's syntax is correct, but it is incorrect mathematically because it tells the computer to divide the contents of the `final` variable by 2 and then add the quotient to the contents

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

of the `midterm` variable. (Recall that division is performed before addition in an arithmetic expression.) The correct instruction for calculating the average is `average = (midterm + final) / 2;`. The parentheses tell the computer to perform the addition before the division.

In order to enter your C++ instructions into the computer, you need to have access to a text editor, more simply referred to as an editor. The instructions you enter are called **source code**. You save the source code in a file on a disk, giving it the filename extension .cpp (which stands for C plus plus). The .cpp file is called the **source file**.

In order to run (execute) the code contained in the source file, you need a C++ compiler. As you learned in Chapter 1, a compiler translates high-level instructions into machine code—the *0*s and *1*s that the computer can understand. Machine code is usually called **object code**. The compiler generates the object code and saves it in a file whose filename extension is .obj (which stands for object). The file containing the object code is called the **object file**.

After the compiler creates the object file, it then invokes another program called a linker. The **linker** combines the object file with other machine code necessary for your C++ program to run correctly, such as machine code that allows the program to communicate with input and output devices. The linker produces an **executable file**, which is a file that contains all of the machine code necessary to run your C++ program as many times as desired without the need for translating the program again. The executable file has an extension of .exe on its filename. (The *exe* stands for executable.) Figure 4-20 illustrates the sequence of steps followed when translating your source code into executable code.

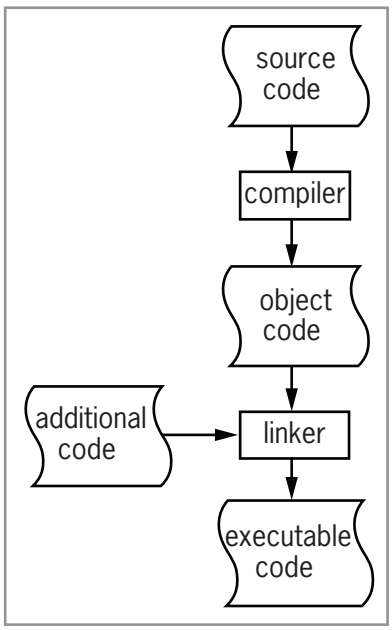


Figure 4-20 Process by which source code is translated into executable code

Many C++ development tools contain both the editor and compiler in one integrated environment, referred to as an **IDE** (**Integrated Development Environment**). Examples include Microsoft Visual C++, Dev C++, Code::Blocks, and Xcode. Other C++ development tools, called command-line compilers, contain only the compiler and require you to use a general-purpose editor (such as Notepad, WordPad, or vi) to enter the program instructions into the computer. As noted in this

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

book's *Read This Before You Begin* page, this book is accompanied by video files that show you how to install various development tools on your computer. Also accompanying this book are files containing step-by-step directions for using these development tools to enter and run some of the C++ programs in this book; the files are in PDF format. However, keep in mind that you are not limited to the development tools supported by the videos and PDF files. You can enter and run the programs in this book using most C++ development tools, often with little or no modification. Your instructor or technical support person will provide you with the appropriate instructions if you are using a different development tool.

Figure 4-21 shows the source code for the Addison O'Reilly program. Each line in the figure is numbered so that it is easier to refer to it in the text; you do not enter the line numbers in the program. The unshaded lines of code are your C++ instructions from Figure 4-12. You do not have to align the initial values in the declaration statements as shown in the figure. However, doing so makes it easier to verify that each memory location has been initialized.

Besides entering your C++ instructions, you also need to enter other instructions in the source file. Some of the additional instructions are required by the C++ compiler, while others are optional but highly recommended. The additional instructions are shaded in Figure 4-21.

```
 1 //Fig4-21.cpp - displays the cost of a TV
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 using namespace std;
 6
 7 int main()                    function header
 8 {
 9    //declare variables
10    double salePrice = 0.0;
11    double taxRate   = 0.0;
12    double salesTax  = 0.0;
13    double cost      = 0.0;
14
15    //enter input items
16    cout << "Enter the sale price: ";
17    cin >> salePrice;
18    cout << "Enter the sales tax rate: ";
19    cin >> taxRate;
20
21    //calculate the sales tax and cost
22    salesTax = salePrice * taxRate;
23    cost = salePrice + salesTax;
24
25    //display output item
26    cout << "Cost: $" << cost << endl;
27
28    return 0;
29 } //end of main function
```

Figure 4-21 Addison O'Reilly program

The two forward slashes (//) on lines 1, 2, 9, 15, 21, 25, and 29 indicate that what follows on that line is a comment. A **comment** is simply a message to the person reading the program and is referred to as internal documentation. The comments on lines 1 and 2 indicate the program's name and purpose, as well as the programmer's name and the date the program was either created or revised. The remaining comments explain various sections of the code.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The C++ compiler does not require you to include comments in a program. However, it is a good programming practice to do so because they make your code more readable and easier to understand by anyone viewing it. The compiler does not process (execute) the comments; instead, it ignores them when it translates the source code into object code. Comments do not end with a semicolon because they are not statements in C++.

Lines 4 and 5 in Figure 4-21 are directives. C++ programs typically contain at least one directive, and most contain many directives. Line 4 is a `#include` directive and line 5 is a `using` directive. A **`#include` directive** provides a convenient way to merge the source code from one file with the source code in another file, without having to retype the code. The `#include <iostream>` directive, for example, tells the C++ compiler to include the contents of the iostream file in the current program. The iostream file must be included in any program that uses the `cin` or `cout` objects. A `#include` directive is not a C++ statement; therefore, it does not end with a semicolon.

A `using` directive, on the other hand, *is* a statement and must end with a semicolon. A **`using` directive** tells the compiler where (in the computer's internal memory) it can find the definitions of keywords and classes, such as `double` or `string`. The `using namespace std;` directive indicates that the definitions of the standard C++ keywords and classes are located in the `std` (which stands for *standard*) namespace. A namespace is a special area in the computer's internal memory.

In line 7 of the program, `main` is the name of a function and must be typed using lowercase letters. A **function** is a block of code that performs a task. Functions have parentheses following their names, like this: `main()`. Depending on the function, you may or may not need to enter information between the parentheses. Every C++ program must have a `main` function because that is where the execution of a C++ program always begins. Although a C++ program can contain many functions, only one can be the `main` function.

Some functions, like `main`, return a value after completing their assigned task. If a function returns a value, the data type of the value it returns appears to the left of the function name; otherwise, the keyword `void` appears to the left of the name. The `int` in line 7 indicates that the `main` function returns an integer. The entire line of code, `int main()`, is referred to as a **function header** because it marks the beginning of the function.

Following the function header is the code that directs the function on how to perform its assigned task. Examples of such code include statements that declare variables, as well as statements that input, calculate, and output data. The code must be enclosed within a set of braces (`{}`). In Figure 4-21, the `main` function's opening brace appears on line 8, immediately below the function's header, and the closing brace appears on line 29.

Everything between the opening and closing braces in Figure 4-21 is included in the `main` function and is referred to as the **function body**. Notice that you can include a comment (in this case, `//end of main function`) on the same line with a C++ instruction. However, you must be sure to enter the comment *after* the instruction because any text appearing after the two forward slashes (`//`) on a line is interpreted as a comment.

The `return 0;` statement on Line 28 in Figure 4-21 returns the number 0 to the operating system to indicate that the program ended normally. (As mentioned earlier, the `main` function returns an integer.) Figure 4-22 shows a sample run of the program, which appears in a Command Prompt window.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

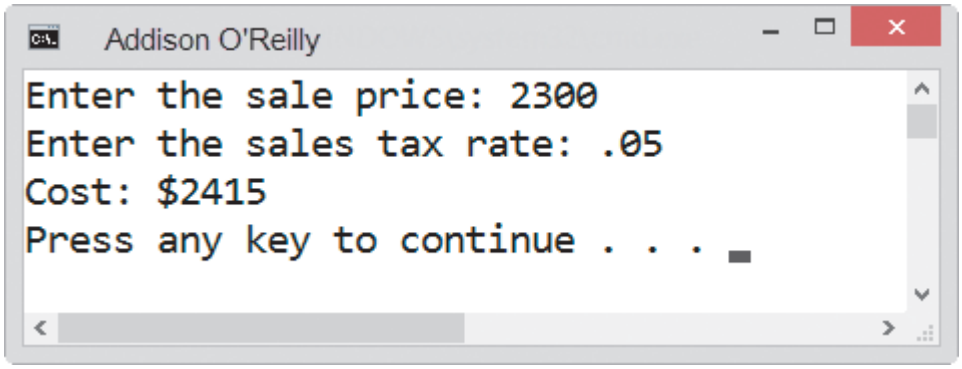


Figure 4-22 Command Prompt window

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 4-3

1. Entering the `cout < "Hello";` statement in a program is an example of a(n) _____ error.
2. The .cpp file that contains your C++ instructions is called the _____ file.
3. In a C++ program, the body of a function is enclosed in _____ .

The answers to the labs are contained in the Answers.pdf file.

LAB 4-1 Stop and Analyze

Study the four examples shown in Figure 4-23 and then answer the questions.

Example 1
```
int quantity = 10;
double itemCost = 5.35;
double amountDue = 0.0;
amountDue = quantity * itemCost + 5;
```
Example 2
```
int total = 30;
total = total / 3;
```
Example 3
```
double store1Sales = 5678.43;
double store2Sales = 8325.72;
double avgSales = 0.0;
avgSales = store1Sales + store2Sales / 2;
```
Example 4
```
int midterm = 74;
int final = 93;
double average = 0.0;
average = (midterm + final) / 2;
```

Figure 4-23 Examples for Lab 4-1

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

QUESTIONS

1. Explain how the computer evaluates the assignment statement in Example 1. What value will be assigned to the `amountDue` variable? Is the value correct? If not, how can you fix the statement so it evaluates correctly?
2. Explain how the computer evaluates the assignment statement in Example 2. What value will be assigned to the `total` variable? Is the value correct? If not, how can you fix the statement so it evaluates correctly?
3. Explain how the computer evaluates the assignment statement in Example 3. What value will be assigned to the `avgSales` variable? Is the value correct? If not, how can you fix the statement so it evaluates correctly?
4. Explain how the computer evaluates the assignment statement in Example 4. What value will be assigned to the `average` variable? Is the value correct? If not, how can you fix the statement so it evaluates correctly?

LAB 4-2 Plan and Create

In Chapter 3's Lab 3-2, you planned, created, and desk-checked an algorithm that displays a 10% commission on a sales amount. Figure 4-24 shows the algorithm, along with the input and output items and their corresponding C++ statements. It also includes the desk-check table you completed in Lab 3-2.

IPO chart information

Input
commission rate (10%)
sales amount

Processing
none

Output
commission

Algorithm
1. enter the sales amount
2. calculate the commission by multiplying the sales amount by the commission rate
3. display the commission

C++ instructions

```
const double COMM_RATE = 0.1;
double sales = 0.0;

double commission = 0.0;
```

commission rate	sales amount	commission
~~.1~~	~~1328.50~~	~~132.85~~
.1	267.90	26.79

Figure 4-24 IPO chart information, C++ instructions, and desk-check table from Lab 3-2

The first instruction in the algorithm is to enter the sales amount. You will code this instruction using both a `cout` statement and a `cin` statement. The `cout` statement will prompt the user for the sales amount, and the `cin` statement will store the user's response in the `sales` variable.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The next instruction in the algorithm calculates the commission by multiplying the sales amount by the commission rate. You will code this instruction using an assignment statement that multiplies the contents of the `sales` variable by the contents of the COMM_RATE named constant, assigning the result to the `commission` variable.

The last instruction in the algorithm displays the commission on the computer screen. You will code this instruction using a `cout` statement. Figure 4-25 shows the program statements entered in the C++ instructions column.

IPO chart information	C++ instructions
Input commission rate (10%) sales amount	`const double COMM_RATE = 0.1;` `double sales = 0.0;`
Processing none	
Output commission	`double commission = 0.0;`

Algorithm
1. enter the sales amount
2. calculate the commission by multiplying the sales amount by the commission rate
3. display the commission
	`cout << "Sales amount: ";`
	`cin >> sales;`
	`commission = sales * COMM_RATE;`
	
	`cout << "Commission: $"`
	`<< commission << endl;`

Figure 4-25 Completed IPO chart and C++ instructions for Lab 4-2

After coding the algorithm, you then need to desk-check the program. You begin by placing the names of the declared variables and named constants in a new desk-check table, along with their values. You then desk-check the remaining C++ instructions in order, recording in the desk-check table any changes made to the variables. Figure 4-26 shows the completed desk-check table for the program. The results agree with the algorithm's desk-check table shown earlier in Figure 4-24.

COMM_RATE	sales	commission	
~~0.1~~	~~0.0~~	~~0.0~~	first desk check
	~~1328.50~~	~~132.85~~	
0.1	~~0.0~~	~~0.0~~	second desk check
	267.90	26.79	

Figure 4-26 Program's desk-check table

The final step in the problem-solving process is to evaluate and modify (if necessary) the program. Recall that you evaluate a program by entering its instructions into the computer and then using the computer to run (execute) it. While the program is running, you enter the same sample data used when desk-checking the program.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

DIRECTIONS

Ch04-Installation *developmentTool*

1. Determine whether your C++ development tool has been installed on your computer. If it hasn't, then you will need to install it before you can enter and run the program shown in Figure 4-27. If you need help with the installation, watch the video that corresponds to your development tool. The videos are named Ch04-Installation *developmentTool*, where *developmentTool* is the name of the C++ development tool covered in the video. If your development tool is not covered in any of these videos, you will need to contact your instructor or technical support person for the appropriate installation instructions.
2. The Cpp8\Chap04 folder contains several files named Ch04-Lab4-2 *developmentTool*.pdf. Each file corresponds to a specific C++ development tool and provides step-by-step directions for creating, entering, saving, and running the program shown in Figure 4-27. If the Cpp8\Chap04 folder contains a PDF file for your C++ development tool, open the PDF file and then follow the directions listed in the file. (You can use Adobe Reader to open a PDF file. If you don't have Adobe Reader on your computer system, you can download it for free at *www.adobe.com*.)
3. If the Cpp8\Chap04 folder does *not* contain a PDF file for your C++ development tool, contact your instructor or technical support person for the appropriate instructions. Follow the instructions you are given for starting and using your C++ development tool. Enter the instructions shown in Figure 4-27 in a source file named Lab4-2.cpp. (Do not enter the line numbers.) Save the file in the Cpp8\Chap04 folder. Now follow the appropriate instructions for running the Lab4-2.cpp file. Run the program twice, using the sample data values of 1328.50 and 267.90 for the sales amount. If necessary, correct any bugs (errors) in the program. **Note**: If your C++ development tool does not automatically pause program execution and display the *Press any key to continue message* when the program ends, enter the `system("pause");` statement above the `return 0;` statement in the program.

```
 1 //Lab4-2.cpp - displays a salesperson's commission
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 using namespace std;
 6
 7 int main()
 8 {
 9    //declare named constant and variables
10    const double COMM_RATE = 0.1;
11    double sales           = 0.0;
12    double commission      = 0.0;
13
14    //enter input item
15    cout << "Sales amount: ";
16    cin >> sales;
17
18    //calculate and display the commission
19    commission = sales * COMM_RATE;
20    cout << "Commission: $"
21         <<    commission << endl;
22
23    return 0;
24 } //end of main function
```

Figure 4-27 Commission program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

LAB 4-3 Modify

In this lab, you will modify the program from Lab 4-2 to allow the user to enter the commission rate (in decimal form).

The Cpp8\Chap04 folder contains several files named Ch04-Lab4-3 *developmentTool*.pdf. Each file corresponds to a specific C++ development tool and provides step-by-step directions for completing Lab 4-3. If the Cpp8\Chap04 folder contains a PDF file for your C++ development tool, open the PDF file, and then follow the directions listed in the file. (You can use Adobe Reader to open a PDF file.)

If the Cpp8\Chap04 folder does *not* contain a PDF file for your C++ development tool, start your development tool and then copy the program instructions from Lab 4-2 into a new source file named Lab4-3.cpp. (You may need to contact your instructor or technical support person for how to perform this task.) Be sure to change Lab4-2.cpp in the first comment to Lab4-3.cpp. Test the program twice. For the first test, use 1328.50 and .1 as the sales amount and commission rate, respectively. For the second test, use 267.90 and .15. (Don't be concerned that the $40.185 has three decimal places. You will learn how to format numbers in Chapter 5.)

LAB 4-4 What's Missing?

The program in this lab should calculate and display the volume of a cylinder, given the cylinder's radius (r) and height (h), and using 3.14 as the value of pi. The formula for calculating the volume is $\pi r^2 h$.

The Cpp8\Chap04 folder contains several files named Ch04-Lab4-4 *developmentTool*.pdf. Each file corresponds to a specific C++ development tool and provides step-by-step directions for completing Lab 4-4. If the Cpp8\Chap04 folder contains a PDF file for your C++ development tool, open the PDF file and then follow the directions listed in the file. (You can use Adobe Reader to open a PDF file.) The directions will guide you in putting the C++ instructions in the proper order and then locating the three missing instructions.

If the Cpp8\Chap04 folder does *not* contain a PDF file for your C++ development tool, start your development tool and then open the Lab4-4.cpp file. Put the C++ instructions in the proper order and then determine the one or more missing instructions. (Hint: Three instructions are missing from the program.) Test the program twice. For the first test, use 10 and 5 as the height and radius, respectively. For the second test, use 15 and 20.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

LAB 4-5 Desk-Check

Desk-check the seven lines of code shown in Figure 4-28.

```
int num 75;
int result1 = 0;
int result2 = 0;
double result3 = 0.0;
result1 = num % 2;
result2 = num / 2;
result3 = num / 2.0;
```

Figure 4-28 Code for Lab 4-5

LAB 4-6 Debug

The Cpp8\Chap04 folder contains several files named Ch04-Lab4-6 *developmentTool*.pdf. Each file corresponds to a specific C++ development tool and provides step-by-step directions for completing Lab 4-6. If the Cpp8\Chap04 folder contains a PDF file for your C++ development tool, open the PDF file and then follow the directions listed in the file. (You can use Adobe Reader to open a PDF file.)

If the Cpp8\Chap04 folder does *not* contain a PDF file for your C++ development tool, start your C++ development tool and then open the Lab4-6.cpp file. The program should calculate and display the area of a triangle, but it is not working correctly. Run and then debug the program.

Chapter Summary

The fourth step in the problem-solving process is to code the algorithm. You begin by declaring a memory location for each unique input, processing, and output item listed in the IPO chart. You then translate each instruction in the algorithm into one or more C++ statements.

In C++, standard input and output operations are performed using stream objects. The standard input stream object is called `cin`. The standard output stream object is called `cout`.

You use `cin` along with the extraction operator (`>>`) to get either numeric or character input from the computer keyboard. You use `cout` along with the insertion operator (`<<`) to display information on the computer screen.

The extraction operator stops removing characters from the `cin` object when it encounters a white-space character.

The `endl` stream manipulator advances the cursor to the next line on the computer screen.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

A program should display (on the computer screen) a separate and meaningful prompt for each item of data the user should enter.

You direct the computer to perform a calculation by writing an arithmetic expression that contains one or more arithmetic operators.

Each arithmetic operator is associated with a precedence number, which controls the order in which the operation is performed in an expression. When an arithmetic expression contains more than one operator having the same priority, those operators are evaluated from left to right. You can use parentheses to override the normal order of precedence.

When an arithmetic operation involves two values having different data types, the computer implicitly promotes the value with the lower-ranking data type to the higher-ranking data type. The value returns to its original data type upon completion of the arithmetic operation.

The quotient obtained by dividing one integer by another integer is always an integer in C++.

You can use the `static_cast` operator to explicitly convert data from one data type to another.

You can use an assignment statement to assign a value to a variable during runtime.

An assignment statement tells the computer to evaluate the expression that appears on the right side of the assignment operator (=) and then store the result in the variable whose name appears on the left side of the assignment operator.

C++ provides arithmetic assignment operators (also called compound assignment operators or shortcut operators) for abbreviating an assignment statement that has the following format, in which *variableName* is the name of the same variable: *variableName = variableName arithmeticOperator value;*. The abbreviated statement will have the following format: *variableName arithmeticAssignmentOperator value;*.

The fifth step in the problem-solving process is to desk-check the program. You should use the same sample data used to desk-check the algorithm.

The sixth (and final) step in the problem-solving process is to evaluate and modify (if necessary) the program.

The errors in a program are called bugs and typically fall into one of two categories: syntax errors or logic errors.

In order for you to enter your C++ instructions into the computer and then run the program, you need to have access to a text editor and a C++ compiler.

The C++ instructions entered in a program are called source code and are saved in a source file, which has a .cpp filename extension.

The compiler translates source code into machine code, also called object code.

The linker produces an executable file that contains all the machine code necessary to run a C++ program. The executable file has an .exe filename extension.

Programmers use comments to document a program internally. Doing this makes the program easier to understand by anyone viewing it. Comments are not statements and are ignored by the compiler. Comments begin with two forward slashes (//).

The `#include <iostream>` directive tells the computer to include the contents of the iostream file in the current program.

The `using namespace std;` directive indicates that the definitions of standard C++ keywords and classes are located in the `std` namespace. A namespace is a special area in the computer's internal memory.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The execution of a C++ program begins with the `main` function. Therefore, every C++ program must have one (and only one) `main` function.

The first line in a function is called the function header. Following the function header is the function body, which must be enclosed in braces.

Key Terms

`#include` directive—an instruction that tells the computer to merge the source code from one file with the source code from another file

`%`—modulus (remainder) operator; divides two integers and returns the remainder as an integer

`<<`—the insertion operator in C++

`>>`—the extraction operator in C++

Arithmetic assignment operators—operators composed of an arithmetic operator followed immediately by the assignment operator; used to abbreviate an assignment statement that follows a specific format; also called compound assignment operators or shortcut operators

Assignment operator—the = symbol in an assignment statement

Assignment statement—used to assign a value to a variable during runtime

Bugs—the errors in a program

`cin`—the standard input stream object in C++; tells the computer to pause program execution while the user enters one or more characters or numbers at the keyboard; temporarily stores the characters or numbers entered at the keyboard

Comment—a message used to document a program internally; begins with two forward slashes (`//`) in C++

`cout`—the standard output stream object in C++; used with the insertion operator to display information on the computer screen

Debugging—the process of locating and correcting any errors in a program

`endl`—a stream manipulator that can be used to advance the cursor to the next line on the computer screen

Executable file—a file that contains all of the machine code necessary to run a program; executable files have an .exe filename extension

Explicit type conversion—the explicit conversion of data from one data type to another; usually performed with the `static_cast` operator; also called a type cast

Extraction operator—two greater-than signs (`>>`); extracts (removes) characters from the `cin` object and sends them "in" to the computer's internal memory

Function—a block of code that performs a task

Function body—the code contained between a function's opening and closing braces

Function header—the first line in a function; marks the beginning of the function

IDE—the acronym for Integrated Development Environment

Insertion operator—two less-than signs (`<<`); used with the `cout` object to send information "out" to the user via the computer screen

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Integrated Development Environment—a system that includes both an editor and a compiler

Linker—a program that combines the code contained in a C++ program's object file with other machine code necessary to run the C++ program

Logic error—an error (bug) that occurs when you neglect to enter a program instruction or enter the instructions in the wrong order; also occurs as a result of calculation statements that are correct syntactically but incorrect mathematically

Modulus operator—the percent sign (%); divides two integers and returns the remainder as an integer

Object code—another name for machine code

Object file—a file that contains the object code associated with a program; automatically generated by the compiler

Prompt—a message (displayed on the computer screen) indicating the type of data the user should enter at the keyboard

Source code—the program instructions you enter using an editor; the instructions are saved in a source file

Source file—a file that contains a program's source code; source files have a .cpp filename extension

`static_cast` operator—explicitly converts (or type casts) data from one data type to another

Stream—a sequence of characters

Stream manipulator—allows a C++ program to manipulate (or manage) the characters in either the input or output stream

Stream objects—objects used to perform standard input and output operations in C++

Syntax error—an error (bug) that occurs when a program instruction violates a programming language's syntax

Type cast—another term for an explicit type conversion

`using` directive—an instruction that tells the computer where it can find the definitions of keywords and classes

White-space character—a newline character, a tab character, or a blank (space) character

Review Questions

1. Which of the following prompts the user to enter an hourly pay rate?

   a. `cout >> "Pay rate per hour? ";`

   b. `cout << "Pay rate per hour? ";`

   c. `cout >> "Pay rate per hour? ":`

   d. `cout << "Pay rate per hour? ":`

2. Which of the following sends keyboard input to a variable named `payRate`?

   a. `cin >> payRate;`

   b. `cin << payRate;`

   c. `cin <> payRate;`

   d. `cin > payRate;`

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

3. If the `payRate` variable has the `double` data type, which of the following statements will require an explicit type conversion to evaluate correctly?

   a. `payRate = 25;`
   b. `payRate = payRate * 1.05;`
   c. `payRate /= 2;`
   d. none of the above

4. The `num1` and `num2` variables have the `int` data type and contain the numbers 13 and 5, respectively. The `answer` variable has the `double` data type. Which of the following statements will require an explicit type conversion to evaluate correctly?

   a. `answer = num1 / 4.0;`
   b. `answer = num1 + num1 / num2;`
   c. `answer = num1 - num2;`
   d. none of the above

5. Which of the following assigns the letter T to a `char` variable named `insured`?

   a. `insured = 'T';`
   b. `insured = "T":`
   c. `insured = "T";`
   d. `insured = 'T':`

6. Which of the following explicitly converts the contents of an `int` variable named `quantity` to the `double` data type?

   a. `castToDouble(quantity)`
   b. `explicit_cast <double>(quantity)`
   c. `static_cast <double>(quantity)`
   d. `type_cast <double>(quantity)`

7. Which of the following statements advances the cursor to the next line on the computer screen?

   a. `cout << endl;`
   b. `cout << endline;`
   c. `cout << newline;`
   d. none of the above

8. Which of the following tells the compiler to merge the code contained in the iostream file with the current file's code?

   a. `#include iostream;`
   b. `#include <iostream>`
   c. `#include <iostream>;`
   d. `#include (iostream)`

9. Which of the following is equivalent to the `rate = rate / 100;` statement?

   a. `rate =/ 100;`
   b. `rate /= 100:`
   c. `rate / = 100;`
   d. `rate /= 100;`

10. Which of the following is a valid comment in C++?

    a. `**This is a comment`
    b. `@/This is a comment`
    c. `/This is a comment`
    d. none of the above

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Exercises

Pencil and Paper

TRY THIS

1. Complete the C++ instructions column in Figure 4-29. (The answers to TRY THIS Exercises are located at the end of the chapter.)

IPO chart information	C++ instructions

Input
food
rent
utilities
car payment
	`double food = 0.0;`
	`double rent = 0.0;`
	`double utilities = 0.0;`
	`double car = 0.0;`

Processing none	
Output total expenses	`double totalExpenses = 0.0;`
Algorithm: 1. enter food, rent, utilities, and car payment 2. calculate total expenses by adding together food, rent, utilities, and car payment 3. display total expenses	

Figure 4-29

TRY THIS

2. Complete the C++ instructions column in Figure 4-30. (The answers to TRY THIS Exercises are located at the end of the chapter.)

IPO chart information	C++ instructions
Input	
latex price	`double latexPrice = 0.0;`
Mylar price	`double mylarPrice = 0.0;`
latex purchased	`int latexPurchased = 0;`
Mylar purchased	`int mylarPurchased = 0;`
sales tax rate (6%)	`const double TAX_RATE = .06;`
Processing	
total latex cost	`double totalLatexCost = 0.0;`
total Mylar cost	`double totalMylarCost = 0.0;`
subtotal	`double subtotal = 0.0;`
sales tax	`double salesTax = 0.0;`
Output	
total cost	`double totalCost = 0.0;`
Algorithm: 1. enter latex price, Mylar price, latex purchased, and Mylar purchased	

Figure 4-30 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

2. calculate total latex cost by multiplying latex purchased by latex price
3. calculate total Mylar cost by multiplying Mylar purchased by Mylar price
4. calculate subtotal by adding total latex cost to total Mylar cost
5. calculate sales tax by multiplying subtotal by sales tax rate
6. calculate total cost by adding sales tax to subtotal
7. display total cost

Figure 4-30

3. Complete TRY THIS Exercise 1, and then modify the IPO chart information and C++ instructions so that the car payment will always be $253.75. MODIFY THIS

4. Complete the C++ instructions column in Figure 4-31. INTRODUCTORY

IPO chart information	**C++ instructions**
Input	
quantity	`int quantity = 0;`
total cost	`double total = 0.0;`
Processing	
none	
Output	
cost per item	`double itemCost = 0.0;`
Algorithm:	
1. enter the quantity and total cost	`cout << "Quantity: ";` `cin >> quantity;`
	_____
	_____
2. calculate the cost per item by dividing the total cost by the quantity	_____
3. display the cost per item	`cout << "Cost per item: $" <<` `itemCost << endl;`

Figure 4-31

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

INTERMEDIATE

5. Complete the C++ instructions column in Figure 4-32.

IPO chart information	C++ instructions
Input	
quantity sold	`int quantity = 0;`
item cost	`double cost = 0.0;`
item selling price	`double sellPrice = 0.0;`
Processing	
price and cost difference	`double  difference = 0.0;`
Output	
profit	`double profit = 0.0;`
Algorithm:	
1. enter the quantity sold, item cost, and item selling price	_____
2. calculate the price and cost difference by subtracting the item cost from the item selling price	_____
3. calculate the profit by multiplying the price and cost difference by the quantity sold	_____
4. display the profit	`cout << "Profit: $ " << profit << endl;`

Figure 4-32

ADVANCED

6. Complete the C++ instructions column in Figure 4-33.

IPO chart information	C++ instructions
Input	
number of pets	`int pets = 0;`
number of owners	`int owners = 0;`
Processing	
none	
Output	
number of pets per owner	`double petsPerOwner = 0.0;`
Algorithm:	
1. enter number of pets and number of owners	_____
2. calculate number of pets per owner by dividing number of pets by number of owners	_____
3. display number of pets per owner	_____

Figure 4-33

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

7. Correct the errors in the lines of code shown in Figure 4-34. (The code contains eight errors.)

SWAT THE BUGS

```
#include <iostream>;
using namespace std

int main
}
      Int quantity = 0;

      cout >> "Enter the quantity ordered: ";
      cin << quantity;
      cout << "You entered " << quantity << endl

      return 0;
{     //end of main function
```

Figure 4-34

Computer

8. First, complete TRY THIS Exercise 1. Then, if necessary, create a new project named TryThis8 Project, and save it in the Cpp8\Chap04 folder. Enter the C++ instructions from TRY THIS Exercise 1 into a source file named TryThis8.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Test the program using 250.75, 900, 100.87, and 205 as the food, rent, utilities, and car payment, respectively. Then test it using your own set of data. (The answers to TRY THIS Exercises are located at the end of the chapter.)

TRY THIS

9. First, complete TRY THIS Exercise 2. Then, if necessary, create a new project named TryThis9 Project, and save it in the Cpp8\Chap04 folder. Enter the C++ instructions from TRY THIS Exercise 2 into a source file named TryThis9.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Test the program using 1.05, 1.65, 10, and 7 as the latex price, Mylar price, number of latex purchased, and number of Mylar purchased, respectively. (Do not be concerned that the output has three decimal places. You will learn how to format numbers in Chapter 5.) Then test it using your own set of data. (The answers to TRY THIS Exercises are located at the end of the chapter.)

TRY THIS

10. First, complete TRY THIS Exercise 8. If necessary, create a new project named ModifyThis10 Project, and save it in the Cpp8\Chap04 folder. Enter (or copy) the instructions from the TryThis8.cpp file into a new source file named ModifyThis10.cpp. The rent and car payment will always be $750 and $125.75, respectively. Modify the code appropriately. Test the program using 199.74 and 126.45 as the food and utilities, respectively. Then test it using your own set of data.

MODIFY THIS

11. Jacob Weinstein wants a program that displays his savings account balance at the end of the month, given the beginning balance, total deposits, and total withdrawals.

INTRODUCTORY

    a. Using the chart shown earlier in Figure 4-12 as a guide, enter the input, processing, and output items, as well as the algorithm, in the first column.
    b. Desk-check the algorithm twice. For the first desk-check, use 2545.75, 409.43, and 210.65 as the beginning balance, total deposits, and total withdrawals. For the second desk-check, use 1125.33, 23, and 800.94.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

c. Enter the C++ instructions in the second column of the chart, and then desk-check the program using the same data used to desk-check the algorithm.
d. If necessary, create a new project named Introductory11 Project, and save it in the Cpp8\Chap04 folder. Enter your C++ instructions into a source file named Introductory11.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Test the program using the same data used to desk-check the program.

INTRODUCTORY

12. The manager of Fish Haven wants a program that displays the number of gallons of water a rectangular aquarium holds, given the aquarium's length, width, and height measurements in inches. (Hint: There are 231 cubic inches in a gallon.)

a. Using the chart shown earlier in Figure 4-12 as a guide, enter the input, processing, and output items, as well as the algorithm, in the first column.
b. Desk-check the algorithm twice. For the first desk-check, use 20.5, 10.5, and 12.5 as the length, width, and height measurements. For the second desk-check, use 30, 9, and 14.
c. Enter the C++ instructions in the second column of the chart, and then desk-check the program using the same data used to desk-check the algorithm.
d. If necessary, create a new project named Introductory12 Project, and save it in the Cpp8\Chap04 folder. Enter your C++ instructions into a source file named Introductory12.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Test the program using the same data used to desk-check the program.

INTERMEDIATE

13. The manager of Keystone Tile wants an application that displays the area of a rectangular floor, given its measurements in feet. It should also display the total cost of tiling the floor, given the price per square foot of tile.

a. Using the chart shown earlier in Figure 4-12 as a guide, enter the input, processing, and output items, as well as the algorithm, in the first column.
b. Desk-check the algorithm twice, using your own sets of data.
c. Enter the C++ instructions in the second column of the chart, and then desk-check the program using the same data used to desk-check the algorithm.
d. If necessary, create a new project named Intermediate13 Project, and save it in the Cpp8\Chap04 folder. Enter your C++ instructions into a source file named Intermediate13.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Test the program using the same data used to desk-check the program.

INTERMEDIATE

14. Silvia's Pizzeria sells four different sizes of pizzas: small, medium, large, and family. The manager of the pizzeria wants a program that displays the total number of pizzas sold, as well as the percentage of the total number contributed by each different size.

a. Using the chart shown earlier in Figure 4-12 as a guide, enter the input, processing, and output items, as well as the algorithm, in the first column.
b. Desk-check the algorithm twice. For the first desk-check, use 25, 50, 50, and 75 as the numbers of small, medium, large, and family pizzas. For the second desk-check, use 30, 25, 85, and 73. Record the percentages with one decimal place.
c. Enter the C++ instructions in the second column of the chart, and then desk-check the program using the same data used to desk-check the algorithm.
d. If necessary, create a new project named Intermediate14 Project, and save it in the Cpp8\Chap04 folder. Enter your C++ instructions into a source file named Intermediate14.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Test the program using the same data used to desk-check the program.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

15. A local zoo offers three different memberships: an individual membership for $99 per year, a dual membership for $175 per year, and a family membership for $225 per year. The membership director wants a program that displays the total membership revenue for the year, as well as the amount of the total revenue contributed by each membership type. ADVANCED

    a. Using the chart shown earlier in Figure 4-12 as a guide, enter the input, processing, and output items, as well as the algorithm, in the first column.
    b. Desk-check the algorithm twice. For the first desk-check, use 50, 75, and 150 as the numbers of individual, dual, and family memberships sold during the year. For the second desk-check, use 35, 150, and 265. Record the percentages with one decimal place.
    c. Enter the C++ instructions in the second column of the chart, and then desk-check the program using the same data used to desk-check the algorithm.
    d. If necessary, create a new project named Advanced15 Project, and save it in the Cpp8\Chap04 folder. Enter your C++ instructions into a source file named Advanced15.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Test the program using the same data used to desk-check the program.

16. In this exercise, you explore the use of integers in monetary calculations. ADVANCED

    a. Follow the instructions for starting C++ and opening the Advanced16.cpp file. Run the program. Enter 256.7 and 223.3 as the sales for Store 1 and Store 2, respectively. The total that appears on the computer screen (504.00) is incorrect because it is not the result of adding together the numbers 269.54 and 234.47. Press any key to stop the program.
    b. Review the code contained in the Advanced16.cpp file. The `#include <iomanip>` directive tells the C++ compiler to include the contents of the iomanip file in the current program. The file contains the definition of the `setprecision` stream manipulator, which appears in the `cout << fixed << setprecision(2) << endl;` statement. The `fixed` stream manipulator, which is defined in the iostream file, forces a real number to display a specific number of decimal places, as specified by the `setprecision` stream manipulator. In this program, the output values will display with two decimal places. You will learn about the directive and both stream manipulators in Chapter 5.
    c. Why does the total appear as 504.00 rather than 504.01? Hint: Change the `cout << fixed << setprecision(2);` statement to a comment, and then save and run the program. Enter 256.7 and 223.3 as the sales for Store 1 and Store 2, respectively. Study the output, and then stop the program. Change the comment back to a statement.
    d. Use the seven comments that appear below the `main` function to modify the program's code. Why do you need to add .5 to the expressions that calculate the increased sales for both stores?
    e. Save, run, and test the program to verify that it is working correctly, and then stop the program.

17. Follow the instructions for starting C++ and opening the SwatTheBugs17.cpp file. The program declares and initializes a `double` variable. It then adds 1.5 to the variable before displaying the variable's value. Run the program. (If you are asked whether you want to run the last successful build, click the No button.) Debug the program. SWAT THE BUGS

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Answers to TRY THIS Exercises

Pencil and Paper

1. See Figure 4-35.

Algorithm:

1. enter food, rent, utilities, and car payment	`cout << "Food: ";`
	`cin >> food;`
	`cout << "Rent: ";`
	`cin >> rent;`
	`cout << "Utilities: ";`
	`cin >> utilities;`
	`cout << "Car payment: ";`
	`cin >> car;`

2. calculate total expenses by adding together food, rent, utilities, and car payment	`totalExpenses = food + rent + utilities + car;`
3. display total expenses	`cout << "Total expenses: $" << totalExpenses << endl;`

Figure 4-35

2. See Figure 4-36.

Algorithm:

1. enter latex price, Mylar price, latex purchased, and Mylar purchased	`cout << "Latex price? ";`
	`cin >> latexPrice;`
	`cout << "Mylar price? ";`
	`cin >> mylarPrice;`
	`cout << "Latex purchased? ";`
	`cin >> latexPurchased;`
	`cout << "Mylar purchased? ";`
	`cin >> mylarPurchased;`

2. calculate total latex cost by multiplying latex purchased by latex price	`totalLatexCost = latexPurchased * latexPrice;`
3. calculate total Mylar cost by multiplying Mylar purchased by Mylar price	`totalMylarCost = mylarPurchased * mylarPrice;`
4. calculate subtotal by adding total latex cost to total Mylar cost	`subtotal = totalLatexCost + totalMylarCost;`
5. calculate sales tax by multiplying subtotal by sales tax rate	`salesTax = subtotal * TAX_RATE;`
6. calculate total cost by adding sales tax to subtotal	`totalCost = subtotal + salesTax;`
7. display total cost	`cout << "Total cost: $" << totalCost << endl;`

Figure 4-36

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Computer

8. See Figure 4-37.

```
 1 //TryThis8.cpp - displays total expenses
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 using namespace std;
 6
 7 int main()
 8 {
 9     double food      = 0.0;
10     double rent      = 0.0;
11     double utilities = 0.0;
12     double car       = 0.0;
13     double totalExpenses = 0.0;
14
15     //enter input
16     cout << "Food: ";
17     cin >> food;
18     cout << "Rent: ";
19     cin >> rent;
20     cout << "Utilities: ";
21     cin >> utilities;
22     cout << "Car payment: ";
23     cin >> car;
24
25     //calculate and display output
26     totalExpenses = food + rent + utilities + car;
27     cout << "Total expenses: $" << totalExpenses << endl;
28
29     return 0;
30 } //end of main function
```

Figure 4-37

9. See Figure 4-38.

```
 1 //TryThis9.cpp - displays total cost
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 using namespace std;
 6
 7 int main()
 8 {
 9     const double TAX_RATE = .06;
10     double latexPrice = 0.0;
11     double mylarPrice = 0.0;
12     double latexPurchased = 0.0;
13     double mylarPurchased = 0.0;
14     double totalLatexCost = 0.0;
15     double totalMylarCost = 0.0;
16     double subtoal = 0.0;
17     double salesTax = 0.0;
18     double totalCost = 0.0;
19
20     //enter input
```

Figure 4-38 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
21     cout << "Latex price? ";
22     cin >> latexPrice;
23     cout << "Mylar price? ";
24     cin >> mylarPrice;
25     cout << "Latex purchased? ";
26     cin >> latexPurchased;
27     cout << "Mylar purchased? ";
28     cin >> mylarPurchased;
29
30     //calculate total latex cost, total Mylar cost,
31     //subtotal, sales tax, and total cost
32     totalLatexCost = latexPurchased * latexPrice;
33     totalMylarCost = mylarPurchased * mylarPrice;
34     subtotal = totalLatexCost + totalMylarCost;
35     salesTax = subtotal * TAX_RATE;
36     totalCost = subtotal + salesTax;
37     //display total cost
38     cout << "Total Cost: $" << totalCost << endl;
39
40     return 0;
41 } //end of main function
```

Figure 4-38

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

CHAPTER 5

The Selection Structure

After studying Chapter 5, you should be able to:

- Include the selection structure in pseudocode and in a flowchart
- Code a selection structure using the `if` statement
- Include comparison operators in a selection structure's condition
- Include logical operators in a selection structure's condition
- Temporarily convert a character to either uppercase or lowercase
- Format numeric output

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Making Decisions

As you learned in Chapter 1, all computer programs are written using one or more of three basic control structures: sequence, selection, and repetition. You used the sequence structure in Chapter 4's programs. Recall that during runtime, the instructions in those programs were processed in the order they appeared in the program. Many times, however, a program will need the computer to make a decision before selecting the next instruction to process. A payroll program, for example, typically has the computer determine whether the number of hours an employee worked is greater than 40. The computer would then select either an instruction that computes regular pay only or an instruction that computes regular pay plus overtime pay. Programs that need the computer to make a decision require the use of the selection structure (also called the decision structure).

Ch05-Dr N

The **selection structure** indicates that a decision (based on some condition) needs to be made, followed by an appropriate action derived from that decision. But how does a programmer determine whether a problem's solution requires a selection structure? The answer to this question is found by studying the problem specification. The first problem specification you will examine in this chapter involves an evil scientist named Dr. N. The problem specification and an illustration of the problem are shown in Figure 5-1 along with an appropriate algorithm. The algorithm, which is written in pseudocode, requires only the sequence structure.

Problem specification
Dr. N is sitting in a chair in his lair, facing a control deck and an electronic screen. At times, visitors come to the door located at the rear of the lair. Before pressing the blue button on the control deck to open the door, Dr. N likes to view the visitor on the screen. He can do so by pressing the orange button on the control deck. Write the instructions that direct Dr. N to view the visitor first, and then open the door and say "Welcome".

Algorithm:
1. press the orange button on the control deck to view the visitor on the screen
2. press the blue button on the control deck to open the door
3. say "Welcome"

Figure 5-1 A problem that requires the sequence structure only
Image by Diane Zak; created with Reallusion CrazyTalk Animator

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Now we will make a slight change to the problem specification shown in Figure 5-1. In this case, Dr. N should open the door only if the visitor knows the secret password. The modified problem specification and algorithm are shown in Figure 5-2. The algorithm contains both the sequence and selection structures. The selection structure's condition, which is enclosed in parentheses in the pseudocode, directs Dr. N to make a decision about the visitor's password. More specifically, he needs to determine whether the visitor's password matches the secret password. As you may remember from Chapter 1, the condition in a selection structure must be phrased so that it evaluates to an answer of either true or false. In this case, either the visitor's password matches the secret password (true) or it doesn't match the secret password (false). Only if both passwords are the same does Dr. N need to follow the two indented instructions. The selection structure in Figure 5-2 is referred to as a **single-alternative selection structure** because it requires one or more actions to be taken *only* when its condition evaluates to true. Other examples of single-alternative selection structures include "if it's raining, take an umbrella" and "if you are driving your car at night, turn on your car's headlights".

Problem specification
Dr. N is sitting in a chair in his lair, facing a control deck and an electronic screen. At times, visitors come to the door located at the rear of the lair. Before pressing the blue button on the control deck to open the door, Dr. N likes to view the visitor on the screen. He can do so by pressing the orange button on the control deck. Write the instructions that direct Dr. N to view the visitor first and then ask the visitor for the password. He should open the door and say "Welcome" only if the visitor knows the secret password.

Algorithm:

```
1. press the orange button on the control deck to view the visitor on the screen
2. ask the visitor for the password

                    condition
3. if (the visitor's password matches the secret password)
       press the blue button on the control deck to open the door
       say "Welcome"                                          followed only when
   end if                                                     the condition is true
```

Figure 5-2 A problem that requires the sequence structure and a single-alternative selection structure

Figure 5-3 shows a modified version of the previous problem specification. In this version, Dr. N will say "Sorry, you are wrong" and then destroy the visitor if the passwords do not match. Also shown in Figure 5-3 are two possible algorithms; both produce the same result. The condition in Algorithm 1's selection structure determines whether the visitor's password is *correct*, whereas the condition in Algorithm 2's selection structure determines whether it is incorrect.

In pseudocode, programmers use the words *if* and *end if* to denote the beginning and end, respectively, of a selection structure.

Unlike the selection structure in Figure 5-2, which provides instructions for Dr. N to follow *only* when the selection structure's condition is true, the selection structures in Figure 5-3 require Dr. N to perform one set of instructions when the condition is true but a different set of instructions when the condition is false. The instructions to follow when the condition evaluates to true are called the **true path**. The true path begins with the instruction following the *if*, and it ends with either the *else* (if there is one) or the *end if*. The instructions to follow when the condition evaluates to false are called the **false path**. The false path begins with the instruction following the *else*, and it ends with the *end if*. For clarity, the instructions in each path should be indented as shown in Figure 5-3. Selection structures that contain instructions in both paths, like the ones in Figure 5-3, are referred to as **dual-alternative selection structures**.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Problem specification
Dr. N is sitting in a chair in his lair, facing a control deck and an electronic screen. At times, visitors come to the door located at the rear of the lair. Before pressing the blue button on the control deck to open the door, Dr. N likes to view the visitor on the screen. He can do so by pressing the orange button on the control deck. Write the instructions that direct Dr. N to view the visitor first and then ask the visitor for the password. He should open the door and say "Welcome" only if the visitor knows the secret password. If the visitor does not know the secret password, Dr. N should say "Sorry, you are wrong" and then destroy the visitor by pressing the big red button on the control deck.

Algorithm 1

```
1. press the orange button on the control deck to view the visitor on the screen
2. ask the visitor for the password
                                   condition—determines
                                   if password is correct
3. if (the visitor's password matches the secret password)
       press the blue button on the control deck to open the door
       say "Welcome"                                          <- true path
   else
       say "Sorry, you are wrong"
       press the big red button on the control deck to destroy the visitor
                                                              <- false path
   end if
```

Algorithm 2

```
1. press the orange button on the control deck to view the visitor on the screen
2. ask the visitor for the password
                                   condition—determines
                                   if password is incorrect
3. if (the visitor's password does not match the secret password)
       say "Sorry, you are wrong"
       press the big red button on the control deck to destroy the visitor
                                                              <- true path
   else
       press the blue button on the control deck to open the door
       say "Welcome"                                          <- false path
   end if
```

Figure 5-3 A problem that requires the sequence structure and a dual-alternative selection structure

Flowcharting a Selection Structure

As you learned in Chapter 2, many programmers use flowcharts (rather than pseudocode) when planning solutions to problems. Figure 5-4 shows a problem specification along with two correct algorithms in flowchart form. The diamond in a flowchart is called the **decision symbol** because it is used to represent the condition (decision) in both the selection and repetition structures. The diamonds in Figure 5-4 represent the conditions in selection structures. Flowchart A contains a single-alternative selection structure because it requires a set of actions to be taken only when its condition evaluates to true. Flowchart B contains a dual-alternative selection structure because it requires two different sets of actions: one to be taken only when its condition evaluates to true, and the other to be taken only when its condition evaluates to false.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Problem specification
Pete's Pizzeria wants a program that displays an employee's weekly gross pay, given the number of hours worked and hourly pay rate. Employees working more than 40 hours are paid an additional one-half of their hourly rate for the hours over 40.

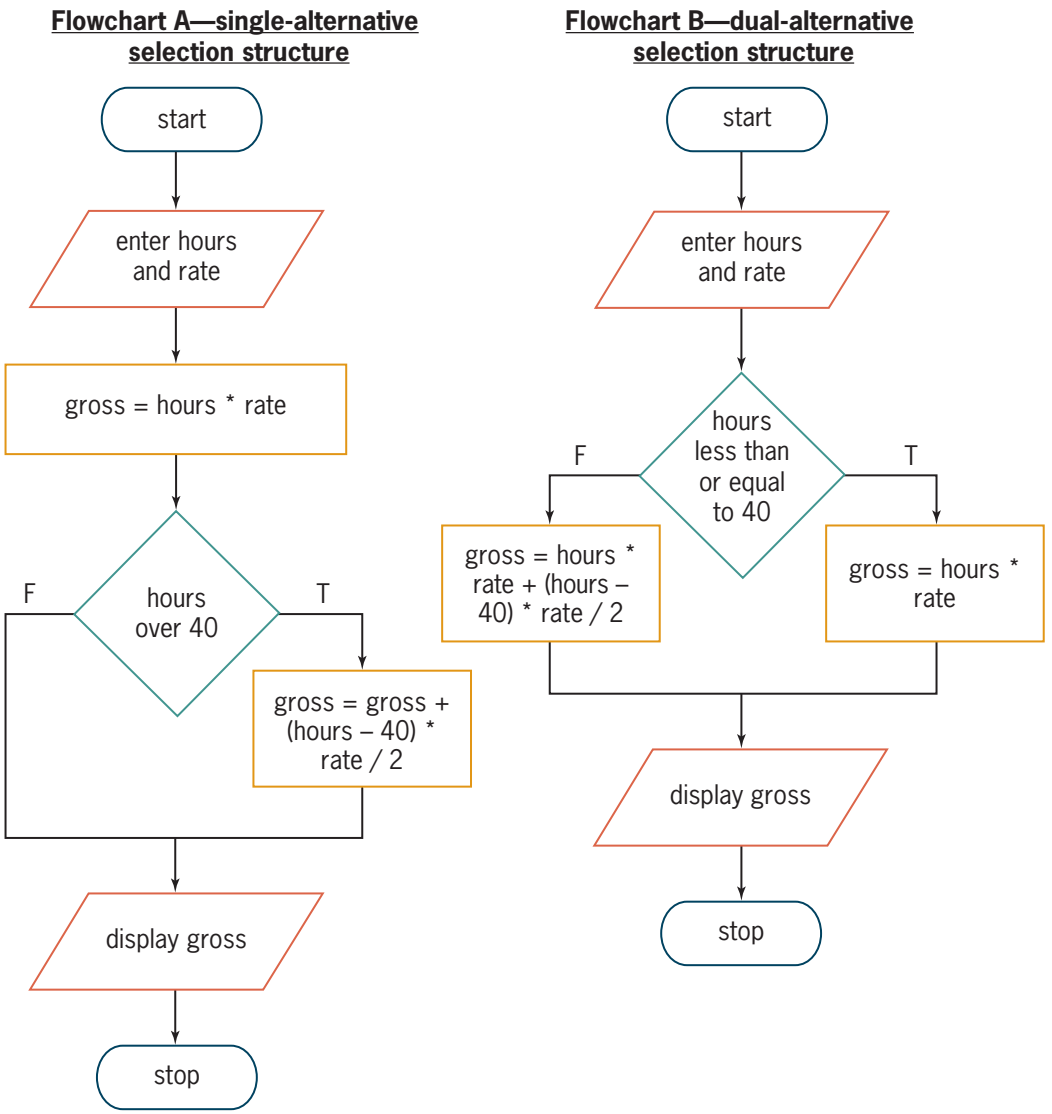


Figure 5-4 Problem specification and two correct algorithms shown in flowchart form

Notice that the conditions in both diamonds evaluate to either true or false only. Also notice that both diamonds have one flowline entering the symbol and two flowlines leaving the symbol. One of the flowlines leading out of a diamond in a flowchart should be marked with a "T" (for true) and the other should be marked with an "F" (for false). The "T" flowline points to the next instruction to be processed when the condition evaluates to true. In Flowchart A, the next instruction calculates the gross pay with overtime; in Flowchart B, it calculates the gross pay without any overtime. The "F" flowline points to the next instruction to be processed when the condition evaluates to false. In Flowchart A, that instruction displays the gross pay; in Flowchart B, it calculates the gross pay with overtime. You can also mark the flowlines leading out of a diamond with a "Y" and an "N" (for yes and no).

For more experience in examining problem specifications, see the Problem Specifications section in the Ch05WantMore.pdf file.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 5-1

1. Most programmers use the words _____ to denote the end of a selection structure in pseudocode.
2. The true path in a selection structure can contain only one instruction.
   a. True
   b. False
3. Which of the following is the decision symbol in a flowchart?
   a. diamond
   b. oval
   c. parallelogram
   d. rectangle
4. A problem specification states that only customers paying with the store's credit card receive a 5% discount on their purchase. Which type of selection structure would be used to calculate the discount?
   a. dual-alternative
   b. single-alternative
5. A problem specification states that customers purchasing at least $100 in product receive free shipping. All other customers must pay a $12 shipping fee. Which type of selection structure would the solution to this problem require?
   a. dual-alternative
   b. single-alternative

Coding Selection Structures in C++

In the C++ programs in this book, you will use the `if` statement to code single-alternative and dual-alternative selection structures. The statement's syntax is shown in Figure 5-5. The square brackets in the syntax indicate that the `else` portion, referred to as the `else` clause, is optional. The boldfaced items in the syntax are required; however, the `else` keyword is necessary only in a dual-alternative selection structure.

Ch05-if Statement

Italicized items in the syntax indicate where the programmer must supply information. In the `if` statement, the programmer must supply the *condition* that the computer needs to evaluate before further processing can occur. The condition must be a Boolean expression, which is an expression that results in a Boolean value (true or false). Besides providing the condition, the programmer must provide the statements to be processed in the true path and (optionally) in the false path. If a path contains more than one statement, the statements must be entered as a **statement block**, which means they must be enclosed in a set of braces (`{}`).

Although not a requirement, using a comment (such as `//end if`) to mark the end of the `if` statement will make your program easier to read and understand. It will also help you keep track of the required `if` and `else` clauses when you nest `if` statements—in other words, when you include one `if` statement inside another `if` statement. You will learn how to nest `if` statements in Chapter 6.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The six examples in Figure 5-5 show various ways of using the `if` statement to code selection structures. Examples 1 and 2 are single-alternative selection structures. The remaining four examples are dual-alternative selection structures. Notice that when a path contains multiple statements, the statements are entered as a statement block by enclosing them in braces. Although not shown in Figure 5-5, you can also include the braces even when a path contains only one statement. By doing this, you won't need to remember to enter the braces when statements are added subsequently to the path. Forgetting to enter the braces is a common error made when typing the `if` statement in a C++ program.

In an `if` statement, you cannot have an `else` clause without a matching `if` clause.

HOW TO Use the `if` Statement

Syntax

```
if (condition)
      one or more statements to be processed when the condition is true
[else
      one or more statements to be processed when the condition is false]
//end if
```

Example 1—one statement in only the true path

```
if (condition)
   one statement
//end if
```

Example 2—multiple statements in only the true path

```
if (condition)
{
   multiple statements enclosed in braces
} //end if
```

Example 3—one statement in each path

```
if (condition)
   one statement
else
   one statement
//end if
```

Example 4—multiple statements in the true path and one statement in the false path

```
if (condition)
{
   multiple statements enclosed in braces
}
else
   one statement
//end if
```

Example 5—one statement in the true path and multiple statements in the false path

```
if (condition)
   one statement
else
{
   multiple statements enclosed in braces
} //end if
```

Figure 5-5 How to use the `if` statement *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Example 6—multiple statements in both paths

```
if (condition)
{
   multiple statements enclosed in braces
}
else
{
   multiple statements enclosed in braces
} //end if
```

Figure 5-5 How to use the `if` statement

As mentioned earlier, an `if` statement's condition must be a Boolean expression, which is an expression that evaluates to either true or false. The expression can contain variables, constants, arithmetic operators, comparison operators, and logical operators. You already know about variables, constants, and arithmetic operators. You will learn about comparison operators and logical operators in this chapter.

Comparison Operators

Figure 5-6 lists the C++ comparison operators (also referred to as relational operators), along with examples of using the operators in an `if` statement's condition. **Comparison operators** are used to compare two values having the same data type. Expressions containing a comparison operator always evaluate to a Boolean value: either true or false. The precedence numbers in Figure 5-6 indicate the order in which the computer performs comparisons in an expression. Comparisons with a precedence number of 1 are performed before comparisons with a precedence number of 2. However, you can use parentheses to override the order of precedence.

HOW TO Use Comparison Operators in an `if` Statement's Condition

Operator	Operation	Precedence number
<	less than	1
<=	less than or equal to	1
>	greater than	1
>=	greater than or equal to	1
==	equal to	2
!=	not equal to	2

Examples (All of the variables have the `int` data type.)

```
if (quantity < 50)
if (age >= 25)
if (onhand == target)
if (quantity != 7500)
```

Note: When making comparisons, keep in mind that equal to (==) is the opposite of not equal to (!=), greater than (>) is the opposite of less than or equal to (<=), and less than (<) is the opposite of greater than or equal to (>=).

Figure 5-6 How to use comparison operators in an `if` statement's condition

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Notice that four of the C++ comparison operators contain two symbols. When entering these operators, be sure you do not enter a space between the symbols, and be sure to enter both symbols in the exact order shown in Figure 5-6.

Because some real numbers (the `float` and `double` data types) cannot be stored precisely in memory, they should never be compared for equality or inequality. In other words, you should not use either the equality operator (==) or the inequality operator (!=) to compare two real numbers. (The exclamation point in the inequality operator stands for *not*.) Instead, you should test that the difference between the real numbers you are comparing is less than some acceptable small value, such as 0.00001. You will learn how to determine whether two real numbers are equal in Computer Exercise 16 at the end of this chapter.

Numbers are compared using their binary equivalents.

When an expression contains more than one comparison operator with the same precedence number, the computer evaluates those comparison operators from left to right in the expression, similar to what is done with arithmetic operators. Comparison operators are evaluated after any arithmetic operators in an expression. Therefore, when processing the expression `7 - 3 + 8 < 9 + 5`, the computer will evaluate the three arithmetic operators before it evaluates the comparison operator. The result of the expression is the Boolean value true, as shown in Figure 5-7.

Original expression	`7 - 3 + 8 < 9 + 5`
The subtraction is performed first	`4 + 8 < 9 + 5`
The first addition is performed next	`12 < 9 + 5`
The second addition is performed next	`12 < 14`
The < comparison is performed last	`true`

Figure 5-7 Evaluation steps for an expression containing arithmetic and comparison operators

It is easy to confuse the equality operator (==) with the assignment operator (=). You use the equality operator to compare two values to determine whether they are equal, as in the condition in the following `if` clause: `if (num == 1)`. You use the assignment operator, on the other hand, to assign a value to a memory location. An example of this is the statement `num = 1;`. In the next two sections, you will view programs that use comparison operators.

Swapping Numeric Values

Ch05-Swapping

Figure 5-8 shows the IPO chart information, C++ code, and a sample run for a program that displays the lowest and highest of two scores entered by the user. The program contains a single-alternative selection structure. The `score1 > score2` condition in the `if` clause compares the contents of the `score1` variable with the contents of the `score2` variable. If the value in the `score1` variable is greater than the value in the `score2` variable, the condition evaluates to true, and the four instructions in the `if` statement's true path swap both values. Swapping the values places the smaller number in the `score1` variable and places the larger number in the `score2` variable. If the condition in the `if` clause evaluates to false, on the other hand, the instructions in the true path are skipped over because the `score1` variable already contains a number that is smaller than (or possibly equal to) the number stored in the `score2` variable.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

IPO chart information

Input
first score
second score

Processing
none

Output
first score (lowest)
second score (highest)

Algorithm:

1. enter the first score and the second score
2. if (the first score is greater than the second score)
   swap the scores so that the first score is
   the lowest score
   end if
3. display the first score and the second score

C++ instructions

```
int score1 = 0;
int score2 = 0;

cout << "First score: ";
cin >> score1;
cout << "Second score: ";
cin >> score2;

if (score1 > score2)
{
    int temp = 0;
    temp = score1;
    score1 = score2;
    score2 = temp;
}  //end if

cout << "Lowest: " <<
score1 << endl;
cout << "Highest: " <<
score2 << endl;
```

swapping instructions in the true path

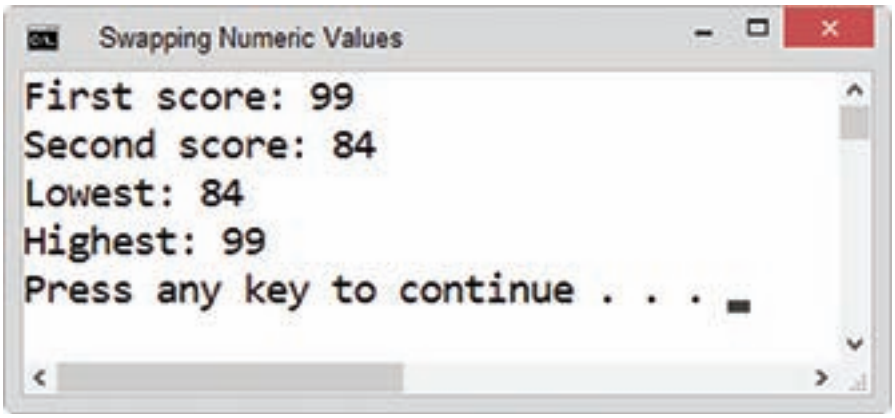


Figure 5-8 Swapping program

Notice that the four instructions in the `if` statement's true path are enclosed in braces. As you learned earlier, when more than one instruction needs to be processed when the `if` statement's condition is true, the C++ syntax requires those instructions to be entered as a statement block.

Study closely the instructions in the true path. The first instruction declares and initializes a variable named `temp`. The `temp` variable must be the same data type as the variables you are swapping. Because the `temp` variable is declared in the `if` statement's true path, it can be used only by the instructions within that path. More specifically, it can be used only by the instructions that follow its declaration statement within the true path. A variable that can be used only within the statement block in which it is defined is referred to as a **local variable**. In this case, the `temp` variable is local to the `if` statement's true path.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

You may be wondering why the `temp` variable was not declared at the beginning of the `main` function, along with the `score1` and `score2` variables. Although there is nothing wrong with declaring the `temp` variable in that location, there is no reason to create the variable until it is needed, which (in this case) is only when a swap is necessary. (You will learn more about local variables in Chapter 9.)

The second instruction in the true path assigns the `score1` variable's value to the `temp` variable. If you do not store that value in the `temp` variable, it will be lost when the computer processes the next statement, `score1 = score2;`, which replaces the contents of the `score1` variable with the contents of the `score2` variable. Finally, the `score2 = temp;` instruction assigns the `temp` variable's value to the `score2` variable; this completes the swap. Figure 5-9 illustrates the concept of swapping, assuming the user enters the numbers 99 and 84 as the first and second scores, respectively. Figure 5-10 shows the corresponding flowchart for the program.

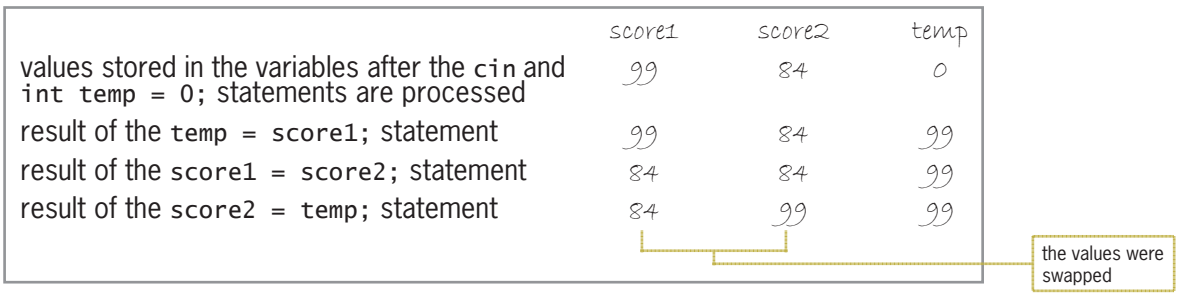


Figure 5-9 Illustration of the swapping concept

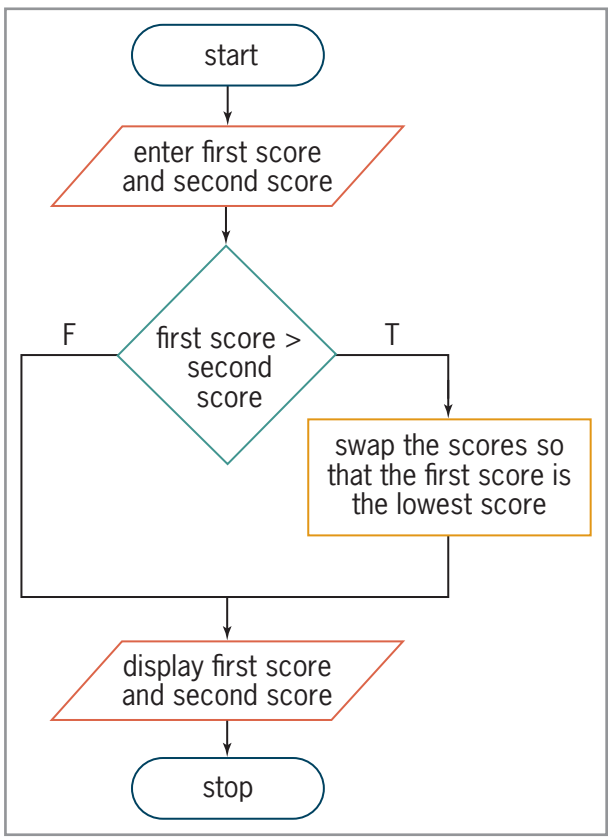


Figure 5-10 Flowchart for the swapping program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Displaying the Area or Circumference

Figure 5-11 shows the IPO chart information, C++ code, and a sample run for a program that displays either a circle's area or its circumference. The program prompts the user to enter the circle's radius, storing the user's response in the `radius` variable. It then prompts the user to enter a number that represents the desired calculation: either 1 (for the area) or 2 (for the circumference). The program stores the user's response in the `choice` variable. If the `choice` variable contains the number 1, the `choice == 1` condition in the dual-alternative selection structure evaluates to true, and the instructions in the true path calculate and display the circle's area. If the `choice` variable contains any number other than the number 1, the `choice == 1` condition evaluates to false, and the instructions in the false path calculate and display the circle's circumference. Notice that the instructions in each path are entered as a statement block. Figure 5-12 shows the corresponding flowchart for the program.

IPO chart information	**C++ instructions**
Input	
pi (3.14)	`const double PI = 3.14;`
radius	`double radius = 0.0;`
choice	`int choice = 0;`
Processing	
none	
Output	
either the area or the circumference	`double answer = 0.0;`
Algorithm:	

1. enter the radius and choice	`cout << "Enter the radius: ";`
	`cin >> radius;`
	`cout << "Enter 1 (area) or 2 (circumference): ";`
	`cin >> choice;`

2. if (the choice is 1)	`if (choice == 1)` `{`
calculate the area by multiplying the radius by itself, and then multiplying the result by pi	`answer = radius * radius * PI;`
display "Area:" and the area	`cout << "Area: " << answer << endl;`
	`}`
else	`else` `{`
calculate the circumference by multiplying the radius by 2, and then multiplying the result by pi	`answer = 2 * radius * PI;`
display "Circumference:" and the circumference	`cout << "Circumference: " << answer << endl;`
end if	`}  //end if`

Figure 5-11 Area or circumference program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

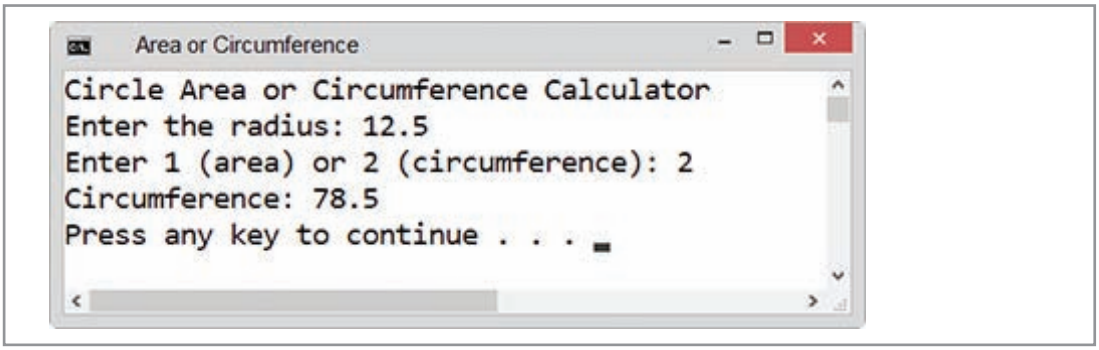


Figure 5-11 Area or circumference program

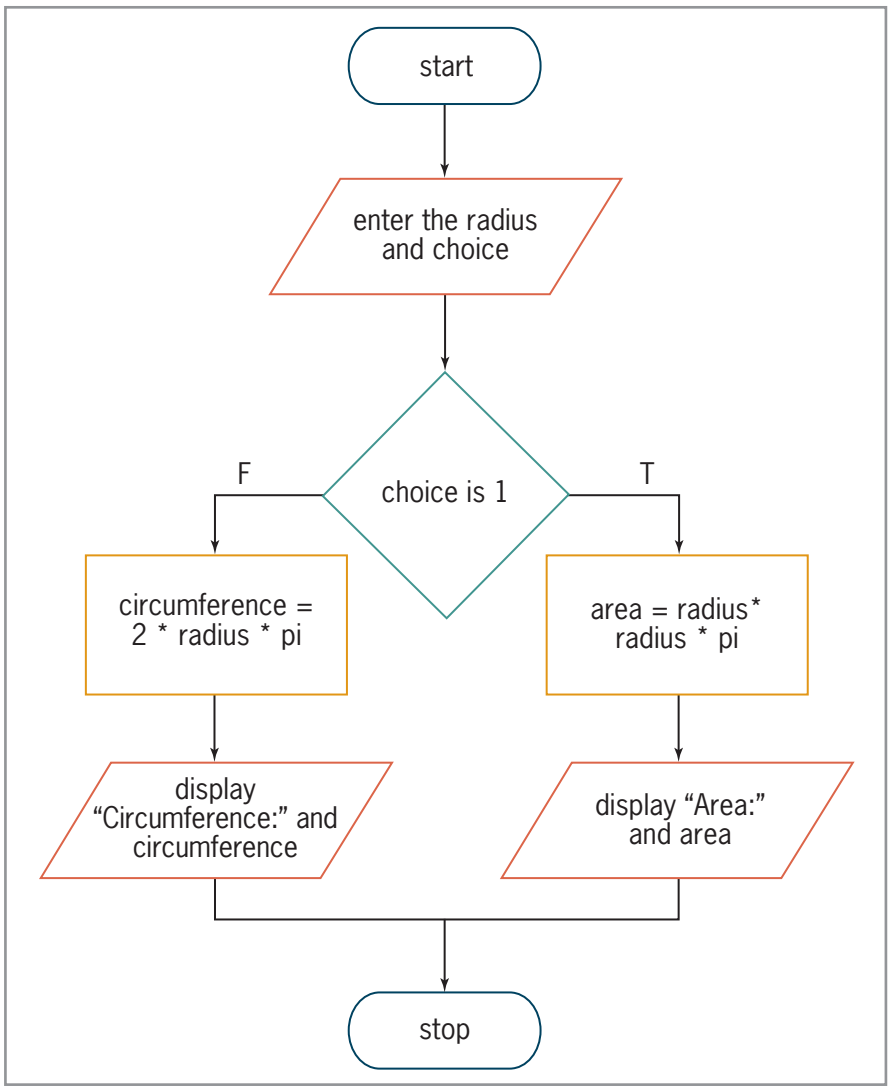


Figure 5-12 Flowchart for the area or circumference program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 5-2

1. You create a statement block by enclosing one or more statements in _____.
   a. braces
   b. parentheses
   c. square brackets
   d. quotation marks

2. Which of the following determines whether an `int` variable named `age` contains the number 21?
   a. `if age = 21`
   b. `if age == 21`
   c. `if (age = 21)`
   d. `if (age == 21)`

3. Which of the following determines whether the value contained in the `price` variable is at least $12.75?
   a. `if (price => 12.75)`
   b. `if (price >= 12.75)`
   c. `if (price <= 12.75)`
   d. `if (price > 12.75)`

4. Which of the following is the inequality operator in C++?
   a. `&=`
   b. `=/`
   c. `!=`
   d. `=!`

5. Which of the following is the opposite of the < operator?
   a. `<=`
   b. `>=`
   c. `=>`
   d. `>`

Logical Operators

An `if` statement's condition can also contain a logical operator. **Logical operators** allow you to combine two or more conditions, referred to as subconditions, into one compound condition. Logical operators are also referred to as **Boolean operators** because the compound condition in which they are contained always evaluates to either true or false only. The two most commonly used logical operators are And and Or. You are already familiar with logical operators because you use them on a daily basis; examples of this are shown in Figure 5-13.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

1. If you finished your homework *and* you studied for tomorrow's exam, watch a movie.
2. If your cell phone rings *and* (it's your spouse calling *or* it's your child calling), answer the phone.
3. If you are driving your car *and* (it's raining *or* it's foggy *or* there is bug splatter on your windshield), turn on your car's wipers.

Figure 5-13 Examples of ways you use logical operators

C++ uses special symbols to represent the And and Or operators in a program. The And operator in C++ is two ampersands (&&); the Or operator is two pipe symbols (||). On most computer keyboards, the pipe symbol (|) is located on the same key as the backslash (\).

When the And (&&) operator is used to create a compound condition, *all* of the subconditions must be true for the compound condition to be true. However, when the Or (||) operator is used, only *one* of the subconditions must be true for the compound condition to be true.

The C++ And and Or operators are listed in Figure 5-14 along with their order of precedence. The figure also includes examples of using the operators in an `if` statement's condition. Notice that the compound condition in each example evaluates to either true or false only. Logical operators are evaluated after any arithmetic or comparison operators in an expression.

HOW TO Use Logical Operators in an `if` Statement's Condition

Operator	Operation	Precedence number
And (&&)	*all* subconditions must be true for the compound condition to evaluate to true	1
Or (||)	only *one* of the subconditions needs to be true for the compound condition to evaluate to true	2

Example 1

```
int population = 0;
cin >> population;
if (population > 2500 && population < 5000)
```

The compound condition evaluates to true when the number stored in the `population` variable is greater than 2500 and, at the same time, less than 5000; otherwise, it evaluates to false.

Example 2

```
int age = 0;
cin >> age;
if (age == 21 || age > 55)
```

The compound condition evaluates to true when the number stored in the `age` variable is either equal to 21 or greater than 55; otherwise, it evaluates to false.

Figure 5-14 How to use logical operators in an `if` statement's condition *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Example 3

```
int quantity = 0;
double price = 0.0;
cin >> quantity;
cin >> price;
if (quantity < 100 && price <= 10.35)
```

The compound condition evaluates to true when the number stored in the `quantity` variable is less than 100 and, at the same time, the number stored in the `price` variable is less than or equal to 10.35; otherwise, it evaluates to false.

Example 4

```
int quantity = 0;
double price = 0.0;
cin >> quantity;
cin >> price;
if (quantity > 0 && quantity < 100 || price > 34.55)
```

The compound condition evaluates to true when either (or both) of the following is true: the number stored in the `quantity` variable is between 0 and 100 or the number stored in the `price` variable is greater than 34.55; otherwise, it evaluates to false. (The `&&` operator is evaluated before the `||` operator because it has a higher precedence.)

Figure 5-14 How to use logical operators in an `if` statement's condition

The tables shown in Figure 5-15, called **truth tables**, summarize how the computer evaluates the logical operators in an expression.

Truth table for the And (&&) operator

subcondition1	subcondition2	subcondition1 && subcondition2
true	true	true
true	false	false
false	true (not evaluated)	false
false	false (not evaluated)	false

evaluates to true only when both subconditions are true

Truth table for the Or (||) operator

subcondition1	subcondition2	subcondition1 || subcondition2
true	true (not evaluated)	true
true	false (not evaluated)	true
false	true	true
false	false	false

evaluates to false only when both subconditions are false

Figure 5-15 Truth tables for the logical operators

Notice that when the computer evaluates the "`subcondition1 && subcondition2`" expression, it does not evaluate subcondition2 when subcondition1 is false. Because both subconditions combined with the And operator need to be true for the compound condition to be true, there is no need to evaluate subcondition2 when subcondition1 is false. Likewise, when the computer evaluates the "`subcondition1 || subcondition2`" expression, it does not evaluate subcondition2 when subcondition1 is true. In this case, because only one of the

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

subconditions combined with the Or operator needs to be true for the compound condition to be true, there is no need to evaluate subcondition2 when subcondition1 is true. The concept of evaluating subcondition2 based on the result of subcondition1 is referred to as **short-circuit evaluation**.

Using the Truth Tables

A program needs to display an employee's weekly gross pay, given the number of hours worked and the hourly pay rate. The number of hours worked must be at least 0 but not more than 40. Before making the gross pay calculation, the program should verify that the number of hours is within the expected range. Programmers refer to the process of verifying the input data as **data validation**. If the number of hours is valid, the program should calculate and display the gross pay; otherwise, it should display the message "Incorrect number of hours". Figure 5-16 shows the problem specification and two partially completed `if` clauses that could be used to verify the number of hours; missing from each clause is the appropriate logical operator.

Problem specification
A program needs to display an employee's weekly gross pay, given the number of hours worked and hourly pay rate. The number of hours worked must be at least 0 but not more than 40. If the number of hours worked is not valid, the program should display the message "Incorrect number of hours".

if clause 1
```
if (hours >= 0 _______ hours <= 40)
```

if clause 2
```
if (hours < 0 _______ hours > 40)
```

Figure 5-16 Problem specification and partially completed `if` clauses

The first `if` clause contains two subconditions that determine whether the number of hours is *within* the expected range of 0 through 40. For the number of hours to be valid, both subconditions must be true at the same time. In other words, the number of hours must be greater than or equal to 0 and also less than or equal to 40. If both subconditions are not true, it means that the number of hours is *outside* the expected range. Which logical operator should you use to combine both subconditions into one compound condition? According to the truth tables shown in Figure 5-15, only the And operator evaluates the compound condition as true when both subconditions are true, while evaluating the compound condition as false when at least one of the subconditions is false. Therefore, the correct compound condition to use here is `hours >= 0 && hours <= 40`.

The second `if` clause in Figure 5-16 contains two subconditions that determine whether the number of hours is *outside* the expected range of 0 through 40. For the number of hours to be invalid, at least one of the subconditions must be true. In other words, the number of hours must be either less than 0 or greater than 40. If neither subcondition is true, it means that the number of hours is *within* the expected range. Which logical operator should you use to combine both subconditions into one compound condition? According to the truth tables, only the Or operator evaluates the compound condition as true when at least one of the subconditions is true, while evaluating the compound condition as false when neither of the subconditions is true. Therefore, the correct compound condition to use here is `hours < 0 || hours > 40`.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

You can use either of the examples shown in Figure 5-17 to code the gross pay program. Both examples produce the same result and simply represent two different ways of performing the same task. Figure 5-17 also includes a sample run of the program.

Example 1

```
const int PAY_RATE = 10;
int hoursWorked    = 0;
int grossPay       = 0;

cout  << "Hours worked (0 through 40): ";
cin  >> hoursWorked;

if (hoursWorked >= 0 && hoursWorked <= 40)
{
    grossPay = hoursWorked * PAY_RATE;
    cout << "Gross pay: $" << grossPay << endl;
}
else
    cout << "Incorrect number of hours" << endl;
//end if
```

And operator

Example 2

```
const int PAY_RATE = 10;
int hoursWorked    = 0;
int grossPay       = 0;

cout << "Hours worked (0 through 40): ";
cin >> hoursWorked;

if (hoursWorked < 0 || hoursWorked > 40)
    cout << "Incorrect number of hours" << endl;
else
{
    grossPay = hoursWorked * PAY_RATE;
    cout << "Gross pay: $" << grossPay << endl;
}   //end if
```

Or operator

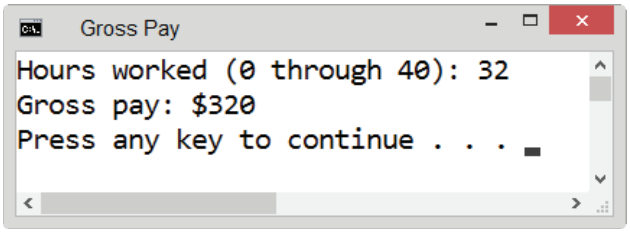


Figure 5-17 Gross pay program

A Different Version of the Area or Circumference Program

The Area or Circumference program shown earlier in Figure 5-11 prompts the user to enter two items: a circle's radius and either the number 1 or the number 2. If the user enters the number 1, the program calculates the circle's area; otherwise, it calculates the circle's circumference. Rather than using numbers to indicate the desired calculation, the examples in Figure 5-18 use letters: A (for the area) or C (for the circumference). The figure also includes a sample run of this version of the program.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Example 1

```
const double PI = 3.14;
double radius = 0.0;
char choice = ' ';
double answer = 0.0;

cout << "Circle Area or Circumference Calculator" << endl;
cout << "Enter the radius: ";
cin >> radius;
cout << "Enter A (area) or C (circumference): ";
cin >> choice;

//calculate and display
if (choice == 'A' || choice == 'a')
{
    answer = radius * radius * PI;
    cout << "Area: " << answer << endl;
}
else
{
    answer = 2 * radius * PI;
    cout << "Circumference: " << answer << endl;
}  //end if
```

character literal constants are enclosed in single quotation marks

Or operator

string literal constants are enclosed in double quotation marks

Example 2

```
const double PI = 3.14;
double radius = 0.0;
char choice = ' ';
double answer = 0.0;

cout << "Circle Area or Circumference Calculator" << endl;
cout << "Enter the radius: ";
cin >> radius;
cout << "Enter A (area) or C (circumference): ";
cin >> choice;

//calculate and display
if (choice != 'A' && choice != 'a')
{
    answer = 2 * radius * PI;
    cout << "Circumference: " << answer << endl;
}
else
{
    answer = radius * radius * PI;
    cout << "Area: " << answer << endl;
}   //end if
```

And operator

```
Area or Circumference
Circle Area or Circumference Calculator
Enter the radius: 15
Enter A (area) or C (circumference): a
Area: 706.5
Press any key to continue . . .
```

Figure 5-18 A different version of the Area or Circumference program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The compound condition in Example 1 uses the Or operator to determine whether the character stored in the `choice` variable is either the uppercase letter A or the lowercase letter a. When the variable contains one of those two letters, the compound condition evaluates to true, and the selection structure's true path calculates and displays the area; otherwise, its false path calculates and displays the circumference.

The compound condition in Example 2, on the other hand, uses the And operator to determine whether the value in the `choice` variable is *not* equal to the uppercase letter A and also *not* equal to the lowercase letter a. When the variable does not contain either of those two letters, the compound condition evaluates to true, and the selection structure's true path calculates and displays the circumference; otherwise, its false path calculates and displays the area.

You may be wondering why you need to compare the contents of the `choice` variable with both the uppercase and lowercase forms of the letter A. As is true in many programming languages, character comparisons in C++ are case sensitive, which means that the uppercase version of a letter is not the same as its lowercase counterpart. So, although a human being recognizes *A* and *a* as being the same letter, a computer does not; to a computer, an *A* is different from an *a*. You learned the reason for this differentiation in Chapter 3. Recall that each character on a computer keyboard is assigned a unique ASCII code, which is stored in the computer's internal memory using a group of *0*s and *1*s. The ASCII code for the uppercase letter A is 65 and is stored using the eight bits 01000001. The ASCII code for the lowercase letter a, on the other hand, is 97 and is stored using the eight bits 01100001. (The full ASCII chart is contained in Appendix B in this book.)

Summary of Operators

Ch05-Operators

Figure 5-19 shows the order of precedence for the arithmetic, comparison, and logical operators you have learned so far. Recall that operators with the same precedence number are evaluated from left to right in an expression. The figure also shows the evaluation steps for an expression that contains three arithmetic operators, two comparison operators, and one logical operator. Notice that the arithmetic operators are evaluated first, followed by the comparison operators, and then the logical operator. (Remember that you can use parentheses to override the order of precedence.)

Operator	Operation	Precedence number
( )	override normal precedence rules	1
–	negation (reverses the sign of a number)	2
*, /, %	multiplication, division, and modulus arithmetic	3
+, –	addition and subtraction	4
<, <=, >, >=	less than, less than or equal to, greater than, greater than or equal to	5
==, !=	equal to, not equal to	6
And (&&)	*all* subconditions must be true for the compound condition to evaluate to true	7
Or (||)	only *one* of the subconditions needs to be true for the compound condition to evaluate to true	8

Figure 5-19 Listing and an example of arithmetic, comparison, and logical operators *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Example	
Original expression	`20 < 80 / 2 + 3 && 25 > 10 * 2`
`80 / 2` is performed first	`20 < 40 + 3 && 25 > 10 * 2`
`10 * 2` is evaluated next	`20 < 40 + 3 && 25 > 20`
`40 + 3` is evaluated next	`20 < 43 && 25 > 20`
`20 < 43` is evaluated next	`true && 25 > 20`
`25 > 20` is evaluated next	`true && true`
`true && true` is evaluated last	`true`

For more examples of using the operators listed in Figure 5-19, see the Operators section in the Ch05WantMore.pdf file.

Figure 5-19 Listing and an example of arithmetic, comparison, and logical operators

Mini-Quiz 5-3

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

1. The compound condition `true && false` will evaluate to _____.
2. The compound condition `75 >= 3 * 25 || 15 < 22` will evaluate to _____.
3. The compound condition `24 * 2 < 20 || false` will evaluate to _____.
4. Which of the following determines whether the value in an `int` variable named `age` is between 30 and 40, including 30 and 40?
   a. `if (age <= 30 || age >= 40)`
   b. `if (age >= 30 && age <= 40)`
   c. `if (age >= 30 || age <= 40)`
   d. `if (age <= 30 && >= 40)`
5. Which of the following determines whether a `char` variable named `code` contains the letter R (in any case)?
   a. `if (code == 'R' || code == 'r')`
   b. `if (code = 'R' || code = 'r')`
   c. `if (code == "R" || code == "r")`
   d. none of the above

Converting a Character to Uppercase or Lowercase

In both examples shown earlier in Figure 5-18, the compound condition in the `if` clause compares the character stored in the `choice` variable with both the uppercase and lowercase forms of the letter A. Rather than using a compound condition, you can use either of the following C++ built-in functions: `toupper` or `tolower`. The **toupper function** temporarily converts a character to uppercase, while the **tolower function** temporarily converts it to lowercase. Figure 5-20 shows the syntax of both functions and includes examples of using the functions.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

HOW TO Use the `toupper` and `tolower` Functions

Syntax
toupper(*charVariable*)
tolower(*charVariable*)

Example 1
```
if (toupper(senior) == 'Y')
```
temporarily converts the contents of the `senior` variable to uppercase and then compares the result with the uppercase letter Y

Example 2
```
if (tolower(senior) == 'y')
```
temporarily converts the contents of the `senior` variable to lowercase and then compares the result with the lowercase letter y

Example 3
```
initial = toupper(initial);
senior = tolower(senior);
```
changes the contents of the `initial` and `senior` variables to uppercase and lowercase, respectively

Figure 5-20 How to use the `toupper` and `tolower` functions

An item that appears between parentheses in a function's syntax is called an **argument**, and it represents information that the function needs to perform its task. In this case, both functions need the name of a variable whose data type is `char`. Both functions copy the character stored in the *charVariable* to a temporary location in the computer's internal memory. The functions convert the temporary character to the appropriate case (if necessary) and then return the temporary character. Keep in mind that the `toupper` and `tolower` functions do not change the contents of the *charVariable*; they change the contents of the temporary location only. In addition, the functions affect only characters that represent letters of the alphabet, which are the only characters that have uppercase and lowercase forms.

When using the `toupper` function in a comparison, be sure that everything you are comparing is uppercase, as shown in Example 1; otherwise, the comparison will not evaluate correctly. For instance, the clause `if (toupper(senior) == 'y')` is not correct. The condition will always evaluate to false because the uppercase version of a letter will never be equal to its lowercase counterpart. Likewise, when using the `tolower` function in a comparison, be sure that everything you are comparing is lowercase, as shown in Example 2. As Example 3 indicates, you can use the `toupper` and `tolower` functions to permanently convert the contents of a `char` variable to uppercase or lowercase, respectively.

Formatting Numeric Output

In a C++ program, numbers with a decimal place are displayed in either fixed-point or e (exponential) notation, depending on the size of the number. Recall that a number with a decimal place is called a real number. Smaller real numbers—those containing six or fewer digits to the left of the decimal point—are usually displayed in fixed-point notation. For example, the number 1234.56 would be displayed in fixed-point notation as 1234.560000. Larger real numbers—those containing more than six digits to the left of the decimal point—typically

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

are displayed in e notation. The number 1,225,000.00, for example, would be displayed in e notation as 1.225e+006. The type of program you are creating determines the appropriate format to use when displaying numbers with a decimal place. Business programs usually display real numbers in fixed-point notation, while many scientific programs use e notation.

C++ provides stream manipulators that allow you to control the format used to display real numbers. You use the **fixed stream manipulator** to display real numbers in fixed-point notation. To display real numbers in e notation, you use the **scientific stream manipulator**. The appropriate manipulator must appear in a `cout` statement, and it must be processed before the real numbers you want formatted are displayed. After being processed, the manipulator remains in effect either until the end of the program or until the computer encounters another manipulator that changes the format, whichever occurs first. The `fixed` and `scientific` stream manipulators are defined in the iostream file.

Figure 5-21 shows examples of using the `fixed` and `scientific` manipulators. As the examples indicate, a stream manipulator can appear by itself in a `cout` statement; or, it can be included with other information in a `cout` statement.

HOW TO Use the `fixed` and `scientific` Stream Manipulators

Example 1

```
double sales = 10575.25;
cout << fixed;
cout << sales << endl;
```

Result: displays 10575.250000

Example 2

```
double rate = 5.9018432;
cout << fixed << rate << endl;
```

Result: displays 5.901843

Example 3

```
double rate = 5.9018436;
cout << fixed << rate << endl;
```

Result: displays 5.901844

Example 4

```
double sales = 10575.25;
cout << scientific << sales << endl;
```

Result: displays 1.057525e+04

Figure 5-21 How to use the `fixed` and `scientific` stream manipulators

Study closely the examples in Figure 5-21. Notice that the code in Example 1 displays 10575.250000 rather than 10575.25. This is because all real numbers formatted by the `fixed` stream manipulator will have six digits to the right of the decimal point. If the unformatted number contains fewer than six decimal places, the `fixed` stream manipulator pads the number with zeros until it has six decimal places. The number 10575.25, for instance, is padded with four zeros to make 10575.250000.

If the unformatted number contains more than six decimal places, the additional decimal places are truncated (dropped off). Before the truncation occurs, however, the number in the sixth decimal place is either rounded up one number or left as is, depending on the value of the number(s) being truncated. The `cout` statement in Example 2, for instance, displays the number 5.9018432 as 5.901843. No rounding occurs in Example 2 because the number in the seventh decimal place (2) is less than 5. The `cout` statement in Example 3, on the other hand, displays the 5.9018436 as 5.901844 because the number in the seventh decimal place (6) is greater than 5.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The `cout` statement in Example 4 displays the contents of the `sales` variable in e notation; the result is 1.057525e+04.

In most programs, especially business programs, numeric output is displayed with either zero or two decimal places. Rarely does a program require numbers to be displayed with the six decimal places you get from the `fixed` stream manipulator. You can use the C++ **setprecision stream manipulator** to control the number of decimal places that appear when a real number is displayed. The `setprecision` manipulator is defined in the iomanip file, which comes with your C++ compiler. (The "io" stands for "input/output.") However, for a program to use the manipulator, it must contain the `#include <iomanip>` directive.

Figure 5-22 shows the `setprecision` manipulator's syntax. The *numberOfDecimalPlaces* argument in the syntax is an integer that specifies the number of decimal places to include when displaying a real number. The `setprecision` manipulator remains in effect either until the end of the program or until the computer encounters another `setprecision` manipulator. Also included in Figure 5-22 are examples of using the manipulator in a C++ statement. As Example 2 shows, you can include the `setprecision` and `fixed` manipulators in the same statement.

Stream manipulators with arguments are defined in the iomanip file. Stream manipulators that do not have arguments are defined in the iostream file.

HOW TO Use the `setprecision` Stream Manipulator

Syntax
setprecision(*numberOfDecimalPlaces*)

Example 1

```
double sales = 3500.6;
cout << fixed;
cout << setprecision(2);
cout << sales << endl;
```

Result: displays 3500.60

Example 2

```
double rate = 10.0732;
cout << fixed << setprecision(3);
cout << rate << endl;
```

Result: displays 10.073

Example 3

```
double sales = 3467.55;
cout << fixed;
cout << setprecision(0) << sales;
```

Result: displays 3468

Figure 5-22 How to use the `setprecision` stream manipulator

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 5-4

1. Which of the following indicates that real numbers should be displayed in fixed-point notation with two decimal places?
   a. `cout << fixed << decimal(2);`
   b. `cout << fixedPoint << precision(2);`
   c. `cout << fixedPoint << setdecimal(2);`
   d. `cout << fixed << setprecision(2);`

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

2. Which of the following changes the contents of a `char` variable named `letter` to lowercase?

   a. `tolower(letter) = letter;`
   b. `letter == tolower(letter);`
   c. `letter = tolower(letter);`
   d. `tolower('letter');`

3. If the `num` variable contains the number 34.65, the `cout << fixed << num;` statement will display the number as _____.

   a. 34.65
   b. 34.650
   c. 34.6500
   d. 34.650000

LAB 5-1 Stop and Analyze

Study the program shown in Figure 5-23, and then answer the questions.

The answers to the labs are contained in the Answers.pdf file.

```
 1 //Lab5-1.cpp - displays projected sales
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <iomanip>
 6 using namespace std;
 7
 8 int main()
 9 {
10    double sales = 0.0;
11    double rate  = 0.0;
12    char code    = ' ';
13
14    cout << "Sales: ";
15    cin >> sales;
16    cout << "Code (1, 2, 3, or 4): ";
17    cin >> code;
18
19    if (code == '1' || code == '3')
20        rate = 0.2;
21    else
22        rate = 0.15;
23    //end if
24
25    //calculate and display the projected sales amount
26    sales = sales + sales * rate;
27    cout << fixed << setprecision(2);
28    cout << "Projected sales: " << sales << endl;
29
30    return 0;
31 }  //end of main function
```

Figure 5-23 Program for Lab 5-1

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

QUESTIONS

1. What rate will be assigned to the `rate` variable when the user enters the following codes: 1, 3, 2, 4, and 5?
2. Why is the directive on Line 5 necessary?
3. Why are the literal constants on Line 19 enclosed in single quotation marks?
4. How would you rewrite the `if` statement on Lines 19 through 23 to use the `!=` operator in the condition?
5. How else could you write the statement on Line 26?
6. What changes would you need to make to the program so that it doesn't use the `rate` variable?

LAB 5-2 Plan and Create

In this lab, you will plan and create an algorithm that can be used to solve the Heaton Boutique problem shown in Figure 5-24.

Problem specification
Heaton Boutique allows customers to purchase items over the phone and have the items shipped to their homes. The shipping fee is $0.99 if the purchase amount after subtracting any discount is at least $100; otherwise, it is $4.99. The only discount Heaton Boutique offers is to customers who are members of the store's Premier Club; the discount rate is 10%. The program should display the total amount the customer owes for his or her purchase.

Figure 5-24 Problem specification for Lab 5-2

First, analyze the problem, looking for the output first and then for the input. In this case, the program should display the final amount the customer owes for his or her purchase. To calculate the final amount owed, the computer will need to know the discount rate, the two shipping charges, the original amount owed, and whether the customer is a member of the store's Premier Club; the latter two items will be entered by the user.

Next, plan the algorithm. As you know, most algorithms begin with an instruction to enter the input items into the computer, followed by instructions that process the input items, typically including the items in one or more calculations. Most algorithms end with one or more instructions that display, print, or store the output items. Figure 5-25 shows the completed IPO chart for the Heaton Boutique problem. Notice that the algorithm requires two selection structures. The single-alternative selection structure determines whether the customer is entitled to a 10% discount for being a member of the Premier Club. The dual-alternative selection structure determines the appropriate shipping charge.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Input	Processing	Output
discount rate (10%) shipping rate 1 (0.99) shipping rate 2 (4.99) amount owed member status (Y or N)	Processing items: none	amount owed

Algorithm:

1. enter amount owed and member status
2. if (member status is Y)
   calculate amount owed by multiplying amount owed by discount rate and then subtracting the result from amount owed
   end if
3. if (amount owed >= 100)
   add shipping rate 1 to amount owed
   else
   add shipping rate 2 to amount owed
   end if
4. display amount owed

Figure 5-25 Completed IPO chart for the Heaton Boutique problem

After completing the IPO chart, you then move on to the third step in the problem-solving process, which is to desk-check the algorithm. Figure 5-26 shows the results of deck-checking the algorithm using five different sets of data.

discount rate	shipping rate 1	shipping rate 2	amount owed	member status	
~~0.1~~	~~0.99~~	~~4.99~~	~~50.25~~ ~~55.24~~	~~N~~	
~~0.1~~	~~0.99~~	~~4.99~~	~~50.25~~ ~~50.22~~	~~Y~~	
~~0.1~~	~~0.99~~	~~4.99~~	~~125.0~~ ~~125.99~~	~~N~~	
~~0.1~~	~~0.99~~	~~4.99~~	~~125.0~~ ~~113.49~~	~~Y~~	
0.1	0.99	4.99	~~125.0~~ 125.99	X	be sure to test with invalid data no discount is given when the member status is invalid

Figure 5-26 Completed desk-check table for the Heaton Boutique algorithm

The fourth step in the problem-solving process is to code the algorithm into a program. You begin by declaring memory locations that will store the values of the input, processing (if any), and output items. The Heaton Boutique program will need five memory locations to store its input and output items. The first three input items will be stored in named constants because

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

their values will not change as the program is running. The remaining two input items—amount owed and member status—will be stored in variables because the user should be allowed to change their values during runtime. The amount owed item in the Output column will use the same variable as the amount owed item in the Input column.

The three named constants and the variable that stores the amount owed will contain real numbers, so those memory locations will be declared using the `double` data type. The variable that stores the member status will be declared using the `char` data type because it needs to store only one character. Figure 5-27 shows the input, processing, and output items from the IPO chart, along with the corresponding C++ instructions.

IPO chart information	**C++ instructions**
Input	
discount rate (10%)	`const double DISCOUNT_RATE = 0.1;`
shipping rate 1 (0.99)	`const double SHIP_CHG1 = 0.99;`
shipping rate 2 (4.99)	`const double SHIP_CHG2 = 4.99;`
amount owed	`double amtOwed = 0.0;`
member status (Y or N)	`char member = ' ';`
Processing	
none	
Output	
amount owed	uses the amtOwed variable declared above
Algorithm:	

1. enter amount owed and member status	`cout << "Amount owed before any discount and shipping: ";`
	`cin >> amtOwed;`
	`cout << "Premier Club member (Y/N)? ";`
	`cin >> member;`

2. if (member status is Y)
calculate amount owed by multiplying amount owed by discount rate and then subtracting the result from amount owed
end if
	`if (toupper(member) == 'Y')`
	`amtOwed -= amtOwed * DISCOUNT_RATE;`
	`//end if`

3. if (amount owed >= 100)
add shipping rate 1 to amount owed
else
add shipping rate 2 to amount owed
end if
	`if (amtOwed >= 100.0)`
	`amtOwed += SHIP_CHG1;`
	`else`
	`amtOwed += SHIP_CHG2;`
	`//end if`

4. display amount owed	`cout << fixed << setprecision(2);` `cout << "Amount owed after any discount and shipping:" << amtOwed << endl;`

Figure 5-27 IPO chart information and C++ instructions for the Heaton Boutique problem

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The fifth step in the problem-solving process is to desk-check the program. You begin by placing the names of the declared variables and named constants (if any) in a new desk-check table, along with their initial values. You then desk-check the remaining C++ instructions in order, recording in the desk-check table any changes made to the variables. Figure 5-28 shows the completed desk-check table for the program. The results agree with those shown in the algorithm's desk-check table in Figure 5-26.

DISCOUNT_RATE	SHIP_CHG1	SHIP_CHG2	amtOwed	member
~~0.1~~	~~0.99~~	~~4.99~~	~~0.0~~	-
			~~50.25~~	~~N~~
			~~55.24~~	
~~0.1~~	~~0.99~~	~~4.99~~	~~0.0~~	-
			~~50.25~~	~~Y~~
			~~50.22~~	
~~0.1~~	~~0.99~~	~~4.99~~	~~0.0~~	-
			~~125.0~~	~~N~~
			~~125.99~~	
~~0.1~~	~~0.99~~	~~4.99~~	~~0.0~~	-
			~~125.0~~	~~Y~~
			~~113.49~~	
0.1	0.99	4.99	~~0.0~~	-
			~~125.0~~	X
			125.99	

Figure 5-28 Completed desk-check table for the Heaton Boutique program

The final step in the problem-solving process is to evaluate and modify (if necessary) the program. Recall that you evaluate a program by entering its instructions into the computer and then using the computer to run (execute) it. While the program is running, you enter the same sample data used when desk-checking the program.

DIRECTIONS

1. Open the Cpp8\Chap05 folder. If the folder contains a Ch05-Lab5-2 *developmentTool*.pdf file for your C++ development tool, open the PDF file, and then follow the directions listed in the file.
2. If the Cpp8\Chap05 folder does *not* contain a PDF file for your C++ development tool, contact your instructor or technical support person for the appropriate instructions. Follow the instructions you are given for starting and using your C++ development tool. Enter the instructions shown in Figure 5-29 into a source file named Lab5-2.cpp. (Do not enter the line numbers.) Save the file in the Cpp8\Chap05 folder. Now follow the appropriate instructions for running the Lab5-2.cpp file. Use the sample data from Figure 5-28 to test the program. If necessary, correct any bugs (errors) in the program. **Note**: If your C++ development tool does not automatically pause program execution and display the *Press any key to continue message* when a program ends, enter the `system("pause");` statement above the `return 0;` statement in the program.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

```
 1 //Lab5-2.cpp - displays the total amount due
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <iomanip>
 6 using namespace std;
 7
 8 int main()
 9 {
10    const double DISCOUNT_RATE = 0.1;
11    const double SHIP_CHG1 = 0.99;
12    const double SHIP_CHG2 = 4.99;
13    double amtOwed = 0.0;
14    char member = ' ';
15
16    //enter input items
17    cout << "Amount owed before any discount and shipping: ";
18    cin >> amtOwed;
19    cout << "Premier Club member (Y/N)? ";
20    cin >> member;
21
22    //subtract discount, if appropriate
23    if (toupper(member) == 'Y')
24        amtOwed -= amtOwed * DISCOUNT_RATE;
25    //end if
26
27    //add shipping
28    if (amtOwed >= 100.0)
29        amtOwed += SHIP_CHG1;
30    else
31        amtOwed += SHIP_CHG2;
32    //end if
33
34    //display final amount owed
35    cout << fixed << setprecision(2);
36    cout << "Amount owed after any discount and shipping: "
37        << amtOwed << endl;
38
39    return 0;
40 }  //end of main function
```

Figure 5-29 Heaton Boutique program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

LAB 5-3 Modify

In this lab, you will modify the Heaton Boutique program from Lab 5-2 to give a 10% discount to members of the store's Premier Club, and a 5% discount to all other customers.

Open the Cpp8\Chap05 folder. If the folder contains a Ch05-Lab5-3 *developmentTool*.pdf file for your C++ development tool, open the PDF file, and then follow the directions listed in the file.

If the Cpp8\Chap05 folder does *not* contain a PDF file for your C++ development tool, copy the program instructions from Lab 5-2 into a new source file named Lab5-3.cpp file. Save the file in the Cpp8\Chap05 folder. Modify the program instructions appropriately. Be sure to change Lab5-2.cpp in the first comment to Lab5-3.cpp. Use the sample data from Figure 5-28 to test the program.

LAB 5-4 What's Missing?

The program in this lab should display the total price of the tickets purchased by a customer. A maximum of 10 tickets can be purchased. Start your C++ development tool, and view the Lab5-4.cpp file, which is contained in either the Cpp8\Chap05\Lab5-4 Project folder or the Cpp8\Chap05 folder. (Depending on your C++ development tool, you may need to open Lab5-4's project/solution file first.) Put the C++ instructions in the proper order, and then determine the one or more missing instructions. Test the program three times using the following data: 8, 12, and –3 (the negative number 3).

LAB 5-5 Desk-Check

Desk-check the code shown in Figure 5-30 using the numbers 5 and 0. Although the code displays the appropriate message, it is considered inefficient. Why? How can you fix the code to make it more efficient?

```
int quantity = 0;
cout << "Quantity: ";
cin >> quantity;

if (quantity <= 0)
    cout << "The quantity must be greater than 0." << endl;
//end if
if (quantity > 0)
    cout << "Valid quantity" << endl;
//end if
```

Figure 5-30 Code for Lab 5-5

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

LAB 5-6 Debug

Open the Cpp8\Chap05 folder. If the folder contains a Ch05-Lab5-6 *developmentTool*.pdf file for your C++ development tool, open the PDF file, and then follow the directions listed in the file.

If the Cpp8\Chap05 folder does *not* contain a PDF file for your C++ development tool, start your C++ development tool and view the Lab5-6.cpp file. Read the comments entered at the beginning of the program. Test the program using codes of 1, 2, and 3. Use 100 as the purchase price. Debug the program.

Chapter Summary

You use the selection structure when you want a program to make a decision before selecting the next instruction to process.

Studying the problem specification will help you determine whether a solution requires a selection structure.

A selection structure's condition must evaluate to either true or false. In single-alternative and dual-alternative selection structures, the instructions to follow when the structure's condition is true are placed in the structure's true path. In a dual-alternative selection structure, the instructions to follow when the structure's condition is false are placed in the structure's false path. You should indent the instructions in both paths.

A diamond, called the decision symbol, is used to represent a selection structure's condition in a flowchart. Each selection structure diamond has one flowline entering the symbol and two flowlines leaving the symbol. One of the flowlines leading out of a diamond should be marked with a "T" (for true), and the other should be marked with an "F" (for false).

You can use the `if` statement to code single-alternative and dual-alternative selection structures. The statement's condition must evaluate to either true or false.

If either an `if` statement's true path or its false path contains more than one statement, the statements in the path must be entered as a statement block, which means the statements must be enclosed in a set of braces (`{}`).

You use comparison operators to compare values in expressions; the values should have the same data type. Expressions containing comparison operators always evaluate to either true or false. If more than one comparison operator with the same precedence number appears in a C++ expression, the computer evaluates those operators from left to right in the expression.

You should not use either the equality operator (==) or the inequality operator (!=) to compare two real numbers because not all real numbers can be stored precisely in memory.

A memory location declared in an `if` statement's true path can be used only by the instructions following its declaration statement within the true path. Likewise, a memory location declared in an `if` statement's false path can be used only by the instructions following its declaration statement within the false path.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The And and Or logical operators are represented in C++ by the symbols && and ||, respectively. All expressions containing a logical operator evaluate to either true or false.

In an expression, arithmetic operators are evaluated first, followed by comparison operators and then logical operators.

Character comparisons in C++ are case sensitive.

The `toupper` and `tolower` functions temporarily convert a character to uppercase and lowercase, respectively.

C++ provides the `fixed` and `scientific` stream manipulators for formatting the display of real numbers. It provides the `setprecision` stream manipulator for controlling the number of decimal places that appear when a real number is displayed. The `fixed` and `scientific` stream manipulators are defined in the iostream file. The `setprecision` stream manipulator is defined in the iomanip file.

Key Terms

Argument—an item that appears between the parentheses that follow a function's name; represents information that the function needs to perform its task

Boolean operators—another term for logical operators

Comparison operators—operators used to compare values having the same data type in an expression; also called relational operators; <, <=, >, >=, ==, !=

Data validation—the process of verifying that a program's input data is within the expected range

Decision symbol—the diamond in a flowchart; used to represent the condition in either a selection or repetition structure

Dual-alternative selection structures—selection structures that require two sets of actions: one to be taken only when the structure's condition is true, and the other only when the condition is false

False path—contains the instructions to be processed when a dual-alternative selection structure's condition evaluates to false

`fixed` stream manipulator—the manipulator used to display real numbers in fixed-point notation

Local variable—a variable declared within a statement block; can be used only by the instructions within the statement block in which it is declared, and the instructions must appear after its declaration statement

Logical operators—operators used to combine two or more subconditions into one compound condition; also called Boolean operators

`scientific` stream manipulator—the manipulator used to display real numbers in scientific (e) notation

Selection structure—one of the three control structures; tells the computer to make a decision before selecting the next instruction to process; also called the decision structure

`setprecision` stream manipulator—the manipulator used to control the number of decimal places that appear when a real number is displayed

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Short-circuit evaluation—refers to the way the computer evaluates two subconditions connected by a logical operator; when the logical operator is And, the computer does not evaluate subcondition2 when subcondition1 is false; when the logical operator is Or, the computer does not evaluate subcondition2 when subcondition1 is true

Single-alternative selection structure—a selection structure that requires a special set of actions to be taken only when the structure's condition is true

Statement block—one or more instructions enclosed in a set of braces ({})

`tolower` function—temporarily converts a character to lowercase

`toupper` function—temporarily converts a character to uppercase

True path—contains the instructions to be processed when a selection structure's condition evaluates to true

Truth tables—tables that summarize how the computer evaluates the logical operators in an expression

Review Questions

1. If an `if` statement's true path contains the statement `double avg = 0.0;`, where can the `avg` variable be used?
   a. in any instruction after the declaration statement in the entire program
   b. in any instruction after the declaration statement in the `if` statement
   c. in any instruction after the declaration statement in the `if` statement's true path
   d. none of the above because you can't declare a variable in an `if` statement's true path

2. Which of the following is a valid `if` clause? (The `average` variable has the `double` data type.)
   a. `if (average > 70.5 && average < 80.5)`
   b. `if (average < 70.5 && average > 80.5)`
   c. `if (average < 70.5 || > 80.5)`
   d. `if (average > 70.5 && < 80.5)`

3. Which of the following conditions evaluates to true when the `letter` variable contains the letter Z in either uppercase or lowercase?
   a. `if (letter = 'Z' || letter = 'z')`
   b. `if (letter == 'Z' || letter == 'z')`
   c. `if (letter = 'Z' && letter = 'z')`
   d. `if (letter == 'Z' && letter = 'z')`

4. The expression `4 > 3 && 7 >= 4` evaluates to _____ .
   a. true
   b. false

5. The computer will perform short-circuit evaluation when processing which of the following `if` clauses?
   a. `if (3 * 2 < 4 && 5 > 3)`
   b. `if (6 < 9 || 5 > 3)`
   c. `if (12 > 4 * 4 && 6 > 2)`
   d. all of the above

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

6. If an expression does not contain any parentheses, which of the following operators is performed first in the expression?
   a. arithmetic
   b. comparison
   c. logical
   d. you can't tell without seeing the expression

7. The expression `4 * 3 < 6 + 7 && 7 < 6 + 9` evaluates to _____ .
   a. true
   b. false

8. Which of the following compares the contents of an `int` variable named `quantity` with the number 5?
   a. `if (quantity = 5)`
   b. `if (quantity == 5)`
   c. `if (quantity is 5)`
   d. `if (quantity =! 5)`

9. Which of the following is required in a program that uses the `setprecision` stream manipulator?
   a. `#include <iostream>`
   b. `#include <setprecision>`
   c. `#include <iomanip>`
   d. `#include <manipulators>`

10. Which of the following tells the computer to display real numbers in fixed-point notation with no decimal places?
    a. `cout << fixed << decimal(0);`
    b. `cout << fixed << precision(0);`
    c. `cout << fixed << setprecision(0);`
    d. `cout << fixed << setdecimal(0);`

Exercises

Pencil and Paper

1. Write the C++ code to compare the contents of two `int` variables named `code1` and `code2`. If both variables contain the same value, display the "Equal" message; otherwise, display the "Not equal" message. (The answers to TRY THIS Exercises are located at the end of the chapter.) TRY THIS

2. Code the partial flowchart shown in Figure 5-31. Use an `int` variable named `ordered`, a `char` variable named `code`, and `double` variables named `price` and `discount`. (The answers to TRY THIS Exercises are located at the end of the chapter.) TRY THIS

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

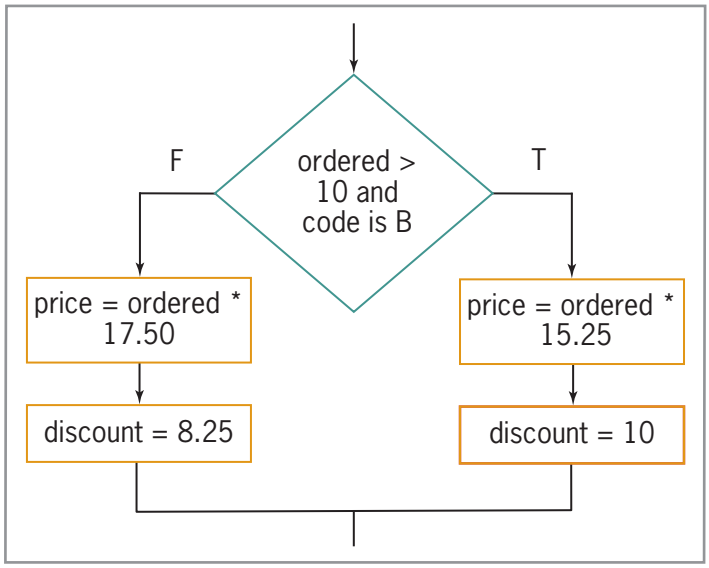


Figure 5-31

MODIFY THIS

3. Complete TRY THIS Exercise 1, and then modify the code so that the true path displays "Not equal" and the false path displays "Equal".

INTRODUCTORY

4. Write the C++ code to display the message "Entry error" when the value in the `units` variable is less than or equal to 0. Otherwise, calculate the total owed by multiplying the `units` variable's value by 5. Store the total owed in the `total` variable, and then display the total owed.

INTERMEDIATE

5. A program stores sales amounts in two `double` variables named `marySales` and `jimSales`. Write the C++ code to assign the highest and lowest sales amounts to the `highSales` and `lowSales` variables, respectively, and then display the contents of those variables. (You can assume that both sales amounts are different.)

ADVANCED

6. A program uses a `char` variable named `department` and two `double` variables named `salary` and `raise`. The `department` variable contains one of the following letters (entered in either uppercase or lowercase): A, B, or C. Employees in departments A and B are receiving a 2% raise. Employees in department C are receiving a 1.5% raise. Write the C++ code to calculate and display the appropriate raise amount. Display the raise amount in fixed-point notation with two decimal places.

SWAT THE BUGS

7. Correct the errors in the lines of code shown in Figure 5-32. The `code` variable has the `char` data type; the other variables have the `int` data type.

```
if (toupper(code) = 'x')
    cout << "Discontinued" << endl;
else
    cout << "How many? ";
    cin >> quantity;
    total = quantity * 10;
//end if
```

Figure 5-32

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Computer

TRY THIS

8. Code the flowchart shown in Figure 5-33. Rate1 and rate2 are 2% and 1.5%, respectively. If necessary, create a new project named TryThis8 Project, and save it in the Cpp8\Chap05 folder. Enter the C++ instructions into a source file named TryThis8.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Display the bonus in fixed-point notation with two decimal places. Test the program using 20500.95 as the sales amount. The answer should be $410.02. Now test it using 9675.50. (The answers to TRY THIS Exercises are located at the end of the chapter.)

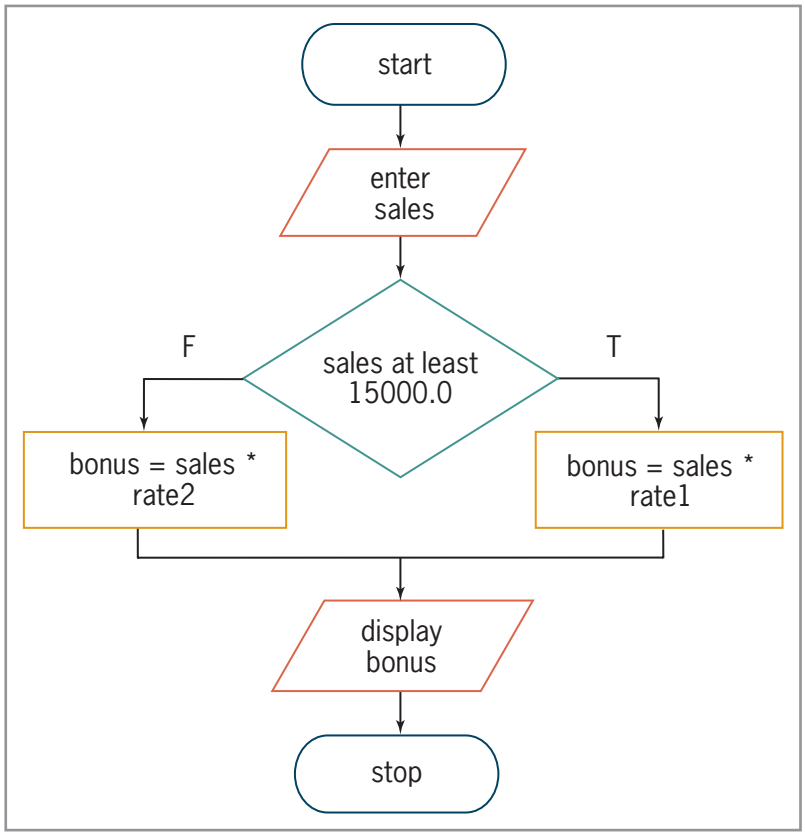


Figure 5-33

TRY THIS

9. Complete Figure 5-34 by writing the algorithm and corresponding C++ instructions. Employees with a pay code of 1, 4, or 9 receive a 4.5% raise; all other employees receive a 3.5% raise. If necessary, create a new project named TryThis9 Project, and save it in the Cpp8\Chap05 folder. Enter the C++ instructions into a source file named TryThis9.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Display the new pay in fixed-point notation with two decimal places. Test the program using 1 and 500 as the pay code and current pay, respectively. The new pay should be $522.50. Now test the program using the following three sets of input values: 4 and 450, 9 and 500, 2 and 625. (The answers to TRY THIS Exercises are located at the end of the chapter.)

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

IPO chart information	C++ instructions
Input	
pay code	`char code = ' ';`
current pay	`double currentPay = 0.0;`
raise rate 1 (4.5%)	`const double RATE1 = 0.045;`
raise rate 2 (3.5%)	`const double RATE2 = 0.035;`
Processing	
raise	`double raise = 0.0;`
Output	
new pay	`double newPay = 0.0;`
Algorithm:	

Figure 5-34

MODIFY THIS

10. In this exercise, you will modify the program from Lab 5-1. If necessary, create a new project named ModifyThis10 Project, and save it in the Cpp8\Chap05 folder. Enter the instructions shown earlier in Figure 5-23 into a new source file named ModifyThis10.cpp. Currently, the 20% rate is assigned to the `rate` variable only when the `code` variable contains either the character 1 or the character 3. Modify the selection structure so that codes 1 and 3 still get the 20% rate, but only when the sales amount is at least $20,000; otherwise, they should get the 15% rate. Test the program appropriately.

INTRODUCTORY

11. Mountain Coffee wants a program that allows a clerk to enter the number of pounds of coffee ordered, the price per pound, and whether the customer should be charged a 3.5% sales tax. The program should calculate and display the total amount the customer owes. Use an `int` variable for the number of pounds, a `double` variable for the price per pound, and a `char` variable for the sales tax information.

    a. Create an IPO chart for the problem, and then desk-check the algorithm twice. For the first desk-check, use 5 as the number of pounds and 13.69 as the price per pound; the customer should be charged the sales tax. For the second desk-check, use 3 as the number of pounds and 11.59 as the price per pound; the customer should not be charged the sales tax.
    b. List the input, processing, and output items, as well as the algorithm, in a chart similar to the one shown earlier in Figure 5-27. Then code the algorithm into a program.
    c. Desk-check the program using the same data used to desk-check the algorithm.
    d. If necessary, create a new project named Introductory11 Project, and save it in the Cpp8\Chap05 folder. Enter your C++ instructions into a source file named Introductory11.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Display the total amount owed in fixed-point notation with two decimal places. Test the program using the same data used to desk-check the program.

INTRODUCTORY

12. A local department store is having a BoGoHo (Buy One, Get One Half Off) sale. The store manager wants a program that allows the salesclerk to enter the prices of two items. The program should calculate and display the total amount the customer owes. The half-off should always be taken on the item having the lowest price. For example, if the items cost $24.99 and $10, the half-off would be taken on the $10 item. If both prices are the same, take the half-off on the second item.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

a. Create an IPO chart for the problem, and then desk-check the algorithm twice. For the first desk-check, use 24.99 and 10 as the prices. For the second desk-check, use 11.50 and 30.99.
b. List the input, processing, and output items, as well as the algorithm, in a chart similar to the one shown earlier in Figure 5-27. Then code the algorithm into a program.
c. Desk-check the program using the same data used to desk-check the algorithm.
d. If necessary, create a new project named Introductory12 Project, and save it in the Cpp8\Chap05 folder. Enter your C++ instructions into a source file named Introductory12.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Display the total amount owed in fixed-point notation with two decimal places. Test the program using the same data used to desk-check the program.

13. INTERMEDIATE

Allenton Water Department wants a program that calculates a customer's monthly water bill. The clerk will enter the current and previous meter readings. The program should calculate and display the number of gallons of water used and the total charge for the water. The charge for water is $7 per 1,000 gallons. However, there is a minimum charge of $16.67. (In other words, every customer must pay at least $16.67.)

a. Create an IPO chart for the problem, and then desk-check the algorithm twice. For the first desk-check, use 16000 and 13000 as the current and previous meter readings, respectively. For the second desk-check, use 3675 and 1650.
b. List the input, processing, and output items, as well as the algorithm, in a chart similar to the one shown earlier in Figure 5-27. Then code the algorithm into a program.
c. Desk-check the program using the same data used to desk-check the algorithm.
d. If necessary, create a new project named Intermediate13 Project, and save it in the Cpp8\Chap05 folder. Enter your C++ instructions into a source file named Intermediate13.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Display the total charge in fixed-point notation with two decimal places. Test the program using the same data used to desk-check the program.

14. INTERMEDIATE

Figure 5-35 shows the Mifflin–St Jeor formulas for calculating a person's basal metabolic rate (BMR), which is the minimum number of calories needed to keep his or her body functioning while resting for 24 hours. A personal trainer at a local health club wants a program that displays a client's BMR.

a. Create an IPO chart for the problem, and then desk-check the algorithm twice. For the first desk-check, display the BMR for a 25-year-old male whose weight and height are 175 pounds and 6 feet, respectively. For the second desk-check, display the BMR for a 31-year-old female whose weight and height are 130 pounds and 5.5 feet, respectively.
b. List the input, processing, and output items, as well as the algorithm, in a chart similar to the one shown earlier in Figure 5-27. Then code the algorithm into a program.
c. Desk-check the program using the same data used to desk-check the algorithm.
d. If necessary, create a new project named Intermediate14 Project, and save it in the Cpp8\Chap05 folder. Enter your C++ instructions into a source file named Intermediate14.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Display the BMR in fixed-point notation with no decimal places. Test the program using the same data used to desk-check the program.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

BMR formulas

Males
BMR = (10 x weight in kg) + (6.25 x height in cm) – (5 x age in years) + 5

Females
BMR = (10 x weight in kg) + (6.25 x height in cm) – (5 x age in years) – 161

Note: One kilogram (kg) equals 2.2 pounds. One inch equals 2.54 centimeters (cm).

Figure 5-35

ADVANCED

15. A third-grade teacher at Potter Elementary School wants a program that allows a student to enter the amount of money a customer owes and the amount of money the customer paid. The program should calculate and display the amount of change, as well as how many dollars, quarters, dimes, nickels, and pennies to return to the customer. Display an appropriate message when the amount paid is less than the amount owed.
    a. Create an IPO chart for the problem, and then desk-check the algorithm three times. For the first desk-check, use 75.34 and 80 as the amount owed and paid, respectively. For the second desk-check, use 39.67 and 50. For the third desk-check, use 10.55 and 9.75.
    b. List the input, processing, and output items, as well as the algorithm, in a chart similar to the one shown earlier in Figure 5-27. Then code the algorithm into a program.
    c. Desk-check the program using the same data used to desk-check the algorithm.
    d. If necessary, create a new project named Advanced15 Project, and save it in the Cpp8\Chap05 folder. Enter your C++ instructions into a source file named Advanced15.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Display the change in fixed-point notation with two decimal places. Display the remaining output in fixed-point notation with no decimal places. Test the program using the same data used to desk-check the program.

ADVANCED

16. As you learned in the chapter, you must be careful when comparing two real numbers for either equality or inequality because some real numbers cannot be stored precisely in memory. To determine whether two real numbers are either equal or unequal, you should test that the difference between both numbers is less than some acceptable small value, such as 0.00001.
    a. Start your C++ development tool, and view the Advanced16.cpp file. The file is contained in either the Cpp8\Chap05\Advanced16 Project folder or the Cpp8\Chap05 folder. (Depending on your C++ development tool, you may need to open this exercise's solution/project file first.) The code divides the contents of the `num1` variable (10.0) by the contents of the `num2` variable (3.0), storing the result (approximately 3.33333) in the `quotient` variable. An `if` statement is used to compare the contents of the `quotient` variable with the number 3.33333. The `if` statement displays a message that indicates whether the numbers are equal.
    b. Run the program. Even though the message on the screen states that the quotient is 3.33333, the message indicates that this value is not equal to 3.33333. Close the Command Prompt window.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

c. If you need to compare two real numbers for equality or inequality, first find the difference between both numbers, and then compare the absolute value of that difference to a small number, such as 0.00001. The absolute value of a number is a positive number that represents the distance the number is from 0 on the number line. For example, the absolute value of the number 5 is 5, and so is the absolute value of the number –5; both numbers are an equal distance from 0 on the number line. You can use the C++ `fabs` function to find the absolute value of a real number; however, your program must contain the `#include <cmath>` directive. Modify the program appropriately. Save and then run the program. This time, the message "Yes, the quotient 3.33333 is equal to 3.33333." appears.

17. Start your C++ development tool, and view the SwatTheBugs17.cpp file. The file is contained in either the Cpp8\Chap05\SwatTheBugs17 Project folder or the Cpp8\Chap05 folder. (Depending on your C++ development tool, you may need to open this exercise's solution/project file first.) The program should display a 10% bonus for sales over $10,000. Correct the syntax errors, and then save, run, and test the program. SWAT THE BUGS

Answers to TRY THIS Exercises

Pencil and Paper

1. See Figure 5-36.

```
if (code1 == code2)
    cout << "Equal" << endl;
else
    cout << "Not equal" << endl;
//end if
```

Figure 5-36

2. See Figure 5-37.

```
if (ordered > 10 && toupper(code) == 'B')
{
    price = ordered * 15.25;
    discount = 10;
}
else
{
    price = ordered * 17.50;
    discount = 8.25;
}   //end if
```

Figure 5-37

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Computer

8. See Figure 5-38.

```
 1 //TryThis8.cpp - displays a bonus
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <iomanip>
 6 using namespace std;
 7
 8 int main()
 9 {
10     //declare named constants and variables
11     const double RATE1 = 0.02;
12     const double RATE2 = 0.015;
13     double sales = 0.0;
14     double bonus = 0.0;
15
16     //enter input item
17     cout << "Sales amount: ";
18     cin >> sales;
19
20     //calculate bonus
21     if (sales >= 15000.0)
22         bonus = sales * RATE1;
23     else
24         bonus = sales * RATE2;
25     //end if
26
27     //display bonus
28     cout << fixed << setprecision(2);
29     cout << "Bonus: $" << bonus << endl;
30     return 0;
31   }  //end of main function
```

Figure 5-38

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

9. See Figures 5-39 and 5-40.

IPO chart information	C++ instructions
Input	
pay code	`char code = ' ';`
current pay	`double currentPay = 0.0;`
rate 1 (4.5%)	`const double RATE1 = 0.045;`
rate 2 (3.5%)	`const double RATE2 = 0.035;`
Processing	
raise	`double raise = 0.0;`
Output	
new pay	`double newPay = 0.0;`
Algorithm:	

1. enter the pay code and current pay	`cout << "Pay code: ";`
	`cin >> code;`
	`cout << "Current pay: ";`
	`cin >> currentPay;`

2. if (the pay code is 1 or 4 or 9)	`if (code == '1' || code == '4'` `|| code == '9')`
calculate the raise by multiplying the current pay by rate 1	`raise = currentPay * RATE1;`
else	`else`
calculate the raise by multiplying the current pay by rate 2	`raise = currentPay * RATE2;`
end if	`//end if`
3. calculate the new pay by adding the raise to the current pay	`newPay = currentPay + raise;`
4. display the new pay	`cout << "New pay: $" <<` `newPay << endl;`

Figure 5-39

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

```
 1 //TryThis9.cpp - displays the new pay
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <iomanip>
 6 using namespace std;
 7
 8 int main()
 9 {
10     //declare named constants and variables
11     const double RATE1 = 0.045;
12     const double RATE2 = 0.035;
13     char code = ' ';
14     double currentPay = 0.0;
15     double raise = 0.0;
16     double newPay = 0.0;
17
18     //enter input items
19     cout << "Pay code: ";
20     cin >> code;
21     cout << "Current pay: ";
22     cin >> currentPay;
23
24     //calculate raise and new pay
25     if (code == '1' || code == '4' || code == '9')
26         raise = currentPay * RATE1;
27     else
28         raise = currentPay * RATE2;
29     //end if
30     newPay = currentPay + raise;
31
32     //display new pay
33     cout << fixed << setprecision(2);
34     cout << "New pay: $" << newPay << endl;
35     return 0;
36 }    //end of main function
```

Figure 5-40

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

CHAPTER 6

More on the Selection Structure

After studying Chapter 6, you should be able to:

- Include a nested selection structure in pseudocode and in a flowchart
- Code a nested selection structure
- Recognize common logic errors in selection structures
- Include a multiple-alternative selection structure in pseudocode and in a flowchart
- Code a multiple-alternative selection structure in C++

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Nested Selection Structures

Both paths in a selection structure can include instructions that declare variables, perform calculations, and so on. Both paths can also include other selection structures. When either a selection structure's true path or its false path contains another selection structure, the inner structure is referred to as a **nested selection structure** because it is contained (nested) entirely within the outer structure.

Ch06-Maleek

A programmer determines whether a problem's solution requires a nested selection structure by studying the problem specification. The first problem specification you will examine in this chapter involves a basketball player named Maleek. The problem specification and an illustration of the problem are shown in Figure 6-1, along with an appropriate algorithm. The algorithm requires only one selection structure because only one decision—whether the basketball went through the hoop—is necessary.

Problem specification and algorithm

Maleek is practicing for an upcoming basketball game. Write the instructions that direct him to shoot the basketball and then say one of two phrases, depending on whether or not the basketball went through the hoop.

Result of shot	Phrase
Basketball went through the hoop	I did it!
Basketball did not go through the hoop	Missed it!

```
1. shoot the basketball
2. if (the basketball went through the hoop)
       say "I did it!"
   else
       say "Missed it!"
   end if
```

condition

true path

false path

Figure 6-1 A problem that requires the selection structure
Image by Diane Zak; created with Reallusion CrazyTalk Animator

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Now we'll make a slight change to the problem specification. This time, Maleek should say either one or two phrases, depending not only on whether or not the ball went through the hoop but also on where he was standing when he made the basket. Figure 6-2 shows the modified problem specification and algorithm. The modified algorithm contains an outer dual-alternative selection structure and a nested dual-alternative selection structure. The outer structure begins with *if (the basketball went through the hoop)*, and it ends with the last *end if*. The last *else* belongs to the outer structure and separates the structure's true path from its false path. Notice that the instructions in both paths are indented within the outer selection structure. Indenting in this manner clearly indicates the instructions to be followed when the basketball went through the hoop, as well as the ones to be followed when the basketball did not go through the hoop.

The nested selection structure in Figure 6-2 appears in the outer structure's true path. The nested structure begins with *if (Maleek was either inside or on the 3-point line)*, and it ends with the first *end if*. The indented *else* belongs to the nested structure and separates the nested structure's true path from its false path. For clarity, the instructions in the nested structure's true and false paths are indented within the structure. For a nested structure to work correctly, it must be contained entirely within either the outer structure's true path or its false path. In Figure 6-2, the nested selection structure appears entirely within the outer selection structure's true path.

Problem specification and algorithm

Maleek is practicing for an upcoming basketball game. Write the instructions that direct him to shoot the basketball and then say either one or two of four phrases, depending on whether or not the basketball went through the hoop and also where he was standing when he made the basket.

Result of shot	Phrase
Basketball went through the hoop	I did it!
Maleek made the basket from either inside or on the 3-point line	2 points for me
Maleek made the basket from behind the 3-point line	3 points for me
Basketball did not go through the hoop	Missed it!

```
1. shoot the basketball
2. if (the basketball went through the hoop)
       say "I did it!"
       if (Maleek was either inside or on the 3-point line)
              say "2 points for me"
       else
              say "3 points for me"
       end if
   else
        say "Missed it!"
   end if
```

nested dual-alternative selection structure

outer dual-alternative selection structure

Figure 6-2 A problem that requires a nested selection structure

Figure 6-3 shows a modified version of the previous problem specification, along with the modified algorithm. In this version of the problem, Maleek should still say "Missed it!" when the basketball misses its target. However, if the basketball hits the rim, he should also say "So close". In addition to the nested dual-alternative selection structure from the previous algorithm, the modified algorithm also contains a nested single-alternative selection

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

structure, which appears in the outer structure's false path. The nested structure begins with *if (the basketball hit the rim)*, and it ends with the second *end if*. In this case, the nested structure is contained entirely within the outer structure's false path.

Problem specification and algorithm

Maleek is practicing for an upcoming basketball game. Write the instructions that direct him to shoot the basketball and then say either one or two of five phrases, depending on whether or not the basketball went through the hoop and also where he was standing when he made the basket.

Result of shot	Phrase
Basketball went through the hoop	I did it!
Maleek made the basket from either inside or on the 3-point line	2 points for me
Maleek made the basket from behind the 3-point line	3 points for me
Basketball did not go through the hoop	Missed it!
Maleek's missed shot hit the rim	So close

```
1. shoot the basketball
2. if (the basketball went through the hoop)
       say "I did it!"
       if (Maleek was either inside or on the 3-point line)
           say "2 points for me"
       else
           say "3 points for me"
       end if
   else
       say "Missed it!"
       if (the basketball hit the rim)
           say "So close"
       end if
   end if
```

nested dual-alternative selection structure

nested single-alternative selection structure

outer dual-alternative selection structure

Figure 6-3 A problem that requires two nested selection structures

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 6-1

1. A nested selection structure can appear _____.
   a. only in an outer selection structure's false path
   b. only in an outer selection structure's true path
   c. in either an outer selection structure's false path or its true path

2. Modify the algorithm shown earlier in Figure 6-3 so that the outer structure's condition determines whether the basketball did *not* go through the hoop, and the inner dual-alternative structure's condition determines whether Maleek was *behind* the 3-point line.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

3. Ken would like to use either his credit card or his debit card—but preferably his credit card—to pay for the items he is purchasing from a local department store. However, he is not sure whether the store accepts either card. If the store doesn't accept either card, he will need to pay cash for the items. Write an appropriate algorithm, using only the instructions listed in Figure 6-4. (An instruction can be used more than once.)

```
else
end if
pay for your items using your credit card
pay for your items using your debit card
pay for your items using cash
if (the store accepts your credit card)
if (the store accepts your debit card)
ask the store clerk whether the store accepts your credit card
ask the store clerk whether the store accepts your debit card
```

Figure 6-4 Instructions for Question 3 in Mini-Quiz 6-1

Flowcharting a Nested Selection Structure

Figure 6-5 shows a problem specification for a voter eligibility program. The program determines whether a person can vote and then displays one of three different messages. The appropriate message depends on the person's age and voter registration status. If the person is younger than 18 years old, the program should display the message "You are too young to vote." However, if the person is at least 18 years old, the program should display one of two messages. The correct message to display is determined by the person's voter registration status. If the person is registered, then the appropriate message is "You can vote."; otherwise, it is "You must register before you can vote." Notice that determining the person's registration status is important only *after* his or her age is determined. Because of this, the decision regarding the age is considered the primary decision, while the decision regarding the registration status is considered the secondary decision because whether it needs to be made depends on the result of the primary decision. A primary decision is always made by an outer selection structure, while a secondary decision is always made by a nested selection structure.

Also included in Figure 6-5 is a correct algorithm in flowchart form. The first diamond in the flowchart represents the outer selection structure's condition, which checks whether the age entered by the user is greater than or equal to 18. If the condition evaluates to false, the outer structure's false path displays the "You are too young to vote." message before the outer structure ends. If the outer structure's condition evaluates to true, on the other hand, its true path uses a nested selection structure to determine whether the person is registered. The nested structure's condition is represented by the second diamond in Figure 6-5. If the person is registered, the nested structure's true path displays the "You can vote." message; otherwise, its false path displays the "You must register before you can vote." message. After the appropriate message is displayed, the nested and outer selection structures end. Notice that the nested structure is processed only when the outer structure's condition evaluates to true.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Problem specification and algorithm
The Danville city manager wants a program that determines voter eligibility and displays one of three messages. The messages and the criteria for displaying each message are as follows:

Message	Criteria
You are too young to vote.	person is younger than 18 years old
You can vote.	person is at least 18 years old and is registered to vote
You must register before you can vote.	person is at least 18 years old but is not registered to vote

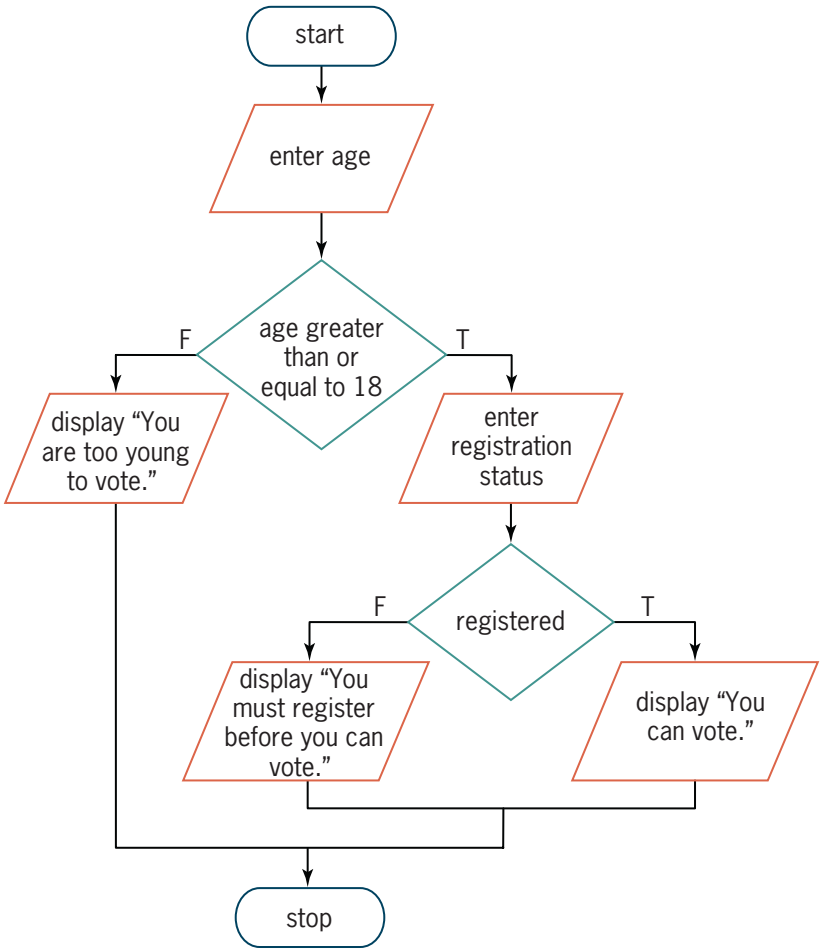


Figure 6-5 Problem specification and a correct algorithm for the voter eligibility problem

Even small problems can have more than one solution. Figure 6-6 shows another correct algorithm for the voter eligibility problem. As in the previous algorithm, the outer selection structure in this algorithm determines the age (the primary decision), and the nested selection structure determines the voter registration status (the secondary decision). In this algorithm, however, the outer structure's condition is the opposite of the one in Figure 6-5: It checks whether the age is less than 18, rather than checking if it is greater than or equal to 18. (Recall that *less than* is the opposite of *greater than or equal to*.) In addition, the nested structure appears in the outer structure's false path in this algorithm, which means it will be processed only when the outer structure's condition evaluates to false. The algorithms in Figures 6-5 and 6-6 produce the same results. Neither algorithm is better than the other; each simply represents a different way of solving the same problem.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Problem specification and algorithm
The Danville city manager wants a program that determines voter eligibility and displays one of three messages. The messages and the criteria for displaying each message are as follows:

Message	Criteria
You are too young to vote.	person is younger than 18 years old
You can vote.	person is at least 18 years old and is registered to vote
You must register before you can vote.	person is at least 18 years old but is not registered to vote

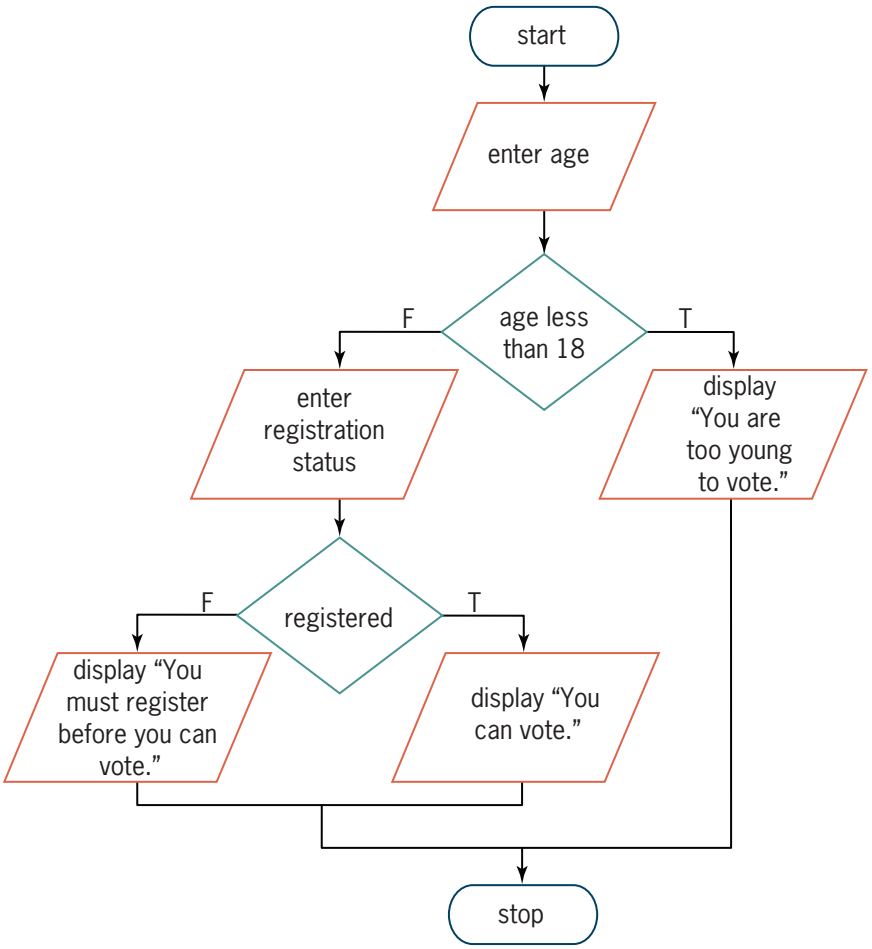


Figure 6-6 Another correct solution for the voter eligibility problem

Coding a Nested Selection Structure

Figure 6-7 shows examples of code that could be used for the voter eligibility program. The first example corresponds to the flowchart in Figure 6-5, and the second example corresponds to the flowchart in Figure 6-6. In the first example, the nested structure is in the outer structure's true path. In the second example, the nested structure is in the outer structure's false path. The figure also includes a sample run of the voter eligibility program.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Example 1: Code for the flowchart in Figure 6-5

```
int age = 0;
char registered = ' ';

//get age
cout << "Age: ";
cin >> age;

if (age >= 18)
{
    //get registration status
    cout << "Registered to vote (Y/N)? ";
    cin >> registered;
    if (toupper(registered) == 'Y')
        cout << "You can vote." << endl;
    else
        cout << "You must register before you can vote." << endl;
    //end if
}
else
    cout << "You are too young to vote." << endl;
//end if
```

nested selection structure

Example 2: Code for the flowchart in Figure 6-6

```
int age = 0;
char registered = ' ';

//get age
cout << "Age: ";
cin >> age;

if (age < 18)
    cout << "You are too young to vote." << endl;
else
{
    //get registration status
    cout << "Registered to vote (Y/N)? ";
    cin >> registered;
    if (toupper(registered) == 'Y')
        cout << "You can vote." << endl;
    else
        cout << "You must register before you can vote." << endl;
    //end if
} //end if
```

nested selection structure

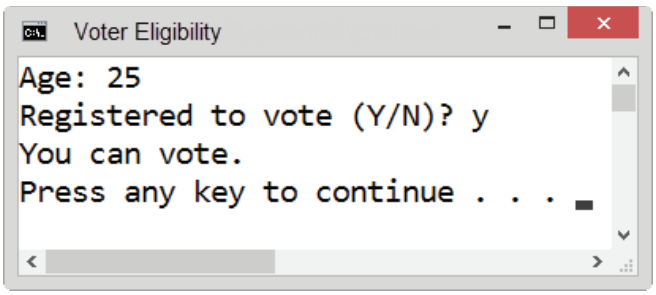


Figure 6-7 Code and a sample run of the voter eligibility program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Mini-Quiz 6-2

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

1. A selection structure should display one of the following messages, depending on a student's test score. Write the pseudocode for the selection structure.

Message	Test score
Great score!	at least 90
Good score	70 through 89
Retake the test	below 70

2. Draw a flowchart of the selection structure from Question 1.
3. Write the C++ code for the selection structure from either Question 1 or Question 2. The test score is stored in an `int` variable named `score`.
4. The manager of a golf course wants a program that displays the appropriate fee to charge a golfer. Club members pay a $5 fee. Nonmembers golfing on Monday through Thursday pay $15. Nonmembers golfing on Friday through Sunday pay $25. The condition in the program's outer selection structure should determine the _____, while the condition in its nested selection structure should determine the _____.
   a. membership status, day of the week
   b. day of the week, membership status
   c. membership status, fee
   d. fee, day of the week

Logic Errors in Selection Structures

In the next few sections, you will observe some of the common logic errors made when writing selection structures. Being aware of these errors will help prevent you from making them. In most cases, logic errors in selection structures are a result of one of the following four mistakes:

1. using a compound condition rather than a nested selection structure
2. reversing the decisions in the outer and nested selection structures
3. using an unnecessary nested selection structure
4. including an unnecessary comparison in a condition

It is easier to understand these four logic errors when viewed in an algorithm. The first three errors will be shown using an algorithm that displays the daily fee for renting a car and the last error using an algorithm that displays an item's price. We will begin with the daily rental fee algorithm. The problem specification and algorithm (written in pseudocode) are shown in Figure 6-8.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Problem specification and algorithm
The daily fee for renting a car from Miller's Car Rental is $55. However, there is an additional charge for renting a luxury car. The additional charge depends on whether the customer belongs to Miller's Rental Club: It is $20 for club members and $30 for nonmembers. Create a program that displays the daily fee for renting a car.

Correct algorithm

```
1. daily fee = 55
2. if (luxury car)
        if (club member)
            add 20 to the daily fee
        else
            add 30 to the daily fee
        end if
   end if
3. display the daily fee
```

Figure 6-8 Problem specification and a correct algorithm for Miller's Car Rental

Notice that the car's classification determines whether the renter is charged an additional amount. If the car is classified as a luxury vehicle, then whether the customer is a club member determines the appropriate additional amount. The decision regarding the car's classification is the primary decision, while the decision regarding the customer's membership status is the secondary decision. Figure 6-9 shows the input values you will use to desk-check the algorithm four times; it also includes the expected output values.

Desk-check	Car classification	Membership status	Expected daily fee
1	standard	member	$55
2	standard	nonmember	$55
3	luxury	member	$75
4	luxury	nonmember	$85

Figure 6-9 Sample data and expected results for the algorithm shown in Figure 6-8

The first set of test data is for a club member renting a standard vehicle. Step 1 in the algorithm assigns $55 as the daily fee. Next, the condition in the outer selection structure determines whether the car is a luxury vehicle; the condition evaluates to false. As a result, the outer structure ends without processing the nested selection structure. This is because the membership information is not important when the car is not a luxury vehicle. The last step in the algorithm displays the expected daily fee of $55.

The second set of test data is for a nonmember renting a standard vehicle. The algorithm begins by assigning $55 as the daily fee. The condition in the outer selection structure determines whether the car is a luxury vehicle. The condition evaluates to false, so the outer selection structure ends. The last step in the algorithm displays the expected daily fee, $55.

The third set of test data is for a club member renting a luxury vehicle. First, the algorithm assigns $55 as the daily fee. Next, the condition in the outer selection structure determines whether the car is a luxury vehicle. In this case, the condition evaluates to true, so the nested selection structure's condition checks whether the customer is a club member. This condition also evaluates to true, so the nested structure's true path adds $20 to the daily fee, giving $75;

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

after doing this, both selection structures end. The last step in the algorithm displays $75 as the daily fee, which is correct.

The last set of test data is for a nonmember renting a luxury vehicle. Step 1 in the algorithm assigns $55 as the daily fee. The condition in the outer selection structure determines whether the car is a luxury vehicle. The condition evaluates to true, so the nested selection structure's condition checks whether the customer is a club member. This condition evaluates to false, so the nested structure's false path adds $30 to the daily fee, giving $85; after doing this, both selection structures end. The last step in the algorithm displays the expected daily fee of $85. The results of desk-checking the algorithm using the data from Figure 6-9 agree with the expected values, as indicated in Figure 6-10.

classification	membership	daily fee	
~~standard~~	~~member~~	~~55~~	(correct result for the first desk-check)
~~standard~~	~~nonmember~~	~~55~~	(correct result for the second desk-check)
~~luxury~~	~~member~~	~~55~~	
		~~75~~	(correct result for the third desk-check)
luxury	nonmember	~~55~~	
		85	(correct result for the fourth desk-check)

Figure 6-10 Result of desk-checking the correct algorithm

First Logic Error: Using a Compound Condition Rather Than a Nested Selection Structure

A common error made when writing selection structures is to use a compound condition in the outer structure when a nested structure is needed. Figure 6-11 shows an example of this error in the car rental algorithm. The correct algorithm is included in the figure for comparison.

Correct algorithm

```
1. daily fee = 55
2. if (luxury car)
       if (club member)
           add 20 to the daily fee
       else
           add 30 to the daily fee
       end if
   end if
3. display the daily fee
```

Incorrect algorithm

```
1. daily fee = 55
2. if (luxury car and club member)
       add 20 to the daily fee
   else
       add 30 to the daily fee
   end if
3. display the daily fee
```

uses a compound condition instead of a nested selection structure

Figure 6-11 Correct algorithm and an incorrect algorithm containing the first logic error

Notice that the incorrect algorithm uses one selection structure rather than two selection structures and that the selection structure contains a compound condition. Consider why the selection structure in the incorrect algorithm cannot be used in place of the selection structures in the correct one. In the correct algorithm, the outer and nested structures indicate that a hierarchy exists between the car classification and membership status decisions: The car classification decision is always made first, followed by the membership status decision (if necessary). In the incorrect algorithm, the compound condition indicates that no hierarchy exists between the classification and membership decisions.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

To understand why the incorrect algorithm in Figure 6-11 will not work correctly, you will desk-check it using the same test data used to desk-check the correct algorithm. Step 1 in the incorrect algorithm assigns $55 as the daily fee. Next, the compound condition in Step 2 determines whether the car is classified as a luxury vehicle and, at the same time, the renter is a club member. Using the first set of test data (standard and member), the compound condition evaluates to false because the car is not a luxury vehicle. As a result, the selection structure's false path adds $30 to the daily fee, giving $85, and then the selection structure ends. The last step in the incorrect algorithm displays $85 as the daily fee, which is not correct; the correct fee is $55, as shown earlier in Figure 6-10.

Now we'll desk-check the incorrect algorithm using the second set of test data: standard and nonmember. The algorithm begins by assigning $55 as the daily fee. The compound condition then determines whether the car is a luxury vehicle and, at the same time, the renter is a club member. The compound condition evaluates to false, so the selection structure's false path adds $30 to the daily fee, giving $85, and then the selection structure ends. The last step in the incorrect algorithm displays $85 as the daily fee, which is not correct; the correct fee is $55.

Next, we'll desk-check the incorrect algorithm using the third set of test data: luxury and member. First, the algorithm assigns $55 as the daily fee. The compound condition then determines whether the car is a luxury vehicle and, at the same time, the renter is a club member. In this case, the compound condition evaluates to true, so the selection structure's true path adds $20 to the daily fee, giving $75, and then the selection structure ends. The last step in the incorrect algorithm displays the expected daily fee, $75. Even though its selection structure is phrased incorrectly, the incorrect algorithm produces the same result as the correct algorithm using the third set of test data.

Finally, we'll desk-check the incorrect algorithm using the fourth set of test data: luxury and nonmember. Step 1 assigns $55 as the daily fee. Next, the compound condition determines whether the car is a luxury vehicle and, at the same time, the renter is a club member. The compound condition evaluates to false because the renter is not a club member. Therefore, the selection structure's false path adds $30 to the daily fee, giving $85, and then the selection structure ends. The last step in the incorrect algorithm displays $85 as the daily fee, which is correct. Here, too, even though its selection structure is phrased incorrectly, the incorrect algorithm produces the same result as the correct algorithm using the fourth set of test data.

Figure 6-12 shows the desk-check table for the incorrect algorithm from Figure 6-11. As indicated in the figure, only the results of the third and fourth desk-checks are correct.

classification	membership	daily fee	
~~standard~~	~~member~~	~~55~~	
		~~85~~	(incorrect result for the first desk-check)
~~standard~~	~~nonmember~~	~~55~~	
		~~85~~	(incorrect result for the second desk-check)
~~luxury~~	~~member~~	~~55~~	
		~~75~~	(correct result for the third desk-check)
luxury	nonmember	~~55~~	
		85	(correct result for the fourth desk-check)

Figure 6-12 Results of desk-checking the incorrect algorithm from Figure 6-11

The importance of desk-checking an algorithm several times using different data cannot be emphasized enough. In this case, if you had used only the last two sets of data to desk-check the incorrect algorithm, you would not have discovered that the algorithm did not work as intended.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Second Logic Error: Reversing the Outer and Nested Decisions

Another common error made when writing selection structures is to reverse the decisions made by the outer and nested structures. Figure 6-13 shows an example of this error in the car rental algorithm. The correct algorithm is included in the figure for comparison.

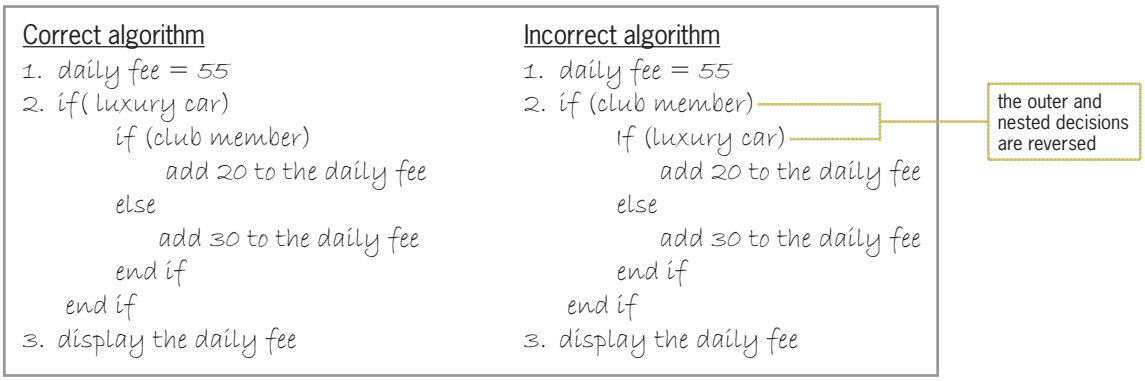


Figure 6-13 Correct algorithm and an incorrect algorithm containing the second logic error

Unlike the selection structures in the correct algorithm, which determine the car classification before determining the membership status, the selection structures in the incorrect algorithm determine the membership status before determining the car classification. Consider how this difference changes the algorithm. In the correct algorithm, the selection structures indicate that only renters of luxury cars pay an additional amount. The selection structures in the incorrect algorithm, on the other hand, indicate that the additional amount is paid by club members only. Figure 6-14 shows the result of desk-checking the incorrect algorithm from Figure 6-13. As indicated in the figure, only two of the four results are correct.

classification	membership	daily fee	
~~standard~~	~~member~~	~~55~~	
		~~85~~	(incorrect result for the first desk-check)
~~standard~~	~~nonmember~~	~~55~~	(correct result for the second desk-check)
~~luxury~~	~~member~~	~~55~~	
		~~75~~	(correct result for the third desk-check)
luxury	nonmember	55	(incorrect result for the fourth desk-check)

Figure 6-14 Results of desk-checking the incorrect algorithm from Figure 6-13

Third Logic Error: Using an Unnecessary Nested Selection Structure

Another common error made when writing selection structures is to include an unnecessary nested selection structure. In most cases, a selection structure containing this error will still produce the correct results. However, it will do so less efficiently than selection structures that are properly structured. Figure 6-15 shows an example of this error in the car rental algorithm. The correct algorithm is included in the figure for comparison.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

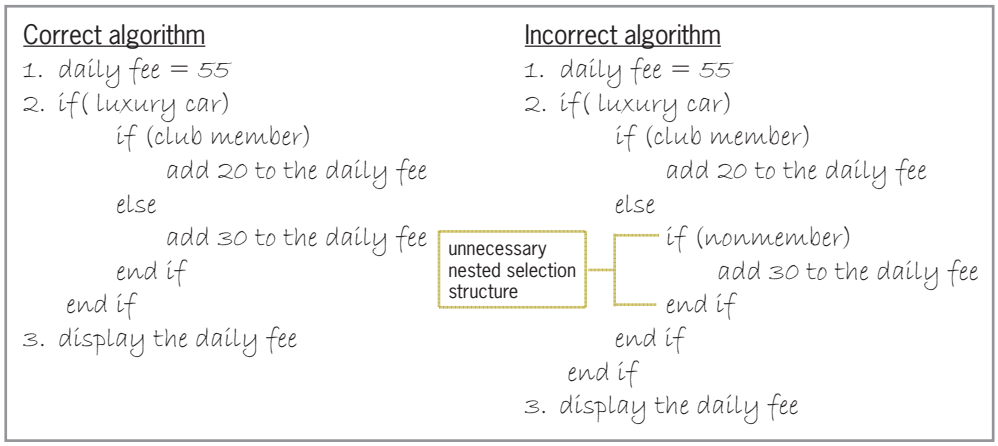


Figure 6-15 Correct algorithm and an inefficient algorithm containing the third logic error

Unlike the correct algorithm, which contains two selection structures, the inefficient algorithm contains three selection structures. The condition in the third structure determines whether the renter is *not* a member of the rental club and is processed only when the second structure's condition evaluates to false. In other words, it is processed only when the algorithm has already determined that the renter is *not* a club member. Therefore, the third selection structure is unnecessary. Figure 6-16 shows the results of desk-checking the inefficient algorithm. Although the results of the four desk-checks are correct, the result of the last desk-check is obtained in a less efficient manner.

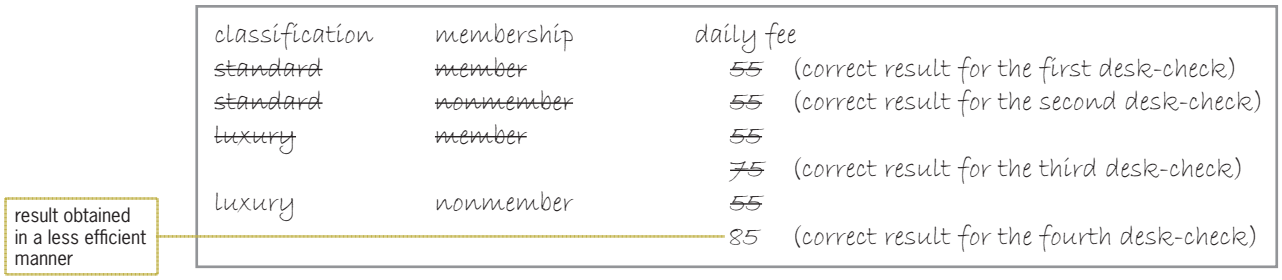


Figure 6-16 Results of desk-checking the inefficient algorithm from Figure 6-15

Fourth Logic Error: Including an Unnecessary Comparison in a Condition

Another common error made when writing selection structures is to include an unnecessary comparison in a condition. Like selection structures containing the third logic error, selection structures containing this error also produce the correct results but sometimes in an inefficient way. We'll demonstrate this error using a procedure that displays an item's price, which is based on the quantity purchased. Figure 6-17 shows the problem specification, a correct algorithm, and an inefficient algorithm that contains the fourth logic error.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Problem specification
A program needs to display an item's price. The price depends on the quantity purchased, as shown here:

Quantity purchased	Price per item ($)
Less than or equal to 0	0.0
1–99	9.50
Over 99	7.75

Correct algorithm

```
1. if (quantity <= 0)
       price = 0
   else
       if (quantity < 100)
           price = 9.50
       else
           price = 7.75
       end if
   end if
2. display the price
```

Inefficient algorithm

```
1. if (quantity <= 0)
       price = 0
   else
       if (quantity > 0 and quantity < 100)
           price = 9.50
       else
           price = 7.75
       end if
   end if
2. display the price
```

unnecessary comparison

In the correct algorithm, the nested selection structure's `if` clause can also be written as `if (quantity >= 100)`, as long as you then reverse the instructions in the two paths.

Figure 6-17 Problem specification, a correct algorithm, and an inefficient algorithm

Unlike the nested structure in the correct algorithm, the nested structure in the inefficient algorithm contains a compound condition that compares the quantity to both 0 and 100. Consider why the comparison to 0 in the compound condition is unnecessary. If the quantity *is* less than or equal to 0, the outer structure's condition will evaluate to true. As a result, the outer structure's true path will assign the number 0 as the price before the outer structure ends. In other words, a quantity that is either less than or equal to 0 will be handled by the outer structure's true path. The nested structure's condition will be evaluated only when the quantity is greater than 0. Therefore, the comparison to 0 is unnecessary in the compound condition. Figure 6-18 shows the results of desk-checking the correct and inefficient algorithms. Although the results of the three desk-checks for the inefficient algorithm are correct, the results of the second and third desk-checks are obtained in a less efficient manner.

Correct algorithm's desk-check

quantity	price	
~~0~~	~~0~~	(correct result for the first desk-check)
~~83~~	~~9.50~~	(correct result for the second desk-check)
105	~~7.75~~	(correct result for the third desk-check)

Inefficient algorithm's desk-check

quantity	price	
~~0~~	~~0~~	(correct result for the first desk-check)
~~83~~	~~9.50~~	(correct result for the second desk-check)
105	~~7.75~~	(correct result for the third desk-check)

results obtained in a less efficient manner

Figure 6-18 Results of desk-checking the algorithms from Figure 6-17

For more experience with problems containing nested selection structures, see the Nested Selection Structures section in the Ch06WantMore.pdf file.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 6-3

1. List the four errors commonly made when writing selection structures.
2. What is wrong with Algorithm A shown in Figure 6-19?
3. Does Algorithm B in Figure 6-19 give you the same results as the algorithm shown earlier in Figure 6-2? If not, why not?
4. Does Algorithm C in Figure 6-19 give you the same results as the algorithm shown earlier in Figure 6-2? If not, why not?

Algorithm A

```
1. shoot the basketball
2. if (the basketball went through the hoop)
        say "I did it!"
   else
        if (the basketball did not go through the hoop)
                say "Missed it!"
        end if
   end if
```

Algorithm B

```
1. shoot the basketball
2. if (the basketball went through the hoop and Maleek was inside the 3-point line)
        say "I did it!"
        say "2 points for me"
   else
         if (Maleek was behind the 3-point line)
                say "I did it!"
                say "3 points for me"
        else
                say "Missed it!"
        end if
   end if
```

Algorithm C

```
1. shoot the basketball
2. if (the basketball did not go through the hoop)
        say "Missed it!"
   else
        say "I did it!"
        if (Maleek was either inside or on the 3-point line)
                say "2 points for me"
        else
                say "3 points for me"
        end if
   end if
```

Figure 6-19 Algorithm for Questions 2 through 4 in Mini-Quiz 6-3

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Multiple-Alternative Selection Structures

Figure 6-20 shows the problem specification and IPO chart for the Snowboard Shop problem. The problem's solution requires a selection structure that can choose from several different item codes. Each code corresponds to the location of a Snowboard Shop warehouse, as shown in the figure. Selection structures containing several alternatives are referred to as **multiple-alternative selection structures** or **extended selection structures**.

Problem specification

Each item the Snowboard Shop sells has a code that identifies the location of the warehouse in which it is stored. The valid codes and the location of their corresponding warehouses are shown here. The shop's owner wants a program that displays the location, given the item code. If the item code is not listed here, the program should display the "Invalid code" message.

Item code	Warehouse location
12	Tennessee
36	Kentucky
40, 43	Louisiana

Input	Processing	Output
code	Processing items: none	location or message

```
Algorithm:
1. enter the code
2. if (the code is one of the following:)
        12          display "Tennessee" as the location
        36          display "Kentucky" as the location
        40 or 43    display "Louisiana" as the location
   else
        display "Invalid code" message
   end if
```

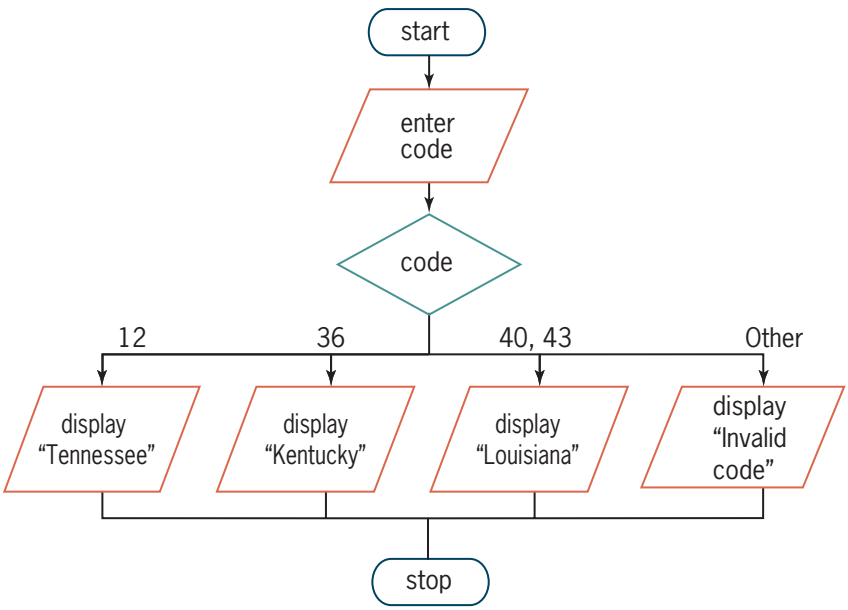


Figure 6-20 Problem specification and IPO chart for the Snowboard Shop problem

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The diamond in the flowchart represents the condition in the multiple-alternative selection structure. Recall that the diamond is also used to represent the condition in both the single-alternative and dual-alternative selection structures. However, unlike the diamond in both of those selection structures, the diamond in a multiple-alternative selection structure has several flowlines (rather than only two flowlines) leading out of the symbol. Each flowline represents a possible path and must be marked appropriately, indicating the value or values necessary for the path to be chosen.

Figure 6-21 shows two ways of coding the multiple-alternative selection structure from Figure 6-20; both versions use If...Then...Else statements. Although both versions produce the same result, Version 2 provides a more convenient way of coding a multiple-alternative selection structure. (In both versions, `code` is an `int` variable that gets its value from the user at the keyboard.) The figure also includes a sample run of the program.

Version 1

```
if (code == 12)
    cout << "Tennessee";
else
    if (code == 36)
        cout << "Kentucky";
    else
        if (code == 40 || code == 43)
            cout << "Louisiana";
        else
            cout << "Invalid code";
        //end if
    //end if
//end if
```

you get here when the code is not 12

you get here when the code is not 12 and not 36

you get here when the code is not 12, 36, 40, or 43

three End If clauses are required

Version 2

```
if (code == 12)
    cout << "Tennessee";
else if (code == 36)
    cout << "Kentucky";
else if (code == 40 || code == 43)
    cout << "Louisiana";
else  //default
    cout << "Invalid code";
//end if
```

you can use one comment to mark the end of the entire structure

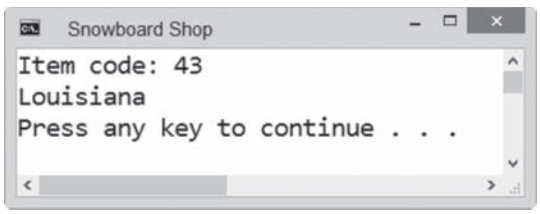


Figure 6-21 Two ways of coding the multiple-alternative selection structure from Figure 6-20

The `switch` Statement

Instead of using the `if` statement to code a multiple-alternative selection structure, you sometimes (but not always) can use the **`switch` statement**. Figure 6-22 shows the `switch` statement's syntax and includes an example of using the statement in place of the `if` statements from Figure 6-21.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

HOW TO Use the `switch` Statement

Syntax

switch (*selectorExpression***)**
{
case *value1:*
    one or more statements
    [break;]
[case *value2:*
    one or more statements
    [break;]]
[case *valueN:*
    one or more statements
    [break;]]
[default:
    one or more statements to be processed when the selectorExpression does not match any of the values in the `case` *clauses*
    [break;]]
} `//end switch`

Example

selectorExpression

```
switch (code)
{
case 12:
     cout << "Tennessee";
     break;
case 36:
     cout << "Kentucky";
     break;
case 40:
case 43:
     cout << "Louisiana";
     break;
default:
     cout << "Invalid code";
}   //end switch
```

Figure 6-22 How to use the `switch` statement

The `switch` statement begins with the `switch` clause, which contains a *selectorExpression* enclosed in parentheses. The *selectorExpression* can contain any combination of variables, constants, functions, and operators; however, the combination must result in a value whose data type is `bool`, `char`, `short`, `int`, or `long`. In the example in Figure 6-22, the *selectorExpression* is an `int` variable named `code`.

Following the `switch` clause is a statement block. Recall from Chapter 5 that a statement block is one or more statements enclosed in a set of braces. Between the `switch` statement's opening and closing braces are one or more `case` clauses. Each `case` clause represents a different alternative.

Notice that each `case` clause contains a value followed by a colon. The value can be a literal constant, a named constant, or an expression composed of literal and named constants. The

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

data type of the value should be compatible with the data type of the *selectorExpression*. When the *selectorExpression* is numeric, the values in the `case` clauses should be numeric. Likewise, when the *selectorExpression* is a character, the values should be characters. In the example in Figure 6-22, the *selectorExpression* has the `int` data type, and so do the values in the `case` clauses. Following the colon in each `case` clause are one or more statements that are processed when the *selectorExpression* matches that `case`'s value.

The `switch` statement is often used in programs that display a menu of choices for the user.

Notice that the last statement in each `case` clause is `break;`. The **break statement** tells the computer to exit ("break out of") the `switch` statement at that point. After processing the `break` statement, the computer processes the instruction that follows the `switch` statement's closing brace. If a `case` clause does not contain a `break` statement, the computer will process that `case` clause's instructions and then continue processing the remaining instructions in the `switch` statement; this may or may not be what you intended.

In addition to the `case` clauses, you can also include one `default` clause in a `switch` statement. Although the `default` clause can appear anywhere within the `switch` statement, it usually is entered as the last clause in the statement, as shown in Figure 6-22. When it is in that position, it does not need a `break` statement; however, some programmers include the `break` statement for clarity. If the `default` clause is not the last clause, a `break` statement is required in order to stop the computer from processing the instructions in the next `case` clause. In Computer Exercise 20, you will observe the result of not using the `break` statement to break out of the `switch` statement.

The `switch` statement sounds more difficult than it really is. When processing the statement, the computer simply compares the value of the *selectorExpression* with the value listed in each of the `case` clauses, one `case` clause at a time beginning with the first. If a match is found, the computer processes the instructions contained in that `case` clause, stopping only when it encounters either a `break` statement or the `switch` statement's closing brace; the computer then skips to the instruction following the closing brace. If a match is *not* found, the next instruction processed depends on whether the `switch` statement contains a `default` clause. If there *is* a `default` clause, the computer processes the instructions in that clause, stopping only when it encounters either a `break` statement or the `switch` statement's closing brace; the computer then skips to the instruction following the closing brace. If there *isn't* a `default` clause, the computer just skips to the instruction following the closing brace.

Desk-checking the code in Figure 6-22 will help you understand how the `switch` statement is processed by the computer. You will desk-check the code using the following three item codes: 12, 40, and 7. The `switch (code)` clause tells the computer to compare the number in the `code` variable (12) with the number listed in the first `case` clause (12). Both numbers match, so the `cout` statement displays "Tennessee" on the screen. The next statement, `break;`, tells the computer to skip the remaining instructions in the `switch` statement and continue processing with the instruction that follows the `switch` statement's closing brace.

Now use the number 40 to desk-check the code. When processing the `switch (code)` clause, the computer compares the number in the `code` variable (40) with the number listed in the first `case` clause (12). The numbers do not match, so the computer compares the number in the `code` variable (40) with the number listed in the second `case` clause (36). Here again, the numbers do not match, so the computer compares the number in the `code` variable (40) with the number listed in the third `case` clause (40). This time, the computer finds a match. However, notice that there is no statement immediately below the `case 40:` clause. So, what (if anything) will appear when the code is 40?

Recall that when the value of the *selectorExpression* matches the value in a `case` clause, the computer processes the instructions contained in that clause until it encounters either a `break` statement or the `switch` statement's closing brace. In this instance, not finding any instructions

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

in the `case 40:` clause, the computer continues processing with the instructions in the next clause, which is the `case 43:` clause. The instructions in that `case` clause display the correct warehouse location (Louisiana) and then exit the `switch` statement. As this example shows, you can process the same instructions for more than one value by listing each value in a separate `case` clause, as long as the clauses appear together in the `switch` statement. The last `case` clause in the group of related clauses should contain the instructions you want the computer to process when one of the values in the group matches the *selectorExpression*. Only the last `case` clause in the group of related clauses should contain the `break` statement.

Finally, you will desk-check the code in Figure 6-22 using the number 7. When processing the `switch` statement, the computer compares the value stored in the `code` variable (7) with the value listed in each of the `case` clauses, one `case` clause at a time beginning with the first. The number 7 does not appear as a value in any of the `case` clauses, so the computer processes the instruction in the `default` clause. That instruction displays the message "Invalid code" on the screen. The computer then skips to the instruction following the `switch` statement's closing brace.

For more experience with problems containing multiple-alternative selection structures, see the Multiple Alternative Selection Structures section in the Ch06WantMore.pdf file.

Mini-Quiz 6-4

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

1. A selection structure should display one of the following messages, depending on a student's test score. Write the C++ code using the shorter form of the `if` statement.

Message	Test score
Great score!	at least 90
Good score	70 through 89
Retake the test	0 through 69
Invalid test score	less than 0

2. If a `switch` statement's *selectorExpression* is a `char` variable named `grade`, which of the following `case` clauses will be processed when the `grade` variable contains the letter B?

   a. `case "B":`
   b. `case 'B':`
   c. `case = 'B':`
   d. `case == 'B':`

3. The _____ statement tells the computer to exit the `switch` statement at that point.

LAB 6-1 Stop and Analyze

Study the flowchart shown in Figure 6-23, and then answer the questions.

The answers to the labs are contained in the Answers.pdf file.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

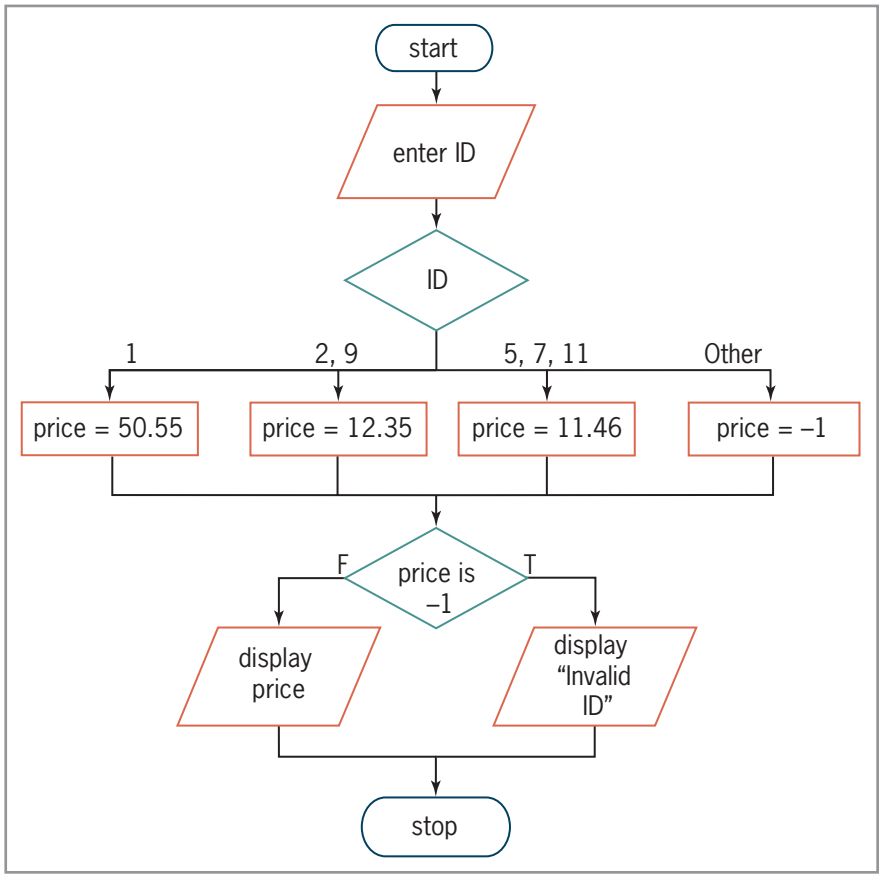


Figure 6-23 Flowchart for Lab 6-1

QUESTIONS

1. What will the program display when the ID is the number 11?
2. How can you write the multiple-alternative selection structure using the longer form of the `if` statement?
3. How can you write the multiple-alternative selection structure using the shorter form of the `if` statement?
4. How can you write the multiple-alternative selection structure using the `switch` statement?
5. What changes would you need to make to the code from Question 4 so that each `case` clause displays the appropriate price and the `default` clause displays the "Invalid ID" message?

LAB 6-2 Plan and Create

In this lab, you will plan and create an algorithm for Sophia's Pizzeria. The problem specification is shown in Figure 6-24.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Problem specification
Sophia's Pizzeria sells two sizes of pizzas: medium and large. A medium pizza is $9.99, and a large pizza is $12.99. Periodically, Sophia e-mails customers a $2 coupon on the purchase of a large pizza only. She wants a program that displays the price of a pizza, given the size of the pizza ordered and whether or not the customer has a $2 coupon.

Figure 6-24 Problem specification for Lab 6-2

First, analyze the problem, looking first for the output and then for the input. In this case, the user wants the program to display the price of a pizza. To calculate the price, the computer will need to know the size of the pizza and whether or not the customer has a $2 coupon.

Next, plan the algorithm. Recall that most algorithms begin with an instruction to enter the input items into the computer, followed by instructions that process the input items, typically including the items in one or more calculations. Most algorithms end with one or more instructions that display, print, or store the output items. Figure 6-25 shows the completed IPO chart for Sophia's Pizzeria.

Input	Processing	Output

size (M or L)
coupon status (Y or N)
	Processing items: none
	
	Algorithm:
	1. enter the size
	2. if (the size is not M or L)
	display "Please enter either M or L."
	else
	if (the size is M)
	price = 9.99
	else
	price = 12.99
	enter coupon status
	if (coupon status is Y)
	price = price – 2
	end if
	end if
	display price
	end if
		price

Figure 6-25 Completed IPO chart for Lab 6-2

After completing the IPO chart, you then move on to the third step in the problem-solving process, which is to desk-check the algorithm. Figure 6-26 shows the test data and completed desk-check table.

Desk-check	Pizza size	Coupon status	Price or message
1	M	Not applicable	9.99
2	L	N	12.99
3	L	Y	10.99
4	X	Not applicable	Please enter either M or L.

Figure 6-26 Test data and completed desk-check table for Lab 6-2's algorithm *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

size	coupon status	price
~~M~~		~~9.99~~
~~L~~	~~N~~	~~12.99~~
~~L~~	Y	10.99
X		

Figure 6-26 Test data and completed desk-check table for Lab 6-2's algorithm

The fourth step in the problem-solving process is to code the algorithm into a program. You begin by declaring memory locations that will store the values of the input, processing (if any), and output items. The Sophia's Pizzeria program will need three memory locations to store the input and output items. The input items (size and coupon status) will be stored in variables because the user should be allowed to change their values during runtime. The output item (price) will also be stored in a variable; this is because its value will change based on the current values of the input items. You will use `char` variables for the letters corresponding to the size and coupon information and use a `double` variable for the price. Figure 6-27 shows the IPO chart information and corresponding C++ instructions.

IPO chart information

Input

size (M or L)
coupon status (Y or N)

Processing

none

Output

price

Algorithm:

```
1. enter the size

2. if (the size is not M or L)
       display "Please enter either
       M or L."
   else

       if (the size is M)
           price = 9.99
       else
           price = 12.99
           enter coupon status

           if (coupon status is Y)
               price = price - 2
           end if
       end if
       display price
   end if
```

C++ instructions

```
char size = ' ';
char coupon = ' ';

double price = 0.0;

cout << "M(edium) or L(arge) pizza? ";
cin >> size;
size = toupper(size);

if (size != 'M' && size != 'L')
    cout << "Please enter either M or L."
    << endl;
else
{

    if (size == 'M')
        price = 9.99;
    else
    {
        price = 12.99;
        cout << "$2 coupon (Y/N)? ";
        cin >> coupon;
        if (toupper(coupon) == 'Y')
            price -= 2;
        //end if
    } //end if
    cout << "Price: $" << price << endl;
} //end if
```

Figure 6-27 IPO chart information and C++ instructions for Lab 6-2

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The fifth step in the problem-solving process is to desk-check the program. You begin by placing the names of the declared variables and named constants (if any) in a new desk-check table, along with their initial values. You then desk-check the remaining C++ instructions in order, recording in the desk-check table any changes made to the variables. Figure 6-28 shows the completed desk-check table. (The two `char` variables are initialized to a space.) The results agree with those shown in the algorithm's desk-check table in Figure 6-26.

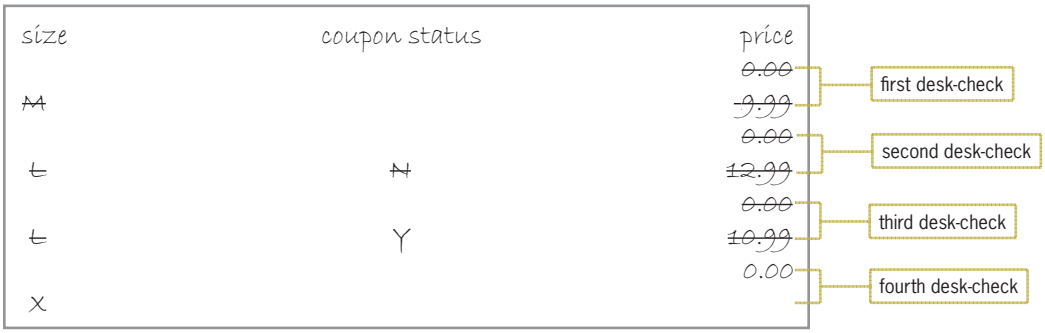


Figure 6-28 Completed desk-check table for Lab 6-2's program

The final step in the problem-solving process is to evaluate and modify (if necessary) the program. Recall that you evaluate a program by entering its instructions into the computer, and then using the computer to run (execute) it. While the program is running, you enter the same sample data used when desk-checking the program.

DIRECTIONS

1. Follow the instructions for starting your C++ development tool. Depending on the development tool you are using, you may need to create a new project; if so, name the project Lab6-2 Project, and save it in the Cpp8\Chap06 folder. Enter the instructions shown in Figure 6-29 in a source file named Lab6-2.cpp. (Do not enter the line numbers.) Save the file in either the project folder or the Cpp8\Chap06 folder. Now follow the appropriate instructions for running the Lab6-2.cpp file. Use the sample data provided in this lab to test the program. If necessary, correct any bugs (errors) in the program.

```
 1 //Lab6-2.cpp - displays the price of a pizza
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <iomanip>
 6 using namespace std;
 7
 8 int main()
 9 {
10    char size = ' ';
11    char coupon = ' ';
12    double price = 0.0;
13
14    cout << "M(edium) or L(arge) pizza? ";
15    cin >> size;
16    size = toupper(size);
17
```

Figure 6-29 Sophia's Pizzeria program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
18    if (size != 'M' && size != 'L')
19        cout << "Please enter either M or L." << endl;
20    else
21    {
22        if (size == 'M')
23            price = 9.99;
24        else
25        {   //large pizza
26            price = 12.99;
27            cout << "$2 coupon (Y/N)? ";
28            cin >> coupon;
29            if (toupper(coupon) == 'Y')
30                price -= 2;
31            //end if
32        } //end if
33        cout << fixed << setprecision(2);
34        cout << "Price: $" << price << endl;
35    } //end if
36
37    return 0;
38 }  //end of main function
```

Figure 6-29 Sophia's Pizzeria program

LAB 6-3 Modify

If necessary, create a new project named Lab6-3 Project and save it in the Cpp8\Chap06 folder. Enter (or copy) the Lab6-2.cpp instructions into a new source file named Lab6-3.cpp. Change Lab6-2.cpp in the first comment to Lab6-3.cpp. Customers can now use the $2 coupon toward the purchase of any size pizza. Modify the program appropriately. Test the program five times using the following test data: M and N, M and Y, L and N, L and Y, and X.

LAB 6-4 What's Missing?

The program in this lab should display the price of a movie ticket. The price is based on the customer's age, as shown in Figure 6-30. If the user enters a negative number, the program should display the "Invalid age" message. Start your C++ development tool, and view the Lab6-4.cpp file, which is contained in either the Cpp8\Chap06\Lab6-4 Project folder or the Cpp8\Chap06 folder. (Depending on your C++ development tool, you may need to open Lab6-4's project/solution file first.) Put the C++ instructions in the proper order, and then determine the one or more missing instructions. Test the program seven times using the following data: 1, 3, 4, 64, 65, 70, and –3.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Age (years)	Price ($)
3 and younger	0
4 to 64	9
65 and older	6

Figure 6-30 Ticket information for Lab 6-4

LAB 6-5 Desk-Check

Desk-check the code shown in Figure 6-31 three times, using the numbers 2, 5, and 100.

```
int number = 0;
cout << "Enter a number: ";
cin >> number;

switch (number)
{
     case 1:
     case 2:
     case 3:
          number = number * 2;
          break;
     case 4:
     case 5:
          number = number + 5;
          break;
     default:
          number = number - 50;
}    //end switch

//display number
cout << "Final number: " << number << endl;
```

Figure 6-31 Code for Lab 6-5

LAB 6-6 Debug

Follow the instructions for starting C++ and viewing the Lab6-6.cpp file, which is contained in either the Cpp8\Chap06\Lab6-6 Project folder or the Cpp8\Chap06 folder. (Depending on your C++ development tool, you may need to open Lab6-6's solution/project file first.) Run the program. Test the program using the following codes: 1, 2, 3, 4, 5, 9, and –3. Debug the program.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Chapter Summary

You can nest a selection structure within either the true or false path of another selection structure.

Logic errors commonly made when writing selection structures usually are a result of one of the following four mistakes: using a compound condition rather than a nested selection structure, reversing the decisions in the outer and nested selection structures, using an unnecessary nested selection structure, or including an unnecessary comparison in a condition.

Some solutions require selection structures that can choose from several alternatives. The selection structures are commonly referred to as multiple-alternative selection structures or extended selection structures. You can code these selection structures using the multiple-alternative form of the `if` statement. You can also use the `switch` statement, as long as the statement's *selectorExpression* evaluates to a value whose data type is `bool`, `char`, `short`, `int`, or `long`.

In a flowchart, a diamond is used to represent the condition in a multiple-alternative selection structure. The diamond has one flowline leading into the symbol and several flowlines leading out of the symbol. Each flowline leading out of the diamond represents a possible path and must be marked to indicate the value or values necessary for the path to be chosen.

In a `switch` statement, the data type of the value in each `case` clause should be compatible with the data type of the statement's *selectorExpression*. The *selectorExpression* must evaluate to a value whose data type is `bool`, `char`, `short`, `int`, or `long`.

Most `case` clauses in a `switch` statement contain a `break` statement, which tells the computer to exit the `switch` statement at that point.

Key Terms

`break` statement—a C++ statement used to tell the computer to exit a `switch` statement

Extended selection structures—another name for multiple-alternative selection structures

Multiple-alternative selection structures—selection structures that contain several alternatives; also called extended selection structures; can be coded using either the multiple-alternative form of the `if` statement or the `switch` statement

Nested selection structure—a selection structure that is wholly contained (nested) within either the true or false path of another selection structure

`switch` statement—a C++ statement that can be used to code some (but not all) multiple-alternative selection structures

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Review Questions

Use the code shown in Figure 6-32 to answer Review Questions 1 through 3.

```
int number = 0;
cout << "Number: ";
cin >> number;

if (number <= 100)
     number *= 2;
else
     if (number < 500)
          number *= 3;
     else
          number = 100;
     //end if
//end if
```

Figure 6-32

1. If the user enters the number 90, what value will be in the `number` variable after the selection structure in Figure 6-32 is processed?

   a. 0
   b. 100
   c. 180
   d. 270

2. If the user enters the number 1000, what value will be in the `number` variable after the selection structure in Figure 6-32 is processed?

   a. 0
   b. 100
   c. 2000
   d. 3000

3. If the user enters the number 200, what value will be in the `number` variable after the selection structure in Figure 6-32 is processed?

   a. 0
   b. 100
   c. 400
   d. 600

Use the code shown in Figure 6-33 to answer Review Questions 4 through 7.

```
if (id == '8')
     cout << "Janet";
else if (id == '2' || id == '7')
     cout << "Mark";
else if (id == '9')
     cout << "Jerry";
else
     cout << "Sue";
//end if
```

Figure 6-33

4. What will the code in Figure 6-33 display when the `id` variable contains the character 9?

   a. Janet
   b. Jerry
   c. Mark
   d. Sue

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

5. What will the code in Figure 6-33 display when the `id` variable contains the character 4?
   a. Janet
   b. Jerry
   c. Mark
   d. Sue
6. What will the code in Figure 6-33 display when the `id` variable contains the character 7?
   a. Janet
   b. Jerry
   c. Mark
   d. Sue
7. What will the code in Figure 6-33 display when the `id` variable contains the character 2?
   a. Janet
   b. Jerry
   c. Mark
   d. Sue

Use the code shown in Figure 6-34 to answer Review Questions 8 through 10.

```
switch (id)
{
      case 8:
           cout << "Janet";
           break;
      case 2:
      case 7:
           cout << "Mark";
           break;
      case 9:
           cout << "Jerry";
           break;
      default:
           cout << "Sue";
}     //end switch
```

Figure 6-34

8. What will the code in Figure 6-34 display when the `id` variable contains the number 2?
   a. Janet
   b. Jerry
   c. Mark
   d. Sue
9. What will the code in Figure 6-34 display when the `id` variable contains the number 4?
   a. Janet
   b. Jerry
   c. Mark
   d. Sue
10. What will the code in Figure 6-34 display when the `id` variable contains the number 9?
    a. Janet
    b. Jerry
    c. Mark
    d. Sue

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Exercises

Pencil and Paper

TRY THIS

1. Write the C++ code for the multiple-alternative selection structure shown in Figure 6-35. First, use the longer form of the `if` statement. Then rewrite the code using the shorter form of the `if` statement. (The answers to TRY THIS Exercises are located at the end of the chapter.)

```
if (sales <= 0)
    bonus = 0
    display "The sales must be greater than 0."
else
    if (sales <= 1000)
        bonus = sales * 0.15
    else
        if (sales <= 5000)
            bonus = sales * 0.20
        else
            bonus = sales * 0.25
        end if
    end if
end if
```

Figure 6-35

TRY THIS

2. Using the `switch` statement, write the C++ code that corresponds to the partial flowchart shown in Figure 6-36. Use a `char` variable named `code` and a `double` variable named `rate`. (The answers to TRY THIS Exercises are located at the end of the chapter.)

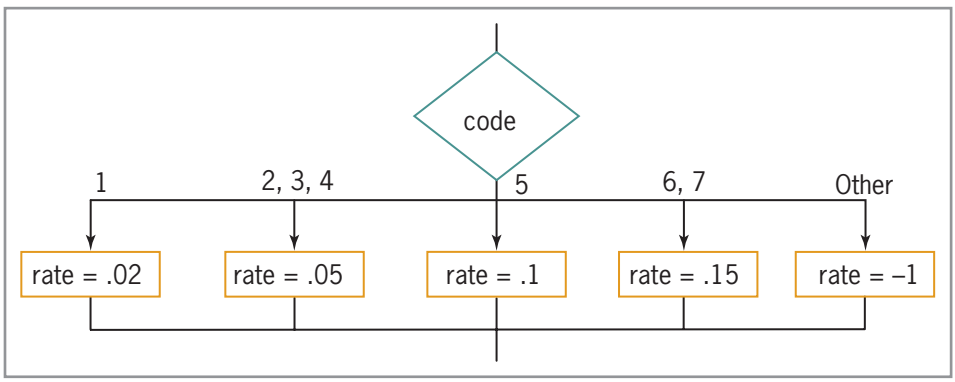


Figure 6-36

MODIFY THIS

3. Complete TRY THIS Exercise 2, and then change the `switch` statement to the multiple-alternative form of the `if` statement.

INTRODUCTORY

4. Travis is standing in front of two containers: one marked Trash and the other marked Recycle. In his right hand, he is holding a bag that contains either trash or recyclables. Travis needs to lift the lid from the appropriate container (if necessary), then drop the bag in the container, and then put the lid back on the container. Write an appropriate algorithm, using only the instructions listed in Figure 6-37. (An instruction can be used more than once.)

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

```
else
end if
drop the bag of recyclables in the Recycle container
drop the bag of trash in the Trash container
if (the bag contains trash)
if (the lid is on the Recycle container)
if (the lid is on the Trash container)
lift the Recycle container's lid using your left hand
lift the Trash container's lid using your left hand
put the lid back on the Recycle container using your left hand
put the lid back on the Trash container using your left hand
```

Figure 6-37

INTRODUCTORY

5. Write the C++ code for a multiple-alternative selection structure that displays one of four different messages, depending on the number of seminar participants stored in an `int` variable named `participants`. Display the message "Use the Kanton room" when the number of seminar participants is at least 75. When the number is 40 through 74, display the message "Use the Harris room". When the number is 10 through 39, display the message "Use the small conference room". When the number is less than 10, display the message "Cancel the seminar".

INTRODUCTORY

6. A program stores test scores in two `int` variables named `myScore` and `expectedScore`. Write the C++ code to compare the two scores and then display one of the following messages: "I met my expectations", "I scored higher than expected", or "I scored lower than expected".

INTERMEDIATE

7. A program uses a `char` variable named `department` and two `double` variables named `salary` and `raise`. The `department` variable contains one of the following letters (entered in either uppercase or lowercase): A, B, C, or D. Employees in departments A and B are receiving a 2% raise. Employees in department C are receiving a 1.5% raise, and employees in department D are receiving a 3% raise. Using the `switch` statement, write the C++ code to calculate the appropriate raise amount.

ADVANCED

8. A program uses a `char` variable named `membership` and an `int` variable named `age`. The `membership` variable contains one of the following letters (entered in either uppercase or lowercase): M or N. The letter M stands for *member*, and the letter N stands for *nonmember*. The program should display the appropriate seminar fee, which is based on a person's membership status and age. The fee schedule is shown in Figure 6-38. Write the C++ code to display the fee.

Seminar fee	Criteria
$10	Club member less than 65 years old
$5	Club member at least 65 years old
$20	Nonmember less than 65 years old
$15	Nonmember at least 65 years old

Figure 6-38

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

9. The C++ code in Figure 6-39 should display one of the four messages listed in the figure. The appropriate message is based on the level entered by the user. Correct the errors in the code.

SWAT THE BUGS

Level	Message
1 or 2	Bronze
3	Silver
4 or 5	Gold
Other	Invalid ID

```
int level = 0;
cout << "Level (1 through 5): ";
cin >> level;
switch (level)
{
      case 1:
      case 2:
                  cout << "Bronze";
      case 3:
                  break;
                  cout << "Silver";
      case 4:
                  cout << "Gold";
      case 5:
                  break;
      default:
                  cout << "Invalid ID";
}     //end switch
```

Figure 6-39

Computer

10. Figure 6-40 shows a partially completed chart for a program that displays the amount of a salesperson's commission. The commission is based on the salesperson's sales amount, as indicated in the figure. Complete the selection structure in the Algorithm section of the chart. Also complete the C++ instructions section. After completing the chart, create a new project (if necessary) named TryThis10 Project, and save it in the Cpp8\Chap06 folder. Enter the C++ instructions into a source file named TryThis10.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Display the commission in fixed-point notation with two decimal places. Save and run the program. Test the program using the following sales amounts: 15250, 251990, 500670, and –5. (The answers to TRY THIS Exercises are located at the end of the chapter.)

TRY THIS

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Sales range ($)	Commission
0–100,000	2% of the sales amount
100,001–400,000	5% of the sales amount
400,001 and over	9% of the sales amount

IPO chart information

Input

sales

Processing

none

Output

commission

Algorithm:

```
1. enter sales
2. if (sales < 0)
      commission = -1
   else


3. if (commission is not -1)
      display commission with 2 decimals
   else
      display "Sales cannot be less than 0."
   end if
```

C++ instructions

```
int sales = 0;

double commission = 0.0;
```

Figure 6-40

TRY THIS

11. Code the algorithm shown in Figure 6-41. Use the `switch` statement to code the multiple-alternative selection structure. If necessary, create a new project named TryThis11 Project, and save it in the Cpp8\Chap06 folder. Enter the C++ instructions into a source file named TryThis11.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Save and run the program. Test the program using the following codes: 1, 2, 3, and 7. (The answers to TRY THIS Exercises are located at the end of the chapter.)

IPO chart information

Input

department code (1, 2, or 3)

Processing

none

Output

salary

C++ instructions

```
int deptCode = 0;

int salary = 0;
```

Figure 6-41 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
Algorithm:
1. enter the department code
2. if (the department code is one of the following:)
        1       salary = 25000
        2       salary = 30000
        3       salary = 32000
        4       display "Invalid code"
                salary = 0
   end if
3. display salary
```

Figure 6-41

MODIFY THIS

12. Complete TRY THIS Exercise 11. If necessary, create a new project named ModifyThis12 Project, and save it in the Cpp8\Chap06 folder. Enter (or copy) the instructions from the TryThis11.cpp file into a new source file named ModifyThis12.cpp. Be sure to change the filename in the first comment. Remove the `default` clause from the `switch` statement. Modify the code to verify that the department code is 1, 2, or 3 only. If the department code is valid, use the `switch` statement to determine the salary, and then display the salary. If the department code is *not* valid, display the "Invalid code" message. Save and run the program. Test the program using the following codes: 1, 2, 3, and 7.

INTRODUCTORY

13. Karlton Learning wants a program that displays the amount of money a company owes for a seminar. The fee per person is based on the number of people the company registers, as shown in Figure 6-42. For example, if the company registers seven people, then the total amount owed is $700. If the user enters a number that is less than or equal to 0, the program should display an appropriate error message.

Number of registrants	Fee per person
1 through 5	$125
6 through 20	$ 100
21 or more	$ 75

Figure 6-42

a. Create an IPO chart for the problem, and then desk-check the algorithm five times, using the numbers 4, 8, 22, 0, and –2 as the number of people registered.
b. List the input, processing, and output items, as well as the algorithm, in a chart similar to the one shown earlier in Figure 6-27. Then code the algorithm into a program.
c. Desk-check the program using the same data used to desk-check the algorithm.
d. If necessary, create a new project named Introductory13 Project, and save it in the Cpp8\Chap06 folder. Enter your C++ instructions into a source file named Introductory13.cpp. Also enter appropriate comments and any additional instructions required by the compiler.
e. Save and run the program. Test the program using the same data used to desk-check the program.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

INTRODUCTORY

14. The owner of Harry's Car Sales pays each salesperson a commission based on his or her quarterly sales. The sales ranges and corresponding commission rates are shown in Figure 6-43. The program should display an error message if the sales amount is less than 0.

Quarterly sales ($)	Commission
0–20,000	multiply the sales by 5%
20,001–50,000	multiply the sales over 20,000 by 7% and then add 1,000 to the result
50,001 or more	multiply the sales over 50,000 by 10% and then add 3,100 to the result

Figure 6-43

a. Create an IPO chart for the problem, and then desk-check the algorithm seven times, using sales of 20000, 20001, 30000, 50000, 50001, 75000, and –3.
b. List the input, processing, and output items, as well as the algorithm, in a chart similar to the one shown earlier in Figure 6-27. Then code the algorithm into a program.
c. Desk-check the program using the same data used to desk-check the algorithm.
d. If necessary, create a new project named Introductory14 Project, and save it in the Cpp8\Chap06 folder. Enter your C++ instructions into a source file named Introductory14.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Display the commission in fixed-point notation with two decimal places.
e. Save and run the program. Test the program using the same data used to desk-check the program.

INTERMEDIATE

15. In this exercise, you will create a program that displays the number of daily calories needed to maintain your current weight. The number of calories is based on your gender, activity level, and weight. The formulas for calculating the daily calories are shown in Figure 6-44.

Moderately active female: total daily calories = weight multiplied by 12 calories per pound
Relatively inactive female: total daily calories = weight multiplied by 10 calories per pound
Moderately active male: total daily calories = weight multiplied by 15 calories per pound
Relatively inactive male: total daily calories = weight multiplied by 13 calories per pound

Gender	Activity	Weight
F	I	150
F	A	120
M	I	180
M	A	200

Figure 6-44

a. Create an IPO chart for the problem, and then desk-check the algorithm using the test data included in Figure 6-44. Also desk-check it using invalid data, such as X as the gender code, K as the activity code, or a negative number for the weight.
b. List the input, processing, and output items, as well as the algorithm, in a chart similar to the one shown earlier in Figure 6-27. Then code the algorithm into a program.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

c. Desk-check the program using the same data used to desk-check the algorithm.
d. If necessary, create a new project named Intermediate15 Project and save it in the Cpp8\Chap06 folder. Enter your C++ instructions into a source file named Intermediate15.cpp. Also enter appropriate comments and any additional instructions required by the compiler.
e. Save and run the program. Test the program using the same data used to desk-check the program.

16. In this exercise, you will create a program that displays both the smallest and largest of three integers entered by the user. For example, if the user enters the numbers 3, 5, and 9, the program should display the messages "Smallest number is 3." and "Largest number is 9." on the computer screen. INTERMEDIATE

a. Create an IPO chart for the problem, and then desk-check the algorithm four times. For the first desk-check, use the numbers 3, 5, and 9. For the second desk-check, use 7, 10, and 2. For the third desk-check, use 8, 4, and 6. For the fourth desk-check, use 1, 9, and 1.
b. List the input, processing, and output items, as well as the algorithm, in a chart similar to the one shown earlier in Figure 6-27. Then code the algorithm into a program.
c. Desk-check the program using the same data used to desk-check the algorithm.
d. If necessary, create a new project named Intermediate16 Project, and save it in the Cpp8\Chap06 folder. Enter your C++ instructions into a source file named Intermediate16.cpp. Also enter appropriate comments and any additional instructions required by the compiler.
e. Save and run the program. Test the program using the same data used to desk-check the program.

17. In this exercise, you will create a program that converts U.S. dollars to a different currency. The number of American dollars should always be an integer that is greater than or equal to zero. The user should be able to choose the currency from the following list: Canadian Dollar, Euro, Indian Rupee, Japanese Yen, Mexican Peso, South African Rand, and British Pound. (Hint: Designate a code for each currency, and use `cout` statements to display a menu that lists each code and the name of its corresponding currency.) Use the Internet to research the exchange rates. If necessary, create a new project named Advanced17 Project, and save it in the Cpp8\Chap06 folder. Enter your C++ instructions into a source file named Advanced17.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Display the results in fixed-point notation with two decimal places. Save and run the program. Test the program nine times. For the first seven tests, convert 10 American dollars to each of the seven different currencies. For the eighth test, use –3 as the number of American dollars. For the last test, use an invalid currency code. ADVANCED

18. A local department store wants a program that displays the number of reward points a customer earns each month. The reward points are based on the customer's membership type and total monthly purchase amount, as shown in Figure 6-45. If necessary, create a new project named Advanced18 Project, and save it in the Cpp8\Chap06 folder. Enter your C++ instructions into a source file named Advanced18.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Display the reward points in fixed-point notation with no decimal places. Save, run, and test the program. ADVANCED

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Membership type	Total monthly purchase ($)	Reward points
Standard	Less than 75	5% of the total monthly purchase
	75–149.99	7.5% of the total monthly purchase
	150 and over	10% of the total monthly purchase
Plus	Less than 150	6% of the total monthly purchase
	150 and over	13% of the total monthly purchase
Premium	Less than 200	4% of the total monthly purchase
	200 and over	15% of the total monthly purchase

Figure 6-45

ADVANCED

19. If necessary, create a new project named Advanced19 Project, and save it in the Cpp8\Chap06 folder. Enter (or copy) the instructions from the Lab6-2.cpp file into a new source file named Advanced19.cpp. Be sure to change the filename in the first comment. The program should now begin by determining the number of medium pizzas and the number of large pizzas the customer is ordering. Be sure to verify that both numbers are greater than or equal to 0. Rather than displaying the price of one pizza, the program should display the number of pizzas ordered and the total cost of the order. The $2 coupon toward the purchase of a large pizza can be used only once. If a customer orders more than four pizzas, the program should deduct 10% from the cost of the order. Modify the program appropriately, and then save, run, and test it.

ADVANCED

20. In this exercise, you will experiment with the `switch` statement.
    a. Follow the instructions for starting C++ and viewing the Advanced20.cpp file, which is contained in either the Cpp8\Chap06\Advanced20 Project folder or the Cpp8\Chap06 folder. (Depending on your C++ development tool, you may need to open this exercise's project/solution file first.) The program uses the `switch` statement to display the names of the gifts mentioned in the song "The Twelve Days of Christmas."
    b. Run the program. When you are prompted to enter the day, type the number 1 and press Enter. The names of the gifts for the first through the twelfth days appear on the screen. Close the Command Prompt window.
    c. Run the program again. When you are prompted to enter the day, type the number 9 and press Enter. The names of the gifts for the ninth through the twelfth days appear on the screen. Close the Command Prompt window.
    d. Modify the program so that it displays only the name of the gift corresponding to the day entered by the user. For example, when the user enters the number 4, the program should display the "4 calling birds" message only.
    e. Save and then run the program. When you are prompted to enter the day, type the number 4 and press Enter. The "4 calling birds" message should appear on the screen. Close the Command Prompt window, and then test the program using the numbers 1 and 9.

ADVANCED

21. In this exercise, you will include a Boolean value in a `switch` statement. Follow the instructions for starting C++ and viewing the Advanced21.cpp file, which is contained in either the Cpp8\Chap06\Advanced21 Project folder or the Cpp8\Chap06 folder. (Depending on your C++ development tool, you may need to open this exercise's project/solution file first.) Replace the dual-alternative `if` statement with a `switch` statement. Save and then run the program. Test the program appropriately.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

22. Follow the instructions for starting C++ and viewing the SwatTheBugs22.cpp file, which is contained in either the Cpp8\Chap06\SwatTheBugs22 Project folder or the Cpp8\Chap06 folder. (Depending on your C++ development tool, you may need to open this exercise's project/solution file first.) The program should calculate and display an item's new price, but it is not working correctly. Test the program using 1 as the code and 10 as the old price. Then test it using the following data: 2 and 10, 3 and 20, and 4 and 50. Debug the program.

SWAT THE BUGS

Answers to TRY THIS Exercises

Pencil and Paper

1. See Figure 6-46.

Longer form of the `if` statement

```
if (sales <= 0)
{
    bonus = 0;
    cout << "The sales must be greater than 0." << endl;
}
else
    if (sales <= 1000)
        bonus = sales * 0.15;
    else
        if (sales <= 5000)
            bonus = sales * 0.2;
        else
            bonus = sales * 0.25;
        //end if
    //end if
//end if
```

Shorter form of the `if` statement

```
if (sales <= 0)
{
    bonus = 0;
    cout << "The sales must be greater than 0." << endl;
}
else if (sales <= 1000)
    bonus = sales * 0.15;
else if (sales <= 5000)
    bonus = sales * 0.2;
else
    bonus = sales * 0.25;
//end if
```

Figure 6-46

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

2. See Figure 6-47.

```
switch (code)
{
     case '1':
                rate = .02;
                break;
     case '2':
     case '3':
     case '4':
                rate = .05;
                break;
     case '5':
                rate = .1;
                break;
     case '6':
     case '7':
                rate = .15;
                break;
     default:
                rate = -1;
}    //end switch
```

Figure 6-47

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Computer

10. See Figures 6-48 and 6-49.

IPO chart information

Input

sales

Processing

none

Output

commission

Algorithm:

```
1. enter sales

2. if (sales < 0)
      commission = -1
   else
      if (sales <= 100000)
         commission = sales * .02
      else
         if (sales <= 400000)
            commission = sales * .05
         else
            commission = sales * .09
         end if
      end if
   end if
3. if (commission is not -1)
      display commission with 2 decimals
   else
      display "Sales cannot be less than
      0."
   end if
```

C++ instructions

```
int sales = 0;


double commission = 0.0;

cout << "Sales: ";
cin >> sales;

if (sales < 0)
    commission = -1;

else if (sales <= 100000)
    commission = sales * .02;

else if (sales <= 400000)
    commission = sales * .05;
else
    commission = sales * .09;
//end if


if (commission != -1)
{
    cout << fixed << setprecision(2);
    cout << "Commission: $" <<
    commission << endl;
}
else
    cout << "Sales cannot be less
    than 0." << endl;
//end if
```

Figure 6-48

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

```
//TryThis10.cpp - displays a salesperson's commission
//Created/revised by <your name> on <current date>

#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    int sales = 0;
    double commission = 0.0;

    cout << "Sales: ";
    cin >> sales;

    //determine commission
    if (sales < 0)
        commission = -1;
    else if (sales <= 100000)
        commission = sales * .02;
    else if (sales <= 400000)
        commission = sales * .05;
    else
        commission = sales * .09;
    //end if

    //display commission or error message
    if (commission != -1)
    {
        cout << fixed << setprecision(2);
        cout << "Commission: $" << commission << endl;
    }
    else
        cout << "Sales cannot be less than 0." << endl;
    //end if
    return 0;
}   //end of main function
```

Figure 6-49

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

11. See Figures 6-50 and 6-51.

IPO chart information

Input

department code (1, 2, or 3)

Processing

none

Output

salary

Algorithm:

1. enter the department code
2. if (the department code is one of the following:)
   - 1 salary = 25000
   - 2 salary = 30000
   - 3 salary = 32000
   - 4 display "Invalid code"
     salary = 0

   end if
3. display salary

C++ instructions

```
int deptCode = 0;

int salary = 0;

cout << "Department code (1, 2, or
3): ";
cin >> deptCode;
switch (deptCode)
{
case 1:
    salary = 25000;
    break;
case 2:
    salary = 30000;
    break;
case 3:
    salary = 32000;
    break;
default:
    cout << "Invalid code" << endl;
    salary = 0;
}   //end switch
cout << "Salary: $" << salary <<
endl;
```

Figure 6-50

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

```
//TryThis11.cpp
//displays the salary associated with a department code
//Created/revised by <your name> on <current date>

#include <iostream>
using namespace std;

int main()
{
    int deptCode = 0;
    int salary = 0;

    cout << "Department code (1, 2, or 3): ";
    cin >> deptCode;

    //determine the salary
    switch (deptCode)
    {
    case 1:
        salary = 25000;
        break;
    case 2:
        salary = 30000;
        break;
    case 3:
        salary = 32000;
        break;
    default:
        cout << "Invalid code" << endl;
        salary = 0;
    }   //end switch

    //display the salary
    cout << "Salary: $" << salary << endl;
    return 0;
}   //end of main function
```

Figure 6-51

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

CHAPTER 7

The Repetition Structure

After studying Chapter 7, you should be able to:

- Differentiate between a pretest loop and a posttest loop
- Include a pretest loop in pseudocode
- Include a pretest loop in a flowchart
- Code a pretest loop using the C++ `while` statement
- Utilize counter and accumulator variables
- Code a pretest loop using the C++ `for` statement

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Repeating Program Instructions

Recall that the three programming control structures are sequence, selection, and repetition.

Programmers use the **repetition structure**, referred to more simply as a **loop**, when they need the computer to repeatedly process one or more program instructions. The loop contains a condition that controls whether the instructions are repeated. In many programming languages, the condition can be phrased in one of two ways. It can either specify the requirement for repeating the instructions or specify the requirement for *not* repeating them. The requirement for repeating the instructions is referred to as the **looping condition** because it indicates when the computer should continue "looping" through the instructions. The requirement for *not* repeating the instructions is referred to as the **loop exit condition** because it tells the computer when to exit (or stop) the loop. Every looping condition has an opposing loop exit condition; one is the opposite of the other.

Two examples may help illustrate the difference between the looping condition and the loop exit condition. You have probably heard the old adage "Make hay while the sun shines." The "while the sun shines" is the looping condition because it tells you when to *continue* making hay. The adage could also be phrased as "Make hay until the sun is no longer shining." In this case, the "until the sun is no longer shining" is the loop exit condition because it specifies when you should *stop* making hay. Similarly, the sentence "Listen while the speaker is talking." uses a looping condition to indicate when you should *continue* listening. The sentence "Listen until the speaker stops talking.", on the other hand, uses a loop exit condition to specify when you should *stop* listening. In the C++ programming language, the repetition structure's condition is always phrased as a looping condition, which means it always contains the requirement for repeating the instructions within the loop.

The programmer determines whether a problem's solution requires a loop by studying the problem specification. The first problem specification you will examine in this chapter involves a superheroine named Sahirah. The problem specification and an illustration of the problem are shown in Figure 7-1, along with a correct algorithm written in pseudocode. The algorithm uses only the sequence and selection structures because no instructions need to be repeated.

Ch07-Sahirah

Problem specification
A superheroine named Sahirah must prevent a poisonous yellow spider from attacking King Khafra and Queen Rashida. Sahirah has one weapon at her disposal: a laser beam that shoots out from her right hand. Unfortunately, Sahirah gets only one shot at the spider, which is flying around the palace looking for the king and queen. Before taking the shot, she needs to position both her right arm and her right hand toward the spider. After taking the shot, she should return her right arm and right hand to their original positions. In addition, she should say "You are safe now. The spider is dead.", if the laser beam hit the spider; otherwise, she should say "Run for your lives, my king and queen!"

Figure 7-1 A problem that requires the sequence and selection structures *(continues)*
Image by Diane Zak; created with Reallusion CrazyTalk Animator

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Algorithm
1. position both your right arm and your right hand toward the spider
2. shoot a laser beam at the spider
3. return your right arm and right hand to their original positions
4. if (the laser beam hit the spider)
        say "You are safe now. The spider is dead."
   else
        say "Run for your lives, my king and queen!"
   end if

Figure 7-1 A problem that requires the sequence and selection structures
Image by Diane Zak; created with Reallusion CrazyTalk Animator

Now we will change the problem specification slightly. Rather than taking only one shot, Sahirah can now take as many shots as needed to destroy the spider. Because of this, she will never need to tell the king and queen to run for their lives again. Figure 7-2 shows the modified problem specification along with the modified algorithm, which contains the sequence and repetition structures. The repetition structure begins with the `repeat while (the laser beam did not hit the spider)` clause and ends with the `end repeat` clause. The instructions between both clauses, called the **loop body**, are indented to indicate that they are part of the repetition structure.

The portion within parentheses in the `repeat while (the laser beam did not hit the spider)` clause specifies the repetition structure's condition. The condition is phrased as a looping condition because it tells Sahirah when to repeat the instructions. In this case, she should repeat the instructions as long as (or while) the laser beam did not hit the spider. Similar to the condition in a selection structure, the condition in a repetition structure must evaluate to a Boolean value: either true or false. The condition in Figure 7-2 evaluates to true when the laser beam did *not* hit the spider and evaluates to false when the laser beam *did* hit the spider. If the condition evaluates to true, Sahirah should follow the loop body instructions. She should skip over those instructions when the condition evaluates to false.

Problem specification
A superheroine named Sahirah must prevent a poisonous yellow spider from attacking King Khafra and Queen Rashida. Sahirah has one weapon at her disposal: a laser beam that shoots out from her right hand. Sahirah can take as many shots as needed to destroy the spider, which is flying around the palace looking for the king and queen. Before taking each shot, she needs to position both her right arm and her right hand toward the spider. When the laser beam hits the spider, she should return her right arm and right hand to their original positions and then say "You are safe now. The spider is dead."

Algorithm
1. position both your right arm and your right hand toward the spider
2. shoot a laser beam at the spider
3. repeat while (the laser beam did not hit the spider) — condition
        position both your right arm and your right hand toward the spider — loop body
        shoot a laser beam at the spider — loop body
   end repeat
4. return your right arm and right hand to their original positions
5. say "You are safe now. The spider is dead."

Figure 7-2 A problem that requires the sequence and repetition structures

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Pretest and posttest loops are also called top-driven and bottom-driven loops, respectively.

A repetition structure can be either a pretest loop or a posttest loop. In both types of loops, the condition is evaluated with each repetition (or iteration) of the loop. In a **pretest loop**, the condition is evaluated *before* the instructions within the loop are processed. In a **posttest loop**, the evaluation occurs *after* the instructions within the loop are processed. The loop shown in Figure 7-2 is a pretest loop.

Depending on the result of the evaluation, the instructions in a pretest loop may never be processed. The algorithm in Figure 7-2 can be used to illustrate this point. If Step 2's shot hits the spider, the instructions in the loop body will not be processed because the loop's condition in Step 3 will evaluate to false. The instructions in a posttest loop, on the other hand, will always be processed at least once. Of the two types of loops, the pretest loop is the most commonly used. You will learn about pretest loops in this chapter; posttest loops are covered in Chapter 8 along with nested loops.

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 7-1

1. Using only the following five instructions, write an algorithm for printing the pages in a document: `end repeat`, `print the next page`, `print the first page`, `repeat while (there is another page to print)`, `say "Done printing"`.
2. Modify the algorithm from Question 1 so that it prints all of the pages except page 3. (You will need to add your own instructions to the algorithm.)
3. Derek is practicing for an upcoming basketball game. Write an appropriate algorithm using only the following instructions: `end repeat`, `repeat while (the basketball did not go through the hoop)`, `say "I did it!"`, `say "Missed it!"`, `shoot the basketball`. (An instruction can be used more than once.)

Using a Pretest Loop to Solve a Real-World Problem

Figure 7-3 shows the problem specification for the commission program. It also shows two algorithms that could be used to calculate and display the amount of each salesperson's commission. However, a program based on Algorithm 1 would need to be executed once for each of the company's salespeople. A more efficient way to calculate and display the commission amounts is provided in Algorithm 2, which contains a loop. After running a program based on Algorithm 2, the company's accountant can calculate and display the commission for any number of salespeople without having to run the program again. The program will end when the accountant enters –1 (a negative number one) as the sales amount.

Problem specification
Create a program that calculates the commission for each of a company's salespeople. The commission is calculated by multiplying the salesperson's sales amount by 20%.

Input	Processing	Output
`commission rate (20%)` `sales`	`Processing items: none`	`commission`

Figure 7-3 Problem specification and IPO chart for the commission program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

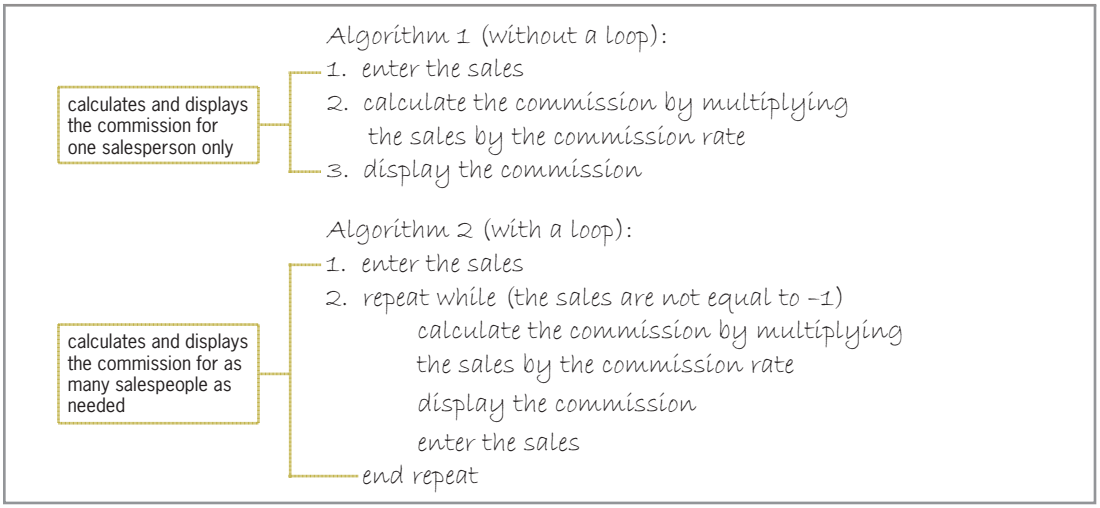


Figure 7-3 Problem specification and IPO chart for the commission program

Figure 7-4 identifies the important components of Algorithm 2. With very rare exceptions, every loop has a condition and a loop body. In a pretest loop, the condition appears at the beginning of the loop. As mentioned earlier, the condition must result in a Boolean value: either true or false. The condition in Figure 7-4 evaluates to true when the sales entry is *not* equal to –1 and evaluates to false when it *is* equal to –1.

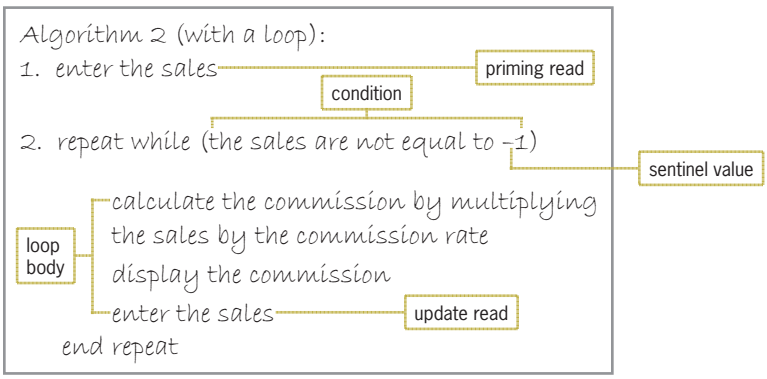


Figure 7-4 Components of Algorithm 2 from Figure 7-3

Some loops, such as the one in Figure 7-4, require the user to enter a special value to end the loop. Values that are used to end loops are referred to as **sentinel values**. The sentinel value should be one that is easily distinguishable from the valid data recognized by the program. In the loop in Figure 7-4, the sentinel value is –1. The number 15 would not be a good sentinel value for the loop because it is possible for a salesperson to sell $15 in product. The number –1, on the other hand, is a good sentinel value for the loop because a salesperson cannot sell a negative amount.

Sentinel values are often referred to as trip values because they release the loop from its task. And, because they are the last values entered before the loop ends, they are also called trailer values.

When a loop's condition evaluates to true, the one or more instructions listed in the loop body are processed; otherwise, the loop body instructions are skipped over. After each processing of the loop body instructions, the loop's condition is reevaluated to determine

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

whether the instructions should be processed again. The loop body instructions are processed and the loop's condition is evaluated until the condition evaluates to false, at which time the loop ends and processing continues with the instruction immediately following the end of the loop.

Keep in mind that because the condition in a pretest loop is evaluated *before* any of the instructions within the loop body are processed, it is possible that the loop body instructions may not be processed at all. For example, if the accountant enters the number –1 as the first sales amount, the condition in Figure 7-4's loop will evaluate to false, and the instructions in the loop body will not be processed during that run of the program.

The algorithm in Figure 7-4 contains two *enter the sales* instructions. One of the instructions appears above the loop, and the other appears as the last instruction in the loop body. The *enter the sales* instruction above the loop is referred to as the **priming read** because it is used to prime (prepare or set up) the loop. The priming read initializes the loop condition by providing its first value—in this case, the first sales amount. This first entry is compared to the sentinel value (–1) and determines whether the loop body instructions are processed at all.

If the loop body instructions *are* processed, the *enter the sales* instruction in the loop body gets the remaining sales entries (if any). This instruction is referred to as the **update read** because it allows the user to update the value of the input item (in this case, the sales amount) that controls the looping condition. The update read is often an exact copy of the priming read.

The importance of the update read cannot be stressed enough. If you do not include the update read in the loop body, there will be no way for the user to enter the sentinel value after the loop body instructions are processed the first time. This is because the priming read is processed only once and gets only the first input entry from the user. Without the update read, the loop will have no way of stopping on its own. You will learn more about this in the section titled "The `while` Statement" later in the chapter.

Flowcharting a Pretest Loop

Figure 7-5 shows the commission program's algorithm from Figure 7-4 in flowchart form. The diamond in the figure indicates the beginning of a repetition structure (loop). Like the diamond in a selection structure, the diamond in a repetition structure contains a condition that evaluates to either true or false only. The condition determines whether the instructions within the loop body are processed. Also like the diamond in a selection structure, the diamond in a repetition structure has one flowline entering the symbol and two flowlines leaving the symbol. The two flowlines leading out of the diamond are marked with a "T" (for true) and an "F" (for false). The flowline marked with a "T" leads to the loop body, which contains the instructions to be processed when the loop's condition evaluates to true. The flowline marked with an "F", on the other hand, leads to the instructions to be processed when the loop's condition evaluates to false. Notice that a circle or loop is formed by the flowline entering the diamond combined with the diamond and the symbols and flowlines within the true path. It is this loop (circle) that distinguishes the repetition structure from the selection structure in a flowchart.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

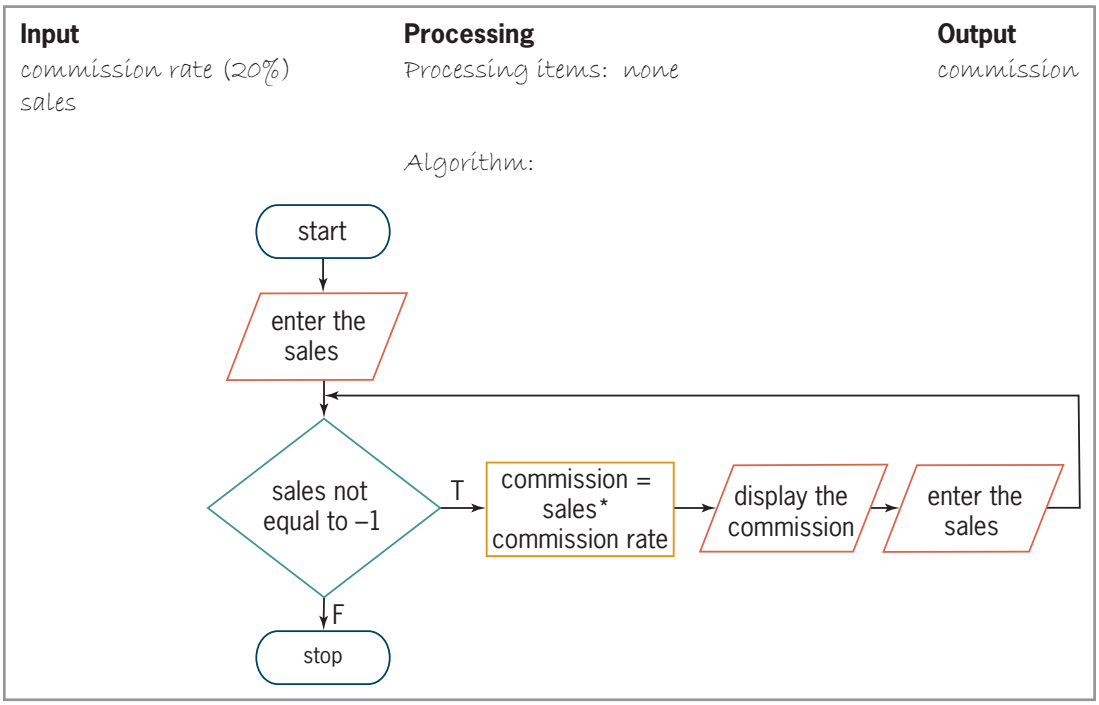


Figure 7-5 Flowchart for the algorithm shown in Figure 7-4

The first and last parallelograms in Figure 7-5 represent the priming and update reads, respectively.

To illustrate how a loop operates in a program, you will desk-check the algorithm in Figure 7-5 using the following sales amounts: 1200, 800, and –1 (the sentinel value). The commission amounts should be $240 and $160, respectively. First, you record the input and output items in a desk-check table. You then follow each of the symbols in the flowchart, from top to bottom, recording in the desk-check table any changes made to those items. The first symbol is the start oval, which merely marks the beginning of the flowchart. The next symbol is a parallelogram that gets the first sales entry from the user; this symbol represents the priming read. Figure 7-6 shows this first entry recorded in the desk-check table.

Ch07-Commission

commission rate	sales	commission
0.2	1200	

Figure 7-6 First sales entry recorded in the desk-check table

The next symbol in the flowchart is a diamond that represents the condition in a pretest loop. You can tell that the loop is a pretest loop (rather than a posttest loop) because the diamond appears *before* the symbols in both the true and false paths. The loop's condition tells the computer to compare the sales amount entered by the user with the sentinel value (–1). In this case, the condition evaluates to true because 1200 is not equal to –1. When the condition evaluates to true, the computer processes the instructions in the loop body. The first two instructions calculate and display the commission. Figure 7-7 shows the first salesperson's information recorded in the desk-check table. The commission amount agrees with the manually calculated results.

commission rate	sales	commission
0.2	1200	240

Figure 7-7 First salesperson's information recorded in the desk-check table

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The last instruction in the loop body in Figure 7-5 is contained in a parallelogram. The instruction allows the user to enter the sales amount for the next salesperson (800). Recall that this instruction is the update read. After the user enters the sales amount, the loop's condition, which is contained in the diamond located at the beginning of the loop, is reevaluated to determine whether the loop should be processed again (a true condition) or end (a false condition). In this case, the condition evaluates to true because 800 is not equal to –1. As a result, the commission is calculated and then displayed on the screen. Figure 7-8 shows the second salesperson's information recorded in the desk-check table. Here too, the commission amount agrees with the manually calculated results.

commission rate	sales	commission
0.2	~~1200~~	~~240~~
	800	160

Figure 7-8 Second salesperson's information recorded in the desk-check table

The update read gets the amount sold by the next salesperson: –1 (the sentinel value). The loop's condition is then reevaluated to determine whether the loop should be processed again (a true condition) or end (a false condition). The condition evaluates to false because the user's entry *is* equal to –1. Therefore, the computer skips over the loop body instructions and processes the instruction immediately following the end of the loop. In Figure 7-5's flowchart, the stop oval follows the loop and marks the end of the flowchart. The completed desk-check table is shown in Figure 7-9.

commission rate	sales	commission
0.2	~~1200~~	~~240~~
	~~800~~	160
	-1	

Figure 7-9 Completed desk-check table

You can code a pretest loop using either the `while` statement or the `for` statement. You will learn about the `while` statement first.

The `while` Statement

Figure 7-10 shows the syntax of the **while statement**, which can be used to code a pretest loop in a C++ program. As the boldfaced text in the syntax indicates, the keyword `while` and the parentheses that surround the *condition* are essential components of the statement. The italicized items in the syntax indicate where the programmer must supply information. In this case, the programmer needs to supply the *condition*, which must be phrased as a looping condition. The condition must be a Boolean expression, which is an expression that evaluates to either true or false. The expression can contain variables, constants, functions, arithmetic operators, comparison operators, and logical operators.

Besides providing the condition, the programmer must also provide the loop body statements, which are the statements to be processed when the condition evaluates to true. If more than one statement needs to be processed, the statements must be entered as a statement block by enclosing them in a set of braces (`{}`). You can also include the braces when a loop body contains only one statement. By doing this, you won't need to remember to enter the braces

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

when statements are added subsequently to the loop body. Forgetting to enter the braces is a common error made when typing the `while` statement in a program. Although not a requirement, using a comment (such as `//end while`) to mark the end of the `while` statement will make your program easier to read and understand.

Also included in Figure 7-10 are examples of using the `while` statement. In Example 1, the `while (age > 0)` clause tells the computer to repeat the loop body instructions as long as (or while) the value in the `age` variable is greater than 0. The loop will stop when the user enters either the number 0 or a negative number. In Example 2, the `while (toupper(anotherSale) == 'Y')` clause indicates that the loop body instructions should be repeated as long as the uppercase equivalent of the value in the `anotherSale` variable is the letter Y. In this case, the loop will stop when the user enters anything other than the letters Y or y.

HOW TO Use the while Statement

Syntax

while (*condition*)
either one statement or a statement block to be processed as long as the condition is true
`//end while`

Example 1

```
int age = 0;

cout << "Enter age: ";
cin >> age;
while (age > 0)
{
      cout << "You entered " << age << endl;
      cout << "Enter age: ";
      cin >> age;
}     //end while
```

Example 2

```
char anotherSale = ' ';
double sales = 0.0;

cout << "Enter a sales amount? (Y/N) ";
cin >> anotherSale;
while (toupper(anotherSale) == 'Y')
{
      cout << "Enter the sales: ";
      cin >> sales;
      cout << "You entered " << sales << endl;
      cout << "Enter a sales amount? (Y/N) ";
      cin >> anotherSale;
}     //end while
```

Note: You could also write the `while` clause in Example 2 as either `while (tolower(anotherSale) == 'y')` or `while (anotherSale == 'Y' || anotherSale == 'y')`.

Figure 7-10 How to use the `while` statement

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Figure 7-11 shows the IPO chart information and corresponding C++ instructions for the commission program. The first `cout` statement prompts the user to enter the amount of the first salesperson's sales, and the `cin >> sales;` statement (the priming read) stores the user's response in the `sales` variable. The looping condition in the `while` clause compares the value stored in the `sales` variable with the sentinel value (–1). If the variable does not contain the sentinel value, the looping condition evaluates to true and the loop body instructions are processed. Those instructions calculate and display the commission. They then use a `cout` statement to prompt the user to enter the sales amount for the next salesperson and use a `cin` statement (the update read) to store the user's response in the `sales` variable. Each time the user enters a sales amount, the looping condition in the `while` clause compares the entry to the sentinel value. When the loop condition evaluates to false, which is when the `sales` variable contains the sentinel value, the loop body instructions are skipped over and processing continues with the instruction located immediately below the end of the loop.

The loop in Figure 7-11 will stop when the sales amount is equal to –1.0 because *equal to* is the opposite of *not equal to*.

IPO chart information	**C++ instructions**
Input	
commission rate (20%) sales	`const double RATE = 0.2;` `int sales = 0;`
Processing	
none	
Output	`double commission = 0.0;`
commission	
Algorithm	`cout << "First salesperson's sales (-1 to stop): ";` `cin >> sales;`
1. enter the sales	
2. repeat while (the sales are not equal to –1)	`while (sales != -1)` `{`
calculate the commission by multiplying the sales by the commission rate	`commission = sales * RATE;`
display the commission	`cout << "Commission: $" << commission << endl << endl;`
enter the sales	`cout << "Next salesperson's sales (-1 to stop): ";` `cin >> sales;`
end repeat	`}  //end while`

Figure 7-11 IPO chart information and C++ instructions for the commission program

The importance of the update read was mentioned earlier. If the loop body in Figure 7-11 does not contain the `cin >> sales;` statement, the computer will process the loop body instructions indefinitely. This is because without that `cin` statement, there will be no way to change the value stored in the `sales` variable once the loop body instructions are processed. A loop whose instructions are processed indefinitely is referred to as either an **endless loop** or an **infinite loop**. Usually, you can stop a program that contains an endless loop by pressing Ctrl+c (press and hold down the Ctrl key as you tap the letter c, and then release both keys); you can also use the Command Prompt window's Close button.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Figure 7-12 shows a sample run of the commission program. The program uses the `fixed` and `setprecision` stream manipulators to display the commission amounts in fixed-point notation with two decimal places.

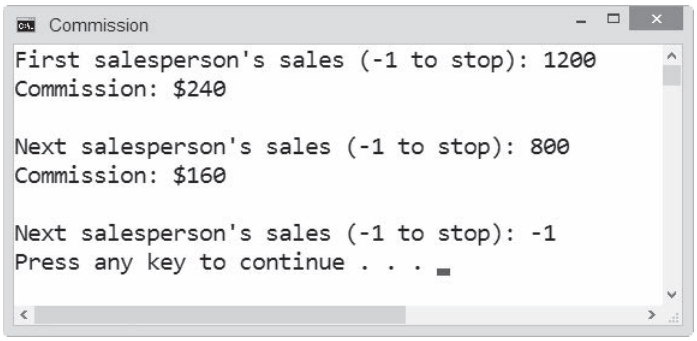


Figure 7-12 A sample run of the commission program

Mini-Quiz 7-2

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

1. Write a C++ `while` clause that processes the loop body instructions as long as the value in the `ordered` variable is greater than the number 100.
2. Write a C++ `while` clause that stops the loop when the value in the `quantity` variable is less than the number 0. (Hint: Change the loop exit condition to a looping condition.)
3. Write a C++ `while` clause that processes the loop body instructions as long as the value in the `inStock` variable is greater than the value in the `reorder` variable.
4. Write a C++ `while` clause that processes the loop body instructions as long as the value in a `char` variable named `letter` is either Y or y.
5. Which of the following is a good sentinel value for a program that inputs a test score?
   a. –9
   b. 32
   c. 45.5
   d. 7

Using Counters and Accumulators

Some algorithms require you to calculate a subtotal, a total, or an average. You make these calculations using a repetition structure that includes a counter, an accumulator, or both. A **counter** is a numeric variable used for counting something, such as the number of employees paid in a week. An **accumulator** is a numeric variable used for accumulating (adding together) something, such as the total dollar amount of a week's payroll.

Two tasks are associated with counters and accumulators: initializing and updating. **Initializing** means to assign a beginning value to the counter or accumulator. Typically, counters and accumulators are initialized to the number 0. However, they can be initialized to any number, depending on the value required by the algorithm. The initialization task is performed before the loop is processed because it needs to be performed only once.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Updating refers to the process of either adding a number to (called **incrementing**) or subtracting a number from (called **decrementing**) the value stored in the counter or accumulator. The number can be either positive or negative, integer or noninteger. A counter is always updated by a constant value—typically the number 1. An accumulator, on the other hand, is usually updated by a value that varies. Accumulators are normally updated by incrementing rather than by decrementing. The assignment statement that updates a counter or an accumulator is placed in the body of a loop because the update task must be performed each time the loop body instructions are processed.

Ch07-Eddie

Game programs make extensive use of counters and accumulators. The partial game program shown in Figure 7-13 uses a counter to keep track of the number of smiley faces that Eddie (the character in the figure) destroys. After he destroys three smiley faces and then jumps through the manhole, he advances to the next level in the game, as shown in the figure.

Problem specification
To advance to the next level in the game, Eddie must destroy the three smiley faces by jumping on each one. He then must jump through the manhole.

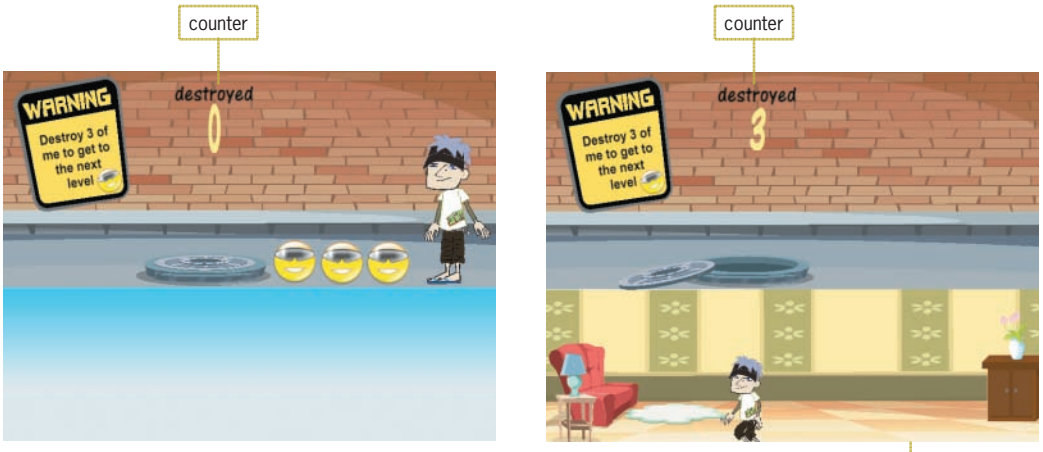


Figure 7-13 Example of a partial game program that uses a counter
Image by Diane Zak; created with Reallusion CrazyTalk Animator

Figure 7-14 shows two versions of the syntax for updating counters and two versions of the syntax for updating accumulators. Both versions of the syntax for updating counters tell the computer to add (or subtract) the *constantValue* to (from) the *counterVariable* first, and then place the result back in the *counterVariable*. Likewise, both versions of the syntax for updating accumulators tell the computer to add (or subtract) the *value* to (from) the *accumulatorVariable* first and then place the result back in the *accumulatorVariable*.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

HOW TO Update Counters and Accumulators

Syntax

counterVariable = *counterVariable* {+ | –} *constantValue;*
counterVariable {+= | –=} *constantValue;*

accumulatorVariable = *accumulatorVariable* {+ | –} *value;*
accumulatorVariable {+= | –=} *value;*

Counter examples

```
years = years + 1;
years += 1;
evenNum = evenNum - 2;
evenNum -= 2;
```

Accumulator examples

```
sum = sum + num;
sum += num;
total = total + score;
total += score;
```

Figure 7-14 Syntax and examples of update statements for counters and accumulators

In the next section, you will view a program that uses a counter, an accumulator, and a repetition structure.

The Stock Price Program

Figure 7-15 shows the problem specification, IPO chart information, and C++ instructions for a program that gets stock prices from the user. The program calculates the average stock price and displays the result on the computer screen. Figure 7-16 shows the corresponding flowchart. The program uses a counter (an `int` variable named `numPrices`) to keep track of the number of stock prices the user enters; the counter variable is initialized to 0. The program also uses an accumulator (a `double` variable named `totalPrices`) to add together (accumulate) the stock prices; the accumulator variable is initialized to 0.0.

Problem specification
Create a program that allows the user to enter the closing price of a specific stock for any number of days. Use a negative number as the sentinel value. If the sentinel value is the first price the user enters, display the "No stock prices entered" message on the screen. Otherwise, use a counter to keep track of the number of prices entered and an accumulator to total the prices. When the user has finished entering the prices, calculate the average price by dividing the accumulator's value by the counter's value, and then display the average price on the screen.

Figure 7-15 Problem specification, IPO chart information, and C++ instructions for the stock price program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

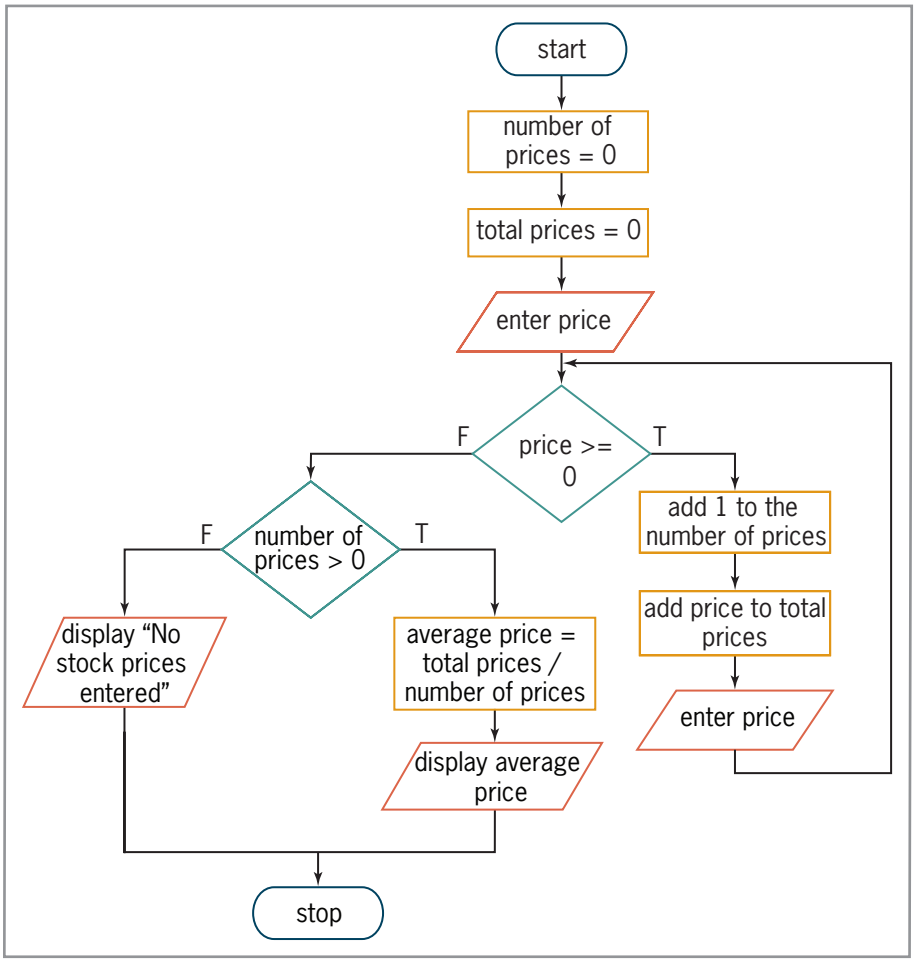
(continued)

IPO chart information	C++ instructions
Input	
price	`double price = 0.0;`
Processing	
number of prices (counter)	`int numPrices = 0;`
total prices (accumulator)	`double totalPrices = 0.0;`
Output	
average price	`double avgPrice = 0.0;`
Algorithm	
1. enter the price	`cout << "Closing price (negative number to stop): ";`
	`cin >> price;`
2. repeat while (the price is at least 0)	`while (price >= 0.0)`
	`{`
add 1 to the number of prices	`numPrices += 1;;`
add the price to the total prices	`totalPrices += price;`
enter the price	`cout << "Closing price (negative number to stop): ";` `cin >> price;`
end repeat	`}  //end while`
3. if (the number of prices is greater than 0)	`if (numPrices > 0)`
	`{`
calculate the average price by dividing the total prices by the number of prices	`avgPrice = totalPrices / numPrices;`
display the average price	`cout << "Average price: $" << avgPrice << endl;`
	`}`
else	`else`
display "No stock prices entered" message	`cout << "No stock prices entered" << endl;`
end if	`//end if`

The loop in Figure 7-15 will stop when the stock price is less than 0.0 because *less than* is the opposite of *greater than or equal to*.

Figure 7-15 Problem specification, IPO chart information, and C++ instructions for the stock price program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.



The "enter price" parallelogram above the loop represents the priming read, and the one within the loop represents the update read.

Figure 7-16 Flowchart for the stock price program

Ch07-Stock Price

You can observe the way counters and accumulators are used in a program by desk-checking the code shown earlier in Figure 7-15. You will do this using 78.75 and 80.05 as the stock prices and –3 as the sentinel value. The average price should be $79.40.

After declaring and initializing the appropriate variables, the code prompts the user to enter the first price and then stores the user's response in the `price` variable. The `while (price >= 0.0)` clause begins a pretest loop that repeats the loop body instructions as long as (or while) the `price` variable contains a value that is greater than or equal to 0.0. The loop stops when the `price` variable contains a sentinel value, which is any value that is less than 0.0. (Unlike the loop in the commission program shown earlier in Figure 7-11, the loop in the stock price program has more than one sentinel value.)

The current value in the `price` variable (78.75) is not less than 0.0, so the computer processes the instructions in the loop body. The first two instructions update the counter by 1 and the accumulator by the value in the `price` variable, respectively. The desk-check table in Figure 7-17 shows the updated values assigned to the counter and accumulator variables.

price	numPrices	totalPrices	avgPrice
~~0.0~~	~~0~~	~~0.0~~	0.0
78.75	1	78.75	

Figure 7-17 Desk-check table after the first update to the counter and accumulator variables

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The last two instructions in the loop body prompt the user to enter the next price and then store the user's response (80.05) in the `price` variable. Next, the condition in the `while` clause is reevaluated to determine whether the loop body instructions should be processed again (a true condition) or skipped over (a false condition). Here again, the loop's condition evaluates to true. As a result, the first two instructions in the loop body update the counter and accumulator. See Figure 7-18.

price	numPrices	totalPrices	avgPrice
~~0.0~~	~~0~~	~~0.0~~	0.0
~~78.75~~	~~1~~	~~78.75~~	
80.05	2	158.80	

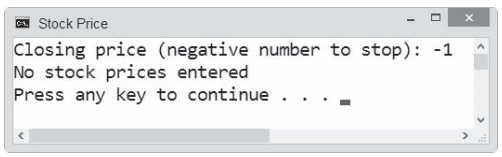
Figure 7-18 Desk-check table after the second update to the counter and accumulator variables

The last two instructions in the loop body prompt the user to enter the next price and then store the user's response (–3) in the `price` variable. The condition in the `while (price >= 0.0)` clause is then reevaluated. This time, the condition evaluates to false because the value in the `price` variable is not greater than or equal to 0.0. As a result, the loop body instructions are skipped over and the loop ends; processing continues with the `if` statement immediately following the loop.

The `if` statement's condition verifies that the value stored in the counter variable (`numPrices`) is greater than the number 0, which is the variable's initial value. This verification is necessary because the first instruction in the statement's true path uses the `numPrices` variable as the divisor when calculating the average price. Before using a variable as the divisor in an expression, you should always verify that the variable contains a value other than 0. Division by 0 is not mathematically possible and will cause the program to end abruptly with an error.

Currently, the `numPrices` variable contains the number 2. Therefore, the instructions in the `if` statement's true path calculate the average price (79.40) and then display that amount on the screen before the program ends. Figure 7-19 shows the completed desk-check table along with two sample runs of the program. (The program uses the `fixed` and `setprecision` stream manipulators to display the average price in fixed-point notation with two decimal places.)

price	numPrices	totalPrices	avgPrice
~~0.0~~	~~0~~	~~0.0~~	~~0.0~~
~~78.75~~	~~1~~	~~78.75~~	79.4
~~80.05~~	2	158.80	
-3.0			



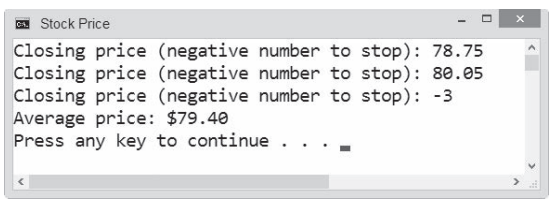


Figure 7-19 Completed desk-check table and sample runs of the stock price program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Mini-Quiz 7-3

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

1. Which of the following is updated by a constant value?
   a. accumulator
   b. counter
2. Write a C++ assignment statement that updates the `quantity` counter variable by 10.
3. Write a C++ assignment statement that updates the `total` counter variable by –5.
4. Write a C++ assignment statement that updates the `totalPurchases` accumulator variable by the value stored in the `purchases` variable.

Counter-Controlled Pretest Loops

In both the commission and stock price programs, the termination of the loop is determined by a sentinel value that is entered by the user at the keyboard. Other loops, like the one in the partial game program shown earlier in Figure 7-13, are controlled using a counter rather than a sentinel value; such loops are referred to as **counter-controlled loops**.

Figure 7-20 shows the problem specification, IPO chart information, and C++ instructions for a modified version of the stock price program. Unlike the loop in the previous version of the program, which allows the user to enter as many prices as needed, the loop in this version uses a counter (the `numDays` variable) to get only five prices from the user—one price for each of the five days.

Modified problem specification (original shown in Figure 7-15)
Create a program that allows the user to enter the closing price of a specific stock for each of five days. Use a counter to keep track of the number of days and an accumulator to total the five prices. Calculate the average price by dividing the accumulator's value by a number that is one less than the counter's value, and then display the average price on the screen.

IPO chart information

Input
price

Processing
number of days (counter: 1 to 5)
total prices (accumulator)

Output
average price

Algorithm
1. repeat while (the number of days is less than 6)
   enter the price

C++ instructions

```
double price = 0.0;

int numDays = 1;
double totalPrices = 0.0;

double avgPrice = 0.0;


while (numDays < 6)
{
    cout << "Day " << numDays <<
    " closing price: ";
    cin >> price;
```

Figure 7-20 Problem specification, IPO chart information, and C++ instructions for the modified stock price program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
      add 1 to the number of days          numDays += 1;;

      add the price to the total prices    totalPrices += price;
   end repeat                           }   //end while

2. calculate the average price by       avgPrice = totalPrices / (numDays - 1);
   dividing the total prices by (the
   number of days - 1)
3. display the average price            cout << "Average price: $" << avgPrice <<
                                        endl;
```

Figure 7-20 Problem specification, IPO chart information, and C++ instructions for the modified stock price program

The program initializes the `numDays` counter variable to the number 1, which corresponds to the first day. It also updates the variable by 1 (day) each time the loop instructions are processed. The initializing and updating of the counter variable in counter-controlled loops are comparable to the priming and update reads, respectively, in loops controlled by a sentinel value.

The `while (numDays < 6)` clause indicates that the loop instructions should be repeated as long as (or while) the number in the `numDays` counter variable is less than 6. The clause could also be written as `while (numDays <= 5)`. In either case, the loop will stop when the `numDays` variable contains the number 6, which occurs after the loop instructions are processed five times. Figure 7-21 shows the corresponding flowchart, and Figure 7-22 shows a completed desk-check table using the following five stock prices: 78.75, 80.05, 81.35, 79.95, and 80.10. Figure 7-22 also contains a sample run of the program.

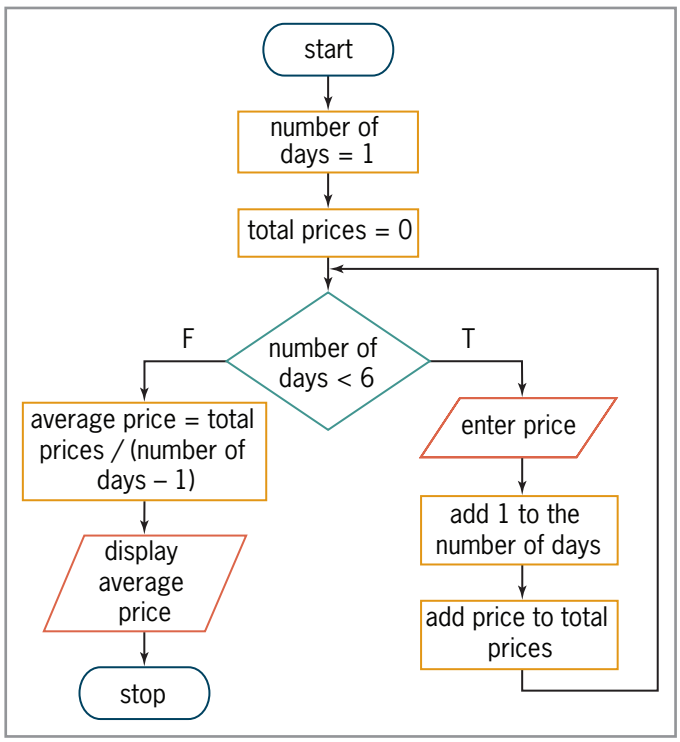


Figure 7-21 Flowchart for the modified stock price program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

price	numDays	totalPrices	avgPrice
~~0.0~~	~~1~~	~~0.0~~	~~0.0~~
~~78.75~~	~~2~~	~~78.75~~	80.04
~~80.05~~	~~3~~	~~158.80~~	
~~81.35~~	~~4~~	~~240.15~~	
~~79.95~~	~~5~~	~~320.10~~	
80.10	6	400.20	

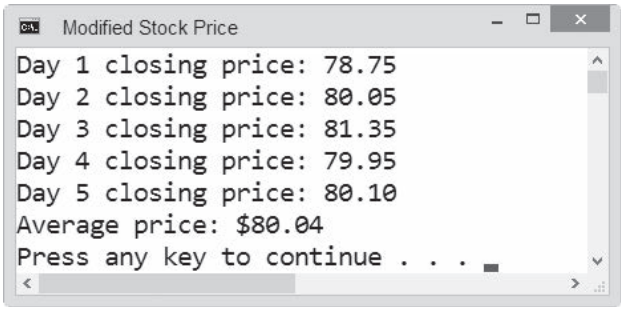


Figure 7-22 Completed desk-check table and sample run for the modified stock price program

For more examples of using the `while` statement, see the Using the `while` Statement section in the Ch07WantMore.pdf file.

The for Statement

Besides using the `while` statement to code pretest loops, you can also use the **`for` statement**. However, the most common use of the `for` statement is to code counter-controlled pretest loops because it provides a more compact and clearer way of writing that type of loop. As Figure 7-23 shows, the statement's `for` clause contains three arguments separated by two semicolons; the first and third arguments are optional.

HOW TO Use the for Statement

Syntax

semicolons

```
for ([initialization]; condition; [update])
    either one statement or a statement block to be processed as long as
    the condition is true
//end for
```

Example 1: displays the numbers 1, 2, and 3 on separate lines on the screen

```
for (int x = 1; x < 4; x += 1)
    cout << x << endl;
//end for
```

Example 2: displays the numbers 3, 2, and 1 on separate lines on the screen

```
for (int x = 3; x > 0; x -= 1)
{
    cout << x << endl;
}  //end for
```

Note: The *condition* and *update* arguments in Example 1 can also be phrased as `x <= 3` and `x = x + 1`, respectively. In Example 2, the *condition* and *update* arguments can also be phrased as `x >= 1` and `x = x - 1`, respectively.

Using commas rather than semicolons is a common error made when typing the `for` clause.

Figure 7-23 How to use the `for` statement

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

In most `for` clauses, the *initialization* argument creates and initializes a counter variable that the computer uses to keep track of the number of times the loop body instructions are processed. The variable is local to the `for` statement, which means it can be used only within the statement's loop body. The variable will be removed from the computer's internal memory when the loop ends.

The *condition* argument in the `for` clause is a looping condition because it specifies the requirement for repeating the loop instructions.

The *condition* argument in the `for` clause specifies the condition that must be true for the loop body instructions to be processed. The condition must be a Boolean expression, which is an expression that evaluates to either true or false. The expression can contain variables, constants, functions, arithmetic operators, comparison operators, and logical operators. The loop stops when its condition evaluates to false. The `for` clause's *update* argument usually contains an expression that updates the counter variable specified in the *initialization* argument.

Following the `for` clause is the loop body, which contains the one or more statements that you want the computer to repeat. If the loop body contains more than one statement, the statements must be entered as a statement block by enclosing them in a set of braces (`{}`). However, you can also include the braces even when the loop body contains only one statement, as shown in Example 2 in Figure 7-23.

Ch07-for Statement

Figure 7-24 describes the way the computer processes the code shown in Example 1 in Figure 7-23. The `for` statement in the example ends when the `x` variable contains the number 4 because that is the first integer that is not less than 4.

Example 1: displays the numbers 1, 2, and 3 on separate lines on the screen

```
for (int x = 1; x < 4; x += 1)
    cout << x << endl;
//end for
```

Processing steps

1. The *initialization* argument (`int x = 1`) creates a variable named `x` and initializes it to 1.
2. The *condition* argument (`x < 4`) checks whether the `x` variable's value is less than 4. It is, so the statement in the loop body displays the `x` variable's value (1) on the screen.
3. The *update* argument (`x += 1`) adds 1 to the contents of the `x` variable, giving 2.
4. The *condition* argument checks whether the `x` variable's value is less than 4. It is, so the statement in the loop body displays the `x` variable's value (2) on the screen.
5. The *update* argument adds `1` to the contents of the `x` variable, giving 3.
6. The *condition* argument checks whether the `x` variable's value is less than 4. It is, so the statement in the loop body displays the `x` variable's value (3) on the screen.
7. The *update* argument adds 1 to the contents of the `x` variable, giving 4.
8. The *condition* argument checks whether the `x` variable's value is less than 4. It's not, so the `for` loop ends. Processing continues with the statement following the end of the loop.

Figure 7-24 Processing steps for Example 1's code

In the remaining sections in this chapter, you will view four programs that use the `for` statement.

The Total Payroll Program

Figure 7-25 shows the problem specification, IPO chart information, and C++ instructions for a program that displays a company's total payroll. The `for` clause's *condition* argument could also be written as `numStores < 4`.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Problem specification
Create a program that allows the user to enter the payroll amount for each of a company's three stores. The program should calculate the total payroll and then display the result on the screen. Use a counter to ensure that the user enters exactly three payroll amounts. Use an accumulator to total the amounts.

IPO chart information

Input
store's payroll

Processing
number of stores (counter: 1 to 3)

Output
total payroll (accumulator)

Algorithm
1. repeat for (number of stores from 1 to 3)
   enter the store's payroll
   add the store's payroll to the total payroll
   end repeat
2. display the total payroll

C++ instructions

```
int storePayroll = 0;
```

this variable is created and initialized in the for clause

```
int totalPayroll = 0;

for (int numStores = 1; numStores
<= 3; numStores += 1)
{
    cout << "Store " << numStores
    << " payroll: ";
    cin >> storePayroll;
    totalPayroll += storePayroll;
}   //end for
cout << "Total payroll: $" <<
totalPayroll << endl;
```

Figure 7-25 Problem specification, IPO chart information, and C++ instructions for the total payroll program

The loop in Figure 7-25 will stop when the value in the `numStores` variable is greater than 3 because *greater than* is the opposite of *less than or equal to*.

Desk-checking the code shown in Figure 7-25 will help you understand how the `for` statement works. You will desk-check the code using the following three payroll amounts: 15000, 25000, and 50000.

Ch07-Payroll

First, the code declares and initializes two `int` variables named `storePayroll` and `totalPayroll`. Next, the `for` clause's *initialization* argument creates an `int` variable named `numStores` and initializes the variable to the number 1. The *initialization* argument is processed only once, at the beginning of the loop. Figure 7-26 shows the desk-check table after the declaration statements and *initialization* argument have been processed.

storePayroll	totalPayroll	numStores
0	0	1

Figure 7-26 Results of processing the declaration statements and *initialization* argument

Next, the `for` clause's *condition* argument is evaluated to determine whether the instructions in the loop body should be processed (a true condition) or skipped over (a false condition). Notice that, like the condition in a `while` statement, the condition in a `for` statement is evaluated *before* the loop body instructions are processed. At this point, the `numStores <= 3` condition evaluates to true because the value in the `numStores` variable (1) is less than 3. As a result, the computer processes the three statements contained in the body of the loop. Those statements prompt the user to enter the amount of Store 1's payroll, then store the user's response (15000) in the `storePayroll` variable, and finally add the value in the `storePayroll` variable (15000) to the value in the `totalPayroll` accumulator variable (0); the sum of both numbers is 15000.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The `for` clause's *update* argument is processed next. The *update* argument adds the number 1 to the value in the `numStores` variable, giving 2. Figure 7-27 shows the desk-check table after the *update* argument is processed the first time.

storePayroll	totalPayroll	numStores
~~0~~	~~0~~	~~1~~
15000	15000	2

Figure 7-27 Desk-check table after the *update* argument is processed the first time

The `for` clause's *condition* argument is reevaluated to determine whether the loop body instructions should be processed again or skipped over. Unlike the *initialization* argument, which is processed only once, the *condition* argument is processed with each repetition (or iteration) of the loop. Currently, the `numStores` variable contains the number 2. Therefore, the `numStores <= 3` condition evaluates to true, and the statements in the loop body are processed again. Those statements prompt the user to enter the amount of Store 2's payroll, then store the user's response (25000) in the `storePayroll` variable, and finally add the value in the `storePayroll` variable (25000) to the value in the `totalPayroll` accumulator variable (15000); the sum of both numbers is 40000.

The `for` clause's *update* argument is processed next. Like the *condition* argument, the *update* argument is processed with each repetition of the loop. The *update* argument adds the number 1 to the value in the `numStores` variable, giving 3. Figure 7-28 shows the desk-check table after the *update* argument is processed the second time.

storePayroll	totalPayroll	numStores
~~0~~	~~0~~	~~1~~
~~15000~~	~~15000~~	~~2~~
25000	40000	3

Figure 7-28 Desk-check table after the *update* argument is processed the second time

The `for` clause's *condition* argument is reevaluated again. The condition evaluates to true because the `numStores` variable contains the number 3. Therefore, the statements in the loop body are processed again. Those statements prompt the user to enter the amount of Store 3's payroll, then store the user's response (50000) in the `storePayroll` variable, and finally add the value in the `storePayroll` variable (50000) to the value in the `totalPayroll` accumulator variable (40000); the sum of both numbers is 90000. The `for` clause's *update* argument then adds the number 1 to the value in the `numStores` variable, giving 4. Figure 7-29 shows the desk-check table after the *update* argument is processed the third (and last) time.

storePayroll	totalPayroll	numStores
~~0~~	~~0~~	~~1~~
~~15000~~	~~15000~~	~~2~~
~~25000~~	~~40000~~	~~3~~
50000	90000	4

Figure 7-29 Desk-check table after the *update* argument is processed the third time

The `for` clause's *condition* argument is reevaluated again. At this point, the condition evaluates to false because the `numStores` variable contains the number 4. Therefore, the instructions in the loop body are skipped over, and the loop ends. As a result, the computer removes the `for` statement's local variable, `numStores`, from internal memory. Processing continues with the

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

instruction located immediately below the end of the loop. Notice that the loop stops when the `numStores` variable contains the number 4, which is the first integer that is not less than or equal to 3. Figure 7-30 shows a sample run of the total payroll program.

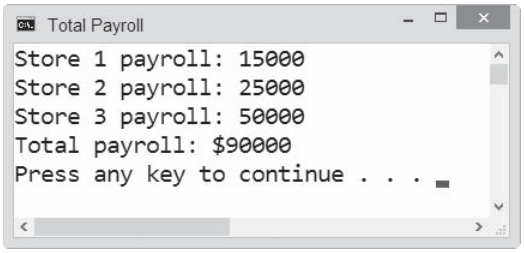


Figure 7-30 A sample run of the total payroll program

The Tip Program

Figure 7-31 shows the problem specification, IPO chart information, and C++ instructions for a program that displays three suggested amounts to tip a waiter on a restaurant bill. Notice that the `for` statement's `rate` variable counts from 0.1 to 0.2 in increments of 0.05.

Problem specification
Create a program that allows the user to enter the amount of a restaurant bill. The program should calculate and display the suggested amounts to tip a waiter using rates of 10%, 15%, and 20%. Use a counter to keep track of the three rates.

IPO chart information

Input
bill
rate (counter: 10% to 20% in increments of 5%)

Processing
none

Output
tip

Algorithm
1. enter the bill

2. repeat for (rate from 10% to 20% in increments of 5%)
   calculate tip by multiplying the bill by the rate

   display the rate and tip

   end repeat

C++ instructions

```
double bill = 0.0;
```
this variable is created and initialized in the for clause

```
double tip = 0.0;

cout << "Bill amount: ";
cin >> bill;

for (double rate = 0.1;
rate <= 0.2; rate += 0.05)
{
    tip = bill * rate;

    cout << rate * 100 << "%
    tip: ";
    cout << "$" << tip << endl;
}   //end for
```

Figure 7-31 Problem specification, IPO chart information, and C++ instructions for the tip program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Figure 7-32 shows the completed desk-check table using a restaurant bill of $90.50. Notice that the `for` loop ends when the value stored in the `rate` variable is 0.25. The figure also includes a sample run of the program.

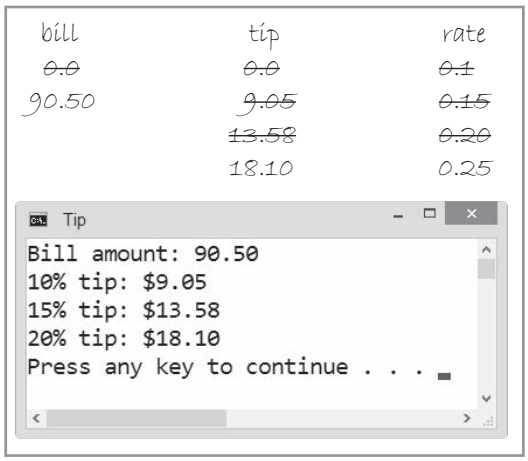


Figure 7-32 Desk-check table and a sample run of the tip program

Many programmers use a hexagon, which is a six-sided figure, to represent the `for` clause in a flowchart; the hexagon contains four items. Going counterclockwise from the top of the hexagon shown in Figure 7-33, the four items are the name of the counter variable (`rate`), the variable's initial value (0.1), the value used to update the variable (0.05), and the last value for which the condition will evaluate to true (0.2). The <= sign that precedes the 0.2 indicates that the loop body instructions will be processed as long as the `rate` variable's value is less than or equal to 0.2.

Some programmers use the hexagon to represent any counter-controlled loop, even those coded with the `while` statement.

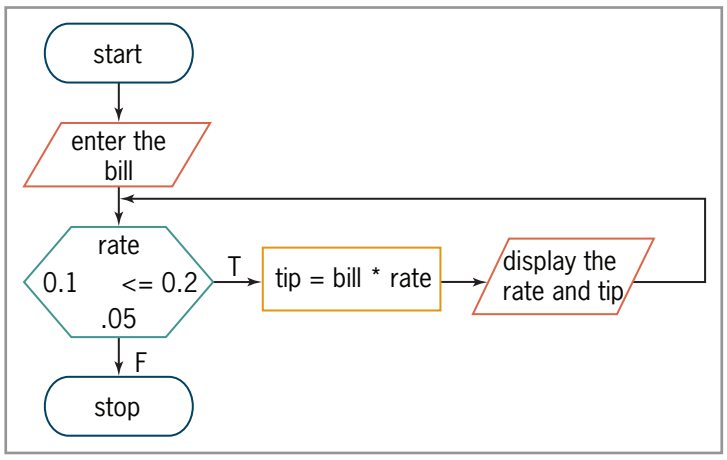


Figure 7-33 Tip program's algorithm shown in flowchart form

Another Version of the Commission Program

Although the `for` statement is more commonly used to code counter-controlled loops, it can be used to code any pretest loop in C++, even the pretest loop in the commission program that you viewed earlier in the chapter. That program's loop calculates and displays the commission amount for as many salespeople as needed without having to run the program again. The loop stops when the user enters –1 (the sentinel value). Figure 7-34 shows how you would code the loop using the `for` statement rather than the `while` statement used in Figure 7-11. The modifications made to the original code are shaded in the figure.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Problem specification
Create a program that calculates the commission for each of a company's salespeople. The commission is calculated by multiplying the salesperson's sales amount by 20%.

IPO chart information

Input
commission rate (20%)
sales

Processing
none

Output
commission

Algorithm
1. enter the sales

2. repeat while (the sales are not equal to -1)
    calculate the commission by multiplying the sales by the commission rate
    display the commission
    enter the sales
   end repeat

C++ instructions

```
const double RATE = 0.2;
int sales = 0;

double commission = 0.0;

cout << "First salesperson's
sales (-1 to stop): ";
cin >> sales;
for (; sales != -1;)
{
    commission = sales * RATE;

    cout << "Commission: $"
    << commission << endl << endl;

    cout << "Next salesperson's
    sales (-1 to stop): ";
    cin >> sales;
}   //end for
```

semicolon after the *initialization* argument
condition argument
semicolon after the *condition* argument

Figure 7-34 IPO chart information and modified C++ instructions for the commission program

Unlike the `for` statements in the total payroll and tip programs, the `for` statement in the commission program contains only the *condition* argument and is controlled by the user at the keyboard instead of by a counter. Although the *initialization* and *update* arguments are omitted from the `for` clause, the semicolons after the *initialization* and *condition* arguments must be included. Figure 7-35 lists the steps the computer follows when processing the program's code using the following sales amounts: 1200, 800, and –1 (the sentinel value). The figure also includes a sample run of the program.

Processing steps
1. The `const` statement creates the `RATE` named constant and initializes it to 0.2.
2. The declaration statements create the `sales` and `commission` variables and initialize them to 0 and 0.0, respectively.
3. The first `cout` statement prompts the user to enter the first sales amount, and the `cin` statement stores the user's response (1200) in the `sales` variable.
4. The `for` clause's *condition* argument (`sales != -1`) checks whether the `sales` variable's value is not equal to –1. The condition evaluates to true, so the statements in the loop body

Figure 7-35 Processing steps and a sample run of the commission program shown in Figure 7-34 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

calculate and display the commission (240). They also prompt the user for the next sales amount and store the user's response (800) in the `sales` variable.

5. The `for` clause's *condition* argument checks whether the `sales` variable's value is not equal to –1. The condition evaluates to true, so the statements in the loop body calculate and display the commission (160). They also prompt the user for the next sales amount and store the user's response (–1) in the `sales` variable.
6. The `for` clause's *condition* argument checks whether the `sales` variable's value is not equal to –1. In this case, the condition evaluates to false, so the `for` loop ends. Processing continues with the statement following the end of the loop.

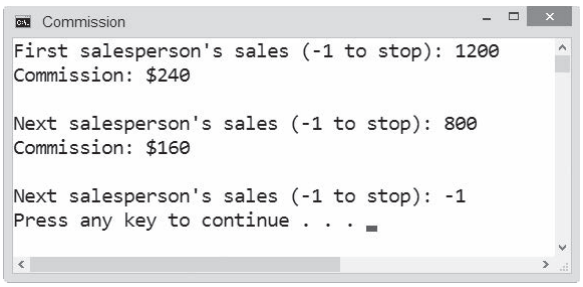


Figure 7-35 Processing steps and a sample run of the commission program shown in Figure 7-34

Whether you use the `for` statement or the `while` statement to code the pretest loop in the commission program is a matter of personal preference. However, most programmers use the `for` statement only when they know the exact number of times they want the loop instructions repeated. For all other pretest loops, they typically use the `while` statement.

The Even Integers Program

Figure 7-36 shows the problem specification for the even integers program, which should calculate and display the sum of three even integers entered by the user. Because the program needs to calculate the sum of three numbers, it may seem logical to use the `for` statement to code the program's loop, as shown in the figure. However, as the sample runs of the program show, the `for` loop will not give you the correct results if at least one of the numbers entered by the user is not even. This is because the `for` loop's *update* argument will increase the counter variable (`x`) by 1 whether the user's entry is even or odd.

Problem specification
Create a program that calculates and displays the sum of three even integers entered by the user.

Incorrect code using the `for` statement

```
for (int x = 1; x < 4; x += 1)
{
    cout << "Enter an even integer: ";
    cin >> evenInteger;
    if (evenInteger % 2 == 0)
        sum += evenInteger;
    else
        cout << "Please enter an even number." << endl << endl;
   //end if
} //end for
cout << "Sum of the three even integers: " << sum << endl;
```

Figure 7-36 Problem specification, code, and sample runs of the incorrect even integers program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

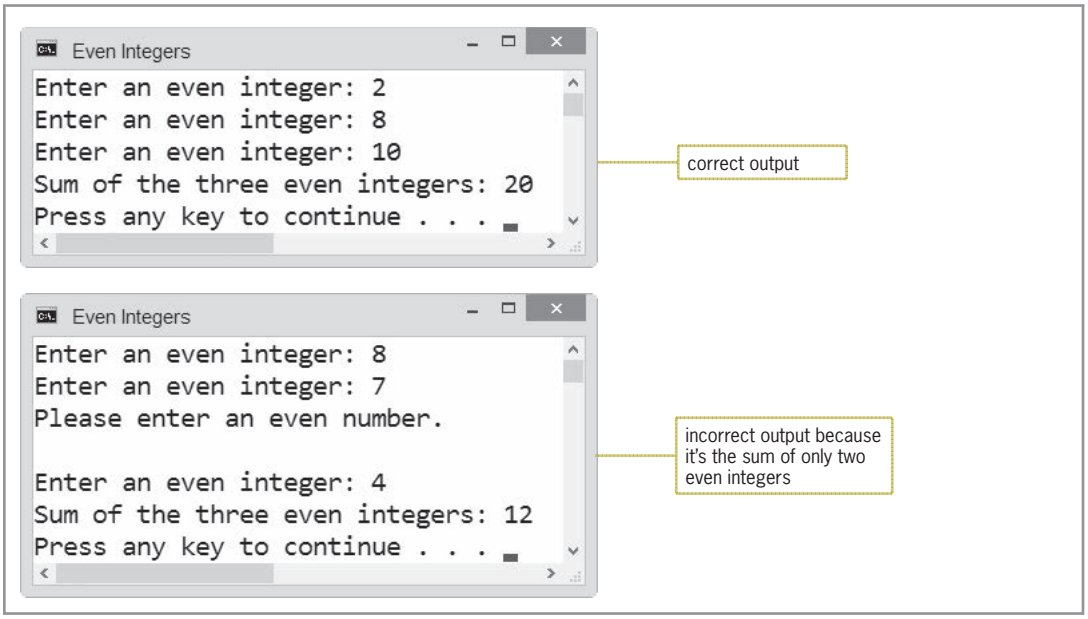


Figure 7-36 Problem specification, code, and sample runs of the incorrect even integers program

You may be tempted to fix the code by modifying the `else` portion of the `if` statement, as shown in Figure 7-37. However, manipulating the value of a `for` statement's counter variable should be avoided because it can lead to errors that are difficult to locate. A better way to fix the program is to change the `for` statement to a `while` statement, as shown in the figure.

Incorrect way to fix the program

```
else
{    x -= 1;
     cout << "Please enter an even number." << endl << endl;
}//end if
```

don't modify the value in a `for` statement's counter variable

Correct way to fix the program

```
int x = 1;
while (x < 4)
{
    cout << "Enter an even integer: ";
    cin >> evenInteger;
    if (evenInteger % 2 == 0)
    {
        sum += evenInteger;
        x += 1;
    }
    else
        cout << "Please enter an even number." << endl << endl;
    //end if
}  //end while
cout << "Sum of the three even integers: " << sum << endl;
```

replace the `for` statement with the `while` statement

For more examples of using the `for` statement, see the Using the `for` Statement section in the Ch07WantMore.pdf file.

Figure 7-37 Incorrect and correct ways to fix the even integers program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

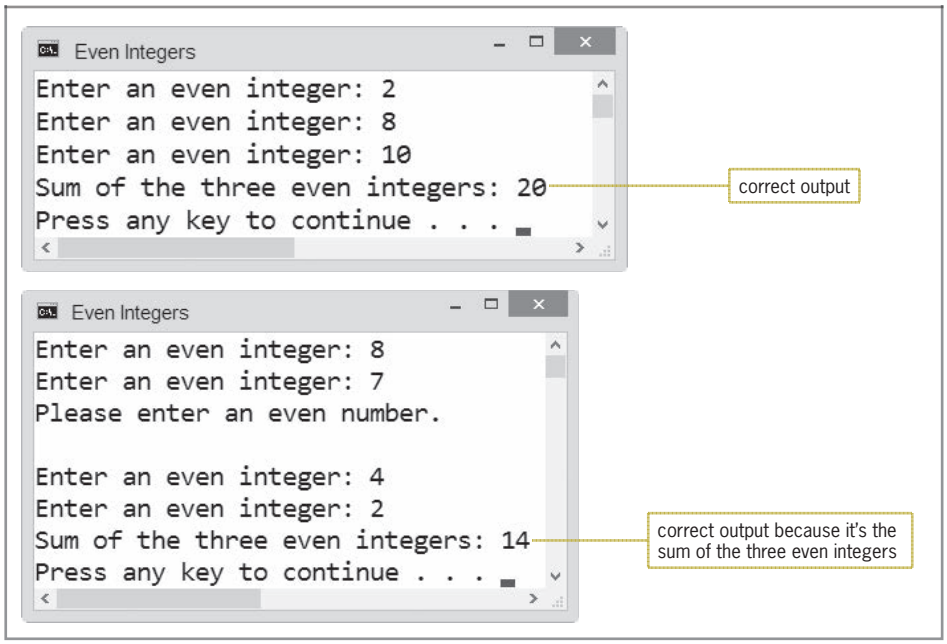


Figure 7-37 Incorrect and correct ways to fix the even integers program

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 7-4

1. A program declares an `int` variable named `evenNum` and initializes it to 2. Write a C++ `while` loop that uses the `evenNum` variable to display the even integers between 1 and 9.
2. Which of the following `for` clauses processes the loop instructions as long as the `x` variable's value is less than or equal to the number 100?

   a. `for (int x = 10; x <= 100; x = x + 10)`
   b. `for (int x = 10, x <= 100, x = x + 10)`
   c. `for (int x == 10; x <= 100; x = x + 10)`
   d. `for (int x = x + 10; x <= 100; x = 10)`

3. The computer will stop processing the loop associated with the `for` clause from Question 2 when the `x` variable contains the number _____.

   a. 100
   b. 111
   c. 101
   d. 110

4. Write a `for` clause that processes the loop instructions as long as the value stored in the `x` variable is greater than the number 0. The `x` variable should be an `int` variable. Initialize the variable to the number 25, and update it by –5 with each repetition of the loop.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

5. The computer will stop processing the loop associated with the `for` clause from Question 4 when the `x` variable contains the number _____.

6. Write a `for` statement that displays the even integers between 2 and 9 (inclusive) on the screen. Use `num` as the name of the counter variable.

LAB 7-1 Stop and Analyze

Study the program shown in Figure 7-38, and then answer the questions. The program calculates and displays the average number of text messages sent each day for seven days.

The answers to the labs are contained in the Answers.pdf file.

```
 1 //Lab7-1.cpp - calculates the average number of text
 2 //messages sent each day for 7 days
 3 //Created/revised by <your name> on <current date>
 4
 5 #include <iostream>
 6 #include <iomanip>
 7 using namespace std;
 8
 9 int main()
10 {
11    int day = 0;
12    int totalTexts = 0;
13    int dailyTexts = 0;
14    double average = 0.0;
15
16    for (day = 1; day < 8; day += 1)
17    {
18        cout << "How many text messages did you send on day "
19            << day << "? ";
20        cin >> dailyTexts;
21        totalTexts += dailyTexts;
22    }   //end for
23
24    average = static_cast<double>(totalTexts) / (day - 1);
25    cout << fixed << setprecision(0);
26    cout << endl << "You sent approximately "
27            << average << " text messages per day." << endl;
28    return 0;
29 }  //end of main function
```

Figure 7-38 Code for Lab 7-1

QUESTIONS

1. What is the name of the program's counter variable? What is the name of its accumulator variable?

2. What is another way of phrasing the `for` clause's *condition* argument?

3. Why is the `for` statement's counter variable declared at the beginning of the program (in Line 11) rather than in the `for` clause? Would the program work correctly if the variable was declared in the `for` clause?

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

4. Desk-check the program using the following data: 76, 80, 100, 43, 68, 70, and 79. What is the average number of text messages, rounded to the nearest whole number?
5. Follow the instructions for starting C++ and viewing the Lab7-1.cpp file, which is contained in either the Cpp8\Chap07\Lab7-1 Project folder or the Cpp8\Chap07 folder. (Depending on your C++ development tool, you may need to open Lab 7-1's project/solution file first.) Run the program, and then enter the data used to desk-check the program in Question 4. What does the program display as the average number of text messages? (The program's output should agree with the results of your desk-check from Question 4.)
6. Change Line 11 to a comment. Also change the *initialization* argument in the `for` clause to `int day = 1`. Save the program. Then, depending on the C++ development tool you are using, either build or compile the program. Explain why the build or compile operation was unsuccessful.
7. Delete the two forward slashes from Line 11. Also delete the `int day = 1` *initialization* argument from the `for` clause.
8. Save and then run the program. Enter your own data. Does the program work correctly? If not, modify the code appropriately, and then save and run the program again. Use the data from Question 4 to test the program.

LAB 7-2 Plan and Create

In this lab, you will plan and create an algorithm for the problem specification shown in Figure 7-39.

Problem specification
Create a program that displays the number of years required for a company's sales amount to reach at least $150,000, using the current year's sales amount and a 5.5% growth rate per year. The program should also display the sales amount at that time.

Figure 7-39 Problem specification for Lab 7-2

First, analyze the problem, looking for the output first and then for the input. In this case, the user wants the program to display two items: the number of years it will take for a company's sales to reach at least $150,000, and the sales amount at that time. The input is the growth rate and the sales amount for the current year. The program will use one processing item: the annual increase.

Next, plan the algorithm. Recall that most algorithms begin with an instruction to enter the input items into the computer, followed by instructions that process the input items, typically including the items in one or more calculations. Most algorithms end with one or more instructions that display, print, or store the output items. Figure 7-40 shows the completed IPO chart for the sales problem.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Input	Processing	Output
growth rate (5.5%) sales	Processing items: annual increase Algorithm: 1. enter the sales 2. repeat while (sales < 150000) calculate the annual increase by multiplying the sales by the growth rate add the annual increase to the sales add 1 to the number of years end repeat 3. display the number of years and the sales	number of years (counter) sales

Figure 7-40 Completed IPO chart for the sales problem

Recall that the third step in the problem-solving process is to desk-check the algorithm. Figure 7-41 shows the completed desk-check table using 125000 as the current sales amount. According to the table, it will take four years for the company to reach its sales goal of at least $150,000.

growth rate	sales	annual increase	number of years
0.055	~~125000.0~~	~~6875.0~~	~~1~~
	~~131875.0~~	~~7253.12~~	~~2~~
	~~139128.12~~	~~7652.05~~	~~3~~
	~~146780.17~~	8072.91	4
	154853.08		

Figure 7-41 Completed desk-check table for the sales problem's algorithm

The fourth step in the problem-solving process is to code the algorithm into a program. You begin by declaring memory locations that will store the values of the input, processing (if any), and output items. The growth rate will be stored in a `double` named constant because its value (0.055) will not change while the program is running. The remaining input, processing, and output items will be stored in variables to allow their values to change during runtime. The sales and annual increase amounts may contain a decimal point, so their variables will be declared using the `double` data type. The number of years will always be an integer, so its variable will be declared using the `int` data type. Figure 7-42 shows the IPO chart information and corresponding C++ instructions.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

IPO chart information	C++ instructions
Input growth rate (5.5%) sales	`const double GROWTH_RATE = 0.055;` `double sales = 0.0;`
Processing annual increase	`double annualIncrease = 0.0;`
Output number of years (counter) sales	`int years = 0;` declared in the Input section

Algorithm
1. enter the sales
2. repeat while (sales < 150000)
calculate the annual increase by multiplying the sales by the growth rate
add the annual increase to the sales
add 1 to the number of years
end repeat
3. display the number of years and the sales
	`cout << "Current year's sales: ";`
	`cin >> sales;`
	`while (sales < 150000.0)`
	`{`
	`annualIncrease = sales * GROWTH_RATE;`
	`sales += annualIncrease;`
	`years += 1;`
	`} //end while`
	`cout << "Sales " << years << " years from now: $" << sales << endl;`

Figure 7-42 IPO chart information and C++ instructions for the sales program

The fifth step in the problem-solving process is to desk-check the program. You begin by placing the names of the declared variables and named constants (if any) in a new desk-check table, along with their initial values. You then desk-check the remaining C++ instructions in order, recording in the desk-check table any changes made to the variables. Figure 7-43 shows the completed desk-check table for the sales program. The results agree with those shown in the algorithm's desk-check table in Figure 7-41.

GROWTH_RATE	sales	annualIncrease	years
0.055	~~0.0~~	~~0.0~~	~~0~~
	~~125000.0~~	~~6875.0~~	~~1~~
	~~131875.0~~	~~7253.12~~	~~2~~
	~~139128.12~~	~~7652.05~~	~~3~~
	~~146780.17~~	8072.91	4
	154853.08		

Figure 7-43 Completed desk-check table for the sales program

The final step in the problem-solving process is to evaluate and modify (if necessary) the program. Recall that you evaluate a program by entering its instructions into the computer and

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

then using the computer to run (execute) it. While the program is running, you enter the same sample data used when desk-checking the program.

DIRECTIONS

1. Follow the instructions for starting your C++ development tool. Depending on the development tool you are using, you may need to create a new project; if so, name the project Lab7-2 Project, and save it in the Cpp8\Chap07 folder. Enter the instructions shown in Figure 7-44 in a source file named Lab7-2.cpp. (Do not enter the line numbers.) Save the file in either the project folder or the Cpp8\Chap07 folder. Now follow the appropriate instructions for running the Lab7-2.cpp file. Test the program using 125000 as the current sales amount. If necessary, correct any bugs (errors) in the program.

```
 1 //Lab7-2.cpp - Displays the number of years required
 2 //for a company's sales to reach at least $150,000
 3 //using a 5.5% annual growth rate. Also displays
 4 //the sales at that time.
 5 //Created/revised by <your name> on <current date>
 6
 7 #include <iostream>
 8 #include <iomanip>
 9 using namespace std;
10
11 int main()
12 {
13    const double GROWTH_RATE = 0.055;
14    double sales = 0.0;
15    double annualIncrease = 0.0;
16    int years = 0;
17
18    cout << "Current year's sales: ";
19    cin >> sales;
20    while (sales < 150000.0)
21    {
22        annualIncrease = sales * GROWTH_RATE;
23        sales += annualIncrease;
24        years += 1;
25    }  //end while
26
27    cout << fixed << setprecision(0);
28    cout << "Sales " << years << " years from now: $"
29        << sales << endl;
30
31    return 0;
32 }  //end of main function
```

Figure 7-44 Sales program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

LAB 7-3 Modify

If necessary, create a new project named Lab7-3 Project, and save it in the Cpp8\Chap07 folder. Enter (or copy) the Lab7-1.cpp instructions into a new source file named Lab7-3.cpp. Change Lab7-1.cpp in the first comment to Lab7-3.cpp. Change the `for` statement to a `while` statement. Test the program using the following data: 76, 80, 100, 43, 68, 70, and 79.

LAB 7-4 What's Missing?

The program in this lab should display the average electric bill. Start your C++ development tool, and view the Lab7-4.cpp file, which is contained in either the Cpp8\Chap07\Lab7-4 Project folder or the Cpp8\Chap07 folder. (Depending on your C++ development tool, you may need to open Lab7-4's project/solution file first.) Put the C++ instructions in the proper order, and then determine the one or more missing instructions. Test the program using the following monthly electric bills: 124.89, 110.65, 99.43, 100.35, and –1 (the sentinel value).

LAB 7-5 Desk-Check

The code shown in Figure 7-45 should display the numbers 2, 4, 6, 8, 10, and 12. Desk-check the code. Did your desk-check reveal any errors in the code? If so, correct the code, and then desk-check it again.

```
for (int number = 2; number < 12; number += 2)
       cout << number << endl;
//end for
```

Figure 7-45 Code for Lab 7-5

LAB 7-6 Debug

Follow the instructions for starting C++ and viewing the Lab7-6.cpp file, which is contained in either the Cpp8\Chap07\Lab7-6 Project folder or the Cpp8\Chap07 folder. (Depending on your C++ development tool, you may need to open Lab 7-6's project/solution file first.) Run the program. When you are prompted to enter a price, type 15.45, and press Enter. The "Next price:" prompt appears over and over again in the Command Prompt window, as shown in Figure 7-46. This is a result of the computer repeatedly processing the `cout << "Next price: ";` statement contained in the body of the `while` loop, and it indicates that the program contains an endless (or infinite) loop. You can stop an endless loop by pressing Ctrl+c (press and hold down the Ctrl key as you tap the letter c, and then release both keys). Or, you can use the Close button on the Command Prompt window's title bar. Use either method to stop the endless loop. Debug the program and then test it appropriately.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

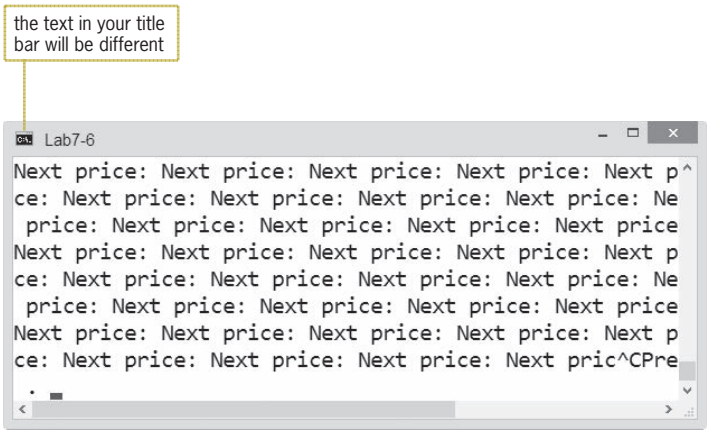


Figure 7-46 Command Prompt window showing that the program is in an endless loop

Chapter Summary

You use the repetition structure, also called a loop, when you need the computer to repeatedly process one or more program instructions.

A repetition structure can be either a pretest loop or a posttest loop. In a pretest loop, the loop condition is evaluated *before* the instructions within the loop are processed. In a posttest loop, which is covered in Chapter 8, the evaluation occurs *after* the instructions within the loop are processed. Of the two types of loops, the pretest loop is the most commonly used.

The condition appears at the beginning of a pretest loop and determines whether the instructions within the loop, referred to as the loop body, are processed. The loop's condition must result in a true or false answer only. When the condition evaluates to true, the instructions listed in the loop body are processed; otherwise, the loop body instructions are skipped over.

Some loops require the user to enter a special value, called a sentinel value, to end the loop. The sentinel value should be easily distinguishable from the valid data recognized by the program. Other loops are terminated through the use of a counter.

The input instruction that appears above the pretest loop's condition is referred to as the priming read and gets only the first value from the user. The input instruction that appears within the loop gets the remaining values (if any) and is referred to as the update read.

In most flowcharts, a diamond is used to represent a repetition structure's condition. The diamond is called the decision symbol.

Counters and accumulators are used within a repetition structure to calculate subtotals, totals, and averages. All counters and accumulators must be initialized and updated. Counters are updated by a constant value, whereas accumulators are updated by an amount that varies.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Many programmers use a hexagon to represent the `for` clause in a `for` statement.

You can use either the `while` statement or the `for` statement to code a pretest loop in C++.

Key Terms

Accumulator—a numeric variable used for accumulating (adding together) something

Counter—a numeric variable used for counting something

Counter-controlled loops—loops whose processing and termination are controlled by a counter variable

Decrementing—decreasing a value

Endless loop—a loop whose instructions are processed indefinitely; also called an infinite loop

`for` statement—a C++ statement that can be used to code a pretest loop

Incrementing—increasing a value

Infinite loop—another name for an endless loop

Initializing—the process of assigning a beginning value to a memory location

Loop—another name for the repetition structure

Loop body—the instructions within a loop

Loop exit condition—the requirement that must be met for the computer to *stop* processing the loop body instructions

Looping condition—the requirement that must be met for the computer to *continue* processing the loop body instructions

Posttest loop—a loop whose condition is evaluated *after* the instructions in its loop body are processed

Pretest loop—a loop whose condition is evaluated *before* the instructions in its loop body are processed

Priming read—the input instruction that appears above the loop that it controls; used to get the first input item from the user

Repetition structure—the control structure used to repeatedly process one or more program instructions; also called a loop

Sentinel values—values that are used to end loops; also called trip values or trailer values

Update read—the input instruction that appears within a loop and is associated with the priming read

Updating—the process of adding a number to the value stored in a counter or accumulator variable

`while` statement—a C++ statement that can be used to code a pretest loop

Review Questions

Refer to Figure 7-47 to answer Review Questions 1 through 4.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

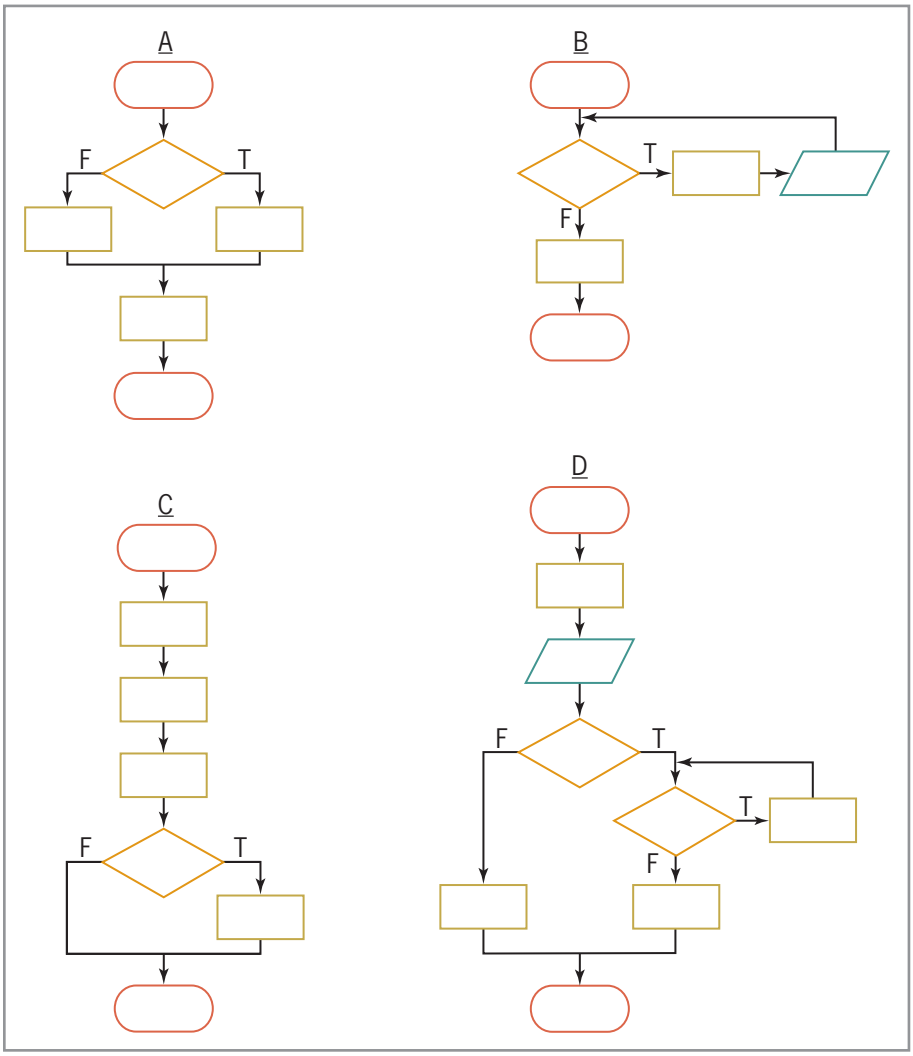


Figure 7-47

1. In addition to the sequence structure, which of the following control structures are used in flowchart A in Figure 7-47?

   a. selection
   b. repetition
   c. both selection and repetition

2. In addition to the sequence structure, which of the following control structures are used in flowchart B in Figure 7-47?

   a. selection
   b. repetition
   c. both selection and repetition

3. In addition to the sequence structure, which of the following control structures are used in flowchart C in Figure 7-47?

   a. selection
   b. repetition
   c. both selection and repetition

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

4. In addition to the sequence structure, which of the following control structures are used in flowchart D in Figure 7-47?

   a. selection
   b. repetition
   c. both selection and repetition

5. Which of the following indicates that the loop should stop when the value in the `quantity` variable is less than the number 50?

   a. `while (quantity >= 50)`
   b. `while (quantity < 50)`
   c. `while (quantity !> 50)`
   d. `while (quantity <= 50)`

6. Which of the following is a good sentinel value for a program that allows the user to enter a person's age?

   a. –4
   b. 350
   c. 999
   d. all of the above

7. Values that are used to end loops are referred to as _____ values.

   a. closing
   b. ending
   c. sentinel
   d. stop

8. A program allows the user to enter one or more numbers. The first input instruction will get the first number only and is referred to as the _____ read.

   a. entering
   b. initializer
   c. initializing
   d. priming

9. How many times will the computer process the `cout` statement in the following code?

```
int numTimes = 1;
while (numTimes > 5)
{
       cout << numTimes << endl;
       numTimes += 1;
}     //end while
```

   a. 0
   b. 1
   c. 3
   d. 4

10. How many times will the computer process the `cout` statement in the following code?

```
for (int numTimes = 1; numTimes < 10; numTimes += 2)
    cout << numTimes << endl;
//end for
```

   a. 0
   b. 5
   c. 6
   d. 9

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

11. What value stops the loop in Review Question 10?

    a. 6
    b. 9
    c. 10
    d. 11

12. How many times will the computer process the `cout` statement in the following code?

```
for (int numTimes = 4; numTimes <= 10; numTimes += 1)
    cout << numTimes << endl;
//end for
```

    a. 0
    b. 7
    c. 10
    d. 11

13. What value stops the loop in Review Question 12?

    a. 4
    b. 9
    c. 10
    d. 11

14. Which of the following updates the `total` accumulator variable by the value in the `sales` variable?

    a. `total = total + sales;`
    b. `total = sales + total;`
    c. `total += sales;`
    d. all of the above

15. Which of the following statements can be used to code a loop whose instructions you want processed 10 times?

    a. `for`
    b. `repeat`
    c. `while`
    d. either a or c

Exercises

Pencil and Paper

TRY THIS

1. Complete a desk-check table for the code shown in Figure 7-48. What will the code display on the computer screen? What value stops the loop? (The answers to TRY THIS Exercises are located at the end of the chapter.)

```
int num = 6;
while (num >= 0)
{
    cout << num << endl;
    num -= 2;
}   //end while
```

Figure 7-48

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

TRY THIS

2. Complete a desk-check table for the code shown in Figure 7-49. What will the code display on the computer screen? What value stops the loop? (The answers to TRY THIS Exercises are located at the end of the chapter.)

```
for (int num = 1; num < 6; num += 2)
{
    cout << "Number: ";
    cout << num * 3 << endl;
}   //end for
```

Figure 7-49

MODIFY THIS

3. Rewrite the code shown in Figure 7-48 to use the `for` statement.

MODIFY THIS

4. Rewrite the code shown in Figure 7-49 to use the `while` statement.

INTRODUCTORY

5. Write an assignment statement that updates a counter variable named `numEmployees` by 1.

INTRODUCTORY

6. Write an assignment statement that updates an accumulator variable named `totalPay` by the value in the `grossPay` variable.

INTRODUCTORY

7. Write a C++ `while` clause that processes the loop instructions as long as the value in the `age` variable is greater than the number 18.

INTRODUCTORY

8. Write a C++ `for` clause that processes the loop instructions 10 times. Use `numTimes` as the counter variable's name.

INTRODUCTORY

9. Figure 7-23 in the chapter showed two examples of the `for` statement. List the processing steps for the code shown in Example 2, using Figure 7-24 as a guide.

INTERMEDIATE

10. Write a C++ `while` clause that stops the loop when the value in the `inStock` variable is less than or equal to the value in the `reorder` variable.

INTERMEDIATE

11. Write an assignment statement that updates a counter variable named `quantity` by –5.

INTERMEDIATE

12. Write an assignment statement that subtracts the contents of the `salesReturns` variable from the `sales` accumulator variable.

INTERMEDIATE

13. Modify the solution shown earlier in Figure 7-2. The solution should now keep track of the number of times Sahirah's laser beam missed the spider. After saying "You are safe now. The spider is dead.", Sahirah should say one of the following: "I got him immediately.", "I missed him one time.", or "I missed him *x* times." (where *x* is the value in the counter).

Computer

TRY THIS

14. Create a program that allows the user to enter any number of integers. The program should display the sum of the integers. Use the `while` statement and a negative number as the sentinel value. If necessary, create a new project named TryThis14 Project, and save it in the Cpp8\Chap07 folder. Enter the C++ instructions into a source file named TryThis14.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Test the program appropriately. (The answers to TRY THIS Exercises are located at the end of the chapter.)

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

15. Create a program that allows the user to enter the ages (in years) of five people. The program should display the average age. Use the `for` statement. Display the average age with one decimal place. If necessary, create a new project named TryThis15 Project, and save it in the Cpp8\Chap07 folder. Enter the C++ instructions into a source file named TryThis15.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Test the program using the following ages: 23, 31, 37, 19, and 43. (The answers to TRY THIS Exercises are located at the end of the chapter.) TRY THIS

16. In this exercise, you will modify the tip program shown earlier in Figure 7-31. Follow the instructions for starting C++ and viewing the ModifyThis16.cpp file, which is contained in either the Cpp8\Chap07\ModifyThis16 Project folder or the Cpp8\Chap07 folder. (You may need to open the project/solution file first.) Change the `for` statement to a `while` statement. Save and then run the program. Test the program using 90.50 as the restaurant bill. The three tips should appear as shown earlier in Figure 7-32. MODIFY THIS

17. In this exercise, you will modify the sales program from Lab7-2. If necessary, create a new project named ModifyThis17 Project, and save it in the Cpp8\Chap07 folder. Enter (or copy) the Lab7-2.cpp instructions into a new source file named ModifyThis17.cpp. Change Lab7-2.cpp in the first comment to ModifyThis17.cpp. In addition to the current output, the program should also display the annual increase amounts. Display each amount on a separate line, using the message "Year *x* increase: $*y*", in which *x* is the year number and *y* is the increase amount. The increase amounts should be displayed with no decimal places. Save and then run the program. Enter 125000 as the current year's sales. (Hint: The first line in the output should say "Year 1 increase: $6875".) MODIFY THIS

18. Create a program that displays the weekly gross pay for any number of employees. The user will input the number of hours the employee worked and the employee's hourly rate. Employees working more than 40 hours receive time and one-half for the hours worked over 40. If necessary, create a new project named Introductory18 Project, and save it in the Cpp8\Chap07 folder. Enter the C++ instructions into a source file named Introductory18.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Save, run, and test the program. INTRODUCTORY

19. Create a program that a teacher can use to enter the midterm and final test scores for any number of students. Each test is worth 200 points. The program should display each student's grade, which is based on the total points the student earned, as indicated in Figure 7-50. If necessary, create a new project named Introductory19 Project, and save it in the Cpp8\Chap07 folder. Enter the C++ instructions into a source file named Introductory19.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Save, run, and test the program. INTRODUCTORY

Total points	Grade
360 – 400	A
320 – 359	B
280 – 319	C
240 – 279	D
below 240	F

Figure 7-50

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

INTRODUCTORY

20. Create a program that displays a multiplication table similar to the one shown in Figure 7-51. If necessary, create a new project named Introductory20 Project, and save it in the Cpp8\Chap07 folder. Enter your C++ instructions into a source file named Introductory20.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Save, run, and test the program.

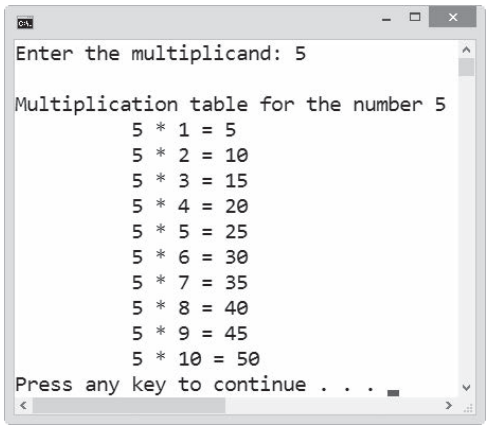


Figure 7-51

INTERMEDIATE

21. A thrift store discounts the price of its items using rates of 10% through 40% in increments of 5%. Create a program that allows the store clerk to enter an item's original price. The program should display the seven discount amounts and discounted prices. If necessary, create a new project named Intermediate21 Project, and save it in the Cpp8\Chap07 folder. Enter your C++ instructions into a source file named Intermediate21.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Save, run, and test the program.

INTERMEDIATE

22. Baxter Skating Rink holds a weekly ice-skating competition. Competing skaters must perform a two-minute program in front of a panel of judges. The number of judges varies from week to week. At the end of a skater's program, each judge assigns a score of 0 through 10 to the skater. Create a program that allows the rink manager to enter each judge's score for a specific skater. The program should display the number of scores entered, the skater's total score, and the skater's average score. If necessary, create a new project named Intermediate22 Project, and save it in the Cpp8\Chap07 folder. Enter the C++ instructions into a source file named Intermediate22.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Save, run, and test the program.

INTERMEDIATE

23. Create a program that allows the user to enter the gender (either F or M) and GPA (0.0 through 4.0) for any number of students. The program should calculate and display the average GPA for all students, the average GPA for male students, and the average GPA for female students. If necessary, create a new project named Intermediate23 Project, and save it in the Cpp8\Chap07 folder. Enter the C++ instructions into a source file named Intermediate23.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Save, run, and test the program.

ADVANCED

24. Create a program that displays the ending balance in a savings account, given the beginning balance, the deposit amounts, and the withdrawal amounts. Use two loops in the program: one to get the deposit amounts, and the other to get the withdrawal amounts.

    a. Create an IPO chart for the problem, and then desk-check the algorithm two times, using the data shown in Figure 7-52.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

b. List the input, processing, and output items, as well as the algorithm, in a chart similar to the one shown earlier in Figure 7-42. Then code the algorithm into a program.
c. Desk-check the program using the same data used to desk-check the algorithm.
d. If necessary, create a new project named Advanced24 Project, and save it in the Cpp8\Chap07 folder. Enter your C++ instructions into a source file named Advanced24.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Display the ending balance with two decimal places.
e. Save and then run the program. Test the program using the same data used to desk-check the program.

	First desk-check	Second desk-check
Beginning balance:	2456.75	9855.89
Deposits:	200, 56.50, 25.78, 3.50	1200, 75
Withdrawals:	25, 100	900.75

Figure 7-52

25. ADVANCED

In this exercise, you create a program for the sales manager at Computer Haven, a small business that offers motivational seminars to local companies. Figure 7-53 shows the charge for attending a seminar. Notice that the charge per person depends on the number of people the company registers. For example, the cost for four registrants is $400; the cost for two registrants is $300. The program should allow the sales manager to enter the number of registrants for as many companies as needed. When the sales manager has finished entering the data, the program should calculate and display the total number of people registered, the total charge for those registrants, and the average charge per registrant. For example, if one company registers four people and another company registers two people, the total number of people registered is six, the total charge is $700, and the average charge per registrant is $116.67.

Number of people a company registers	Charge per person ($)
1 – 3	150
4 – 9	100
10 or more	90

Figure 7-53

a. Create an IPO chart for the problem, and then desk-check the algorithm appropriately.
b. List the input, processing, and output items, as well as the algorithm, in a chart similar to the one shown earlier in Figure 7-42. Then code the algorithm into a program.
c. Desk-check the program using the same data used to desk-check the algorithm.
d. If necessary, create a new project named Advanced25 Project, and save it in the Cpp8\Chap07 folder. Enter your C++ instructions into a source file named Advanced25.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Display the average charge with two decimal places.
e. Save and then run the program. Test the program using the same data used to desk-check the program.

26. ADVANCED

Create a program that displays the first 10 Fibonacci numbers (1, 1, 2, 3, 5, 8, 13, 21, 34, and 55). Notice that beginning with the third number in the series, each Fibonacci number is the sum of the prior two numbers. For example, 2 is the sum of 1 plus 1, 3 is the sum of 1 plus 2, 5 is the sum of 2 plus 3, and so on. Write two versions of the code: one using the `while` statement, and the other using the `for` statement. If necessary,

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

create a new project named Advanced26 Project, and save it in the Cpp8\Chap07 folder. Enter the C++ instructions into a source file named Advanced26.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Save and then run the program. The Fibonacci numbers should appear twice on the screen.

ADVANCED

27. If necessary, create a new project named Advanced27 Project, and save it in the Cpp8\Chap07 folder. Enter the C++ instructions shown earlier in Figure 7-11 into a source file named Advanced27.cpp. Also enter appropriate comments and any additional instructions required by the compiler. When the user has finished entering the sales amounts, the program should display the number of sales amounts entered and the average commission. Save run, and test the program.

SWAT THE BUGS

28. Follow the instructions for starting C++ and viewing the SwatTheBugs28.cpp file, which is contained in either the Cpp8\Chap07\SwatTheBugs28 Project folder or the Cpp8\Chap07 folder. (You may need to open the project/solution file first.) The program should display the number of positive integers and the number of negative integers entered by the user, but it is not working correctly. Debug the program.

SWAT THE BUGS

29. Follow the instructions for starting C++ and viewing the SwatTheBugs29.cpp file, which is contained in either the Cpp8\Chap07\SwatTheBugs29 Project folder or the Cpp8\ Chap07 folder. (You may need to open the project/solution file first.) The program should display the numbers 1, 2, 3, and 4, but it is not working correctly. Debug the program.

SWAT THE BUGS

30. Follow the instructions for starting C++ and viewing the SwatTheBugs30.cpp file, which is contained in either the Cpp8\Chap07\SwatTheBugs30 Project folder or the Cpp8\Chap07 folder. (You may need to open the project/solution file first.) The program should display each salesperson's bonus, which is calculated by multiplying the salesperson's sales by 10%. The program is not working correctly. Debug the program.

Answers to TRY THIS Exercises

Pencil and Paper

1. See Figure 7-54. The code will display the numbers 6, 4, 2, and 0 on separate lines on the computer screen. The loop stops when the `num` variable's value is –2.

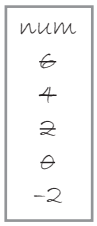


Figure 7-54

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

2. See Figure 7-55. The code will display Number: 3, Number: 9, and Number: 15 on separate lines on the computer screen. The loop stops when the num variable's value is 7.

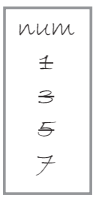


Figure 7-55

Computer

14. See Figure 7-56.

```
//TryThis14.cpp - displays the sum of integers
//Created/revised by <your name> on <current date>

#include <iostream>
using namespace std;

int main()
{
    int num = 0;
    int sum = 0;

    cout << "Enter a number (negative number to end): ";
    cin >> num;
    while (num >= 0)
    {
        sum += num;
        cout << "Enter a number (negative number to end): ";
        cin >> num;
    }  //end while
    cout << "Sum: " << sum << endl;
    return 0;
}    //end of main function
```

Figure 7-56

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

15. See Figure 7-57.

```
//TryThis15.cpp - displays the average age
//Created/revised by <your name> on <current date>

#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    int age = 0;
    int totalAges = 0;
    int people = 0;
    double avgAge = 0.0;

    for (people = 1; people < 6; people += 1)
    {
        cout << "Age: ";
        cin >> age;
        totalAges += age;
    }  //end for
    avgAge = static_cast<double>(totalAges) / (people - 1);
    cout << fixed << setprecision(1);
    cout << "Average age: " << avgAge << endl;
    return 0;
}    //end of main function
```

Figure 7-57

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

CHAPTER 8

More on the Repetition Structure

After studying Chapter 8, you should be able to:

- Include a posttest loop in pseudocode
- Include a posttest loop in a flowchart
- Code a posttest loop using the C++ `do while` statement
- Nest repetition structures

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Posttest Loops

Pretest and posttest loops are also called top-driven and bottom-driven loops, respectively.

As you learned in Chapter 7, a repetition structure can be either a pretest loop or a posttest loop. Both types of loops differ in when their loop condition is evaluated. Unlike the condition in a pretest loop, which is evaluated *before* the instructions within the loop are processed, the condition in a posttest loop is evaluated *after* the instructions within the loop are processed. As a result, the instructions in a posttest loop will always be processed at least once, whereas the instructions in a pretest loop may never be processed. Although pretest loops are more commonly used, it is essential to understand the way posttest loops work because you may encounter a situation where a posttest loop is the better choice. Or, you may encounter a posttest loop in another programmer's code that you are either modifying or debugging.

The problem specification, illustrations, and algorithms shown in Figure 8-1 will help clarify the difference between pretest and posttest loops. Both algorithms contain the instructions for getting Sherri from her current location to one that is directly in front of the fountain. Algorithm 1 contains a pretest loop, and Algorithm 2 contains a posttest loop. In the pretest loop, the condition appears in the first line, which indicates that Sherri should evaluate it *before* she follows the `walk forward one complete step` instruction in the loop body. In the posttest loop, the condition appears in the last line, indicating that Sherri should evaluate it only *after* following the `walk forward one complete step` instruction. The pretest loop in Algorithm 1 will work when Sherri is zero or more steps away from the fountain. The posttest loop in Algorithm 2, however, will work only when Sherri is at least one step away from the fountain.

Problem specification
Sherri is standing an unknown number of steps away from the Burlington fountain. Write the instructions that direct Sherri to walk from her current location to the fountain.

Illustration A

Illustration B

Algorithm 1 – pretest loop

```
repeat while (you are not directly in front of the fountain)
      walk forward one complete step
end repeat
```

condition

works when Sherri is zero or more steps away from the fountain

Algorithm 2 – posttest loop

```
repeat
      walk forward one complete step
end repeat while (you are not directly in front of the fountain)
```

works only when Sherri is at least one step away from the fountain

condition

Figure 8-1 Problem specification, illustrations, and algorithms containing pretest and posttest loops
Image by Diane Zak; created with Reallusion CrazyTalk Animator

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

To understand why the loops in Figure 8-1 are not interchangeable, you will desk-check them using the two illustrations shown in the figure. For the first desk-check, we will use Illustration A, which shows Sherri at least one step away from the fountain. For the purposes of this desk-check, we will assume that Sherri is three steps away from her final destination. In the pretest loop, the loop's condition checks Sherri's current location. Sherri is not directly in front of the fountain, so she is told to walk forward one complete step; after she takes this first step, the loop's condition is evaluated again. The condition still evaluates to true, so Sherri is told to walk forward one complete step (her second step), after which the loop's condition is evaluated again. The condition still evaluates to true, so Sherri is told to walk forward one complete step (her third step), and then the loop's condition is evaluated again. At this point, Sherri is directly in front of the fountain, so the condition evaluates to false, and the pretest loop ends.

The posttest loop, on the other hand, instructs Sherri to walk forward one complete step (her first step) before evaluating the loop's condition. Sherri is still not directly in front of the fountain, so she is told to walk forward one complete step (her second step), after which the loop's condition is evaluated again. The condition still evaluates to true, so Sherri is told to walk forward one complete step (her third step), and then the loop's condition is evaluated again. At this point, Sherri is directly in front of the fountain, so the condition evaluates to false, and the posttest loop ends. Notice that when Sherri is three steps away from the fountain, the pretest and posttest loops produce the same result: Both loops place her right in front of the fountain. If you desk-check both loops using other values for the number of steps, you will find that both loops are interchangeable when Sherri is at least one step away from her final destination.

For the second desk-check, we will use Illustration B, which shows Sherri already standing in front of the fountain. The condition in the pretest loop checks Sherri's current location. Sherri is already positioned correctly, so the walk forward one complete step instruction is bypassed and the loop ends. The posttest loop, on the other hand, instructs Sherri to walk forward one complete step before the loop's condition is evaluated. But if Sherri walks forward, she will bump into the fountain. Obviously, the posttest loop in Algorithm 2 does not work correctly when Sherri starts out directly in front of the fountain. You can fix this problem by adding a selection structure to the algorithm, as shown in Figure 8-2.

Modified Algorithm 2 – posttest loop within a selection structure

```
if (you are not directly in front of the fountain)
    repeat
        walk forward one complete step
    end repeat while (you are not directly in front of the fountain)
end if
```

works when Sherri is zero or more steps away from the fountain

Figure 8-2 Selection structure added to Algorithm 2 from Figure 8-1

The posttest loop in Figure 8-2 is identical to the posttest loop in Figure 8-1 except it is processed only when the selection structure's condition evaluates to true, which is when Sherri is not directly in front of the fountain. Although the modified algorithm in Figure 8-2 works correctly, most programmers prefer to use a pretest loop, rather than a posttest loop within a selection structure, because it is easier to write and understand. Posttest loops should be used only when their instructions must be processed at least once. You often will find a posttest loop

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

in programs that allow the user to select from a menu, such as a game program. This type of program uses the posttest loop to control the display of the menu, which must appear on the screen at least once.

Flowcharting a Posttest Loop

For many people, it is easier to understand the difference between a pretest loop and a posttest loop by viewing both loops in a flowchart. Figure 8-3 shows the problem specification for the commission program from Chapter 7. It also shows two correct algorithms in flowchart form. Algorithm 1 (which is from Figure 7-5 in Chapter 7) uses a pretest loop, and Algorithm 2 uses a posttest loop. Notice that the decision diamond, which contains the loop's condition, appears at the top of a pretest loop in a flowchart; however, it appears at the bottom of a posttest loop. The instructions in Algorithm 2's flowchart will always be processed at least once. The instructions in Algorithm 1's flowchart, on the other hand, may never be processed.

Problem specification
Create a program that calculates the commission for each of a company's salespeople. The commission is calculated by multiplying the salesperson's sales amount by 20%.

Algorithm 1 (pretest loop):

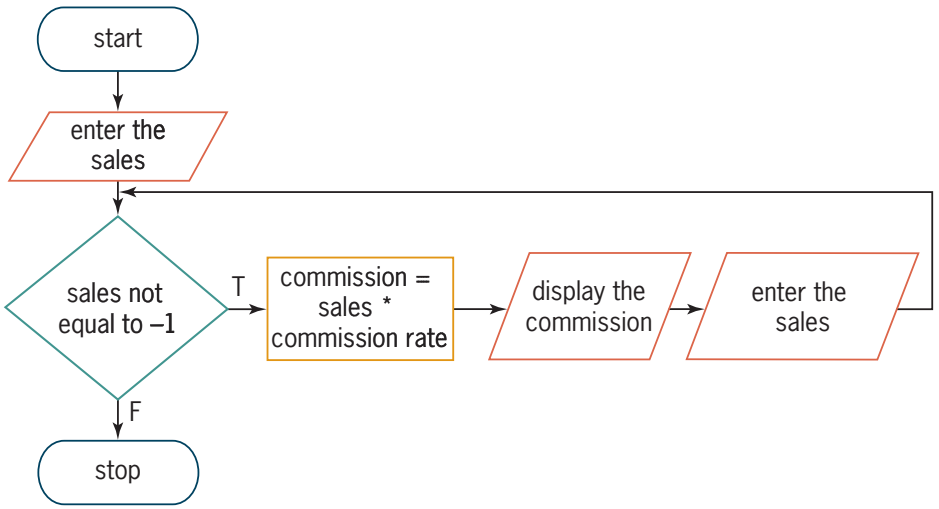


Figure 8-3 Commission program's problem specification and flowcharts *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

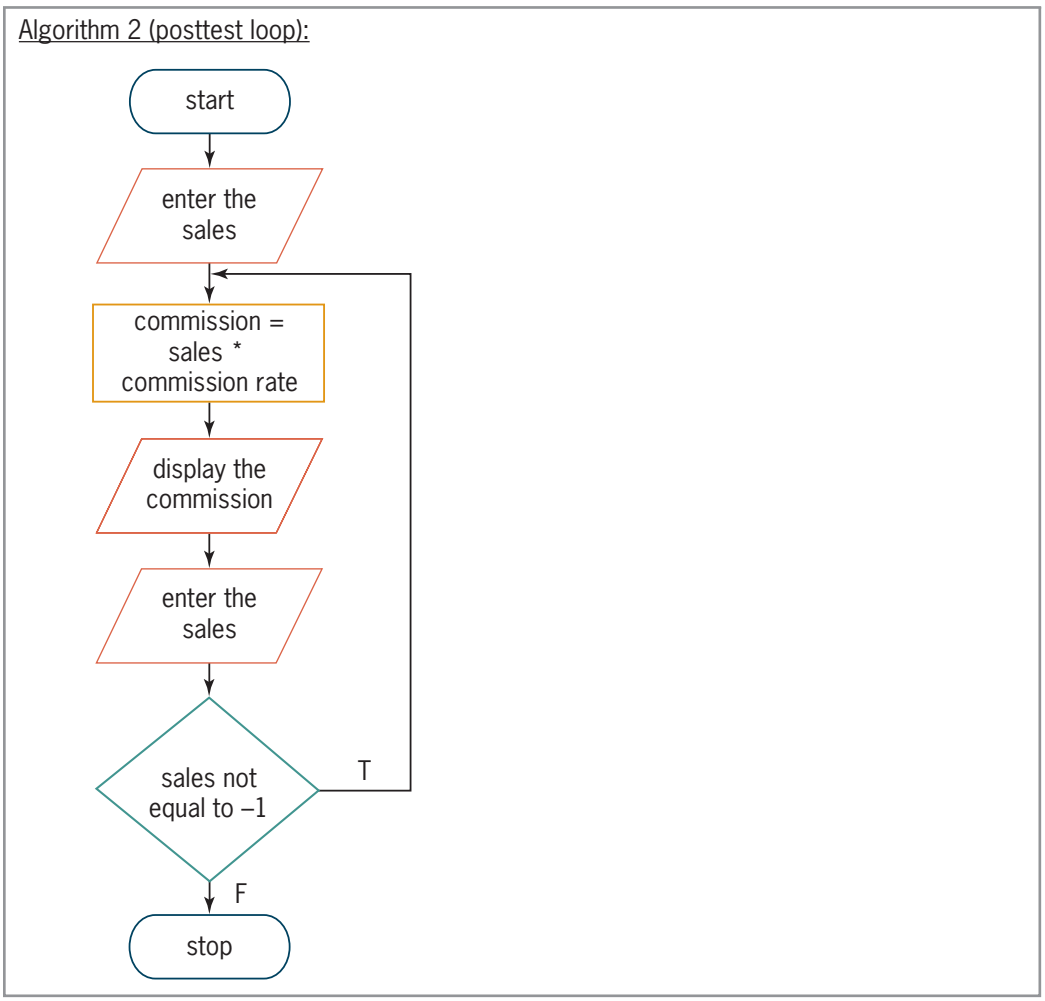


Figure 8-3 Commission program's problem specification and flowcharts

Mini-Quiz 8-1

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

1. If the user enters the numbers 5, 8, 9, and –1, how many times will the condition in the following algorithm be evaluated, and what will the algorithm display?

```
1. enter number
2. repeat while (number is greater than or equal to 0)
        display number
        enter number
   end repeat
3. display "Done"
```

2. If the user enters the number –4, how many times will the condition in Question 1's algorithm be evaluated, and what will the algorithm display?
3. If the user enters the numbers 5, 8, 9, and –1, how many times will the condition in the following algorithm be evaluated, and what will the algorithm display?

```
1. enter number
2. repeat
        display number
```

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

```
        enter number
    end repeat while (number is greater than or equal to 0)
3. display "Done"
```

4. If the user enters the numbers 0 and –4, how many times will the condition in Question 3's algorithm be evaluated, and what will the algorithm display?

The do while Statement

C++ provides the **do while statement** for coding a posttest loop. The statement's syntax is shown in Figure 8-4, along with examples of using the statement. As in the `while` statement, the condition in the `do while` statement must be phrased as a looping condition that evaluates to either true or false. Recall that a looping condition indicates the requirement for repeating the loop body instructions. The condition can contain variables, constants, functions, and operators (arithmetic, comparison, or logical). Although not a requirement, some programmers use a comment (such as `//begin loop`) to mark the beginning of the `do while` statement because it makes the program easier to read and understand.

The braces in a `do while` statement are not required when the loop body contains only one statement. However, a one-statement loop body is rare.

HOW TO Use the `do while` Statement

Syntax

```
do  //begin loop
{
    one or more statements to be processed one time, and
    thereafter as long as the condition is true
}  while (condition);
```

the statement ends with a semicolon

Example 1

```
int hours = 0;

cout << "Hours worked: ";
cin >> hours;
do    //begin loop
{
     cout << "Gross pay: $" << hours * 10 << endl << endl;
     cout << "Hours worked: ";
     cin >> hours;
}  while (hours > 0);
```

priming read

update read

semicolon

Example 2

```
char another = ' ';
double number = 0.0;
double sum = 0.0;

cout << "Enter a number? (Y/N) ";
cin >> another;
do  //begin loop
{
     cout << "Number: ";
     cin >> number;
     sum += num;
     cout << "Enter another number? (Y/N)";
     cin >> another;
}  while (toupper(another) == 'Y');
```

priming read

update read

semicolon

Figure 8-4 How to use the `do while` statement

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The `while` clause in Example 1 indicates that the loop body instructions should be repeated as long as (or while) the value in the `hours` variable is greater than 0. The loop will stop when the second number entered by the user is either less than or equal to 0. When processing the `while` clause in Example 2, the computer temporarily converts the `another` variable's value to uppercase and then compares the result to the uppercase letter Y. In this case, the loop will stop when the user's second entry is anything other than the letter Y entered in either uppercase or lowercase.

Earlier, in Figure 8-3, you viewed the problem specification and algorithms (in flowchart form) for the commission problem from Chapter 7. Figure 8-5 shows the pseudocode and C++ instructions corresponding to Algorithm 2 in Figure 8-3. Recall that the algorithm contains a posttest loop. The beginning and end of the loop are shaded in the figure, which also contains a sample run of the program.

For more examples of using the `do while` statement, see the Using the `do while` Statement section in the Ch08WantMore.pdf file.

IPO chart information

Input

commission rate (20%)
sales

Processing

none

Output

commission

Algorithm

1. enter the sales

2. repeat

   calculate the commission by multiplying the sales by the commission rate

   display the commission

   enter the sales

   end repeat while (the sales are not equal to -1)

C++ instructions

```
const double RATE = 0.2;
int sales = 0;

double commission = 0.0;

cout << "First salesperson's sales
(-1 to stop): ";
cin >> sales;

do   //begin loop
{
    commission = sales * RATE;

    cout << "Commission: $"
    << commission << endl << endl;

    cout << "Next salesperson's
    sales (-1 to stop): ";
    cin >> sales;
}  while(sales != -1);
```

```
Commission
First salesperson's sales (-1 to stop): 1200
Commission: $240

Next salesperson's sales (-1 to stop): 800
Commission: $160

Next salesperson's sales (-1 to stop): -1
Press any key to continue . . .
```

Figure 8-5 Commission program containing a posttest loop

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 8-2

1. Which clause marks the beginning of the C++ `do while` statement?

   a. `do`
   b. `do while`
   c. `repeat`
   d. `while`

2. The `while` clause in the C++ `do while` statement ends with a _____.

   a. brace
   b. colon
   c. comma
   d. semicolon

3. Write a C++ `while` clause that processes the body of a posttest loop as long as the value in the `ordered` variable is less than or equal to the value in the `inventory` variable.

4. Write a C++ `while` clause that processes the body of a posttest loop as long as the value in a `char` variable named `letter` is either Y or y. Use the built-in `toupper` function.

Nested Repetition Structures

Like selection structures, repetition structures can be nested, which means you can place one loop (called the nested or inner loop) within another loop (called the outer loop). Both loops can be pretest loops, or both can be posttest loops. Or, one can be a pretest loop and the other a posttest loop.

A programmer determines whether a problem's solution requires a **nested loop** by studying the problem specification. The first problem specification you will examine in this chapter involves a waitress named Trixie. The problem specification and an illustration of the problem are shown in Figure 8-6, along with an appropriate algorithm. The algorithm requires a loop because the instructions for telling each table about the daily specials must be repeated for every table that needs to be waited on. However, it does not require a nested loop. This is because the instructions within the loop should be followed only once per table.

Problem specification and algorithm
Trixie works as a waitress at a local diner. The diner just opened for the day, and there are customers already sitting at several of the tables. Write the instructions that direct Trixie to go to each table that needs to be waited on and tell the customers about the daily specials.

```
repeat for (each table that needs to be waited on)
        go to a table that needs to be waited on
        tell the customers at the table about the daily specials
end repeat
```

follow these instructions for each table

Figure 8-6 Problem specification and algorithm that requires a loop
Image by Diane Zak; created with Reallusion CrazyTalk Animator

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Now we will add some additional tasks for Trixie to perform. After telling the customers at a table about the daily specials, Trixie should now take each customer's order and then submit the order for the entire table to the cook. Figure 8-7 shows the modified problem specification along with the modified algorithm, which requires a nested loop. The nested loop begins with the *repeat for (each customer at the table)* clause and ends with the first *end repeat* clause. Notice that the nested loop is contained entirely within the outer loop. This must be true for the loop to be nested and work correctly.

Problem specification and algorithm
Trixie works as a waitress at a local diner. The diner just opened for the day, and there are customers already sitting at several of the tables. Write the instructions that direct Trixie to go to each table that needs to be waited on and tell the customers about the daily specials. While at each table, Trixie should take each customer's order. She should then submit the order for the entire table to the cook.

```
repeat for (each table that needs to be waited on)
        go to a table that needs to be waited on
        tell the customers at the table about the daily specials
        repeat for (each customer at the table)
            ask the customer for his or her order
            record the order on the order slip for that table
        end repeat
        go over to the cook at the counter
        tear the appropriate order slip from the order pad
        give the order slip to the cook
end repeat
```

follow these instructions for each customer at the current table

follow these instructions for each table

Figure 8-7 Modified problem specification and algorithm that requires a nested loop

The Clock Program

A clock uses nested repetition structures to keep track of the time. For simplicity, consider a clock's minute and second hands only. The second hand on a clock moves one position, clockwise, for every second that has elapsed. After the second hand moves 60 positions, the minute hand moves one position, also clockwise. The second hand then begins its journey around the clock again.

The algorithm used by a clock's minute and second hands is shown in Figure 8-8. The outer loop controls the minute hand, while the inner (nested) loop controls the second hand. Here, too, the nested loop (which is shaded in the figure) is contained entirely within the outer loop, which is a requirement for the loop to be nested and work correctly. The figure also contains the C++ instructions for a program that simulates the minute and second hands, along with a sample run of the program. However, to make it easier to desk-check the instructions, the nested loop uses three seconds per minute and the outer loop stops after two minutes.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The next iteration of the outer loop (which controls the minute hand) occurs only after the nested loop (which controls the second hand) has finished processing.

```
1. start minutes at 0
2. repeat while (minutes are less than 60)
       start seconds at 0
       repeat while (seconds are less than 60)
           move second hand 1 position, clockwise
           add 1 to seconds
       end repeat
       move minute hand 1 position, clockwise
       add 1 to minutes
   end repeat
```

outer loop

```
int main()
{
    cout << "Minutes    Seconds" << endl;
    for (int minutes = 0; minutes < 2; minutes += 1)
    {
        for (int seconds = 0; seconds < 3; seconds += 1)
            cout << "   " << minutes
                 << "          " << seconds << endl;
        //end for
        cout << endl;
    } //end for
    return 0;
}   //end of main function
```

outer loop

```
Clock
Minutes    Seconds
   0          0
   0          1
   0          2

   1          0
   1          1
   1          2

Press any key to continue . . .
```

Figure 8-8 Algorithm, code, and a sample run for the clock program

Recall that the `for` clause's *condition* argument must be phrased as a looping condition, which means it must specify the requirement for processing the loop body instructions.

In the code in Figure 8-8, the outer loop's `for` clause directs the computer to repeat the loop body instructions two times. Braces are required in the outer loop because its loop body contains more than one statement. The nested loop's `for` clause, on the other hand, directs the computer to repeat the one instruction in its loop body three times. Although both loops in Figure 8-8 are coded using the `for` statement, one or both could be coded using the `while` statement. In addition, the algorithm could have been written using one or more posttest loops. Recall that you use the `do while` statement to code a posttest loop in C++.

You can observe the way the computer processes a nested loop by desk-checking the loops shown in Figure 8-8. First, the *initialization* argument in the outer loop's `for` clause creates the `minutes` variable and initializes it to 0. The *condition* argument then checks whether the variable's value is less than 2. It is, so the instructions in the outer loop are processed.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The first instruction in the outer loop is the nested loop's `for` clause. The clause's *initialization* argument creates the `seconds` variable and initializes it to 0. Its *condition* argument then checks whether the value in the variable is less than 3. It is, so the nested loop's `cout` statement displays the contents of the `minutes` (0) and `seconds` (0) variables on the screen.

Ch08-Clock

Next, the nested `for` clause's *update* argument adds 1 to the value in the `seconds` variable; the result is 1. Its *condition* argument then checks whether the variable's value is less than 3. It is, so the nested loop's `cout` statement displays the contents of the `minutes` (0) and `seconds` (1) variables on the screen.

The *update* argument in the nested `for` clause adds 1 to the value in the `seconds` variable, giving 2. The *condition* argument checks whether the variable's value is less than 3. It is, so the nested loop's `cout` statement displays the contents of the `minutes` (0) and `seconds` (2) variables on the screen.

The nested `for` clause's *update* argument increases the value in the `seconds` variable by 1, giving 3. The *condition* argument checks whether the variable's value is less than 3. It's not, so the nested loop ends, and the `seconds` variable is removed from the computer's internal memory. (Recall that the variable created in the `for` clause is local to the `for` statement and is removed from memory when the `for` loop ends.) Processing continues with the first statement following the end of the nested loop. That statement is the `cout << endl;` statement, which positions the cursor on the next line on the computer screen.

Notice that the nested loop is completely processed prior to the next iteration of the outer loop.

After processing the `cout << endl;` statement, which is the last statement in the body of the outer loop, the computer returns to the outer loop's `for` clause to process its *update* and *condition* arguments. The *update* argument adds 1 to the value in the `minutes` variable, giving 1. The *condition* argument checks whether the variable's value is less than 2. It is, so the outer loop's instructions are processed again.

The first instruction in the outer loop is the nested loop's `for` clause, whose *initialization* and *condition* arguments create the `seconds` variable and initialize it to 0 and then check whether its value is less than 3. At this point, the variable's value is less than 3, so the `cout` statement in the nested loop displays the contents of the `minutes` (1) and `seconds` (0) variables on the screen. Figure 8-9 shows the desk-check table and output at this point.

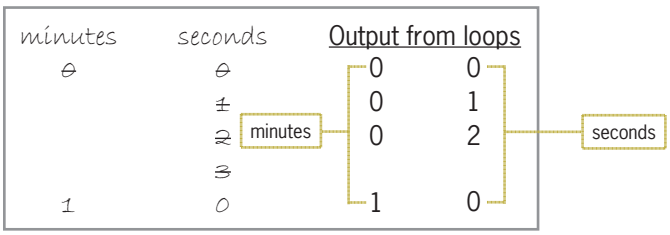


Figure 8-9 Current desk-check table and output

The nested `for` clause's *update* argument adds 1 to the value in the `seconds` variable, giving 1. The *condition* argument checks whether the variable's value is less than 3. It is, so the nested loop's `cout` statement displays the contents of the `minutes` (1) and `seconds` (1) variables on the screen.

The *update* argument in the nested `for` clause increases the value in the `seconds` variable by 1, giving 2. The *condition* argument checks whether the variable's value is less than 3. It is, so the nested loop's `cout` statement displays the contents of the `minutes` (1) and `seconds` (2) variables on the screen.

Here again, the nested `for` clause's *update* argument increases the value in the `seconds` variable by 1; the result is 3. The *condition* argument checks whether the variable's value is less than 3.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

It's not, so the nested loop ends, and the `seconds` variable is removed from the computer's internal memory. The `cout` statement below the nested loop then positions the cursor on the next line on the computer screen.

After processing the `cout` statement, which is the last statement in the body of the outer loop, the computer returns to the outer loop's `for` clause to process its *update* and *condition* arguments. The *update* argument adds 1 to the value in the `minutes` variable, giving 2. The *condition* argument checks whether the variable's value is less than 2. It's not, so the outer loop ends, and the `minutes` variable is removed from the computer's internal memory. Figure 8-10 shows the completed desk-check table and output.

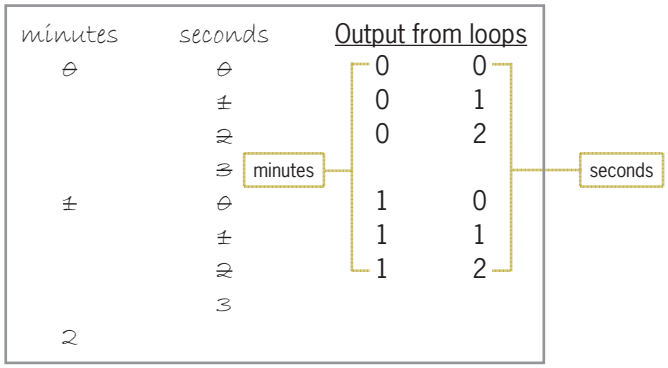


Figure 8-10 Completed desk-check table and output

The Car Depreciation Program

Typically, new cars depreciate—in other words, lose their value—by 15% to 25% per year. Figure 8-11 shows the problem specification and C++ code for the car depreciation program. The program uses a counter-controlled loop to display the value of a new car at the end of each of five years, using a 15% annual depreciation rate. The figure also shows a sample run of the program.

Problem specification
Create a program that displays the value of a new car at the end of each of five years, using a 15% annual depreciation rate.

```
int main()
{
    double originalValue = 0.0;
    double depreciation = 0.0;
    double currentValue = 0.0;

    cout << "Original value: ";
    cin >> originalValue;
    cout << endl << fixed << setprecision(0);

    cout << "Depreciation rate: 15%" << endl;
    cout << "Value after year: " << endl;
```

Figure 8-11 Car depreciation program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
    currentValue = originalValue;
    for (int year = 1; year < 6; year += 1)
    {
        depreciation = currentValue * 0.15;
        currentValue -= depreciation;
        cout << year << "   $" << currentValue << endl;
    } //end for

    return 0;
}   //end of main function
```

loop

```
Car Depreciation
Original value: 25000

Depreciation rate: 15%
Value after year:
1   $21250
2   $18063
3   $15353
4   $13050
5   $11093
Press any key to continue . . .
```

Figure 8-11 Car depreciation program

Now we will modify the problem specification slightly. The program should now display the depreciated values using three different rates: 15%, 20%, and 25%. Figure 8-12 shows the modified problem specification and C++ code, as well as a sample run of the program. The modifications made to the original code are shaded in the figure. Notice that the modified code contains two loops rather than one loop. Both loops are counter-controlled loops, and one is nested within the other. The outer loop keeps track of the depreciation rates, and the nested loop keeps track of the years. The flowchart for the modified program is shown in Figure 8-13.

Problem specification
Create a program that displays the value of a new car at the end of each of five years, using annual depreciation rates of 15%, 20%, and 25%.

```
int main()
{
    double originalValue = 0.0;
    double depreciation = 0.0;
    double currentValue = 0.0;

    cout << "Original value: ";
    cin >> originalValue;
    cout << endl << fixed << setprecision(0);
```

Figure 8-12 Modified car depreciation program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
    for (double rate = 0.15; rate < 0.26; rate += 0.05)
    {
        cout << "Depreciation rate: " << rate * 100 << "%" << endl;
        cout << "Value after year: " << endl;

        currentValue = originalValue;
        for (int year = 1; year < 6; year += 1)
        {
            depreciation = currentValue * rate;
            currentValue -= depreciation;
            cout << year << "  $" << currentValue << endl;
        }  //end for
        cout << endl;
    }  //end for

    return 0;
}   //end of main function
```

outer loop

nested loop

```
Car Depreciation
Original value: 25000

Depreciation rate: 15%
Value after year:
1  $21250
2  $18063
3  $15353
4  $13050
5  $11093

Depreciation rate: 20%
Value after year:
1  $20000
2  $16000
3  $12800
4  $10240
5  $8192

Depreciation rate: 25%
Value after year:
1  $18750
2  $14063
3  $10547
4  $7910
5  $5933

Press any key to continue . . .
```

Figure 8-12 Modified car depreciation program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

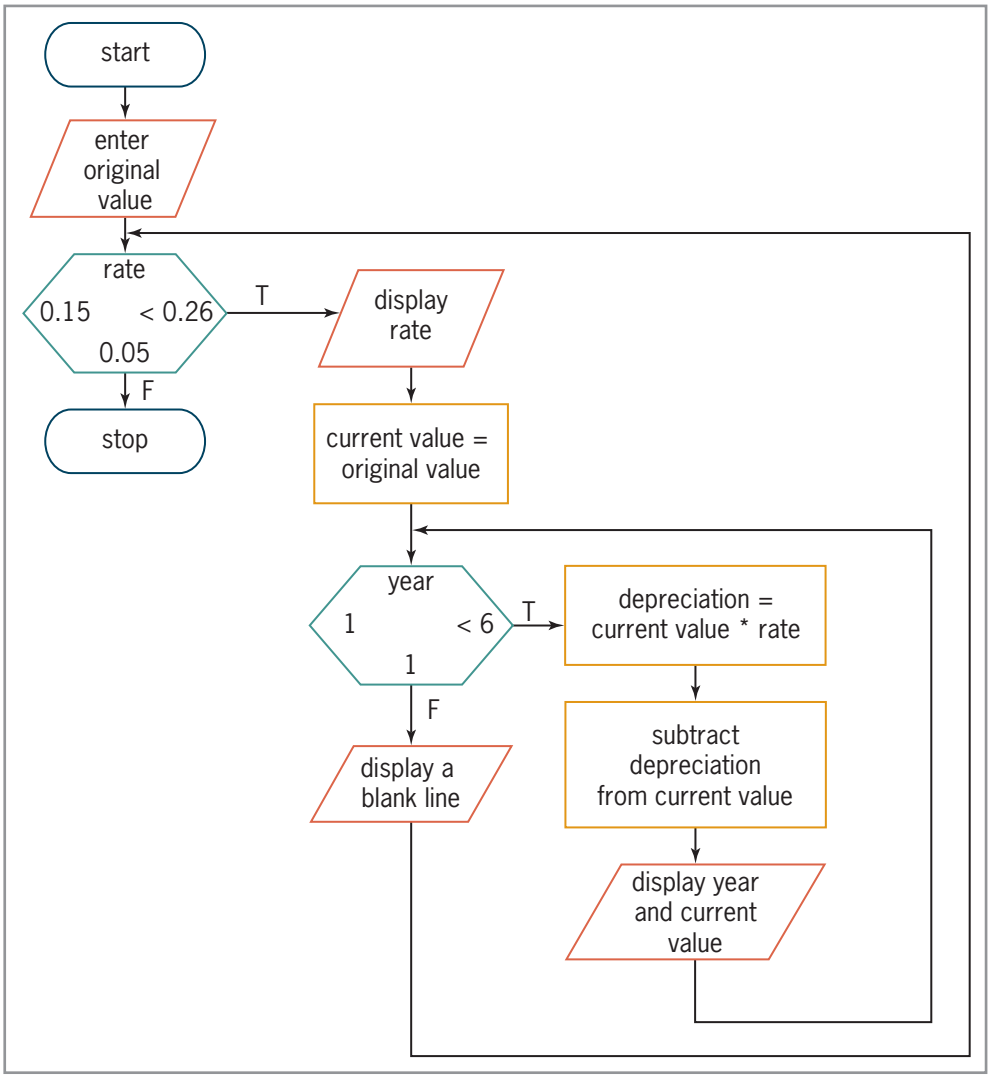


Figure 8-13 Flowchart for the modified car depreciation program

For more examples of nested loops, see the Nested Loops section in the Ch08WantMore.pdf file.

Mini-Quiz 8-3

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

1. A nested loop must be a pretest loop; it cannot be a posttest loop.
   a. True
   b. False
2. For a(n) _____ loop to work correctly, it must be contained entirely within a(n) _____ loop.
   a. outer, nested
   b. nested, outer
3. Consider a clock's hour and minute hands. The hour hand is controlled by a(n) _____ loop, while the minute hand is controlled by a(n) _____ loop.
   a. outer, nested
   b. nested, outer

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The answers to the labs are contained in the Answers.pdf file.

LAB 8-1 Stop and Analyze

Study the program shown in Figure 8-14, and then answer the questions.

```
 1 //Lab8-1.cpp
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 using namespace std;
 6
 7 int main()
 8 {
 9    int maxRows = 0;
10
11    cout << "Maximum number of rows: ";
12    cin >> maxRows;
13
14    for (int row = 0; row < maxRows; row += 1)
15    {
16       for (int space = 0; space < maxRows - row; space += 1)
17            cout << " ";
18       //end for
19
20       for (int asterisk = 0; asterisk <= row; asterisk += 1)
21            cout << "* ";
22       //end for                    notice the space
23
24       cout << endl;
25    }  //end for
26    return 0;
27 }  //end of main function
```

Figure 8-14 Code for Lab 8-1

QUESTIONS

1. How many loops does the program contain? How many are nested?
2. Desk-check the program using the number 3 as the maximum number of rows. What will the program display?
3. Follow the instructions for starting C++ and viewing the Lab8-1.cpp file, which is contained in either the Cpp8\Chap08\Lab8-1 Project folder or the Cpp8\Chap08 folder. (Depending on your C++ development tool, you may need to open Lab8-1's project/solution file first.) Run the program. Test the program using the number 3 as the maximum number of rows. What does the program display?
4. Run the program again. This time, use the number 10 as the maximum number of rows. What does the program display?
5. Replace the outer `for` statement with a `while` statement. Save and then run the program. Test the program using the number 10 as the maximum number of rows. The output should agree with the results from Question 4.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

LAB 8-2 Plan and Create

In this lab, you will plan and create an algorithm for the problem specification shown in Figure 8-15. You begin by analyzing the problem, looking first for the output and then for the input. In this case, the program should display a person's total earnings before retirement at age 65, using annual raise rates of 3%, 4%, and 5%. To calculate the amounts, the program will need to know the person's age and current salary.

Next, you plan the algorithm and then desk-check it. Figure 8-15 shows the completed IPO chart and desk-check table, which (for simplicity) uses an age of 62 and a salary of $25000. The algorithm contains two loops. The outer loop keeps track of the three annual raise rates (3%, 4%, and 5%). The nested loop keeps track of the number of years until retirement.

Problem specification

Create a program that allows the user to enter a person's age (in years) and current salary. Both input items should be entered as integers. The program should display a person's total earnings before retirement at age 65, using annual raise rates of 3%, 4%, and 5%. Display the total earning amounts as integers.

Input	Processing	Output
age (1-64 years) current salary	Processing items: raise rate (3%, 4%, 5%) years until retirement new salary year (counter)	total earnings (at end of each of the years until retirement)

```
Algorithm:
1. enter the age
2. if (the age is less than 1 or greater than 64)
      display reenter message
   else
      enter current salary
      calculate years until retirement = 65 - age
      repeat for (each raise rate)
         assign current salary to new salary
         assign current salary to total earnings
         repeat for (year 2 to years until retirement)
               new salary = new salary * (1 + raise rate)
               add new salary to total earnings
         end repeat
         display raise rate and total earnings
      end repeat
   end if
```

at this point, the current salary is year 1's salary

current age	current salary	raise rate	years until retirement
62	25000	~~.03~~	3
		~~.04~~	
		~~.05~~	
		.06	

Figure 8-15 Problem specification, IPO chart, and desk-check table for the retirement algorithm *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

new salary	total earnings		year
~~25000.0~~	~~25000.0~~		~~2~~
~~25750.0~~	~~50750.0~~		~~3~~
~~26522.5~~	~~77272.5~~	3% rate total	~~4~~
~~25000.0~~	~~25000.0~~		~~2~~
~~26000.0~~	~~51000.0~~		~~3~~
~~27040.0~~	~~78040.0~~	4% rate total	~~4~~
~~25000.0~~	~~25000.0~~		~~2~~
~~26250.0~~	~~51250.0~~		~~3~~
27562.5	78812.5	5% rate total	4

Figure 8-15 Problem specification, IPO chart, and desk-check table for the retirement algorithm

The fourth step in the problem-solving process is to code the algorithm into a program. Figure 8-16 shows the IPO chart information and corresponding C++ instructions.

IPO chart information

Input

age (1-64 years)
current salary

Processing

raise rate (3%, 4%, 5%)
years until retirement
new salary
year (counter)

Output

total earnings (at end of each of the years until retirement)

Algorithm

1. enter the age

2. if (the age is less than 1 or greater than 64)
   display reenter message
   else
   enter current salary

   calculate years until retirement = 65 - age
   repeat for (each raise rate)

C++ instructions

```
int age = 0;
int currentSalary = 0;

this variable is created and initialized
in the for clause
int yearsToRetire = 0;
double newSalary = 0.0;
this variable is created and initialized
in the for clause

double total = 0.0;



cout << "Current age in years (1
to 64): ";
cin >> age;

if (age < 1 || age > 64)
    cout << "Please enter an age
    from 1 to 64." << endl;
else
{
    cout << "Current salary as a whole
    number: ";
    cin >> currentSalary;

    yearsToRetire = 65 - age;
    for (double rate = 0.03;rate <
    0.06; rate += 0.01)
```

Figure 8-16 IPO chart information and C++ instructions for the retirement program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
                                                {
        assign current salary to new salary         newSalary = currentSalary;
        assign current salary to total earnings     total = currentSalary;

        repeat for (year 2 to years until retirement)   for (int year = 2; year <=
                                                        yearsToRetire; year += 1)
                                                        {
            new salary = new salary * (1                    newSalary *= (1 + rate);
            + raise rate)
            add new salary to total earnings                total += newSalary;
        end repeat                                       } //end for

        display raise rate and total earnings            cout << "Total with a " <<
                                                         rate * 100 <<
                                                         "% annual raise: $"
    end repeat                                           << total << endl;
end if                                               } //end for
                                                  }  //end if
```

Figure 8-16 IPO chart information and C++ instructions for the retirement program

The fifth step in the problem-solving process is to desk-check the program. Figure 8-17 shows the completed desk-check table for the retirement program. The results agree with those shown in the algorithm's desk-check table in Figure 8-15.

age	currentSalary	rate	yearsToRetire
~~0~~	~~0~~	~~0.0~~	~~0~~
62	25000	~~0.03~~	3
		~~0.04~~	
		~~0.05~~	
		0.06	

newSalary	total		year
~~0.0~~	~~0.0~~		~~0~~
~~25000.0~~	~~25000.0~~		~~2~~
~~25750.0~~	~~50750.0~~		~~3~~
~~26522.5~~	~~77272.5~~	3% rate total	~~4~~
~~25000.0~~	~~25000.0~~		~~2~~
~~26000.0~~	~~51000.0~~		~~3~~
~~27040.0~~	~~78040.0~~	4% rate total	~~4~~
~~25000.0~~	~~25000.0~~		~~2~~
~~26250.0~~	~~51250.0~~		3
27562.5	78812.5	5% rate total	4

Figure 8-17 Desk-check table for the retirement program

The final step in the problem-solving process is to evaluate and modify (if necessary) the program. Recall that you evaluate a program by entering its instructions into the computer and then using the computer to run (execute) it. While the program is running, you enter the same sample data used when desk-checking the program.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

DIRECTIONS

1. Follow the instructions for starting your C++ development tool. Depending on the development tool you are using, you may need to create a new project; if so, name the project Lab8-2 Project, and save it in the Cpp8\Chap08 folder. Enter the instructions shown in Figure 8-18 in a source file named Lab8-2.cpp. (Do not enter the line numbers.) Save the file in either the project folder or the Cpp8\Chap08 folder. Now follow the appropriate instructions for running the Lab8-2.cpp file. Test the program using 62 as the age and 25000 as the current salary. The total earning amounts should be $77273, $78040, and $78813. If necessary, correct any bugs (errors) in the program.

```
 1 //Lab8-2.cpp - displays a person's total
 2 //earnings before retirement at age 65,
 3 //using annual raise rates of 3%, 4%, and 5%
 4 //Created/revised by <your name> on <current date>
 5
 6 #include <iostream>
 7 #include <iomanip>
 8 using namespace std;
 9
10 int main()
11 {
12    int age = 0;
13    int currentSalary = 0;
14    int yearsToRetire = 0;
15    double newSalary = 0.0;
16    double total = 0.0;
17
18    cout << fixed << setprecision(0);
19
20    cout << "Current age in years (1 to 64): ";
21    cin >> age;
22
23    if (age < 1 || age > 64)
24        cout << "Please enter an age from 1 to 64." << endl;
25    else
26    {
27        cout << "Current salary as a whole number: ";
28        cin >> currentSalary;
29        cout << endl;
30
31        yearsToRetire = 65 - age;
32        for (double rate = 0.03; rate < 0.06; rate += 0.01)
33        {
34            newSalary = currentSalary;  //year 1 salary
35            total = currentSalary;      //year 1 salary
36            for (int year = 2; year <= yearsToRetire; year += 1)
37            {
38                newSalary *= (1 + rate);
39                total += newSalary;
```

Figure 8-18 Retirement program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
40            } //end for
41            cout << "Total with a " << rate * 100
42                 << "% annual raise: $" << total << endl;
43        } //end for
44    }  //end if
45    return 0;
46 }  //end of main function
```

Figure 8-18 Retirement program

LAB 8-3 Modify

If necessary, create a new project named Lab8-3 Project, and save it in the Cpp8\Chap08 folder. Enter (or copy) the Lab8-2.cpp instructions into a new source file named Lab8-3.cpp. Change Lab8-2.cpp in the first comment to Lab8-3.cpp. Change the outer `for` loop to a posttest loop. Save, run, and test the program.

LAB 8-4 What's Missing?

The program in this lab should display the pattern of numbers shown in Figure 8-19. Start your C++ development tool, and view the Lab8-4.cpp file, which is contained in either the Cpp8\Chap08\Lab8-4 Project folder or the Cpp8\Chap08 folder. (Depending on your C++ development tool, you may need to open Lab8-4's project/solution file first.) Put the C++ instructions in the proper order, and then determine the one or more missing instructions. Test the program appropriately.

If the user enters the number 4 in response to the "How many rows?" prompt, the program should display the following pattern of numbers:

```
1
12
123
1234
```

Figure 8-19 Sample output for Lab 8-4

LAB 8-5 Desk-Check

Desk-check the code shown in Figure 8-20. What will the code display on the computer screen? What is the value in the `x` variable when the outer `for` loop ends? What is the value in the `y` variable when the nested `for` loop ends?

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

```
int sumX = 0;
int sumY = 0;

for (int x = 2; x < 7; x += 2)
{
     sumX += x;
     for (int y = 1; y < 4; y += 2)
          sumY += y;
     //end for
} //end for
cout << "sumX value: " << sumX << endl;
cout << "sumY value: " << sumY << endl;
```

Figure 8-20 Code for Lab 8-5

LAB 8-6 Debug

Follow the instructions for starting C++ and viewing the Lab8-6.cpp file, which is contained in either the Cpp8\Chap08\Lab8-6 Project folder or the Cpp8\Chap08 folder. (Depending on your C++ development tool, you may need to open Lab8-6's project/solution file first.) The program should display a store's quarterly sales, but it is not working correctly. Debug the program.

Chapter Summary

A repetition structure can be either a pretest loop or a posttest loop. In a pretest loop, the loop condition is evaluated *before* the instructions within the loop are processed. In a posttest loop, the evaluation occurs *after* the instructions within the loop are processed.

The condition appears at the end of a posttest loop and determines whether the instructions within the loop body will be processed more than once. The loop's condition must result in either a true or false answer only. When the condition evaluates to true, the instructions listed in the loop body are processed again; otherwise, the loop is exited.

You use the `do while` statement to code a posttest loop in C++. To code a pretest loop in C++, you can use either the `while` statement or the `for` statement.

Repetition structures can be nested, which means one loop can be placed inside another loop. For nested repetition structures to work correctly, the inner (nested) loop must be contained entirely within the outer loop.

Key Terms

`do while` statement—the statement used to code a posttest loop in C++

Nested loop—a loop (repetition structure) contained entirely within another loop (repetition structure)

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Review Questions

1. The condition in the `do while` statement is evaluated _____ the instructions in the loop body are processed.

   a. after
   b. before

2. The instructions in the body of the _____ statement are always processed at least once during runtime.

   a. `do while`
   b. `for`
   c. `while`
   d. both a and b

3. It is possible that the instructions in the body of the _____ statement will not be processed during runtime.

   a. `do while`
   b. `for`
   c. `while`
   d. both b and c

4. What numbers will the following code display on the computer screen?

```
int x = 0;
do
{
        cout << x << endl;
        x += 1;
} while (x < 5);
```

   a. 0, 1, 2, 3, 4
   b. 0, 1, 2, 3, 4, 5
   c. 1, 2, 3, 4
   d. 1, 2, 3, 4, 5

5. What numbers will the following code display on the computer screen?

```
int x = 16;
do
{
        cout << x << endl;
        x -= 4;
} while (x > 10);
```

   a. 16, 12, 8
   b. 16, 12
   c. 20, 16, 12, 8
   d. 20, 16, 12

6. What value of `x` causes the loop in Review Question 5 to end?

   a. 0
   b. 8
   c. 10
   d. 12

7. What numbers will the following code display on the computer screen?

```
int total = 1;
do
{
        total += 2;
        cout << total << endl;
} while (total <= 3);
```

   a. 1, 2
   b. 1, 3
   c. 3
   d. 3, 5

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

8. What pattern of asterisks will the following code display on the computer screen?

```
for (int x = 1; x < 3; x += 1)
{
        for (int y = 1; y < 4; y += 1)
                cout << "*";
        //end for
        cout << endl;
} //end for
```

a.
```
***
***
```

b.
```
***
***
***
```

c.
```
**
**
**
```

d.
```
***
***
***
***
```

9. What number will the following code display on the computer screen?

```
int sum = 0;
int y = 0;
do
{
        for (int x = 1; x <= 5; x += 1)
                sum += x;
        //end for
        y += 1;
} while (y < 2);
cout << sum << endl;
```

a. 5
b. 8
c. 15
d. 30

Exercises

Pencil and Paper

TRY THIS

1. Complete a desk-check table for the code shown in Figure 8-21. What will the code display on the computer screen? What value causes the nested loop to end? What value causes the outer loop to end? (The answers to TRY THIS Exercises are located at the end of the chapter.)

```
int nested = 0;
for (int outer = 1; outer <= 2; outer += 1)
{
     nested = 3;
     do    //begin loop
     {
          cout << nested << "     ";
          nested -= 1;
     }  while (nested > 0);
     cout << endl;
}  //end for
```

Figure 8-21

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

2. Complete a desk-check table for the code shown in Figure 8-22. What will the code display on the computer screen? What value causes the nested loop to end? What value causes the outer loop to end? (The answers to TRY THIS Exercises are located at the end of the chapter.) TRY THIS

```
for (int region = 3; region > 0; region -= 1)
{
     cout << "Region: " << region << endl;
     for (int store = 1; store <= 2; store += 1)
          cout "      Store: " << store << endl;
     //end for
     cout << endl;
}  //end for
```

Figure 8-22

3. Rewrite the code from Figure 8-21 so it uses the `for` statement for the nested loop. MODIFY THIS

4. Chakura is sitting at a table in a bookstore, attending her book signing. Customers are standing in line waiting for her to sign their copy of her book. However, it is possible that some customers in line may not have a book; they are in line simply to meet her. It's also possible that some customers may have more than one book for her to sign. Write an appropriate algorithm using only the instructions listed in Figure 8-23. INTRODUCTORY

```
accept the book from the customer
close the book
end repeat
end repeat
open the front cover of the book
place the book on the table
repeat while (the customer has a book that needs signing)
repeat while (there are customers in line)
return the book to the customer
sign your name on the first page
thank the customer
```

Figure 8-23

5. Write a C++ `while` clause that processes a posttest loop's instructions as long as the value in the `inStock` variable is greater than the value in the `reorder` variable. INTRODUCTORY

6. A program declares an `int` variable named `evenNum` and initializes it to 2. Write the C++ code to display the even integers 2, 4, 6, 8, and 10 on separate lines on the computer screen. Use the `do while` statement. INTERMEDIATE

7. Write the C++ code to display the integers 15, 12, 9, 6, 3, and 0 on separate lines on the computer screen. Use the `for` statement and an `int` variable named `num`. INTERMEDIATE

8. The code in Figure 8-24 should display the pattern of ampersands shown in the figure, but it is not working correctly. Debug the code. SWAT THE BUGS

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

```
for (int row = 1; row < 4; row += 1)
{
     for (int col = 1; col <= 5; col += 1)
          cout << "& ";
     //end for
     cout << endl;
}  //end for

Pattern
&
& &
& & &
& & & &
```

Figure 8-24

Computer

TRY THIS

9. In this exercise, you will create a program that uses two `for` statements to display the pattern of asterisks shown in Figure 8-25. If necessary, create a new project named TryThis9 Project, and save it in the Cpp8\Chap08 folder. Enter the C++ instructions into a source file named TryThis9.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Save and then run the program. (The answers to TRY THIS Exercises are located at the end of the chapter.)

```
**********
********
******
****
**
```

Figure 8-25

TRY THIS

10. In this exercise, you will create a program that displays the pattern of asterisks shown in Figure 8-26. Use the `while` statement for the outer loop. Use the `do while` statement for the nested loop. If necessary, create a new project named TryThis10 Project, and save it in the Cpp8\Chap08 folder. Enter the C++ instructions into a source file named TryThis10.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Save and then run the program. (The answers to TRY THIS Exercises are located at the end of the chapter.)

```
*****
****
***
**
*
```

Figure 8-26

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

11. In this exercise, you will modify the code from Exercise 10. If necessary, create a new project named ModifyThis11 Project, and save it in the Cpp8\Chap08 folder. Enter (or copy) the TryThis10.cpp instructions into a new source file named ModifyThis11.cpp. Change TryThis10.cpp in the first comment to ModifyThis11.cpp. Replace the `while` and `do while` statements with `for` statements. Save and then run the program. MODIFY THIS

12. In this exercise, you will modify the car depreciation program from Figure 8-12. Follow the instructions for starting C++ and viewing the ModifyThis12.cpp file, which is contained in either the Cpp8\Chap08\ModifyThis12 Project folder or the Cpp8\Chap08 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) Replace the nested `for` statement with a `while` statement. Test the program appropriately. MODIFY THIS

13. Create a program that a professor can use to display a grade for any number of students. Each student's grade is based on four test scores, with each test worth 100 points. The program should total the student's test scores and then assign the appropriate grade using the information shown in Figure 8-27. Display the student's number and grade in a message, such as "Student 1's grade: A". If necessary, create a new project named Introductory13 Project, and save it in the Cpp8\Chap08 folder. Enter the C++ instructions into a source file named Introductory13.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Save, run, and test the program. INTRODUCTORY

Total points earned	Grade
372–400	A
340–371	B
280–339	C
240–279	D
below 240	F

Figure 8-27

14. Create a program that displays a table consisting of four rows and five columns. The first column should display the numbers 1 through 4. The second and subsequent columns should display the result of multiplying the number in the first column by the numbers 2 through 5. If necessary, create a new project named Introductory14 Project, and save it in the Cpp8\Chap08 folder. Enter the C++ instructions into a source file named Introductory14.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Save and then run the program. INTRODUCTORY

15. Create a program that displays a table consisting of ten rows and four columns. The first column should display prices from $5 to $50 in increments of $5. The second and subsequent columns should display the discounted prices, using discount rates of 10%, 15%, and 20%, respectively. If necessary, create a new project named Intermediate15 Project, and save it in the Cpp8\Chap08 folder. Enter the C++ instructions into a source file named Intermediate15.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Save and then run the program. INTERMEDIATE

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

INTERMEDIATE

16. In this exercise, you will modify the program from Lab 7-2 in Chapter 7. If necessary, create a new project named Intermediate16 Project, and save it in the Cpp8\Chap08 folder. Copy the instructions from the Lab7-2.cpp file into a source file named Intermediate16.cpp. The Lab7-2.cpp file is contained in either the Cpp8\Chap07\Lab7-2 Project folder or the Cpp8\Chap07 folder. (Alternatively, you can enter the instructions from Figure 7-44 into the Intermediate16.cpp file.) Change the filename in the first comment. Modify the program to allow the user to display the output for any number of sales amounts. Save and then run the program. Test the program twice, using sales amounts of 125000 and 96000.

INTERMEDIATE

17. In this exercise, you will modify the program from Lab 7-1 in Chapter 7. Follow the instructions for starting C++ and viewing the Intermediate17.cpp file, which is contained in either the Cpp8\Chap08\Intermediate17 Project folder or the Cpp8\Chap08 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) Currently, the program displays the average number of text messages sent each day for one week. Modify the program so that it also displays the average number of text messages sent each week for four weeks. Test the program appropriately.

INTERMEDIATE

18. At the beginning of every year, Khalid receives a raise on his previous year's salary. He wants a program that calculates and displays the amount of his annual raises for the next three years, using rates of 3%, 4%, 5%, and 6%. The program should end when Khalid enters a sentinel value as the salary.

    a. Create an IPO chart for the problem, and then desk-check the algorithm using annual salaries of 30000 and 50000, followed by your sentinel value.
    b. List the input, processing, and output items, as well as the algorithm, in a chart similar to the one shown earlier in Figure 8-16. Then code the algorithm into a program.
    c. Desk-check the program using the same data used to desk-check the algorithm.
    d. If necessary, create a new project named Intermediate18 Project, and save it in the Cpp8\Chap08 folder. Enter your C++ instructions into a source file named Intermediate18.cpp. Also enter appropriate comments and any additional instructions required by the compiler.
    e. Save and then run the program. Test the program using the same data used to desk-check the program.

INTERMEDIATE

19. Create a program that allows the user to enter an unknown number of sales amounts for each of three car dealerships: Dealership 1, Dealership 2, and Dealership 3. The program should calculate the total sales. Display the total sales with a dollar sign and two decimal places.

    a. Create an IPO chart for the problem, and then desk-check the algorithm using 23000 and 15000 as the sales amounts for Dealership 1; 12000, 16000, 34000, and 10000 for Dealership 2; and 64000, 12000, and 70000 for Dealership 3.
    b. List the input, processing, and output items, as well as the algorithm, in a chart similar to the one shown earlier in Figure 8-16. Then code the algorithm into a program.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

c. Desk-check the program using the same data used to desk-check the algorithm.
d. If necessary, create a new project named Intermediate19 Project, and save it in the Cpp8\Chap08 folder. Enter your C++ instructions into a source file named Intermediate19.cpp. Also enter appropriate comments and any additional instructions required by the compiler.
e. Save and then run the program. Test the program using the same data used to desk-check the program.

20. ADVANCED

Follow the instructions for starting C++ and viewing the Advanced20.cpp file, which is contained in either the Cpp8\Chap08\Advanced20 Project folder or the Cpp8\Chap08 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.)

a. Run the program, which displays five rows of asterisks. Close the Command Prompt window.
b. Modify the program to allow the user to specify the outer loop's ending and increment values. The ending value determines the maximum number of asterisks to display. The increment value determines the number of asterisks to repeat.
c. Save and then run the program. Test the program by entering the numbers 4 and 1 as the maximum number of asterisks and the number of asterisks to repeat, respectively. The program should display four rows of asterisks as follows: one asterisk, two asterisks, three asterisks, and four asterisks.
d. Run the program again. This time, enter the numbers 9 and 3 as the maximum number of asterisks and the number of asterisks to repeat, respectively. The program should display three rows of asterisks as follows: three asterisks, six asterisks, and nine asterisks.
e. Run the program again. Enter 7 and 3 as the maximum number of asterisks and the number of asterisks to repeat, respectively. The program displays only two rows of asterisks. The first row contains the expected three asterisks, but the second row contains six asterisks rather than seven asterisks. This is because the maximum number of asterisks (7) is not evenly divisible by the number of asterisks to repeat (3). Modify the program so that it displays the asterisks only when the maximum number is evenly divisible by the number to repeat; otherwise, display the message "The maximum number must be evenly divisible by the number to repeat."
f. Save and then run the program. Test the program three times, using the data from Steps c, d, and e.

21. ADVANCED

Create a program that displays movie ratings in a bar chart, similar to the one shown in Figure 8-28. If necessary, create a new project named Advanced21 Project, and save it in the Cpp8\Chap08 folder. Enter your C++ instructions into a source file named Advanced21.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Save and then run the program. Test the program appropriately.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

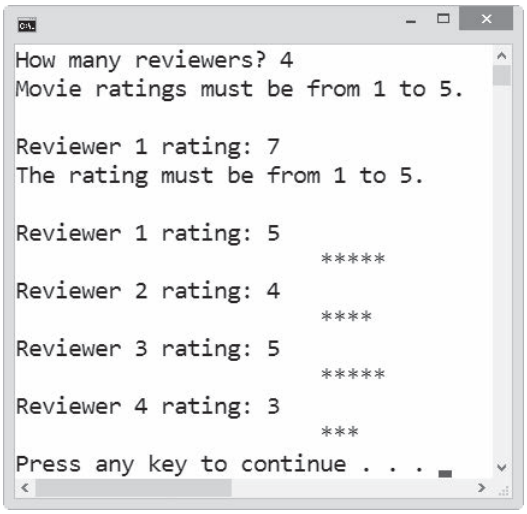


Figure 8-28

SWAT THE BUGS

22. Follow the instructions for starting C++ and viewing the SwatTheBugs22.cpp file, which is contained in either the Cpp8\Chap08\SwatTheBugs22 Project folder or the Cpp8\Chap08 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) Read the program's comments, and then debug the program.

Answers to TRY THIS Exercises

Pencil and Paper

1. See Figure 8-29. The number 0 causes the nested loop to end. The number 3 causes the outer loop to end.

outer	nested	The program displays the following:
1	0	3 2 1
2	3	3 2 1
3	2	
	1	
	0	
	3	
	2	
	1	
	0	

Figure 8-29

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

2. See Figure 8-30. The number 3 causes the nested loop to end. The number 0 causes the outer loop to end.

region	store	The program displays the following:
3	1	Region: 3
2	2	Store: 1
1	3	Store: 2
0	1	
	2	Region: 2
	3	Store: 1
	1	Store: 2
	2	
	3	Region: 1
		Store: 1
		Store: 2

Figure 8-30

Computer

9. See Figure 8-31.

```
//TryThis9.cpp - displays a pattern of asterisks
//Created/revised by <your name> on <current date>

#include <iostream>
using namespace std;

int main()
{
     for (int row = 5; row > 0; row -= 1)
     {
          for (int col = 1; col <= row * 2; col += 1)
               cout << "*";
          //end for
          cout << endl;
     }  //end for

     return 0;
}  //end of main function
```

Figure 8-31

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

10. See Figure 8-32.

```
//TryThis10.cpp - displays a pattern of asterisks
//Created/revised by <your name> on <current date>

#include <iostream>
using namespace std;

int main()
{
     int row = 5;
     int col = 0;

     while (row > 0)
     {
          col = 1;
          do    //begin loop
          {
               cout << "*";
               col += 1;
          } while (col <= row);

          cout << endl;
          row -= 1;
     }  //end while
     return 0;
}  //end of main function
```

Figure 8-32

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

CHAPTER 9

Value-Returning Functions

After studying Chapter 9, you should be able to:

- Raise a number to a power using the **pow** function
- Return the square root of a number using the `sqrt` function
- Generate random numbers
- Create and invoke a function that returns a value
- Pass information *by value* to a function
- Write a function prototype
- Understand a variable's scope and lifetime

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Functions

In some programming languages, functions are called methods, subroutines, or procedures.

As you learned in Chapter 4, a function is a block of code that performs a task. Every C++ program contains at least one function, which is named `main`. However, most C++ programs contain many functions. Some of the functions used in a program are built into the C++ language. The code for these **built-in functions** resides in C++ libraries, which are special files that come with the C++ language. You are already familiar with two built-in functions: `toupper` and `tolower`.

Other functions, like `main`, are created by the programmer. These functions are often referred to as **program-defined functions** because the function definitions are typically contained in the program itself rather than in a different file. But why would a programmer need more than the `main` function? One reason is to avoid the duplication of code. If the same task needs to be performed in more than one section of a program, it is more efficient for the programmer to enter the code in a function and then have each section call (or invoke) the function when needed.

Program-defined functions also allow large and complex programs, which are usually written by a team of programmers, to be broken into small and manageable tasks. Each member of the team is assigned one or more tasks to code as a function. Doing this allows more than one programmer to work on a program at the same time, decreasing the time it takes to write the program. When each programmer completes his or her function, all of the functions are gathered together into one program. Typically, a program's `main` function is responsible for calling (or invoking) each of the other program-defined functions. However, any program-defined function can call any other program-defined or built-in function.

All program-defined and built-in functions are categorized as either value-returning functions or void functions. Value-returning functions return a value after performing their assigned task. Void functions, on the other hand, do *not* return a value after completing their task. You are already familiar with the concept of something being either value-returning or void. The two illustrations shown in Figure 9-1 can be used to demonstrate this fact.

Illustration A

Helen:
1. ask ticket agent for a senior ticket
2. give ticket agent $5
3. receive senior ticket from ticket agent

Ticket agent (value-returning function):
1. take $5 from Helen
2. give Helen a senior ticket

Illustration B

Helen:
1. tell Penelope to have fun playing games
2. give Penelope $5

Penelope (void function):
1. take $5 from Helen
2. buy game tickets with the $5
3. play games and have fun

Figure 9-1 Illustrations of value-returning and void functions
Image by Diane Zak; created with Reallusion CrazyTalk Animator

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

In Illustration A, Helen is at the ticket counter in her local movie theater, requesting a ticket for the current movie. Helen gives the ticket agent a $5 bill and expects a ticket in return. The ticket agent is similar to a value-returning function. He performs his task (fulfilling her request for a ticket) and then returns a value (a ticket) to Helen. Compare that with Illustration B, where Helen and her granddaughter, Penelope, are at the Blast Off Games arcade. Helen wants Penelope to have fun, so she gives Penelope a $5 bill to play some games. But, unlike with the ticket agent, Helen expects nothing from Penelope in return. This is similar to the way a void function works. Penelope performs her task (having fun by playing games) but doesn't need to return any value to her grandmother.

You will learn about value-returning functions in this chapter. Void functions are covered in Chapter 10.

Value-Returning Functions

All **value-returning functions**, whether built-in or program-defined, perform a task and then return precisely one value after the task is completed. The built-in value-returning `toupper` function, for example, temporarily converts a character to uppercase and then returns the result. Likewise, the `tolower` function returns the result of temporarily converting a character to lowercase.

In almost all cases, a value-returning function returns its one value to the statement from which it was called (invoked). One exception is the `main` function, which returns its one value to the operating system. Typically, the statement that invokes a function assigns the return value to a variable. However, it may also use the return value in a calculation or comparison; or it may simply display the return value.

The value returned by the `main` function indicates whether the program ended normally.

The first part of this chapter covers four of the value-returning functions built into C++: `pow`, `sqrt`, `rand`, and `time`. It also covers a built-in void function named `srand`. Later in the chapter, you will learn how to create program-defined value-returning functions.

At this point, it is important to point out that functions are one of the more challenging topics for beginning programmers. Therefore, do not be concerned if you do not understand everything right away. If you still feel overwhelmed by the end of the chapter, try reading the chapter again, paying particular attention to the examples and programs shown in the figures. And be sure to complete the six labs at the end of the chapter.

The pow Function

Some mathematical expressions require a number (called the base number) to be raised to a power (called the exponent). For example, in the expression 4^2, 4 is the base number, and 2 is the exponent. The expression indicates that the number 4 should be squared—in other words, multiplied by itself, like this: 4 * 4. The result of the expression is the number 16. Similarly, the expression 5^3 means to cube the number, which means to multiply it by itself three times, like this: 5 * 5 * 5. This expression evaluates to 125. Raising a number to a power is referred to as **exponentiation**.

Recall from Chapter 4 that `#include` directives provide a convenient way to merge the source code from one file with the source code in another file, without having to retype the code.

C++ provides the **pow function** for performing exponentiation in a program. The function's code is contained in the cmath library file. Therefore, a program must contain the `#include <cmath>` directive in order to use the function. The directive tells the C++ compiler the location of the function's code.

Figure 9-2 shows the `pow` function's syntax. Recall from Chapter 5 that the items within parentheses in a function's syntax—in this case, *x* and *y*—are called arguments. More specifically, they are called actual arguments. An **actual argument** represents information that the function

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

needs to perform its task, and it can be a variable, named constant, literal constant, or keyword; however, in most cases it will be a variable. The pow function contains two actual arguments because it requires two items of information: the base number (*x*) and the exponent (*y*).

The actual arguments are passed to the pow function when it is invoked. Invoking a function is also referred to as calling a function. You call a function simply by including its name and actual arguments (if any) in a program statement, as shown in the examples in Figure 9-2. When the pow function is called, it raises *x* to power *y* and then returns the result as a `double` number.

HOW TO Use the pow Function

Syntax
pow(*x*, *y***)** — requires the `#include <cmath>` directive

Example 1
```
double cube = 0.0;
cube = pow(4.0, 3);
```
The assignment statement assigns the number 64.0, which is 4.0 raised to the third power, to the `cube` variable.

Example 2
```
cout << pow(100, 0.5);
```
The statement displays the number 10, which is 100 raised to the 0.5 power. The `pow(100, 0.5)` expression is equivalent to finding the square root of the number 100.

Example 3
```
double area = 0.0;
double radius = 5.0;
area = 3.14 * pow(radius, 2);
```
The assignment statement raises the value stored in the `radius` variable to the second power, giving 25.0. It then multiplies the 25.0 by 3.14 and assigns the product (78.5) to the `area` variable.

Figure 9-2 How to use the pow function

The `sqrt` Function

Although you can use the pow function to find the square root of a number, as shown in Example 2 in Figure 9-2, C++ provides the `sqrt` function specifically for that purpose. Like the pow function, the `sqrt` function is defined in the cmath library file. Therefore, a program must contain the `#include <cmath>` directive in order to use the function. The **sqrt function** calculates a number's square root and then returns the result as a `double` number.

As the syntax in Figure 9-3 indicates, the `sqrt` function requires one actual argument: the number whose square root you want to find. The number must have a data type of either `double` or `float`. Also included in Figure 9-3 are examples of statements that invoke the function.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

How To Use the `sqrt` Function

Syntax
sqrt(*x*) — requires the `#include <cmath>` directive

Example 1
```
double squareRoot = 0.0;
squareRoot = sqrt(100.0);
```
The `sqrt` function finds the square root of the `double` number 100.0 and then returns the result (the `double` number 10.0) to the assignment statement, which assigns the return value to the `squareRoot` variable.

Example 2
```
double num = 0.0;
cout << "Enter a number: ";
cin >> num;
cout << sqrt(num);
```
The `sqrt` function finds the square root of the `double` number stored in the `num` variable and then returns the result to the `cout` statement, which displays the return value on the computer screen.

Figure 9-3 How to use the `sqrt` function

The Hypotenuse Program

The hypotenuse program covered in this section will use the Pythagorean theorem to calculate the length of a right triangle's hypotenuse, which is the longest side of the triangle. Figure 9-4 shows the theorem and provides an example of using it. Notice that the theorem requires raising a number to the second power (in other words, squaring the number) and also taking the square root of a number.

Pythagorean theorem
The theorem states that the length of a right triangle's hypotenuse is equal to the square root of the sum of the squares of the lengths of the triangle's two adjacent sides. In other words, the hypotenuse's length is equal to the square root of the following sum: (*side a length*)^2 + (*side b length*)^2.

Example: *side a length* is 10 and *side b length* is 24

1. square *side a length*	10 * 10 = 100
2. square *side b length*	24 * 24 = 576
3. sum the squares from Steps 1 and 2	100 + 576 = 676
4. find the square root of the sum from Step 3	26 — length of the hypotenuse

Figure 9-4 Pythagorean theorem

Figure 9-5 shows the problem specification, IPO chart information, and C++ instructions for the hypotenuse program. The pow and `sqrt` functions are shaded in the figure.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The flowchart for the hypotenuse program is contained in the Hypotenuse.pdf file.

Problem specification
Create a program that uses the Pythagorean theorem to calculate the length of a right triangle's hypotenuse, given the lengths of its two adjacent sides (*side a* and *side b*). Display the length with one decimal place.

IPO chart information	**C++ instructions**
Input	
side a length	`double sideA = 0.0;`
side b length	`double sideB = 0.0;`
Processing	
sum of the squares	`double sumSqrs = 0.0;`
Output	
hypotenuse length	`double hypotenuse = 0.0;`
Algorithm	

1. enter side a length and side b length	`cout << "Side a length: ";`
	`cin >> sideA;`
	`cout << "Side b length: ";`
	`cin >> sideB;`

2. calculate the sum of the squares = (side a length)^2 + (side b length)^2	`sumSqrs = pow(sideA, 2) + pow(sideB, 2);`
3. calculate the hypotenuse length by finding the square root of the sum of the squares	`hypotenuse = sqrt(sumSqrs);`
4. display the hypotenuse length	`cout << "Hypotenuse length: " << hypotenuse << endl;`

Figure 9-5 Problem specification, IPO chart information, and C++ instructions for the hypotenuse program

Figure 9-6 shows the completed program, along with a sample run of the program.

```
 1 //Hypotenuse.cpp - displays the length of a right
 2 //triangle's hypotenuse
 3 //Created/revised by <your name> on <current date>
 4
 5 #include <iostream>
 6 #include <iomanip>
 7 #include <cmath>
 8 using namespace std;
 9
10 int main()
11 {
12    double sideA = 0.0;
13    double sideB = 0.0;
14    double sumSqrs = 0.0;
15    double hypotenuse = 0.0;
16
17    //get lengths of two sides
```

required for the pow and sqrt functions

Figure 9-6 Hypotenuse program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
18    cout << "Side a length: ";
19    cin >> sideA;
20    cout << "Side b length: ";
21    cin >> sideB;
22
23    //calculate the hypotenuse length
24    sumSqrs = pow(sideA, 2) + pow(sideB, 2);
25    hypotenuse = sqrt(sumSqrs);
26
27    //display the hypotenuse length
28    cout << fixed << setprecision(1);
29    cout << "Hypotenuse length: " << hypotenuse << endl;
30    return 0;
31 }  //end of main function
```

use the pow and sqrt functions

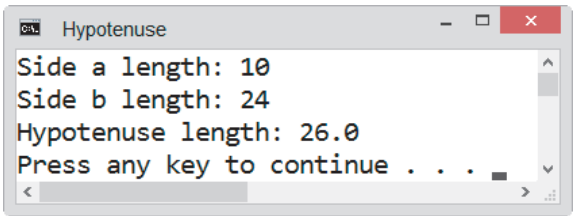


Figure 9-6 Hypotenuse program

The assignment statement on Line 24 invokes the pow function twice. The first time the function is invoked, it is passed the value stored in the `sideA` variable (10.0) and the integer 2. The function squares the `sideA` value passed to it and then returns the result (100.0) to the assignment statement that invoked it. The return value is stored in a temporary location in the computer's internal memory until the assignment statement's expression is fully evaluated.

The second time the pow function is invoked, it is passed the value stored in the `sideB` variable (24.0) and the integer 2. The function squares the `sideB` value passed to it and then returns the result (576.0) to the assignment statement that invoked it. At this point, the computer adds the function's first return value (100.0) to its second return value (576.0) and then stores the sum (676.0) in the `sumSqrs` variable.

Next, the assignment statement on Line 25 invokes the `sqrt` function, passing it the value stored in the `sumSqrs` variable (676.0). The function calculates the square root of the value passed to it and then returns the result (26.0) to the assignment statement, which assigns it to the `hypotenuse` variable.

The rand, srand, and time Functions

Many programs, such as game programs and lottery programs, require the use of random numbers. Because of this, most programming languages provide a specific function for producing random numbers. The numbers are not completely random, however, because a definite mathematical algorithm is used to select them; but they are sufficiently random for practical purposes. The function that produces the pseudo-random numbers is often referred to as a **pseudo-random number generator**.

In C++, the pseudo-random number generator is the **rand function**. The function returns an integer that is greater than or equal to 0 but less than or equal to the value stored in the **RAND_MAX** constant, which is one of many constants built into the C++ language. Although the value of RAND_MAX varies with different compilers, its value is always at least 32,767. (You can display your compiler's RAND_MAX value using the statement `cout << RAND_MAX;`.)

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

In C++, every function's name is followed by a set of parentheses, which may or may not contain actual arguments.

Figure 9-7 shows the `rand` function's syntax and includes examples of using the function to generate random integers. The empty set of parentheses after the function's name is required. Depending on your C++ development tool, you may need to include the `#include <cstdlib>` directive in your program in order to use `rand` and `RAND_MAX`.

How To Use the `rand` Function

Syntax
rand() — your compiler may require the `#include <cstdlib>` directive

Example 1
```
int randomNum = 0;
randomNum = rand();
```
The `rand` function generates a random integer that is greater than or equal to 0 but less than or equal to `RAND_MAX`. It then returns the random integer to the assignment statement, which assigns it to the `randomNum` variable.

Example 2
```
cout << rand();
```
The `rand` function generates a random integer that is greater than or equal to 0 but less than or equal to `RAND_MAX`. It then returns the random integer to the `cout` statement, which displays it on the computer screen.

Example 3
```
int tripleNum = 0;
tripleNum = rand() * 3;
```
The `rand` function generates a random integer that is greater than or equal to 0 but less than or equal to `RAND_MAX`. It then returns the random integer to the assignment statement, which multiplies it by 3 and assigns the result to the `tripleNum` variable.

Figure 9-7 How to use the `rand` function

Most programs that use random numbers require the numbers to be within a specific range. A program that simulates rolling dice, for example, will require integers from 1 through 6 only. Figure 9-8 shows the syntax of an expression that you can use to specify the desired range of integers. In the syntax, *lowerBound* and *upperBound* are the lowest integer and highest integer, respectively, in the range. Also included in Figure 9-8 are examples of using the expression in a C++ statement. The expression in Example 1 produces random integers in the range of 1 through 6. Example 2's expression produces random integers from 10 through 100. You can also write the expressions in Figure 9-8 as `1 + rand() % 6` and `10 + rand() % 91`. However, including the *lowerBound* and *upperBound* values within the parentheses makes the expression clearer and more self-documenting. The figure also shows how the computer evaluates both examples using sample `rand` function values. The resulting random integers are shaded in the figure.

As you learned in Chapter 4, the modulus operator divides two integers and then returns the remainder as an integer.

How To Generate Random Integers within a Specific Range

Syntax
lowerBound **+ rand() % (***upperBound* **–** *lowerBound* **+ 1)**

Example 1
```
cout << 1 + rand() % (6 - 1 + 1);
```
displays a random integer from 1 through 6 on the computer screen

Figure 9-8 How to generate random integers within a specific range *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Ch09-Random Range

Results using different rand values:

rand value: 27	1 + 27 % (6 - 1 + 1)
6 - 1 + 1 is evaluated first and results in 6	1 + 27 % 6
27 % 6 is evaluated next and results in 3	1 + 3
1 + 3 is evaluated last and results in 4	4
rand value: 8	1 + 8 % (6 - 1 + 1)
6 - 1 + 1 is evaluated first and results in 6	1 + 8 % 6
8 % 6 is evaluated next and results in 2	1 + 2
1 + 2 is evaluated last and results in 3	3
rand value: 324	1 + 324 % (6 - 1 + 1)
6 - 1 + 1 is evaluated first and results in 6	1 + 324 % 6
324 % 6 is evaluated next and results in 0	1 + 0
1 + 0 is evaluated last and results in 1	1

Example 2

```
int num = 0;
num = 10 + rand() % (100 - 10 + 1);
```

assigns a random integer from 10 through 100 to the num variable

Results using different rand values:

rand value: 352	10 + 352 % (100 - 10 + 1)
100 - 10 + 1 is evaluated first and results in 91	10 + 352 % 91
352 % 91 is evaluated next and results in 79	10 + 79
10 + 79 is evaluated last and results in 89	89
rand value: 4	10 + 4 % (100 - 10 + 1)
100 - 10 + 1 is evaluated first and results in 91	10 + 4 % 91
4 % 91 is evaluated next and results in 4	10 + 4
10 + 4 is evaluated last and results in 14	14
rand value: 2500	10 + 2500 % (100 - 10 + 1)
100 - 10 + 1 is evaluated first and results in 91	10 + 2500 % 91
2500 % 91 is evaluated next and results in 43	10 + 43
10 + 43 is evaluated last and results in 53	53

Figure 9-8 How to generate random integers within a specific range

As indicated in Figure 9-8, the expression in Example 1 evaluates to the integers 4, 3, and 1 when the rand values are 27, 8, and 324, respectively. Notice that the three random integers (4, 3, and 1) are within the range of 1 through 6. The expression in Example 2 evaluates to 89, 14, and 53 when the rand values are 352, 4, and 2500, respectively. In this case, the three random integers (89, 14, and 53) are within the required range of 10 through 100.

You should initialize the random number generator in each program in which it is used. Otherwise, it will generate the same series of numbers each time the program is executed. Typically, the initialization task is performed at the beginning of the program. You initialize the generator using the **srand function**. Like the rand function, the srand function is a built-in C++ function. However, unlike the rand function, the srand function is a void function, which

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

means it does not return a value. (You will learn more about void functions in Chapter 10.) Depending on your C++ development tool, you may need to include the `#include <cstdlib>` directive in your program in order to use `srand`.

Figure 9-9 shows the syntax of the `srand` function and includes examples of using the function. The *seed* actual argument in the syntax is an integer that represents the starting point for the random number generator. The computer uses the starting point (or seed) in the mathematical algorithm it employs when selecting the random numbers. You can have the user enter the seed, as shown in Example 1 in Figure 9-9. However, a more common way to initialize the generator is to use the C++ `time` function as the seed, as shown in Examples 2 and 3.

The **time function** is a built-in value-returning function that returns the current time (according to your computer system's clock) as seconds elapsed since midnight on January 1, 1970. However, because the `time` function returns a `time_t` object, you will need to use a type cast to convert the function's return value to an integer, as shown in Examples 2 and 3 in Figure 9-9. In both examples, the `time` function is passed one actual argument: the number 0. Using the `time` function as the `srand` function's seed ensures that the random number generator is initialized with a unique integer each time the program is executed. The unique integer will produce a unique series of random numbers. To use the `time` function in a program, the program must contain the `#include <ctime>` directive.

HOW TO Use the `srand` Function

Syntax

srand(*seed***)**

your compiler may require the `#include <cstdlib>` directive

Example 1

```
int x = 0;
cout << "Enter an integer: ";
cin >> x;
srand(x);
cout << rand() << endl;
```

The `srand` function initializes the random number generator using the integer entered by the user. The `cout` statement displays a random integer that will be greater than or equal to 0 but less than or equal to `RAND_MAX`.

Example 2

the `time` function requires the `#include <ctime>` directive

```
srand(static_cast<int>(time(0)));
cout << rand() << endl;
```

The `srand` function initializes the random number generator using the value returned by the `time` function after it has been converted to the `int` data type. The `cout` statement displays a random integer that will be greater than or equal to 0 but less than or equal to `RAND_MAX`.

Example 3

the `time` function requires the `#include <ctime>` directive

```
int randNum = 0;
srand(static_cast<int>(time(0)));
randNum = 1 + rand() % (10 - 1 + 1);
```

The `srand` function initializes the random number generator using the value returned by the `time` function after it has been converted to the `int` data type. The assignment statement assigns a random integer that is greater than or equal to 1 but less than or equal to 10 to the `randNum` variable.

Figure 9-9 How to use the `srand` function

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The Guessing Game Program

Figure 9-10 shows the IPO chart information and corresponding C++ instructions for the guessing game program. The program generates a random number from 1 through 10 and then allows the user as many chances as needed to guess the number. The `srand`, `time`, and `rand` functions are shaded in the figure.

Problem specification
Create a program that generates a random number from 1 through 10 and then allows the user as many chances as needed to guess the number. If the user guesses the number, display the message "Yes, the number is" followed by the number. Otherwise, display the "Sorry, guess again:" message.

IPO chart information	C++ instructions
Input	
random number (from 1 through 10)	`int randomNumber = 0;`
guess	`int numberGuess = 0;`
Processing	
none	
Output	
appropriate message	
Algorithm	
1. generate random number	`srand(static_cast<int>(time(0)));` `randomNumber = 1 + rand() % (10 - 1 + 1);`
2. enter guess	`cout << "Guess a number from 1 through 10: ";` `cin >> numberGuess;`

3. repeat while (guess is not random number)
display "Sorry, guess again:" message
enter guess
end repeat
	`while (numberGuess != randomNumber)`
	`{`
	`cout << "Sorry, guess again: ";`
	`cin >> numberGuess;`
	`} //end while`

4. display "Yes, the number is" and the random number	`cout << endl << "Yes, the number is "` `<< randomNumber << "." << endl;`

Figure 9-10 Problem specification, IPO chart information, and C++ instructions for the guessing game program

Figure 9-11 shows the completed guessing game program and includes a sample run of the program. The statement on Line 15 uses the `srand` and `time` functions to initialize the random number generator. The `rand` function, which appears in the assignment statement on Line 16, generates a random integer from 1 through 10. The statement assigns the random integer to the `randomNumber` variable.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

```
 1 //Guessing Game.cpp - number guessing game
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <ctime>
 6 //#include <cstdlib>
 7 using namespace std;
 8
 9 int main()
10 {
11    int randomNumber = 0;
12    int numberGuess = 0;
13
14    //generate a random number from 1 through 10
15    srand(static_cast<int>(time(0)));
16    randomNumber = 1 + rand() % (10 - 1 + 1);
17
18    //get first guess from user
19    cout << "Guess a number from 1 through 10: ";
20    cin >> numberGuess;
21
22    while (numberGuess != randomNumber)
23    {
24        cout << "Sorry, guess again: ";
25        cin >> numberGuess;
26    }   //end while
27
28    cout << endl << "Yes, the number is "
29        << randomNumber << "." << endl;
30    return 0;
31 }  //end of main function
```

required for the `time` function

your compiler may require this directive to use the `rand` and `srand` functions

```
Guessing Game
Guess a number from 1 through 10: 5
Sorry, guess again: 8
Sorry, guess again: 2

Yes, the number is 2.
Press any key to continue . . .
```

Figure 9-11 Guessing game program

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 9-1

1. Which of the following expressions will return the square root of the number 16?
   a. `pow(16.0, 2)`
   b. `sqrt(16.0)`
   c. `sqrt(16.0, 0.5)`
   d. both a and b

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

2. Which of the following expressions will generate a random integer from 25 through 50, inclusive?

   a. `1 + rand() % (50 - 25 + 1)`
   b. `50 + rand() % (50 - 25 + 1)`
   c. `25 + rand() % (50 - 25 + 1)`
   d. `25 + rand() % (50 - 25 - 1)`

3. Which of the following functions initializes the random number generator in C++?

   a. `initialize()`
   b. `startRand()`
   c. `rand(time(0))`
   d. none of the above

4. Which of the following directives is necessary for a program to use the C++ `time` function?

   a. `#include <ctime>`
   b. `#include <stime>`
   c. `#include <time>`
   d. none of the above

Creating Program-Defined Value-Returning Functions

As mentioned earlier, you can create your own functions in C++. The functions you create are referred to as program-defined functions because their definitions are typically contained in the program itself rather than in a different file. You already know how to create one program-defined value-returning function: `main`.

Figure 9-12 shows the syntax for creating (or defining) a value-returning function in a C++ program. The figure also shows examples of program-defined value-returning functions. The `getRandomNumber` function in Example 1 returns a random integer from 1 through 10. The `getRectangleArea` function in Example 2 calculates and returns the area of a rectangle, using the values stored in the `len` and `wid` variables. Both values will be passed to the function by the statement from which it is called. The function returns the area as a `double` number. The `getSubtotal` function in Example 3 uses the values passed to it to calculate a subtotal, which the function returns as a `double` number. At this point, you should not be concerned if you do not fully understand the examples. They will become clearer to you as you progress through the chapter.

HOW TO Create a Program-Defined Value-Returning Function

Syntax

```
returnDataType functionName([parameterList])      <- function header
{                                                 <- function body
        one or more statements
        return expression;
}       //end of functionName function
```

Figure 9-12 How to create a program-defined value-returning function *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Rather than using an empty set of parentheses when a function is not passed any information, as in Example 1, some programmers enter the keyword `void` within the parentheses.

Example 1

```
int getRandomNumber()
{
     int randInteger = 0;
     randInteger = 1 + rand() % (10 - 1 + 1);
     return randInteger;
}    //end of getRandomNumber function
```

function definition

The function generates a random integer from 1 through 10 and then returns the random integer.

Example 2

```
double getRectangleArea(double len, double wid)
{
     return len * wid;
}    //end of getRectangleArea function
```

The function calculates the area of a rectangle and then returns the result as a `double` number.

Example 3

```
double getSubtotal(int sold, double costPerItem)
{
     double subtotal = 0.0;
     subtotal = sold * costPerItem;
     return subtotal;
}    //end of getSubtotal function
```

The function calculates the subtotal and then returns the result as a `double` number.

Figure 9-12 How to create a program-defined value-returning function

Recall that a value-returning function can return only one value.

As Figure 9-12 indicates, a function definition contains a function header and a function body. The function header for a value-returning function begins with *returnDataType*, which indicates the data type of the value the function returns. The `getRandomNumber` function in the figure returns an integer; therefore, its *returnDataType* is `int`. The `getRectangleArea` and `getSubtotal` functions, on the other hand, return a `double` number and have a *returnDataType* of `double`.

The function header also specifies the function's name. The rules for naming functions are the same as for naming variables. However, it is a common practice to begin a function's name with a verb. To make your programs more self-documenting and easier to understand, you should use meaningful names that describe the task the function performs. The names of the functions in Figure 9-12 indicate that the functions return a random number, the area of a rectangle, and a subtotal.

The function header also contains an optional *parameterList* enclosed in parentheses. Keep in mind that only the *parameterList* is optional; the parentheses are a required part of the syntax. The *parameterList* contains the data type and name of one or more memory locations. The memory locations in a function's *parameterList* are called **formal parameters**. Each formal parameter will store an item of information that is passed to the function when the function is called. In Example 1 in Figure 9-12, the empty set of parentheses in the function header indicates that the `getRandomNumber` function will not be passed any information by the statement that calls it. The function header in Example 2, however, contains two formal parameters and indicates that the `getRectangleArea` function will be passed two items of information

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

when it is invoked. Both items will have the `double` data type. The `getSubtotal` function in Example 3 will receive an `int` item followed by a `double` item from the statement that invokes it. You will learn more about the *parameterList* later in the chapter and also in Chapter 10.

The function body in a function definition contains the instructions for performing the function's assigned task. The function body begins with the opening brace ({) and ends with the closing brace (}). Using a comment to mark the end of a program-defined function will make your programs easier to read and understand.

In most cases, the last statement in the function body of a value-returning function is `return` *expression*`;`, in which *expression* represents the function's one and only return value. The data type of the *expression* must agree with the *returnDataType* specified in the function header. The **return statement** returns the *expression*'s value to the statement that called the function. After the `return` statement is processed, the function ends, and program execution continues in the calling function.

Mini-Quiz 9-2

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

1. The header in a value-returning function consists of _____.
   a. the data type of the function's return value
   b. the function's name
   c. an optional *parameterList*
   d. all of the above

2. Which of the following is a valid function header for the `getArea` function? The function returns a `double` number and does not have any formal parameters.
   a. `double getArea()`
   b. `double getArea`
   c. `double getArea();`
   d. `double getArea;`

3. Write the function header for the `getGrossPay` function. The function returns a `double` number and has two formal parameters: an `int` variable named `hours` and a `double` variable named `rate`.

4. The `getGrossPay` function from Question 3 calculates and returns an employee's gross pay. Write a C++ statement that returns the gross pay to the statement that called the function. The gross pay is stored in a `double` variable named `gross`.

Calling a Function

For a function to perform its task, it must be called (or invoked). The `main` function in a C++ program is invoked automatically when the program is executed. Functions other than `main`, however, must be called by a statement within the program. As you observed in the programs from earlier in the chapter, you call a built-in function by including its name and actual arguments (if any) in the statement, like this: `hypotenuse = sqrt(sumSqrs);`. The statement invokes the built-in value-returning `sqrt` function, passing it one actual argument: the value stored in the `sumSqrs` variable. You use the same method to call a program-defined value-returning function, as shown in Figure 9-13.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

How To Call (Invoke) a Value-Returning Function

Syntax
functionName(**[***argumentList***]**)

Example 1
```
int num1 = 0;
num1 = getRandomNumber();
```
The assignment statement calls the `getRandomNumber` function and then assigns the function's return value to the `num1` variable.

Example 2
```
cout << getRectangleArea(7.25, 21.0);
```
The `cout` statement calls the `getRectangleArea` function, passing it the `double` numbers 7.25 and 21.0. It then displays the function's return value on the computer screen.

Example 3
```
int quantity = 0;
double itemPrice = 0.0;
double salesTax = 0.0;
cin >> quantity;
cin >> itemPrice;
salesTax = getSubtotal(quantity, itemPrice) * 0.05;
```
The assignment statement calls the `getSubtotal` function, passing it the integer stored in the `quantity` variable and the `double` number stored in the `itemPrice` variable. It then multiplies the function's return value by the `double` number 0.05 and stores the result in the `salesTax` variable.

Figure 9-13 How to call (invoke) a value-returning function

The number of actual arguments passed to a function should match the number of formal parameters in its function header. In addition, the data type and order (or position) of each actual argument must agree with the data type and order (position) of its corresponding formal parameter. This is because when the function is called, the computer stores the value of the first actual argument in the function's first formal parameter, the value of the second actual argument in its second formal parameter, and so on.

The assignment statement in Example 1 in Figure 9-13 calls the `getRandomNumber` function shown earlier in Figure 9-12. The empty set of parentheses in the function's header indicates that it requires no information to be passed to it in order to complete its task. Therefore, the function call—`getRandomNumber()`—contains an empty set of parentheses. The function generates a random integer from 1 through 10 and then returns the integer to the assignment statement that invoked it. The statement assigns the return value to the `num1` variable.

The function call in Example 2 passes two `double` numbers (7.25 and 21.0) to the `getRectangleArea` function from Figure 9-12. This is because the function's header contains two formal parameters, both of which have the `double` data type. The computer stores the number 7.25 in the `len` formal parameter and stores the number 21.0 in the `wid` formal parameter. The function uses the values stored in its formal parameters to calculate the area. It then returns the area to the `cout` statement that called it. That statement displays the area on the computer screen.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Ch09-Pass By Value

As you learned earlier, an actual argument can be a variable, named constant, literal constant, or keyword; however, in most cases it will be a variable. Each variable you declare in a program has both a value and a unique address that represents the location of the variable in the computer's internal memory. C++ allows you to pass either a copy of the variable's value or its address to a function. Passing a copy of a variable's value is referred to as **passing by value.** Passing a variable's address is referred to as **passing by reference**. Unless you specify otherwise, variables in C++ are automatically passed *by value*. For now, you do not need to concern yourself with passing *by reference* because all variables passed to functions in this chapter are passed *by value*. You will learn how to pass variables *by reference* in Chapter 10.

The function call in Example 3 in Figure 9-13 passes a copy of the values stored in two variables to the `getSubtotal` function from Figure 9-12. The function definition and the statement containing the function call are shown together in Figure 9-14.

```
salesTax = getSubtotal(quantity, itemPrice) * 0.05;
                       ↓          ↓

double getSubtotal(int sold, double costPerItem)
{
     double subtotal = 0.0;
     subtotal = sold * costPerItem;
     return subtotal;
}    //end of getSubtotal function
```

Figure 9-14 Function call and function definition

The `getSubtotal` function header indicates that the function is expecting to receive two values, in this order: an integer that represents the number of items sold and a `double` number that represents the cost per item. Because of this, the function call passes two actual arguments in the required data type and order: the `int quantity` variable first and the `double itemPrice` variable second. As the arrows in Figure 9-14 indicate, the computer stores a copy of the value contained in the first actual argument (`quantity`) in the first formal parameter (`sold`) and a copy of the value contained in the second actual argument (`itemPrice`) in the second formal parameter (`costPerItem`). Notice that the names of the actual arguments do not have to be identical to the names of their corresponding formal parameters. In fact, to avoid confusion, you should use different names for the arguments and their corresponding parameters.

The `getSubtotal` function uses the values in its formal parameters to calculate the subtotal. It then returns the subtotal (as a `double` number) to the assignment statement that called it. The assignment statement multiplies the function's return value by 0.05 and assigns the result to the `salesTax` variable.

Keep in mind that when the computer encounters a statement that calls a function, it temporarily leaves the calling function to process the code contained in the called function. It returns to the calling function only after the called function ends.

The Savings Account Program

Figure 9-15 shows the problem specification, IPO chart information, and C++ instructions for the savings account program. The program allows the user to enter the initial deposit made into a savings account and the annual interest rate. It then displays the amount of money in the account at the end of 1 through 3 years, assuming no additional deposits or withdrawals are made. As the problem specification states, you can calculate the savings account balances using the following formula: $b = p * (1 + r)^n$. In the formula, p is the principal (the amount of the initial deposit), r is the annual interest rate, n is the number of years, and b is the balance in the savings account at the end of the n^{th} year.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Problem specification
Create a program that allows the user to enter the initial deposit made into a savings account and the annual interest rate. The program should display the amount of money in the account at the end of 1 through 3 years, assuming no additional deposits or withdrawals are made. You can calculate the savings account balances using the following formula: $b = p * (1 + r)^n$. In the formula, p is the principal (the amount of the initial deposit), r is the annual interest rate, n is the number of years, and b is the balance in the savings account at the end of the n^{th} year.

Example 1: using 1000 for *p*, 0.02 for *r*, and 1 for *n*
$b = 1000 * (1 + 0.02)^1$
$b = 1020$

Example 2: using 5000 for *p*, 0.03 for *r*, and 2 for *n*
$b = 5000 * (1 + 0.03)^2$
$b = 5304.50$

main function **IPO chart information**	**main function** **C++ instructions**
Input	

deposit
rate
year (counter: 1 to 3)
	`int deposit = 0;`
	`double interestRate = 0.0;`
	this variable is created and initialized in the for clause

Processing	
none	
Output	
account balance (at end of each of the 3 years)	`double acctBalance = 0.0;`
Algorithm	

1. enter deposit and rate	`cout << "Deposit: ";`
	`cin >> deposit;`
	`cout << "Rate (in decimal form): ";`
	`cin >> interestRate;`

2. repeat for (year from 1 to 3)	`for (int year = 1; year < 4; year += 1)` `{`
call the getBalance function to calculate the current account balance; pass the deposit, rate, and year	`acctBalance = getBalance(deposit, interestRate, year);`
display the year and current account balance	`cout << "Year " << year << ": $"` `<< acctBalance << endl;`
end repeat	`} //end for`

Figure 9-15 Problem specification, IPO chart information, and C++ instructions for the savings account program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

getBalance function **IPO chart information**	**getBalance function** **C++ instructions**
Input	

deposit (formal parameter)
rate (formal parameter)
year (formal parameter)
	`int amount`
	`double rate`
	`int y`

Processing	
none	
Output	
account balance	`double balance = 0.0;`
Algorithm	
1. calculate the account balance 2. return the account balance	`balance = amount * pow((1 + rate), y);` `return balance;`

Figure 9-15 Problem specification, IPO chart information, and C++ instructions for the savings account program

Figure 9-16 shows the flowcharts for the savings account program. The call to a function is often placed in a rectangle with side borders, as shown in the figure.

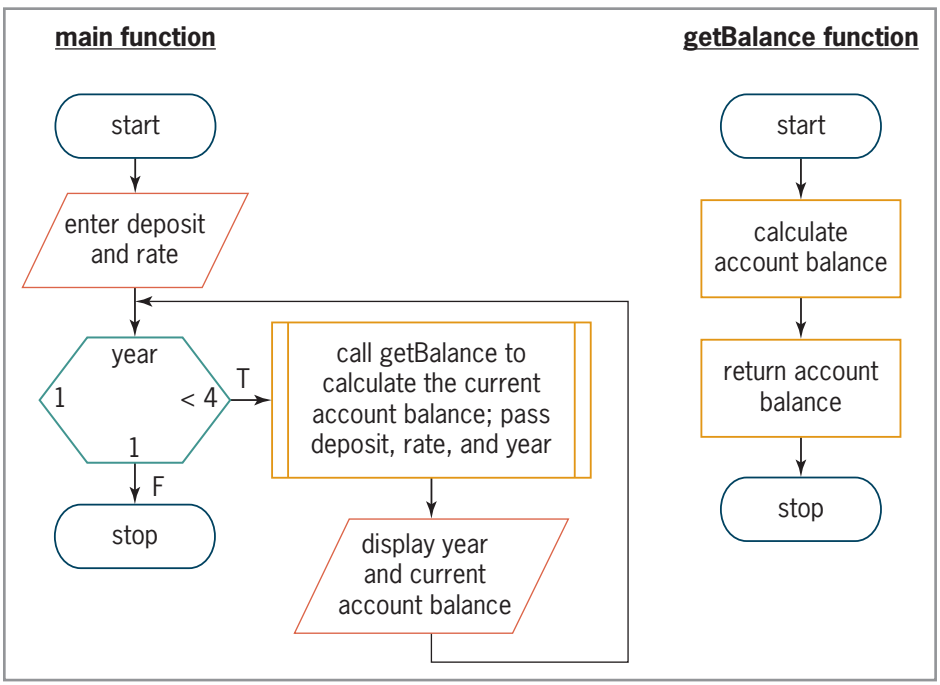


Figure 9-16 Flowcharts for the savings account program

Before you can enter the C++ instructions from Figure 9-15 into a file, you need to learn about function prototypes.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Function Prototypes

Most C++ programmers enter the function definitions below the `main` function in a program. When a function definition appears below the `main` function, you must enter a function prototype *above* the `main` function. Otherwise, the compiler won't recognize the function's name when it is used in the `main` function. A **function prototype** is a statement that specifies the function's name, the data type of its return value, and the data type of each of its formal parameters (if any). You can also include each formal parameter's name; however, that is not a requirement.

A program will have one function prototype for each function defined *below* the `main` function. You usually place the function prototypes at the beginning of the program, after the `#include` directives and `using namespace std;` statement. A function prototype alerts the C++ compiler that the function will be defined later in the program. The function prototypes in a program are similar to the table of contents in a book. Like each entry in a table of contents, each prototype is simply a preview of what will be expanded on later in the program (or in the book).

Keep in mind that a function prototype is necessary only when the function is defined *below* the `main` function in the program. It is not needed for a function whose definition appears *above* the `main` function. In this book, the function definitions will be entered below the `main` function because that is the format used by most C++ programmers. This means that each program-defined function will need a corresponding function prototype above the `main` function.

Figure 9-17 shows a function prototype's syntax, which is almost identical to a function header's syntax. However, unlike a function header, a function prototype ends with a semicolon. The figure also shows two ways of writing the function prototype for the `getBalance` function in the savings account program. As Example 2 indicates, it is not necessary to include the names of the formal parameters in a function prototype. However, many programmers include the names to make the program easier to read and understand. Some also include the names for convenience because it makes entering the function prototype an easy task: Simply copy the function's header, then paste it in the function prototype section of the program, and then type a semicolon at the end of it.

HOW TO Write a Function Prototype

Syntax

returnDataType functionName(*[parameterList]*);

semicolon

each formal parameter's data type and (optionally) name

Example 1 – with the optional names

```
double getBalance(int amount, double rate, int y);
```

Example 2 – without the optional names

```
double getBalance(int, double, int);
```

Figure 9-17 How to write a function prototype

Completing the Savings Account Program

Figure 9-18 shows all of the code entered in the savings account program. The function prototype on Line 11 alerts the computer that the `getBalance` function is defined somewhere below the `main` function in the program. In this case, the function definition appears on Lines 37 through 42. Some programmers use a comment to separate the function definitions from

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

the `main` function, as shown on Line 36; however, that is not a requirement. Figure 9-18 also includes a sample run of the program.

```
 1 //Savings.cpp - displays the account balance at
 2 //the end of 1 through 3 years
 3 //Created/revised by <your name> on <current date>
 4
 5 #include <iostream>
 6 #include <iomanip>
 7 #include <cmath>
 8 using namespace std;
 9
10 //function prototype
11 double getBalance(int amount, double rate, int y);
12
13 int main()
14 {
15    int deposit = 0;
16    double interestRate = 0.0;
17    double acctBalance = 0.0;
18
19    cout << "Deposit: ";
20    cin >> deposit;
21    cout << "Rate (in decimal form): ";
22    cin >> interestRate;
23
24    cout << fixed << setprecision(2);
25    for (int year = 1; year < 4; year += 1)
26    {
27        acctBalance =
28            getBalance(deposit, interestRate, year);
29        cout << "Year " << year << ": $"
30            << acctBalance << endl;
31    }  //end for
32
33    return 0;
34 }  //end of main function
35
36 //*****function definitions*****
37 double getBalance(int amount, double rate, int y)
38 {
39    double balance = 0.0;
40    balance = amount * pow((1 + rate), y);
41    return balance;
42 }  //end of getBalance function
```

function prototype

function call

function definition

```
Savings Account
Deposit: 1000
Rate (in decimal form): 0.02
Year 1: $1020.00
Year 2: $1040.40
Year 3: $1061.21
Press any key to continue . . .
```

Figure 9-18 Savings account program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Ch09-Savings Desk-Check

Desk-checking the savings account program will help you understand how the computer processes a program-defined function. The declaration statements on Lines 15 through 17 create and initialize three variables. The `cout` and `cin` statements on Lines 19 through 22 prompt the user to enter the initial deposit and annual interest rate and then store the user's responses in two of the `main` function's variables. The statement on Line 24 tells the computer to display the output in fixed-point notation with two decimal places. The `for` clause on Line 25 creates a counter variable named `year` and initializes it to 1. Figure 9-19 shows the desk-check table at this point, assuming the user enters 1000 and 0.02 as the initial deposit and annual interest rate, respectively.

deposit	interestRate	acctBalance	year
~~0~~	~~0.0~~	0.0	1
1000	0.02		

these four variables belong to the `main` function

Figure 9-19 Desk-check table after the instructions on Lines 15 through 25 are processed

The `for` clause checks whether the value in the `year` variable is less than 4. It is, so the statements in the loop body are processed. The first statement is an assignment statement that invokes the `getBalance` function, passing it a copy of the values stored in three variables. At this point, the computer temporarily leaves the `main` function to process the `getBalance` function's code, beginning with the function header on Line 37 in the program.

The `getBalance` function's header contains three formal parameters. The parameters tell the computer to reserve three memory locations: an `int` variable named `amount`, a `double` variable named `rate`, and an `int` variable named `y`. After reserving the variables, the computer stores a copy of the values passed to the function—in this case, the numbers 1000, 0.02, and 1—in the variables. Figure 9-20 shows the desk-check table after the function header is processed.

Note: The names in black indicate variables that belong to the `main` function. The names in red indicate variables that belong to the `getBalance` function.

deposit	interestRate	acctBalance	year
~~0~~	~~0.0~~	0.0	1
1000	0.02		

amount	rate	y
1000	0.02	1

Figure 9-20 Desk-check table after the function header on Line 37 is processed

The first statement in the `getBalance` function creates and initializes a `double` variable named `balance`. The second statement calculates the current balance using the values stored in the `amount` (1000), `rate` (0.02), and `y` (1) variables. The statement assigns the current balance (1020.0) to the `balance` variable, as shown in Figure 9-21.

Note: The names in black indicate variables that belong to the `main` function. The names in red indicate variables that belong to the `getBalance` function.

deposit	interestRate	acctBalance	year
~~0~~	~~0.0~~	0.0	1
1000	0.02		

amount	rate	y	balance
1000	0.02	1	~~0.0~~
			1020.0

Figure 9-21 Desk-check table after the first two statements in the `getBalance` function are processed

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The `getBalance` function's `return` statement is processed next and returns the value stored in the `balance` variable (1020.0) to the statement that called the function. That statement, which appears on Lines 27 and 28 in the `main` function, assigns the value to the `main` function's `acctBalance` variable.

After the `getBalance` function's `return` statement is processed, the function ends, and the computer removes the `amount`, `rate`, `y`, and `balance` variables from its internal memory. Figure 9-22 shows the desk-check table at this point in the program. Notice that only the `main` function's variables are still in the computer's internal memory. (When desk-checking, you do not need to cross out the columns that contain the `getBalance` function's variables when the function ends. You can simply cross out the ending values and then reuse the columns for the next call to the function, as shown later in Figure 9-33. The columns are crossed out in this section to make it obvious that the variables no longer exist when the `getBalance` function ends.)

Note: The names in black indicate variables that belong to the `main` function. The names in red indicate variables that belong to the `getBalance` function.

deposit	interestRate	acctBalance	year
~~0~~	~~0.0~~	~~0.0~~	1
1000	0.02	1020.0	

~~amount~~	rate	y	~~balance~~
1000	0.02	1	~~0.0~~
			~~1020.0~~

Figure 9-22 Desk-check table after the `getBalance` function's `return` statement is processed

After the assignment statement in the `for` loop is processed, the loop's `cout` statement displays the message "Year 1: $1020.00" on the computer screen. Next, the `for` clause updates the `year` variable by 1, giving 2. The clause then checks whether the variable's value is less than 4. It is, so the statements in the loop body are processed again.

The first statement in the loop body invokes the `getBalance` function, passing it a copy of the values stored in its actual arguments. At this point, the computer temporarily leaves the `main` function to process the `getBalance` function's code, beginning with the function header. The function's formal parameters tell the computer to create an `int` variable named `amount`, a `double` variable named `rate`, and an `int` variable named `y`. After reserving the variables, the computer stores a copy of the values passed to the function—in this case, the numbers 1000, 0.02, and 2—in the variables.

The first statement in the `getBalance` function creates and initializes a `double` variable named `balance`. The second statement calculates the current balance using the values stored in the `amount` (1000), `rate` (0.02), and `y` (2) variables. The statement assigns the current balance (1040.4) to the `balance` variable.

The `getBalance` function's `return` statement is processed next and returns the value stored in the `balance` variable (1040.4) to the statement that called the function. That statement, which appears on Lines 27 and 28 in the `main` function, assigns the value to the `main` function's `acctBalance` variable, as shown in Figure 9-23.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

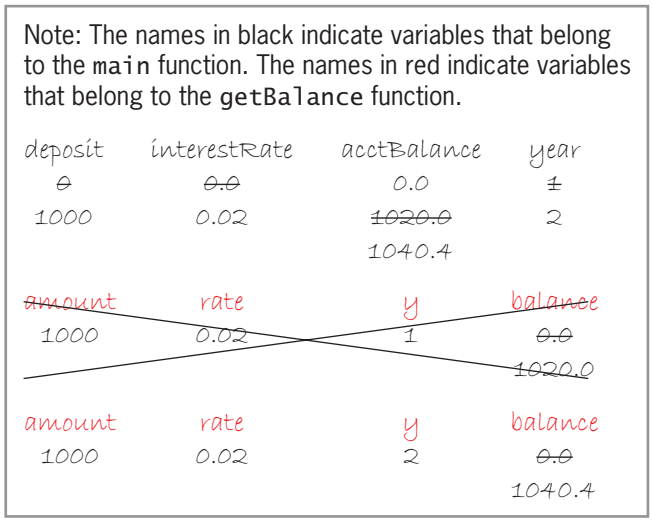


Figure 9-23 Desk-check table *after* the `return balance;` statement is processed the second time

After the `getBalance` function's `return` statement is processed, the function ends, and the computer removes the `amount`, `rate`, `y`, and `balance` variables from its internal memory. As Figure 9-24 indicates, only the `main` function's variables are still in the computer's internal memory.

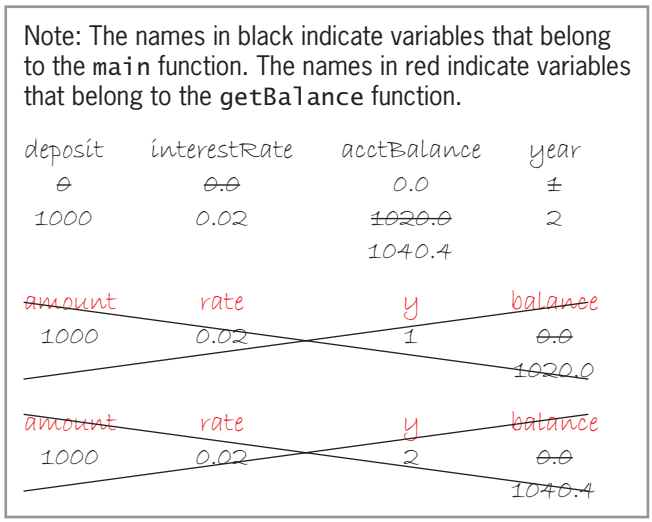


Figure 9-24 Desk-check table *after* the `getBalance` function ends the second time

The loop's `cout` statement is processed next and displays the message "Year 2: $1040.40" on the computer screen. Then, the `for` clause updates the `year` variable by 1, giving 3, and checks whether the variable's value is less than 4. It is, so the statements in the loop body are processed again.

The first statement in the loop body invokes the `getBalance` function, passing it a copy of the values stored in its actual arguments. Here again, the computer temporarily leaves the `main` function to process the `getBalance` function's code. Using the formal parameters as a guide, the computer creates three variables and stores a copy of the values passed to the function (1000, 0.02, and 3) in the variables.

The first statement in the `getBalance` function creates and initializes a `double` variable named `balance`. The second statement calculates the current balance using the values stored in the `amount` (1000), `rate` (0.02), and `y` (3) variables. The statement assigns the current balance (1061.208) to the `balance` variable.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The `getBalance` function's `return` statement is processed next and returns the value stored in the `balance` variable (1061.208) to the assignment statement on Lines 27 and 28 in the `main` function. The statement assigns the value to the `main` function's `acctBalance` variable.

After the `getBalance` function's `return` statement is processed, the function ends and the computer removes the `amount`, `rate`, `y`, and `balance` variables from its internal memory. At this point, only the `main` function's variables are still in the computer's internal memory, as shown in Figure 9-25. (As mentioned earlier, you do not need to cross out the columns that contain the called function's variables each time the function ends. You can simply cross out the ending values and then reuse the columns for the next call to the function, as shown later in Figure 9-33. The columns are crossed out in this section to make it obvious that the variables no longer exist when the `getBalance` function ends.)

Note: The names in black indicate variables that belong to the `main` function. The names in red indicate variables that belong to the `getBalance` function.

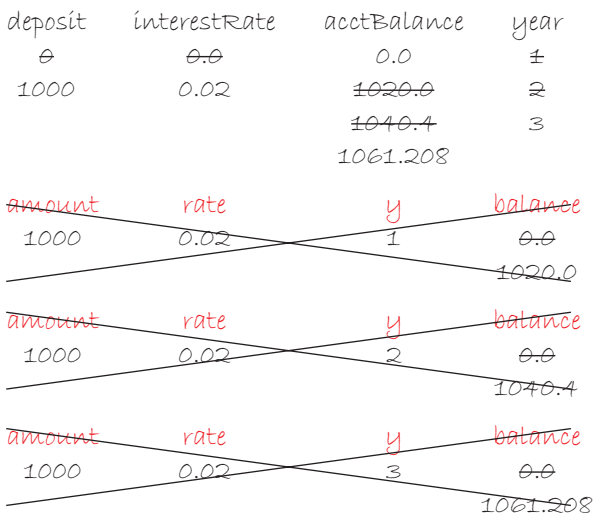


Figure 9-25 Desk-check table *after* the `getBalance` function ends the third time

The loop's `cout` statement is processed next and displays the message "Year 3: $1061.21" on the computer screen. Then, the `for` clause updates the `year` variable by 1, giving 4, and checks whether the variable's value is less than 4. It isn't, so the `for` loop ends, and the computer removes the `year` variable from its internal memory.

The `return 0;` statement on Line 33 is processed next. After the statement is processed, the `main` function ends, and the computer removes the `deposit`, `interestRate`, and `acctBalance` variables from its internal memory.

At this point, you may be wondering why the `main` function needs to pass a copy of the values stored in the `deposit`, `interestRate`, and `year` variables to the `getBalance` function. Why can't the `getBalance` function just use the `main` function's variables? You may also be wondering why the computer removes the variables at different times while the program is running. To answer these questions, you will need to learn about the scope and lifetime of a variable. The scope and lifetime of a variable are the last topics covered in this chapter.

For more examples of value-returning functions, see the Value-Returning Functions section in the Ch09WantMore.pdf file.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The Scope and Lifetime of a Variable

A variable's **scope** indicates where in the program the variable can be used, and its **`lifetime`** indicates how long the variable remains in the computer's internal memory. Although variables can have either local or global scope, most of the variables used in a program will have local scope. This is because fewer unintentional errors occur in programs when the variables are declared using the minimum scope needed, which usually is local scope.

A variable's scope and lifetime are determined by where you declare the variable in the program. Variables declared within a statement block, a function's *parameterList*, or a `for` clause have local scope and are referred to as **local variables**. Recall that a statement block is a set of instructions enclosed in braces. A program-defined function is an example of a statement block. Variables declared within the function, and those that appear in its *parameterList*, can be used only by that function. You also observed the use of a local variable in the swapping program in Figure 5-8 in Chapter 5. As you may remember, the statement block in the `if` statement's true path declares a local variable named `temp`. The `temp` variable is local to the true path and can be used only by the statements in that path. A variable declared in a `for` clause also has a local scope; it can be used only by the `for` loop and remains in memory until the loop ends.

Unlike local variables, **global variables** are declared outside of any function in the program, and they remain in memory until the program ends. Also unlike a local variable, any statement in the program can use a global variable. Declaring a variable as global rather than local allows unintentional errors to occur when a function that should not have access to the variable inadvertently changes the variable's contents. Because of this, you should avoid using global variables in your programs. If more than one function needs access to the same variable, it is better to create a local variable in one of the functions and then pass that variable to the other functions that need it.

In the savings account program shown earlier in Figure 9-18, the `deposit`, `interestRate`, and `acctBalance` variables are declared on Lines 15 through 17 in the `main` function. As a result, the variables are local to the `main` function and can be used only by statements below Line 17 within the `main` function. In other words, their scope begins with Line 18 and ends with Line 34. The `getBalance` function, which begins on Line 37, is not even aware of the existence of these variables in memory. If you want the `getBalance` function to use the values stored in those variables, you will need to pass the values to the function. The variables are removed from memory when the `main` function ends.

The `year` variable, which is declared in the `for` clause on Line 25, is local to the `for` loop. This means its scope is limited to the loop—in this case, the statements from Line 25 through Line 31. The `year` variable's lifetime lasts only as long as the loop is processing. When the loop ends, the `year` variable is removed from the computer's internal memory.

The `amount`, `rate`, `y`, and `balance` variables are local to the `getBalance` function—the first three because they appear in the function's *parameterList* and the last because it is declared within the function. Only the `getBalance` function can use the four variables, and they will be removed from the computer's internal memory when the `getBalance` function ends.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Mini-Quiz 9-3

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

1. The `getArea` function returns a `double` number and has no formal parameters. Which of the following calls the `getArea` function and assigns its return value to a `double` variable named `area`?

   a. `area = getArea`
   b. `area = getArea();`
   c. `area = getArea(double);`
   d. `getArea(area);`

2. Which of the following is a valid function prototype for the `getArea` function from Question 1?

   a. `double getArea()`
   b. `double getArea`
   c. `double getArea();`
   d. `double getArea;`

3. Write a C++ statement that will display the value returned by the `getArea` function from Question 1.
4. Write a function prototype for the `getGrossPay` function. The function returns a `double` number and has two formal parameters: an `int` variable named `hours` and a `double` variable named `rate`.
5. Write a statement that invokes the `getGrossPay` function from Question 4. The statement should pass the function the integer 40 and a copy of the value stored in a `double` variable named `payRate`. The statement should assign the function's return value to a `double` variable named `weekGross`.

LAB 9-1 Stop and Analyze

Study the program shown in Figure 9-26, and then answer the questions.

The answers to the labs are contained in the Answers.pdf file.

```
 1 //Lab9-1.cpp - circle calculations
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <cmath>
 6 using namespace std;
 7
 8 //function prototypes
 9 double getArea(double rad);
10 double getDiameter(double rad);
11
```

Figure 9-26 Code for Lab 9-1 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
12 int main()
13 {
14    int choice = 0;
15    double radius = 0.0;
16
17    cout << "1   Circle area" << endl;
18    cout << "2   Circle diameter" << endl;
19    cout << "Enter your choice (1 or 2): ";
20    cin >> choice;
21
22    if (choice < 1 || choice > 2)
23        cout << "Invalid choice" << endl;
24    else
25    {
26        cout << "Radius: ";
27        cin >> radius;
28        if (choice == 1)
29            cout << "Area: " << getArea(radius);
30        else
31            cout << "Diameter: " << getDiameter(radius);
32        //end if
33        cout << endl;
34    }  //end if
35    return 0;
36 }  //end of main function
37
38 //*****function definitions*****
39 double getArea(double rad)
40 {
41    const double PI = 3.141593;
42    double area = 0.0;
43    area = PI * pow(rad, 2);
44    return area;
45 }  //end getArea function
46
47 double getDiameter(double rad)
48 {
49    return 2 * rad;
50 }  //end getDiameter function
```

Figure 9-26 Code for Lab 9-1

QUESTIONS

1. Why are the statements on Lines 9 and 10 necessary?
2. How else could you write the statement on Line 9?
3. If the program does not include the outer selection structure, what will the program display if the user enters the numbers 3 and 10 as the choice and radius, respectively?
4. What are the scope and lifetime of the `choice` and `radius` variables?
5. What are the scope and lifetime of the `rad` variable used in the `getArea` function?
6. What are the scope and lifetime of the `PI` constant and the `area` variable?
7. What are the scope and lifetime of the `rad` variable used in the `getDiameter` function?

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

8. Follow the instructions for starting C++ and viewing the Lab9-1.cpp file, which is contained in either the Cpp8\Chap09\Lab9-1 Project folder or the Cpp8\Chap09 folder. (Depending on your C++ development tool, you may need to open Lab9-1's project/solution file first.) Use the program to display the area of a circle with a radius of 25.5. Then use it to display the diameter of a circle with a radius of 10.
9. Modify the program to allow the user to display a circle's circumference, given its radius. Save and then run the program. Test the program appropriately.

LAB 9-2 Plan and Create

In this lab, you will plan and create an algorithm for the problem specification shown in Figure 9-27. (Hint: You can use a calculator or a spreadsheet program such as Microsoft Excel to verify the payments shown in the figure and also to perform your own calculations using the periodic payment formula.)

Problem specification
Many car dealers offer customers a choice of either a large cash rebate or an extremely low financing rate that is much lower than a local credit union charges. Create a program that calculates the monthly car payment for each of the following two scenarios:

- Scenario 1: The user accepts the dealer's rebate offer and finances the car through his or her local credit union.
- Scenario 2: The user declines the dealer's rebate offer but accepts the dealer's lower financing rate.

The formula for calculating a periodic payment on a loan is shown below. In the formula, *principal* is the amount of the loan, *rate* is the periodic interest rate, and *term* is the number of periodic payments. Also shown below are two examples that use the formula to calculate a periodic payment. Example 1 calculates the annual payment for a $9,000 loan for three years at 5% interest. The annual payment rounded to the nearest cent is $3,304.88. In other words, if you borrow $9,000 for three years at 5% interest, you would need to make three annual payments of $3,304.88 to pay off the loan. Example 2 calculates the monthly payment for a $12,000 loan for five years at 6% interest. To pay off this loan, you would need to make 60 payments of $231.99.

When you apply for a loan, the lender typically quotes you an annual interest rate and expresses the term in years. Therefore, when calculating a monthly payment, you must convert the annual interest rate to a monthly rate by dividing the annual rate by 12. You also need to convert the term from years to months by multiplying the number of years by 12.

Periodic payment formula
$principal * rate / (1 - (rate + 1)^{-term})$

Example 1	annual payment for a $9,000 loan for 3 years at 5% interest
Principal:	9,000
Annual rate:	0.05
Term (years):	3
Formula:	$9{,}000 * 0.05 / (1 - (0.05 + 1)^{-3})$
Annual payment:	$3,304.88 (rounded to the nearest cent)

Figure 9-27 Problem specification and sample calculations for Lab 9-2 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Example 2	monthly payment for a $12,000 loan for 5 years at 6% interest
Principal:	12,000
Monthly rate:	0.005 (annual rate of 0.06 divided by 12
Term (months):	60 (5 years multiplied by 12)
Formula:	$12{,}000 * 0.005 / (1 - (0.005 + 1)^{-60})$
Monthly payment:	$231.99 (rounded to the nearest cent)

Figure 9-27 Problem specification and sample calculations for Lab 9-2

First, analyze the problem, looking for the output first and then for the input. In this case, the user wants the program to display two monthly payments: one if the car is financed through the credit union and the other if the car is financed through the dealer. To calculate the monthly payments, the computer will need to know the following information: the price of the car (after any trade-in), the rebate amount, the credit union's annual interest rate, the dealer's annual interest rate, and the term (in years). After analyzing the problem, you plan the algorithm. Recall that most algorithms follow the format of entering the input items, processing the input items, and displaying, printing, or storing the output items.

Figures 9-28 and 9-29 show the completed IPO charts for the program's `main` and `getPayment` functions, respectively. Notice that the `main` function calls the `getPayment` function twice: once to determine the credit union payment and again to determine the car dealer payment. When calling the `getPayment` function to calculate the credit union payment, the `main` function will pass the difference between the car price and the rebate as the principal. It will also pass the monthly credit union rate (which is the annual credit union rate divided by 12) and the number of months (which is the term times 12). Similarly, when calling the `getPayment` function to calculate the dealer payment, the `main` function will pass the car price as the principal and also pass the monthly dealer rate and the number of months.

`main` function

Input

car price
rebate
credit union rate (annual)
dealer rate (annual)
term (years)

Processing

Processing items: none

Algorithm:
1. enter the car price, rebate, credit union rate, dealer rate, and term
2. call the getPayment function to calculate the credit union payment; pass the car price minus the rebate, the credit union rate / 12, and the term * 12
3. call the getPayment function to calculate the dealer payment; pass the car price, the dealer rate / 12, and the term * 12

Output

credit union payment
dealer payment

Figure 9-28 IPO chart for the `main` function *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

4. display the credit union payment and the dealer payment

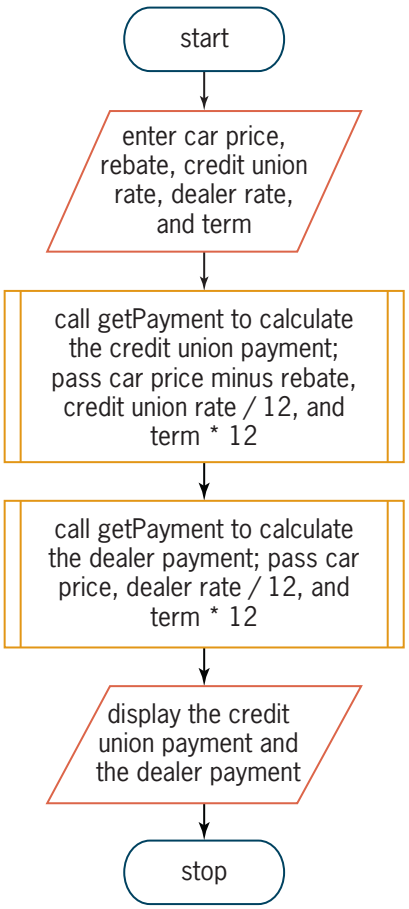


Figure 9-28 IPO chart for the `main` function

`getPayment` function

Input	**Processing**	**Output**
principal monthly rate number of months	Processing items: none	monthly payment

Algorithm:
1. calculate the monthly payment using the periodic payment formula
2. return the monthly payment

Figure 9-29 IPO chart for the `getPayment` function *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

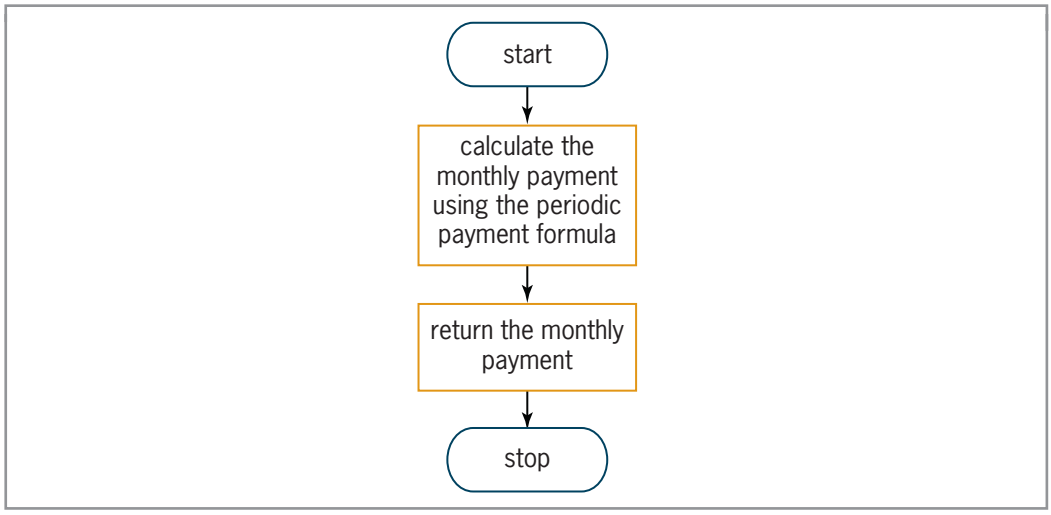


Figure 9-29 IPO chart for the `getPayment` function

The third step in the problem-solving process is to desk-check the algorithm. You will desk-check the algorithms for the `main` and `getPayment` functions using 16000, 3000, 0.08, 0.03, and 4 as the car price (after any trade-in), rebate, annual credit union rate, annual dealer rate, and term (in years), respectively. Using these values, the monthly payments should be $317.37 (credit union) and $354.15 (dealer). Therefore, it will be cheaper to finance the car through the credit union. Figure 9-30 shows the completed desk-check table.

Note: The names in black indicate items that belong to the `main` function. The names in red indicate items that belong to the `getPayment` function.

car price	rebate	credit union rate	dealer rate	term
16000	3000	0.08	0.03	4

credit union payment	dealer payment
317.37	354.15

principal	monthly rate	number of months	monthly payment
~~13000~~	~~.0067~~	~~48~~	~~317.37~~
16000	.0025	48	354.15

Figure 9-30 Completed desk-check table for the car payment algorithms

The fourth step in the problem-solving process is to code the algorithm into a program. The IPO chart information and C++ instructions for the `main` and `getPayment` functions are shown in Figures 9-31 and 9-32, respectively. The variables declared in Figure 9-31 are local to the `main` function and remain in memory until the `main` function ends. The variables in Figure 9-32 are local to the `getPayment` function and remain in memory until the `getPayment` function ends.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

main function **IPO chart information**	**main function** **C++ instructions**
Input	

car price
rebate
credit union rate (annual)
dealer rate (annual)
term (years)
	`int carPrice = 0;`
	`int rebate = 0;`
	`double creditRate = 0.0;`
	`double dealerRate = 0.0;`
	`int term = 0;`

Processing	
none	
Output	
credit union payment dealer payment	`double creditPayment = 0.0;` `double dealerPayment = 0.0;`
Algorithm	

1. enter the car price, rebate, credit union rate, dealer rate, and term	`cout << "Car price (after any trade-in): ";`
	`cin >> carPrice;`
	`cout << "Rebate: ";`
	`cin >> rebate;`
	`cout << "Credit union rate: ";`
	`cin >> creditRate;`
	`cout << "Dealer rate: ";`
	`cin >> dealerRate;`
	`cout << "Term in years: ";`
	`cin >> term;`

2. call the getPayment function to calculate the credit union payment; pass the car price minus the rebate, the credit union rate / 12, and the term * 12	`creditPayment = getPayment (carPrice - rebate, creditRate / 12, term * 12);`
3. call the getPayment function to calculate the dealer payment; pass the car price, the dealer rate / 12, and the term * 12	`dealerPayment = getPayment(carPrice, dealerRate / 12, term * 12);`
4. display the credit union payment and the dealer payment	`cout << "Credit union payment: $" << creditPayment << endl;` `cout << "Dealer payment: $" << dealerPayment << endl;`

Figure 9-31 IPO chart information and C++ instructions for the `main` function

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

getPayment function **IPO chart information**	**getPayment function** **C++ instructions**

Input
principal (formal parameter)
monthly rate (formal parameter)
number of months (formal parameter)
	`int prin`
	`double monthRate`
	`int months`

Processing none	
Output monthly payment	`double monthPay = 0.0;`

Algorithm
1. calculate the monthly payment using the periodic payment formula
2. return the monthly payment
	`monthPay = prin * monthRate /`
	`(1 - pow(monthRate + 1, -months));`
	`return monthPay;`

Figure 9-32 IPO chart information and C++ instructions for the `getPayment` function

The fifth step in the problem-solving process is to desk-check the program. Figure 9-33 shows the completed desk-check table for the car payment program. The results agree with those shown in the algorithm's desk-check table in Figure 9-30. (Rather than crossing out the called function's variables each time the function ends, as shown earlier in Figures 9-22 through 9-25, you can simply cross out the last values in those columns and then reuse them for the next call to the function. You would then cross out the variables only after the last call to the function, as shown in Figure 9-33.)

Note: The names in black indicate variables that belong to the `main` function. The names in red indicate variables that belong to the `getPayment` function.

carPrice	rebate	creditRate	dealerRate	term
~~0~~	~~0~~	~~0.0~~	~~0.0~~	~~0~~
16000	3000	0.08	0.03	4

creditPayment	dealerPayment
~~0.0~~	~~0.0~~
~~317.37~~	354.15

~~prin~~	monthRate	months	~~monthPay~~
~~13000~~	~~.0067~~	~~48~~	~~0.0~~
16000	.0025	48	~~317.37~~
			~~0.0~~
			~~354.15~~

Figure 9-33 Completed desk-check table for the car payment program

The final step in the problem-solving process is to evaluate and modify (if necessary) the program. Recall that you evaluate a program by entering its instructions into the computer and then using the computer to run (execute) it. While the program is running, you enter the same sample data used when desk-checking the program.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

DIRECTIONS

Follow the instructions for starting your C++ development tool. Depending on the development tool you are using, you may need to create a new project; if so, name the project Lab9-2 Project, and save it in the Cpp8\Chap09 folder. Enter the instructions shown in Figure 9-34 in a source file named Lab9-2.cpp. (Do not enter the line numbers.) Save the file in either the project folder or the Cpp8\Chap09 folder. Now follow the appropriate instructions for running the Lab9-2.cpp file. Test the program using the same data you used to desk-check the program. If necessary, correct any bugs (errors) in the program.

```
 1 //Lab9-2.cpp - displays two monthly car payments
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <cmath>
 6 #include <iomanip>
 7 using namespace std;
 8
 9 //function prototype
10 double getPayment(int, double, int);
11
12 int main()
13 {
14    int carPrice = 0;
15    int rebate = 0;
16    double creditRate = 0.0;
17    double dealerRate = 0.0;
18    int term = 0;
19    double creditPayment = 0.0;
20    double dealerPayment = 0.0;
21
22    cout << "Car price (after any trade-in): ";
23    cin >> carPrice;
24    cout << "Rebate: ";
25    cin >> rebate;
26    cout << "Credit union rate: ";
27    cin >> creditRate;
28    cout << "Dealer rate: ";
29    cin >> dealerRate;
30    cout << "Term in years: ";
31    cin >> term;
32
33    //call function to calculate payments
34    creditPayment = getPayment(carPrice - rebate,
35        creditRate / 12, term * 12);
36    dealerPayment = getPayment(carPrice,
37        dealerRate / 12, term * 12);
38
39    //display payments
40    cout << fixed << setprecision(2) << endl;
41    cout << "Credit union payment: $"
42        << creditPayment << endl;
43    cout << "Dealer payment: $"
44        << dealerPayment << endl;
45    return 0;
```

the names of the formal parameters are not required

Figure 9-34 Car payment program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
46 }//end of main function
47
48 //*****function definitions*****
49 double getPayment(int prin,
50                   double monthRate,
51                   int months)
52 {
53    //calculates and returns a monthly payment
54    double monthPay = 0.0;
55    monthPay = prin * monthRate /
56        (1 - pow(monthRate + 1, -months));
57    return monthPay;
58 } //end of getPayment function
```

Figure 9-34 Car payment program

LAB 9-3 Modify

If necessary, create a new project named Lab9-3 Project, and save it in the Cpp8\Chap09 folder. Enter (or copy) the Lab9-2.cpp instructions into a new source file named Lab9-3.cpp. Change Lab9-2.cpp in the first comment to Lab9-3.cpp. Make the three modifications listed in Figure 9-35. Save, run, and test the program.

1. Compare both monthly payments and then display one of the following three messages:
   a. Take the rebate and finance through the credit union.
   b. Don't take the rebate. Finance through the dealer.
   c. You can finance through the dealer or the credit union.
2. The user should be able to calculate the monthly payments as many times as needed without having to run the program again.
3. Allow the user to enter an interest rate as either a whole number or a decimal number. For example, if the interest rate is 5%, the user should be able to enter the rate as either 5 or .05.

Figure 9-35 Modifications for Lab 9-3

LAB 9-4 What's Missing?

The program in this lab should display the total due, given the quantity purchased and the item price. Start your C++ development tool, and view the Lab9-4.cpp file, which is contained in either the Cpp8\Chap09\Lab9-4 Project folder or the Cpp8\Chap09 folder. (Depending on your C++ development tool, you may need to open Lab9-4's project/solution file first.) Put the C++ instructions in the proper order, and then determine the one or more missing instructions. Test the program appropriately.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

LAB 9-5 Desk-Check

Use the data shown in Figure 9-36 to desk-check the figure's code. What current total will the code display on the screen?

Test data: 100 (the beginning number), 4, –3, 10, 9, and –5

```
 1 //Lab9-5.cpp - displays the current total
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <cmath>
 6 using namespace std;
 7
 8 //function prototype
 9 double getCurrentTotal(int current, int num);
10
11 int main()
12 {
13    int currentTotal = 0;
14    int number = 0;
15
16    cout << "Beginning number: ";
17    cin >> currentTotal;
18
19    for (int x = 1; x < 6; x += 1)
20    {
21        cout << "Enter a positive or negative number: ";
22        cin >> number;
23        currentTotal = getCurrentTotal(currentTotal, number);
24    }  //end for
25
26    cout << endl << "Current total: " << currentTotal << endl;
27    return 0;
28 }  //end of main function
29
30 //*****function definitions*****
31 double getCurrentTotal(int current, int num)
32 {
33    if (num >= 0)
34        current += pow(num, 2);
35    else
36        current += num * 2;
37    //end if
38    return current;
39 }  //end of getCurrentTotal function
```

Figure 9-36 Test data and code for Lab 9-5

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

LAB 9-6 Debug

Follow the instructions for starting C++ and viewing the Lab9-6.cpp file, which is contained in either the Cpp8\Chap09\Lab9-6 Project folder or the Cpp8\Chap09 folder. (Depending on your C++ development tool, you may need to open Lab9-6's project/solution file first.) Test the program using 20500, 3500, and 10 as the asset cost, salvage value, and useful life, respectively. The depreciation should be $1700.00. Debug the program.

Chapter Summary

Functions allow a programmer to avoid duplicating code in different parts of a program. They also allow large and complex programs to be broken into small and manageable tasks.

Some of the functions used in a program are built-in functions. Others, like `main`, are program-defined functions.

All functions are classified as either value-returning functions or void functions. A value-returning function returns precisely one value after completing its assigned task. The value is returned to the statement that called the function. Void functions, which you will learn about in Chapter 10, do not return a value.

The C++ `pow` function raises a number to a power and then returns the result as a `double` number. To use the `pow` function, a program must contain the `#include <cmath>` directive.

The C++ `sqrt` function finds the square root of a number and then returns the result as a `double` number. To use the `sqrt` function, a program must contain the `#include <cmath>` directive.

The items within parentheses in a function call are referred to as actual arguments.

The C++ language provides the `rand` function for generating random numbers. The `rand` function is a value-returning function. It returns an integer that is greater than or equal to 0 but less than or equal to `RAND_MAX`, whose value is always at least 32,767.

You can use the expression *lowerBound* + `rand() %` (*upperBound* – *lowerBound* + 1) to produce random integers within a specific range.

You can initialize the `rand` function using the C++ built-in void `srand` function. Most programmers use the built-in value-returning `time` function as the `srand` function's *seed* argument. To use the `time` function, a program must contain the `#include <ctime>` directive.

A function definition is composed of a function header and a function body.

The function header for a value-returning function specifies the type of data the function returns, the function's name, and an optional *parameterList* enclosed in parentheses.

The items listed in a function's *parameterList* are called formal parameters.

The *parameterList* in a function header contains the data type and name of each formal parameter. The quantity, data type, and position (order) of the formal parameters in the *parameterList* should agree with the quantity, data type, and position (order) of the actual arguments passed

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

to the function. In most cases, the name of each formal parameter is different from the name of its corresponding actual argument. Functions that do not require a *parameterList* will have an empty set of parentheses after the function's name.

The function body in a function definition contains the instructions that the function must follow to perform its assigned task. The function body begins with an opening brace and ends with a closing brace. Typically, the `return` statement, which instructs the function to return a value, is the last statement in the function body of a value-returning function.

You call a function by including its name and actual arguments (if any) in a statement.

Unless specified otherwise, variables in C++ are passed to a function *by value*, which means that only a copy of the value stored in the variable is passed.

A program will have one function prototype for each function defined below the `main` function. Functions defined above the `main` function in a program do not need a function prototype.

A variable's scope, which can be either local or global, indicates where in a program a variable can be used. A variable's lifetime indicates how long the variable remains in the computer's internal memory.

Local variables can be used only within the statement block, *parameterList*, or `for` clause in which they are declared, and they remain in memory until the end of the statement block, function, or `for` loop, respectively. Global variables, which you should avoid using, can be used anywhere in the program. Unlike local variables, global variables remain in memory until the program ends.

Key Terms

Actual argument—an item of information passed (sent) to a function when the function is called (invoked)

Built-in functions—blocks of code that perform a task and are included in libraries that come with the C++ language; examples include the `pow`, `sqrt`, `rand`, `srand`, and `time` functions

Exponentiation—the process of raising a number to a power

Formal parameters—the memory locations listed in a function header's *parameterList*; a formal parameter stores an item of information passed to a function when the function is invoked (called)

Function prototype—a statement that specifies the function's name, the data type of its return value (if any), and the data type and (optionally) name of each of its formal parameters (if any); required for every function that is defined below the `main` function in a program

Global variables—variables that are declared outside of any function in a program; global variables can be used by any statement below their variable declaration in the program, and they remain in memory until the program ends; you should avoid using global variables in a program

Lifetime—indicates how long an item, such as a variable, remains in the computer's internal memory

Local variables—variables that are declared within a statement block, a function's *parameterList*, or a `for` clause; local variables can be used only by the statement block, function, or `for` loop in which they are declared; local variables remain in memory until the end of the statement block, function, or `for` loop

Passing by reference—refers to the process of passing a variable's address to a function

Passing by value—refers to the process of passing a copy of a variable's value to a function

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

pow function—a built-in C++ function that raises a number to a power and then returns the result as a `double` number

Program-defined functions—functions created by the programmer and whose definitions typically reside in the current program

Pseudo-random number generator—a function that produces a sequence of numbers that meet certain statistical requirements for randomness; the `rand` function is the pseudo-random number generator in C++

rand function—a built-in C++ function that returns a random integer that is greater than or equal to 0 but less than or equal to the value stored in the `RAND_MAX` constant; the pseudo-random number generator in C++

RAND_MAX—a C++ built-in constant that represents the largest integer generated by the `rand` function; its value is always at least 32,767

return statement—in most cases, the last statement in a value-returning function; it returns a value to the statement that called the function and then alerts the computer that the function has completed its task

Scope—indicates where in the program an item, such as a variable, can be used

sqrt function—a C++ built-in function whose purpose is to return the square root of a real number; returns the square root as a `double` number

srand function—a C++ built-in function used to initialize the `rand` function

time function—a C++ built-in function that returns the current time (according to your computer system's clock) as seconds elapsed since midnight on January 1, 1970; often used as the *seed* argument in the `srand` function

Value-returning functions—functions that return precisely one value after they complete their assigned task

Review Questions

1. Value-returning functions can return _____ .
   a. one value only
   b. one or more values
   c. the number 0 only
   d. none of the above

2. The function header specifies _____ .
   a. the data type of the function's return value (if any)
   b. the name of the function
   c. the function's formal parameters (if any)
   d. all of the above

3. Which of the following is false?
   a. The number of actual arguments should agree with the number of formal parameters.
   b. The data type of each actual argument should match the data type of its corresponding formal parameter.
   c. The name of each actual argument should be identical to the name of its corresponding formal parameter.
   d. When you pass information to a function *by value*, the function stores the value of each item it receives in a separate memory location.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

4. Each memory location listed in a function header's *parameterList* is referred to as _____ .

   a. an actual argument
   b. an actual parameter
   c. a formal argument
   d. a formal parameter

5. A program contains the statement `tax = calcTax(sales);`. The `tax` and `sales` variables have the `double` data type. Which of the following is a valid function header for the `calcTax` function?

   a. `calcTax(double sales);`
   b. `double calcTax(salesAmount)`
   c. `double calcTax(double salesAmount)`
   d. `double calcTax(int sales);`

6. Which of the following is a valid function header for the `getFee` function, which receives an integer first and a number with a decimal place second? The function returns a number with a decimal place.

   a. `getFee(int base, double rate);`
   b. `double getFee(int base, double rate);`
   c. `double getFee(double base, int rate)`
   d. `double getFee(int base, double rate)`

7. Which of the following is a valid function prototype for the function described in Review Question 6?

   a. `getFee(int base, double rate);`
   b. `int getFee(int, double)`
   c. `double getFee(int base, double rate)`
   d. `double getFee(int, double);`

8. Which of the following directs a function to return the contents of the `stateTax` variable to the statement that invoked it, which is contained in the `main` function?

   a. `restore stateTax;`
   b. `return stateTax`
   c. `return to main(stateTax);`
   d. none of the above

9. Unless specified otherwise, variables in C++ are passed _____ .

   a. *by address*
   b. *by content*
   c. *by reference*
   d. *by value*

10. A variable's _____ indicates where in the program a variable can be used.

    a. lifetime
    b. range
    c. scope
    d. span

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

11. A program contains three functions named `main`, `calcGross`, and `displayGross`. Two of the functions—`main` and `calcGross`—declare a variable named `pay`. The `pay` variable name also appears in the `displayGross` function header. When the computer processes the statement `pay = hours * rate;` in the `calcGross` function, it multiplies the contents of the `hours` variable by the contents of the `rate` variable. It then stores the result in which function's `pay` variable?

    a. `calcGross`
    b. `displayGross`
    c. `main`
    d. none of the above because you can't have more than one memory location with the same name

12. The variables in a function header have local scope.

    a. True
    b. False

Exercises

Pencil and Paper

TRY THIS

1. Write the C++ code for a function that receives an integer followed by a `double` number from the calling statement. The function should multiply the integer by the `double` number and then return the result as a `double` number. Name the function `getProduct`. Name the formal parameters `num1` and `num2`. (The answers to TRY THIS Exercises are located at the end of the chapter.)

TRY THIS

2. Write the function prototype for the `getProduct` function from Pencil and Paper Exercise 1. Also write the statement to call the function, assigning its return value to the `product` variable. Name the actual arguments `firstNum` and `secondNum`. (The answers to TRY THIS Exercises are located at the end of the chapter.)

INTRODUCTORY

3. Write an assignment statement that raises the number 3 to the 16th power and then assigns the result to a `double` variable named `answer`.

INTRODUCTORY

4. Write a C++ statement that displays a random integer from 50 through 100 on the computer screen.

INTRODUCTORY

5. Write a C++ statement that assigns the square root of a number to a `double` variable named `sqRoot`. The number is stored in a `double` variable named `num`.

INTRODUCTORY

6. Write the C++ code for a function that prompts the user to enter a character and then stores the character in a `char` variable named `response`. The function should return the contents of the `response` variable. Name the function `getCharacter`. (The function will not have any actual arguments passed to it.) Also write an appropriate function prototype for the `getCharacter` function. In addition, write a statement that invokes the `getCharacter` function and assigns its return value to a `char` variable named `custCode`.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

7. Write a C++ statement that adds the cube of the number stored in the `num1` variable to the square root of the number stored in the `num2` variable. The statement should assign the result to the `answer` variable. All of the variables have the `double` data type. INTERMEDIATE

8. Write a C++ statement that calculates the square root of the following expression: $x^2 * y^3$. The `x` and `y` variables have the `double` data type. Assign the result to a `double` variable named `answer`. INTERMEDIATE

9. Write a C++ statement that calculates the result of the following expression: $(x / y)^4 - 10$. The `x` and `y` variables have the `double` data type. Assign the result to a `double` variable named `answer`. INTERMEDIATE

10. A program's `main` function declares three `double` variables named `salesTax`, `sales`, and `taxRate`. It also declares a `char` variable named `status`. The `main` function contains the following statement: `salesTax = getSalesTax(sales, status, taxRate);`. The `getSalesTax` function header is shown here. Correct the function header. SWAT THE BUGS

```
int getSalesTax(char code, int sold, double rate)
```

Computer

11. If necessary, create a new project named TryThis11 Project, and save it in the Cpp8\Chap09 folder. Create a program that displays the average of three integers. The program should pass the three integers to a program-defined function named `calcAvg`, which should return the result as a `double` number. Name the formal parameters `n1`, `n2`, and `n3`. Use the following names for the actual arguments: `num1`, `num2`, and `num3`. Assign the function's return value to a `double` variable named `avg`. Enter your C++ instructions into a source file named TryThis11.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Display the average in fixed-point notation with one decimal place. Test the program using the numbers 20, 35, and 67; the average should be 40.7. (The answers to TRY THIS Exercises are located at the end of the chapter.) TRY THIS

12. If necessary, create a new project named TryThis12 Project, and save it in the Cpp8\Chap09 folder. Code the IPO charts shown in Figure 9-37. Enter your C++ instructions into a source file named TryThis12.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Display the Celsius temperature in fixed-point notation with no decimal places. Test the program using the following Fahrenheit temperatures: 32 and 212. (The answers to TRY THIS Exercises are located at the end of the chapter.) TRY THIS

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

`main` function

Input	Processing	Output
Fahrenheit temperature	Processing items: none	Celsius temperature

Algorithm:
1. enter Fahrenheit temperature
2. call calcCelsius to calculate the Celsius temperature; pass the Fahrenheit temperature
3. display the Celsius temperature

`calcCelsius` function

Input	Processing	Output
Fahrenheit temperature	Processing items: none	Celsius temperature

Algorithm:
1. Celsius temperature = 5.0 / 9.0 * (Fahrenheit temperature - 32.0)
2. return the Celsius temperature

Figure 9-37

MODIFY THIS

13. In this exercise, you will modify the hypotenuse program shown earlier in Figure 9-6. Follow the instructions for starting C++ and viewing the ModifyThis13.cpp file, which is contained in either the Cpp8\Chap09\ModifyThis13 Project folder or the Cpp8\Chap09 folder. (Depending on your C++ development tool, you may need to open this exercise's project/solution file first.) Remove both calculation tasks from the `main` function, and assign both to a program-defined value-returning function named `getHypotenuse`. Test the program appropriately.

MODIFY THIS

14. In this exercise, you will modify the program from Computer Exercise 12. If necessary, create a new project named ModifyThis14 Project, and save it in the Cpp8\Chap09 folder. Enter (or copy) the TryThis12.cpp instructions into a new source file named ModifyThis14.cpp. Change TryThis12.cpp in the first comment to ModifyThis14.cpp. Modify the program so that the user can convert as many temperatures as desired without having to run the program again. Test the program appropriately.

MODIFY THIS

15. In this exercise, you will modify the guessing game program shown earlier in Figure 9-11. Follow the instructions for starting C++ and viewing the ModifyThis15.cpp file, which is contained in either the Cpp8\Chap09\ModifyThis15 Project folder or the Cpp8\Chap09 folder. (Depending on your C++ development tool, you may need to open this exercise's project/solution file first.) Modify the program so that it allows the user only five chances to guess the number. After the fifth wrong guess, display the number on the computer screen. Test the program appropriately.

INTRODUCTORY

16. In this exercise, you will create a program that displays a table consisting of four rows and three columns. The first column should contain the numbers 10 through 13. The second and third columns should contain the results of squaring and cubing, respectively, the numbers 10 through 13. The table will look similar to the one shown in Figure 9-38. If necessary, create a new project named Introductory16 Project, and save it in the Cpp8\Chap09 folder. Enter your C++ instructions into a source file named Introductory16.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Test the program appropriately.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Number	Square	Cube
10	100	1000
11	121	1331
12	144	1728
13	169	2197

Figure 9-38

17. In this exercise, you will create a program that displays the gross pay for one or more employees. If necessary, create a new project named Introductory17 Project, and save it in the Cpp8\Chap09 folder. The program should allow the user to enter the number of hours the employee worked and his or her hourly pay rate. Use a negative sentinel value to stop the program. Employees are paid at their regular pay rate for hours worked from 1 through 37. They are paid time and one-half for the hours worked from 38 through 50, and double-time for the hours worked over 50. Use a program-defined function to calculate and return the employee's overtime pay, if applicable. Enter your C++ instructions into a source file named Introductory17.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Test the application appropriately. (Hint: If an employee earns $10 per hour and works 37 hours, the gross pay is $370.00. If he or she works 38 hours, the gross pay is $385.00. If he or she works 51 hours, the gross pay is $585.00.) INTRODUCTORY

18. In this exercise, you will create a program that displays a measurement in either inches or centimeters. If necessary, create a new project named Introductory18 Project, and save it in the Cpp8\Chap09 folder. The program should allow the user the choice of converting a measurement from inches to centimeters or vice versa. Use two program-defined functions: one for each different conversion type. Enter your C++ instructions into a source file named Introductory18.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Test the application appropriately. INTRODUCTORY

19. In this exercise, you will modify the car payment program from Lab 9-2. If necessary, create a new project named Intermediate19 Project, and save it in the Cpp8\Chap09 folder. Copy the instructions from the Lab9-2.cpp file into a source file named Intermediate19.cpp. (Alternatively, you can enter the instructions from Figure 9-34 into the Intermediate19.cpp file.) Change the filename in the first comment. Make the modifications listed in Figure 9-39. Test the program appropriately. INTERMEDIATE

1. Before calculating a monthly payment, verify that the denominator in the periodic payment formula is not the number 0. If it is 0, the function should return the number –1 (the negative number 1).
2. In addition to displaying the monthly payments, the program should also display the following two amounts:
   a. The total amount the user will pay for the car if the loan is financed through the credit union.
   b. The total amount the user will pay for the car if the loan is financed through the car dealer.

Figure 9-39

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

INTERMEDIATE

20. In this exercise, you will modify the savings account program shown earlier in Figure 9-18. Follow the instructions for starting C++ and viewing the Intermediate20.cpp file, which is contained in either the Cpp8\Chap09\Intermediate20 Project folder or the Cpp8\Chap09 folder. (Depending on your C++ development tool, you may need to open this exercise's project/solution file first.) Modify the program to allow the user to enter the minimum and maximum interest rates, as shown in Figure 9-40. Test the program appropriately.

```
Deposit: 1000
Minimum rate (in decimal form): 0.02
Maximum rate (in decimal form): 0.04
Rate 2%:
     Year 1: $1020.00
     Year 2: $1040.40
     Year 3: $1061.21
Rate 3%:
     Year 1: $1030.00
     Year 2: $1060.90
     Year 3: $1092.73
Rate 4%:
     Year 1: $1040.00
     Year 2: $1081.60
     Year 3: $1124.86
```

Figure 9-40

INTERMEDIATE

21. In this exercise, you will modify the program that you created in Chapter 6's Lab 6-2. If necessary, create a new project named Intermediate21 Project, and save it in the Cpp8\Chap09 folder. Copy the instructions from the Lab6-2.cpp file (which is contained in either the Cpp8\Chap06\Lab6-2 Project folder or the Cpp8\Chap06 folder) into a source file named Intermediate21.cpp. (Alternatively, you can enter the instructions from Figure 6-29 into the Intermediate21.cpp file.) Change the filename in the first comment. Modify the program so that it uses two value-returning functions: one to calculate and return the price of a medium pizza and the other to calculate and return the price of a large pizza. In addition to the $2 coupon on the purchase of a large pizza, Sophia is now e-mailing customers a $1 coupon on the purchase of a medium pizza. Test the program appropriately.

ADVANCED

22. A local department store wants a program that displays the number of reward points a customer earns each month. The reward points are based on the customer's membership type and total monthly purchase amount, as shown in Figure 9-41. The program should use a separate function for each membership type. If necessary, create a new project named Advanced22 Project, and save it in the Cpp8\Chap09 folder. Enter your C++ instructions into a source file named Advanced22.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Display the reward points in fixed-point notation with no decimal places. Test the program appropriately.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Membership type	Total monthly purchase ($)	Reward points
Standard	Less than 75	5% of the total monthly purchase
	75–149.99	7.5% of the total monthly purchase
	150 and over	10% of the total monthly purchase
Plus	Less than 150	6% of the total monthly purchase
	150 and over	13% of the total monthly purchase
Premium	Less than 200	4% of the total monthly purchase
	200 and over	15% of the total monthly purchase

Figure 9-41

23. ADVANCED

An online retailer wants a program that displays the total amount a customer owes, including shipping. The user will enter the total amount due before shipping. The amount to charge for shipping is based on the customer's membership status, which can be either Standard or Premium. The appropriate shipping charges are shown in Figure 9-42. The program should use two program-defined functions: one to determine the shipping charge for a Standard member and the other to determine the shipping charge for a Premium member. If necessary, create a new project named Advanced23 Project, and save it in the Cpp8\Chap09 folder. Enter your C++ instructions into a source file named Advanced23.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Display the total due in fixed-point notation with two decimal places. Test the program appropriately.

Membership type	Total due before shipping ($)	Shipping ($)
Standard	0–100	12.99
	Over 100	4.99
Premium	0–49.99	4.99
	Over 49.99	0

Figure 9-42

24. ADVANCED

Create a program that displays five random addition problems, one at a time, on the computer screen. Each problem should be displayed as a question, like this: *What is the sum of x + y?*. The *x* and *y* in the question represent random numbers from 1 to 10, inclusive. After displaying the question, the program should allow the user to enter the answer. It should then compare the user's answer with the correct answer. If the user's answer matches the correct answer, the program should display the "Correct!" message. Otherwise, it should display the "Sorry, the answer is" message followed by the correct answer and a period. If necessary, create a new project named Advanced24 Project, and save it in the Cpp8\Chap09 folder. Enter your C++ instructions into a source file named Advanced24.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Test the program appropriately.

25. ADVANCED

In this exercise, you will create a program that displays the amount of a cable bill. The amount is based on the type of customer, as shown in Figure 9-43. For a residential customer, the user will need to enter the number of premium channels only. For a business customer, the user will need to enter the number of connections and the number of premium channels. Use a separate program-defined function for each customer type.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

If necessary, create a new project named Advanced25 Project, and save it in the Cpp8\Chap09 folder. Enter your C++ instructions into a source file named Advanced25.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Test the program appropriately.

Residential customers:	
Processing fee:	$4.50
Basic service fee:	$30
Premium channels:	$5 per channel
Business customers:	
Processing fee:	$16.50
Basic service fee:	$80 for the first 5 connections; $4 for each additional connection
Premium channels:	$50 per channel for any number of connections

Figure 9-43

SWAT THE BUGS

26. Follow the instructions for starting C++ and viewing the SwatTheBugs26.cpp file, which is contained in either the Cpp8\Chap09\SwatTheBugs26 Project folder or the Cpp8\Chap09 folder. (Depending on your C++ development tool, you may need to open this exercise's project/solution file first.) The program should calculate and display the miles per gallon, but it is not working correctly. Debug the program.

Answers to TRY THIS Exercises

Pencil and Paper

1.
```
double getProduct(int num1, double num2)
{
   return num1 * num2;
} //end of getProduct function
```
2.
```
double getProduct(int num1, double num2);
```
or
```
double getProduct(int, double);
```
```
product = getProduct(firstNum, secondNum)
```

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Computer

11. See Figure 9-44.

```
//TryThis11.cpp - displays an average
//Created/revised by <your name> on <current date>

#include <iostream>
#include <iomanip>
using namespace std;

//function prototype
double calcAvg(int n1, int n2, int n3);

int main()
{
    int num1 = 0;
    int num2 = 0;
    int num3 = 0;
    double avg = 0.0;

    cout << "First number: ";
    cin >> num1;
    cout << "Second number: ";
    cin >> num2;
    cout << "Third number: ";
    cin >> num3;

    avg = calcAvg(num1, num2, num3);
    cout << fixed << setprecision(1);
    cout << "Average: " << avg << endl;
    return 0;
}   //end of main function

    //*****function definitions*****
double calcAvg(int n1, int n2, int n3)
{
    return (n1 + n2 + n3) / 3.0;
}   //end of calcAvg function
```

Figure 9-44

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

12. See Figure 9-45.

```
//TryThis12.cpp - converts Fahrenheit to Celsius
//Created/revised by <your name> on <current date>

#include <iostream>
#include <iomanip>
using namespace std;

//function prototype
double calcCelsius(double tempF);

int main()
{
    double fahrenheit = 0.0;
    double celsius = 0.0;

    cout << "Enter Fahrenheit temperature: ";
    cin >> fahrenheit;
    celsius = calcCelsius(fahrenheit);

    cout << fixed << setprecision(0);
    cout << "Celsius temperature: " << celsius << endl;
    return 0;
}   //end of main function

//*****function definitions*****
double calcCelsius(double tempF)
{
    double tempC = 0.0;
    tempC = 5.0 / 9.0 * (tempF - 32.0);
    return tempC;
}   //end of calcCelsius function
```

Figure 9-45

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

CHAPTER 10

Void Functions

After studying Chapter 10, you should be able to:

- Create a void function
- Invoke a void function
- Pass information *by reference* to a function

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Functions

As discussed in Chapter 9, all functions are categorized as either value-returning functions or void functions. Recall that a value-returning function performs a task and then returns precisely one value to the statement that called it. **Void functions**, on the other hand, do not return a value after completing their task.

The illustration shown in Figure 10-1 may help clarify the difference between the two categories of functions. Sarah and her two siblings are planning a surprise birthday party for their mother. Being the oldest of the three children, Sarah will handle most of the party plans herself. However, she does need to delegate some tasks to her brother (Jacob) and sister (Sonja). She delegates the task of putting up the decorations (streamers, balloons, and so on) to Jacob and delegates the task of getting the birthday present (a bottle of perfume) to Sonja. Like a void function, Jacob will perform his task but will not need to return anything to Sarah after doing so. However, like a value-returning function, Sonja will perform her task and then return a value (the bottle of perfume) to Sarah for wrapping.

Figure 10-1 Illustration of value-returning and void functions
Image by Diane Zak; created with Reallusion CrazyTalk Animator

In Chapter 9, you learned how to use one built-in void function, `srand`, to initialize the C++ random number generator. In this chapter, you will learn how to create and invoke program-defined void functions.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Creating Program-Defined Void Functions

A program might use a void function to display information (such as a title and column headings) at the top of each page in a report. Rather than duplicating the required code several times in the program, the code can be entered in a void function that can then be called whenever and wherever it is needed in the program. A void function is appropriate in this situation because it does not need to return a value after completing its task.

Figure 10-2 shows the syntax for creating a void function in a C++ program. When you compare this syntax with the one for creating a value-returning function (shown in Figure 9-12 in Chapter 9), you will notice two differences. First, a void function's header begins with the keyword `void` rather than with a data type. The `void` keyword indicates that the function does not return a value. Second, the function body in a void function does not contain a `return` statement, which is required in the function body of a value-returning function. The `return` statement is not necessary in a void function body because a void function does not return a value. Also included in Figure 10-2 are examples of program-defined void functions.

HOW TO Create a Program-Defined Void Function

Syntax

```
void functionName([parameterList])          <- function header
{                                           <- function body
      one or more statements
} //end of functionName function
```

Example 1

```
void displayLine()                          <- function definition
{
     cout << "-------------------------" << endl;
}    //end of displayLine function
```

The function displays a straight line composed of 25 hyphens.

Example 2

```
void displayCompanyInfo()
{
     cout << "Martin Sports" << endl;
     cout << "Atlanta, GA" << endl << endl;
}    //end of displayCompanyInfo function
```

The function displays a company's name, city, and state.

Example 3

```
void displayTotalSales(double total)
{
     cout << fixed << setprecision(2);
     cout << "Total sales: $" << total << endl;
}    //end of displayTotalSales function
```

The function displays the total sales it receives from the statement that invoked it.

Figure 10-2 How to create a program-defined void function

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Recall that value-returning functions are typically called from statements that display the return value, use the return value in a calculation or comparison, or assign the return value to a variable.

You use the same method to invoke (call) a void function as you do to invoke a value-returning function: You simply include the function's name and actual arguments (if any) in a program statement. However, unlike a call to a value-returning function, a call to a void function is an independent statement. Figure 10-3 shows the statements you would use to call the void functions defined in Figure 10-2.

HOW TO Call (Invoke) a Void Function

Syntax
functionName(*[argumentList]*)

Statements for calling the void functions from Figure 10-2

```
displayLine();
displayCompanyInfo();
displayTotalSales(totalSales);
```

Figure 10-3 How to call (invoke) a void function

Figure 10-4 shows the problem specification, IPO chart information, and C++ instructions for the Martin Sports program, which uses the void functions and function calls from Figures 10-2 and 10-3. The program displays two horizontal lines, as well as the company's name, city, and state. It also displays the total sales made in the company's two stores.

Problem specification
Create a program that allows the sales manager of Martin Sports to enter the sales made in two stores. The program should total both sales amounts and then display the following information, in which *total* is the total sales:

```
------------------------------
Martin Sports
Atlanta, GA

Total sales: $total
------------------------------
```

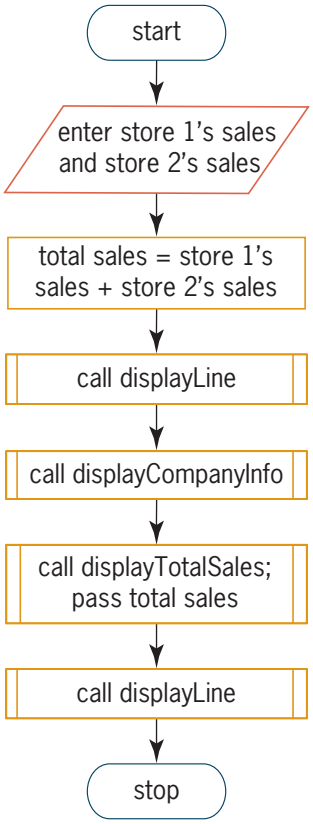
main function **IPO chart information**	**main function** **C++ instructions**
Input	
store 1's sales	`double store1Sales = 0.0;`
store 2's sales	`double store2Sales = 0.0;`
Processing	
none	
Output	
total sales	`double totalSales = 0.0;` (displayed by the displayTotalSales function)
straight line (2 of them)	displayed by the displayLine function
name, city, and state	displayed by the displayCompanyInfo function

Figure 10-4 Problem specification, IPO chart information, and C++ instructions for the Martin Sports program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Algorithm




```
cout << "Store 1's sales: ";
cin >> store1Sales;
cout << "Store 2's sales: ";
cin >> store2Sales;
```

```
totalSales = store1Sales + store2Sales;
```

```
displayLine();
```

```
displayCompanyInfo();
```

```
displayTotalSales(totalSales);
```

```
displayLine();
```

displayLine function
IPO chart information

Input

none

Processing

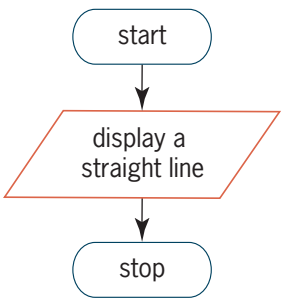
none

Output

straight line (25 hyphens) — *displayed using a string literal constant*

displayLine function
C++ instructions

Algorithm




```
cout << "-------------------------" << endl;
```

Figure 10-4 Problem specification, IPO chart information, and C++ instructions for the Martin Sports program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

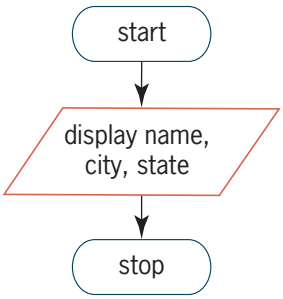
displayCompanyInfo function
IPO chart information

Input
none

Processing
none

Output
name, city, and state

Algorithm



displayCompanyInfo function
C++ instructions

displayed using string literal constants

```
cout << "Martin Sports" << endl;
cout << "Atlanta, GA" << endl << endl;
```

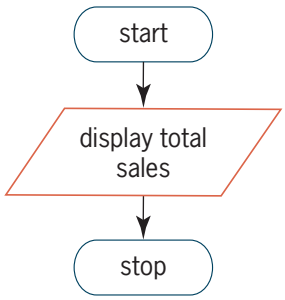
displayTotalSales function
IPO chart information

Input
total sales (formal parameter)

Processing
none

Output
total sales

Algorithm



displayTotalSales function
C++ instructions

```
double total
```

stored in the total formal parameter

```
cout << fixed << setprecision(2);
cout << "Total sales: $" << total << endl;
```

Figure 10-4 Problem specification, IPO chart information, and C++ instructions for the Martin Sports program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Figure 10-5 shows all of the code entered in the Martin Sports program. The function prototypes and calls are shaded in the figure. Notice that each call to a void function is a self-contained statement.

```
 1 //Martin.cpp - displays the total sales
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <iomanip>
 6 using namespace std;
 7
 8 //function prototypes
 9 void displayLine();
10 void displayCompanyInfo();
11 void displayTotalSales(double total);
12
13 int main()
14 {
15    double store1Sales = 0.0;
16    double store2Sales = 0.0;
17    double totalSales = 0.0;
18
19    cout << "Store 1's sales: ";
20    cin >> store1Sales;
21    cout << "Store 2's sales: ";
22    cin >> store2Sales;
23
24    totalSales = store1Sales + store2Sales;
25
26    displayLine();
27    displayCompanyInfo();
28    displayTotalSales(totalSales);
29    displayLine();
30
31    return 0;
32 }    //end of main function
33
34 //*****function definitions*****
35 void displayLine()
36 {
37    cout << "-------------------------" << endl;
38 }   //end of displayLine function
39
40 void displayCompanyInfo()
41 {
42    cout << "Martin Sports" << endl;
43    cout << "Atlanta, GA" << endl << endl;
44 }   //end of displayCompanyInfo function
45
46 void displayTotalSales(double total)
47 {
48    cout << fixed << setprecision(2);
49    cout << "Total sales: $" << total << endl;
50 }   //end of displayTotalSales function
```

Figure 10-5 Martin Sports program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

When the computer processes a statement that calls a program-defined void function, the computer first locates the function's code in the program. If the function call contains an *argumentList*, the computer passes a copy of the values stored in the actual arguments (assuming the variables included in the *argumentList* are passed *by value*) to the called function. The function receives the values and stores them in the formal parameters listed in its *parameterList*. Then, the computer processes the function's code. When the function ends, the computer continues program execution with the statement immediately below the one that called the function.

In the program shown in Figure 10-5, for example, the statement on Line 26 calls the `displayLine` function. After processing the function's code, the computer returns to the `main` function to process the statement on Line 27. The statement on Line 27 calls the `displayCompanyInfo` function.

When the computer finishes processing the code in the `displayCompanyInfo` function, it returns to the `main` function and processes the statement on Line 28. That statement calls the `displayTotalSales` function, passing it a copy of the value stored in the `totalSales` variable. The function stores the value in its formal parameter (`total`) and then displays the value in a message on the computer screen.

When the `displayTotalSales` function ends, the computer returns to the `main` function to process the `displayLine();` statement on Line 29. After the `displayLine` function completes its task, the computer returns to the `main` function to process the `return 0;` statement on Line 31. Figure 10-6 shows a sample run of the program. As the figure indicates, the sample run contains the output from each of the program's four functions.

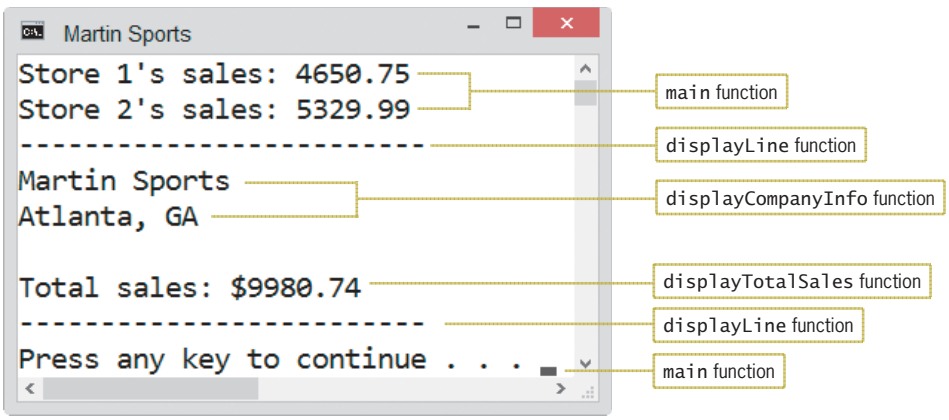


Figure 10-6 Sample run of the Martin Sports program

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 10-1

1. In C++, the function header for a function that does not return a value begins with the keyword _____.
2. Write a C++ statement that calls a void function named `displayTaxes`, passing it a copy of the values stored in two `double` variables named `federalTax` and `localTax`.
3. Write the function header for the `displayTaxes` function from Question 2. Use `fedTax` and `stateTax` as the names for the formal parameters.
4. The `return` statement is typically the last statement in a C++ void function.
   a. True
   b. False

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Passing Variables to a Function

As you learned in Chapter 9, the items passed to a function are called actual arguments. An actual argument can be a variable, named constant, literal constant, or keyword; however, in most cases, it will be a variable. Recall that each variable declared in a program has both a value and a unique address that represents the location of the variable in the computer's internal memory. C++ allows you to pass either a copy of the variable's value or its address to a function. Passing a copy of a variable's value is referred to as **passing by value**, whereas passing its address is referred to as **passing by reference**. The method you choose—*by value* or *by reference*—depends on whether you want the receiving function to have access to the variable in memory. In other words, it depends on whether you want to allow the receiving function to change the contents of the variable.

The internal memory of a computer is similar to a large post office. Like each post office box, each memory cell has a unique address.

You already are familiar with the concept of passing information *by value* and *by reference*. The illustrations shown in Figure 10-7 can be used to demonstrate this fact. Assume you have a savings account at a local bank. (Think of the savings account as a variable.) During a conversation with your friend Melissa, you mention the amount of money you have in the account, as shown in Illustration A. Sharing this information with Melissa is similar to passing a variable *by value*. Knowing the balance in your account does not give Melissa access to your bank account. It merely provides information that she can use to compare with the amount of money she has saved.

Only variables can be passed *by reference*.

Now we'll use the savings account example to demonstrate passing information *by reference*. (Here again, think of your savings account as a variable.) To either deposit money into your account or withdraw money from your account, you must provide the bank teller with your account number, as shown in Illustration B in Figure 10-7. The account number represents the location of your account at the bank and allows the teller to change the account balance. Giving the teller your bank account number is similar to passing a variable *by reference*. The account number allows the teller to change the contents of your bank account, similar to the way a variable's address allows the receiving function to change the contents of the variable.

Illustration A

Illustration B

Figure 10-7 Illustrations of passing *by value* and passing *by reference*
Image by Diane Zak; created with Reallusion CrazyTalk Animator

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Before learning how to pass a variable *by reference*, we will review the concept of passing *by value*, which you learned about in Chapter 9.

Reviewing Passing Variables *by Value*

Recall that unless you specify otherwise, variables are passed *by value* in C++. This means that the computer passes only a copy of the variable's contents to the receiving function. When only a copy of the contents is passed, the receiving function is not given access to the variable in memory, and, therefore, it cannot change the value stored inside of the variable. It is appropriate to pass a variable *by value* when the receiving function needs to *know* the variable's contents but does not need to *change* the contents.

The company ratings program shown in Figure 10-8 passes a variable *by value* to a void function named `displayRating`. The function definition is located below the `main` function (on Lines 28 through 34). Therefore, the program includes an appropriate function prototype above the `main` function (on Line 8). Because the `displayRating` function is a void function, its function call (on Line 18) appears as a statement by itself. The function call passes the `numStars` variable *by value* to the void function. This means that only a copy of the variable's value is passed to the function, which stores that value in its formal parameter (`num`). The `displayRating` function does not have access to the `numStars` variable. It is not even aware of the variable's existence in the computer's internal memory.

```
 1 //Ratings.cpp - displays company ratings
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 using namespace std;
 6
 7 //function prototype
 8 void displayRating(int num);
 9
10 int main()
11 {
12    int numStars = 0;
13
14    cout << "Rate the XYZ Company (1 to 5 stars): ";
15    cin >> numStars;
16    while (numStars > 0 && numStars < 6)
17    {
18        displayRating(numStars);
19        cout << "Rate the XYZ Company (1 to 5 stars): ";
20        cin >> numStars;
21    }  //end while
22    cout << "End of ratings" << endl;
23
24    return 0;
25 }    //end of main function
26
27 //*****function definitions*****
28 void displayRating(int num)
```

the name is not required in the function prototype

function call

function header

Figure 10-8 Company ratings program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
29 {
30    for (int star = 1; star <= num; star += 1)
31        cout << "*";
32    //end for
33    cout << endl;
34 }  //end of displayRating function
```

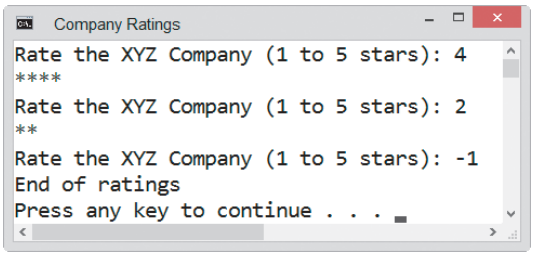


Figure 10-8 Company ratings program

Notice that the data type of the actual argument in the function call—in this case, `int`—matches the data type of the formal parameter listed in both the void function's header and its prototype. Also notice that the names of the actual argument (`numStars`) and the formal parameter (`num`) are not the same. Recall from Chapter 9 that to avoid confusion, you should use different names for an actual argument and its corresponding formal parameter.

To review the concept of passing *by value*, we will desk-check the program in Figure 10-8 using the following ratings: 4, 2, and –1 (a sentinel value). The first statement in the `main` function creates and initializes an `int` variable named `numStars`. The variable is local to the `main` function and will remain in memory until the `main` function ends. The next two statements prompt the user to enter a rating and then store the user's response (4) in the `numStars` variable.

The `while` statement's condition is evaluated next. The condition evaluates to true, so the first statement in the loop invokes the `displayRating` function, passing it the value stored in the `numStars` variable (4). At this point, the computer temporarily leaves the `main` function to process the code contained in the `displayRating` function.

The `displayRating` function's header tells the computer to create a local `int` variable named `num`. The computer stores the value passed to the function in the variable. The `for` loop in the function uses an `int` variable named `star` to display the appropriate number of asterisks. The number of asterisks to display was passed to the `displayRating` function from the `main` function and is stored in the `num` variable. When the `for` loop ends, the `star` variable is removed from the computer's internal memory.

The last statement in the `displayRating` function moves the cursor to the next line on the computer screen. When the function ends, the `num` variable is removed from memory, as indicated in the desk-check table shown in Figure 10-9. Only the `main` function's local `numStars` variable remains in the computer's memory.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

As you learned in Chapter 9, the variables listed in a function header are local to the function, and they remain in memory until the function ends.

Note: The names in black indicate variables that belong to the `main` function. The names in red indicate variables that belong to the `displayRating` function.

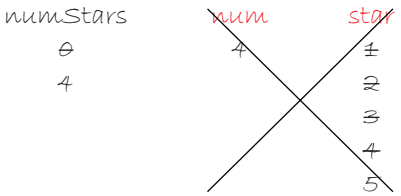


Figure 10-9 Desk-check table after the `displayRating` function ends the first time

Now the computer returns to the `main` function to process the statement immediately following the one that called the `displayRating` function. That statement and the one immediately below it prompt the user to enter a rating and then store the user's response (2) in the `numStars` variable. Next, the `while` statement's condition is evaluated again. The condition still evaluates to true, so the first statement in the loop calls the `displayRating` function and passes it the value stored in the `numStars` variable (2). Here again, the computer temporarily leaves the `main` function to process the code contained in the `displayRating` function.

Using the `displayRating` function's header as a guide, the computer recreates the `num` variable and then stores the value passed to the function (2) in the variable. Next, the `for` loop in the function recreates its `star` variable and uses it to keep track of the appropriate number of asterisks to display. When the `for` loop ends, the `star` variable is removed from the computer's internal memory.

The last statement in the `displayRating` function moves the cursor to the next line on the computer screen. When the function ends, the `num` variable is removed from memory, as indicated in the desk-check table shown in Figure 10-10. Only the `main` function's local `numStars` variable remains in the computer's memory.

Note: The names in black indicate variables that belong to the `main` function. The names in red indicate variables that belong to the `displayRating` function.

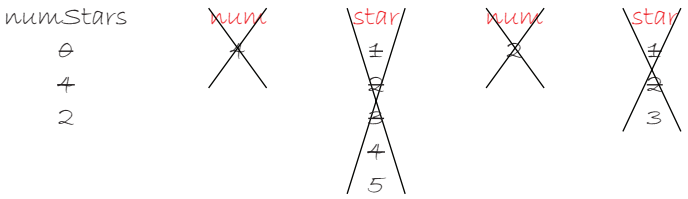


Figure 10-10 Desk-check table after the `displayRating` function ends the second time

After processing the code in the `displayRating` function, the computer returns to the `main` function to process the statement immediately following the one that called the function. That statement and the one immediately below it prompt the user to enter a rating and then store the user's response (–1) in the `numStars` variable. The `while` statement's condition is evaluated once again. This time, the condition evaluates to false, so the computer processes the `cout` statement on Line 22. That statement displays the "End of ratings" message on the

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

computer screen. Finally, the `return 0;` statement is processed and ends the program. At this point, the computer removes the `main` function's local `numStars` variable from its internal memory.

Passing Variables *by Reference*

Instead of passing a copy of a variable's value to a function, you can pass the variable's location in the computer's internal memory—in other words, its address. As you learned earlier, passing a variable's address is referred to as passing *by reference*, and it gives the receiving function access to the variable being passed. You pass a variable *by reference* when you want the receiving function to change the contents of the variable.

To pass a variable *by reference* in C++, you include an ampersand (&) before the name of the corresponding formal parameter in the receiving function's header and in its prototype (if there is one). The **&** (ampersand) is called the **address-of operator**, and it tells the computer to pass the variable's address rather than a copy of its contents. If the function's prototype does not include the formal parameter's name, you enter a space followed by the address-of operator after the formal parameter's data type.

Recall that only functions defined below the `main` function require a function prototype above the `main` function.

The tips program shown in Figure 10-11 demonstrates how you pass a variable *by reference*. The program uses a void function named `getTips` to calculate both a 15% tip and a 20% tip on a restaurant bill. The statement that calls the function appears on Line 22. The statement passes three variables to the `getTips` function. The first variable (`totalBill`) is passed *by value*, whereas the second and third variables (`tip15` and `tip20`) are passed *by reference*. You can tell that the `tip15` and `tip20` variables are passed *by reference* because the address-of operator precedes the names of their corresponding formal parameters in the `getTips` function's header (on Lines 30 and 31) and in its prototype (on Lines 9 through 11). As Figure 10-11 shows, the statement that calls a function does not indicate whether an item is passed *by value* or *by reference*. That information can be determined only by examining the *parameterList* in either the receiving function's header or its prototype. Figure 10-11 also shows a sample run of the program.

```
 1 //Tips.cpp - displays the tips on a restaurant bill
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <iomanip>
 6 using namespace std;
 7
 8 //function prototype
 9 void getTips(double bill,
10              double &percent15,
11              double &percent20);
12
13 int main()
14 {
15    double totalBill = 0.0;
16    double tip15 = 0.0;
17    double tip20 = 0.0;
```

function prototype

Figure 10-11 Tips program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
18
19    cout << "Total bill: ";
20    cin >> totalBill;
21
22    getTips(totalBill, tip15, tip20);
23    cout << fixed << setprecision(2);
24    cout << "15% tip: $" << tip15 << endl;
25    cout << "20% tip: $" << tip20 << endl;
26    return 0;
27 }  //end of main function
28
29 //*****function definitions*****
30 void getTips(double bill,
31              double &percent15, double &percent20)
32 {
33    percent15 = bill * 0.15;
34    percent20 = bill * 0.2;
35 }  //end getTips function
```

function call

function header

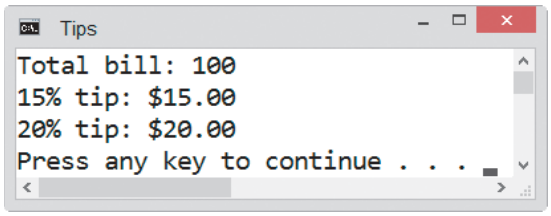


Figure 10-11 Tips program

Notice that the data types of the actual arguments in the function call match the data types of the `getTips` formal parameters, which are listed in both the function's header and prototype. Also notice that the names of the actual arguments are different from the names of the formal parameters.

Ch10-Tips

Desk-checking the tips program shown in Figure 10-11 will help you understand the concept of passing *by reference*. The first three statements in the `main` function create and initialize three `double` variables. The variables are local to the `main` function and will remain in memory until the `main` function ends. Next, the `cout` and `cin` statements prompt the user to enter the total bill and then store the user's response in the `totalBill` variable. Figure 10-12 shows the desk-check table at this point, assuming the user enters 100 as the total bill.

totalBill	tip15	tip20
~~0.0~~	0.0	0.0
100.0		

Figure 10-12 Desk-check table before the `getTips` function is called

The statement on Line 22 calls the `getTips` function, passing it a copy of the value stored in the `totalBill` variable, the address of the `tip15` variable, and the address of the `tip20` variable. At this point, the computer temporarily leaves the `main` function to process the code contained in the `getTips` function, beginning with the function header.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The first formal parameter in the `getTips` function header tells the computer to create a local `double` variable named `bill`. The computer stores a copy of the first actual argument's value (100.0) in the variable. (Recall that if the formal parameter does not include an ampersand, it means that the actual argument is being passed *by value*.)

The ampersands in the next two formal parameters indicate that the function receives the addresses of two `double` variables. (Recall that the ampersand indicates that an actual argument is being passed *by reference*.) When you pass a variable's address to a function, the computer uses the address to locate the variable in its internal memory. It then assigns the formal parameter's name to the memory location. In this case, the computer locates the `tip15` variable in memory and assigns the name `percent15` to it. It then locates the `tip20` variable and assigns it the name `percent20`. As indicated in the desk-check table in Figure 10-13, two of the memory locations now have two names: one assigned by the `main` function and the other assigned by the `getTips` function. Although both functions can access the memory locations, each function uses different names to do so. The `main` function uses the names `tip15` and `tip20`, whereas the `getTips` function uses the names `percent15` and `percent20`.

Note: The names in black indicate variables that belong to the `main` function. The names in red indicate variables that belong to the `getTips` function.

	percent15	percent20	
totalBill	tip15	tip20	bill
~~0.0~~	0.0	0.0	100.0
100.0			

Figure 10-13 Desk-check table after the `getTips` function header is processed

Next, the computer processes the two statements contained in the `getTips` function body. The first statement calculates a 15% tip and stores the result in the `percent15` variable. The second statement calculates a 20% tip and stores the result in the `percent20` variable. Figure 10-14 shows the desk-check table after the statements are processed. Notice that changing the value in the `percent15` variable also changes the value in the `tip15` variable. This is because both variable names refer to the same location in memory. Likewise, changing the value in the `percent20` variable also changes the value in the `tip20` variable.

Note: The names in black indicate variables that belong to the `main` function. The names in red indicate variables that belong to the `getTips` function.

	percent15	percent20	
totalBill	tip15	tip20	bill
~~0.0~~	~~0.0~~	~~0.0~~	100.0
100.0	15.0	20.0	

Figure 10-14 Desk-check table after the statements in the `getTips` function are processed

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The `getTips` function ends when the computer encounters the function's closing brace. At that point, the computer removes the `bill` variable from its internal memory. It also removes the `percent15` and `percent20` names from their locations in memory. Figure 10-15 shows the desk-check table after the `getTips` function ends. Notice that only the `main` function's variables remain in internal memory.

Note: The names in black indicate variables that belong to the `main` function. The names in red indicate variables that belong to the `getTips` function.

totalBill	~~percent15~~ tip15	~~percent20~~ tip20	~~bill~~
~~0.0~~	~~0.0~~	~~0.0~~	~~100.0~~
100.0	15.0	20.0	

Figure 10-15 Desk-check table after the `getTips` function ends

Next, the computer returns to the `main` function to process the statement immediately following the function call. That statement tells the computer to display the program output in fixed-point notation with two decimal places. The next two statements display the contents of the `tip15` and `tip20` variables on the computer screen. The last statement in the `main` function, `return 0;`, returns the number 0 to the operating system to indicate that the program ended normally. When the program ends, the `main` function's variables are removed from the computer's internal memory.

For more examples of void functions, see the Void Functions section in the Ch10WantMore.pdf file.

Keep in mind that when you pass a variable *by value*, the computer uses the data type and name of its corresponding formal parameter to create a separate variable in which to store the passed value. When you pass a variable *by reference*, on the other hand, the computer locates the variable in memory and then assigns the name of its corresponding formal parameter to the variable. This means that when you pass a variable *by reference*, the variable will have two names: one assigned by the calling function and the other assigned by the receiving function. Void functions use variables that are passed *by reference* to send information back to the calling function. Value-returning functions, on the other hand, send information back to the calling function through their return value.

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 10-2

1. Write the function header for a void function named `calcTaxes`. The function is passed the value of a `double` variable named `gross` and the addresses of two `double` variables named `federal` and `state`. Use `pay`, `fedTax`, and `stateTax` for the names of the formal parameters.
2. Write a C++ statement to call the `calcTaxes` function from Question 1.
3. Write the function prototype for the `calcTaxes` function from Question 1.
4. Unless specified otherwise, a variable's address is passed to a function in C++.
   a. True
   b. False

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

LAB 10-1 Stop and Analyze

Figure 10-16 shows a sample run of the program for Lab 10-1. Study the program's code shown in Figure 10-17, and then answer the questions.

The answers to the labs are contained in the Answers.pdf file.

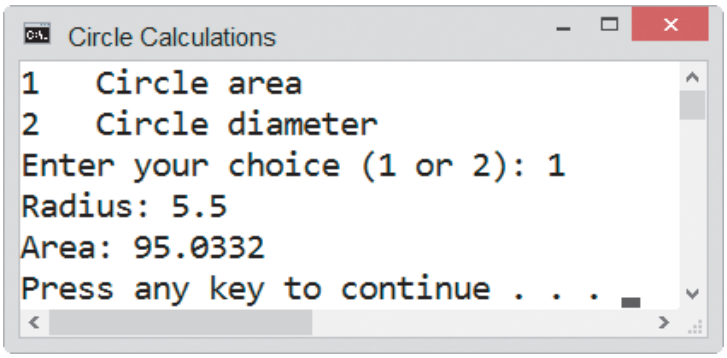


Figure 10-16 Sample run of the program for Lab 10-1

```
 1 //Lab10-1.cpp - circle calculations
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <cmath>
 6 using namespace std;
 7
 8 //function prototypes
 9 void displayChoices();
10 void getArea(double rad, double &area);
11 void getDiameter(double rad, double &diameter);
12
13 int main()
14 {
15    int choice = 0;
16    double radius = 0.0;
17    double circleArea = 0.0;
18    double circleDiameter = 0.0;
19
20    displayChoices();
21    cout << "Enter your choice (1 or 2): ";
22    cin >> choice;
23
24    if (choice < 1 || choice > 2)
25        cout << "Invalid choice" << endl;
26    else
27    {
28        cout << "Radius: ";
```

Figure 10-17 Code for Lab 10-1 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
29        cin >> radius;
30        if (choice == 1)
31        {
32            getArea(radius, circleArea);
33            cout << "Area: " << circleArea;
34        }
35        else
36        {
37            getDiameter(radius, circleDiameter);
38            cout << "Diameter: " << circleDiameter;
39        } //end if
40        cout << endl;
41    }  //end if
42    return 0;
43 }  //end of main function
44
45 //*****function definitions*****
46 void displayChoices()
47 {
48    cout << "1   Circle area" << endl;
49    cout << "2   Circle diameter" << endl;
50 }
51
52 void getArea(double rad, double &area)
53 {
54    const double PI = 3.141593;
55    area = PI * pow(rad, 2);
56 }  //end getArea function
57
58 void getDiameter(double rad, double &diameter)
59 {
60    diameter = 2 * rad;
61 }  //end getDiameter function
```

Figure 10-17 Code for Lab 10-1

QUESTIONS

1. The `main` function passes two variables to the `getArea` function. Which lines in the program indicate whether the variables are passed *by value* or *by reference*?
2. Why is the `radius` variable passed *by value*? Why are the `circleArea` and `circleDiameter` variables passed *by reference*?
3. Why is the `displayChoices` function a void function?
4. How do the `getArea` and `getDiameter` functions, which are void functions, send information back to the `main` function?
5. Follow the instructions for starting C++ and viewing the Lab10-1.cpp file, which is contained in either the Cpp8\Chap10\Lab10-1 Project folder or the Cpp8\Chap10 folder. (Depending on your C++ development tool, you may need to open Lab10-1's project/solution file first.) Run the program. Type 1 and press Enter, and then type 5.5 and press Enter. The program displays the number 95.0332, as shown earlier in Figure 10-16.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

6. Use the program to calculate the diameter for a circle whose radius is 25.7. The diameter is 51.4.
7. Change the `getArea` function to a value-returning function. Save and then run the program. Use the program to calculate the area of a circle whose radius is 5.5. Then use it to calculate the diameter of a circle whose radius is 25.7.

LAB 10-2 Plan and Create

In this lab, you will plan and create an algorithm for Patterson Windows. The problem specification and sample calculations are shown in Figure 10-18.

Create a program for Patterson Windows, a company that sells energy-efficient replacement windows for homes. The program should display the total amount a customer owes, given the number of windows ordered and the price per window. The total owed is calculated by multiplying the number of windows ordered by the price per window. However, several times during the year, the company has a BOGO (buy one, get one free) offer.

Example using regular pricing	Example using BOGO pricing
Number of windows: 11	Number of windows: 15
Price per window: 300	Price per window: 200
Total owed (11 * 300): $3300.00	Total owed (8 * 200): $1600.00

Figure 10-18 Problem specification and a sample calculation for Lab 10-2

First, analyze the problem, looking for the output first and then for the input. In this case, the program needs to display the total amount the customer owes. To calculate that amount, the computer will need to know the number of windows ordered, the price per window, and the pricing option (either regular or BOGO).

After analyzing the problem, you plan the algorithm. In this case, in addition to the `main` function, the program will use three void functions named `displayOptions`, `getRegular`, and `getBoGo`. The `displayOptions` function will display the pricing options on the computer screen. A void function is appropriate for this task because the function will not need to return a value.

The `getRegular` function will calculate the total owed using the regular pricing option. The `getBoGo` function, on the other hand, will calculate the total owed using the BOGO pricing option. The `getRegular` and `getBoGo` functions could be coded as either value-returning or void functions. For this lab, you will use void functions. (You will change the functions to value-returning functions in Lab 10-3.)

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

To calculate the total owed, the `getRegular` and `getBoGo` functions will need to know the number of windows ordered and the price per window. The calling statement will pass that information *by value* to the functions. Because both functions will be void functions, they will also need the calling statement to pass them the address of a variable in which to store the calculated results. Figure 10-19 shows the completed IPO charts for the program's four functions.

`main` function

Input	**Processing**	**Output**
pricing option number of windows window price	Processing items: none	total owed

Algorithm:

```
1. call the displayOptions function to display
   the pricing options
2. get the pricing option
3. if (the pricing option is either 1 or 2)
       get the number of windows and the window price
       if (the pricing option is 1)
           call the getRegular function to calculate
           the total owed; pass the number of windows
           and the window price, as well as the address
           of a variable to store the total owed
       else
           call the getBoGo function to calculate
           the total owed; pass the number of windows
           and the window price, as well as the address
           of a variable to store the total owed
       end if
       display the total owed
   else
       display "Invalid option" message
   end if
```

Figure 10-19 IPO charts for the functions in the Lab10-2 program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

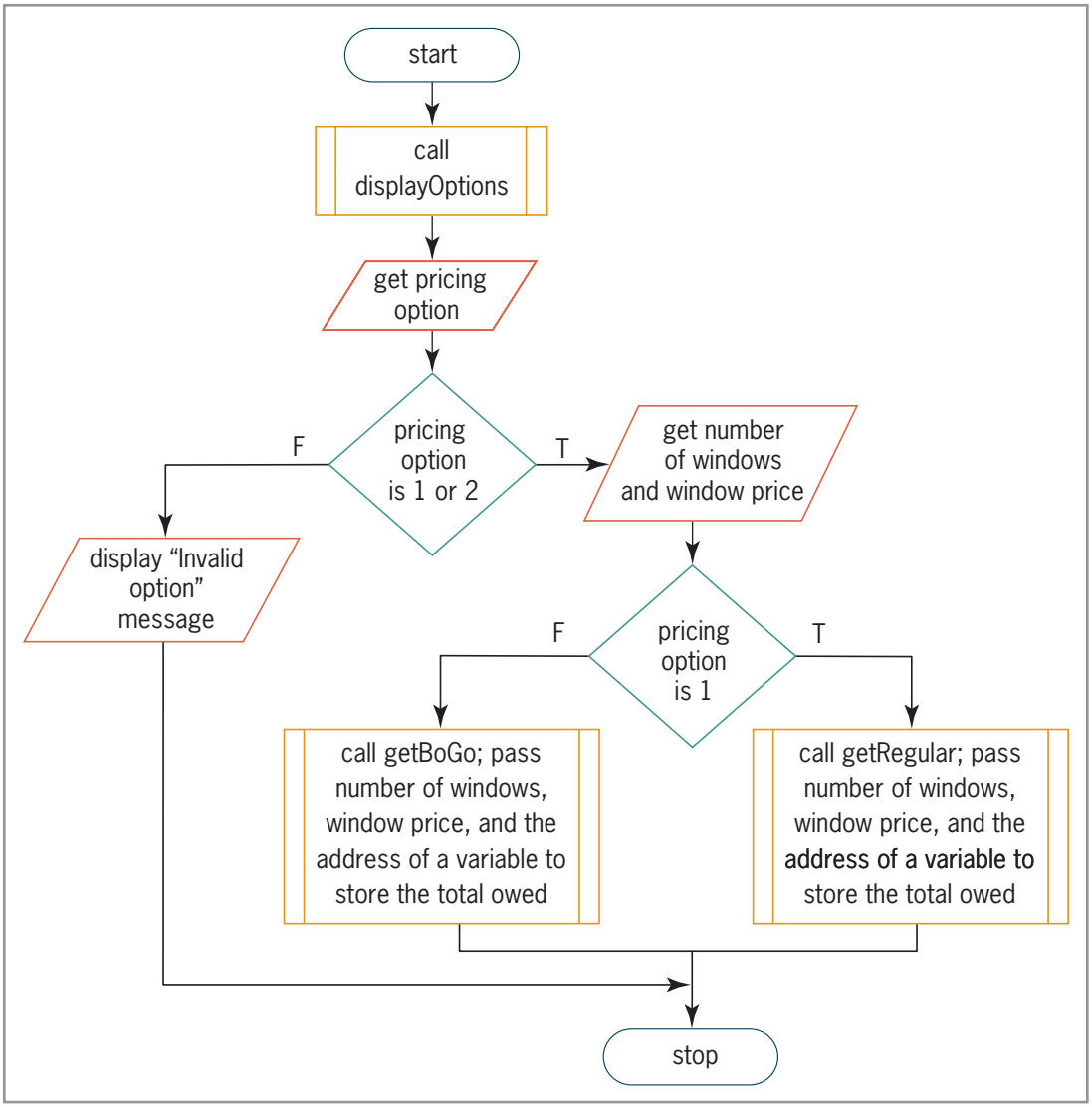


Figure 10-19 IPO charts for the functions in the Lab10-2 program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

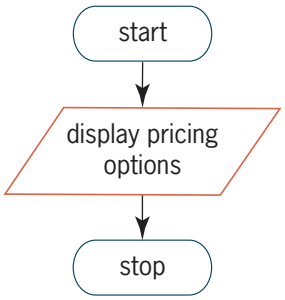
(continued)

displayOptions function

Input
none

Processing
Processing items: none

Algorithm:
1. display pricing options



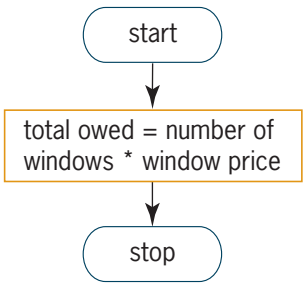
Output
pricing options

getRegular function

Input
number of windows
window price
address of a variable to store the total owed

Processing
Processing items: none

Algorithm:
1. calculate total owed = number of windows * window price



Output
total owed

getBoGo function

Input
number of windows
window price
address of a variable to store the total owed

Processing
Processing items: none

Algorithm:
1. calculate total owed = (number of windows / 2 + the remainder of number of windows / 2) * window price

Output
total owed

Figure 10-19 IPO charts for the functions in the Lab10-2 program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

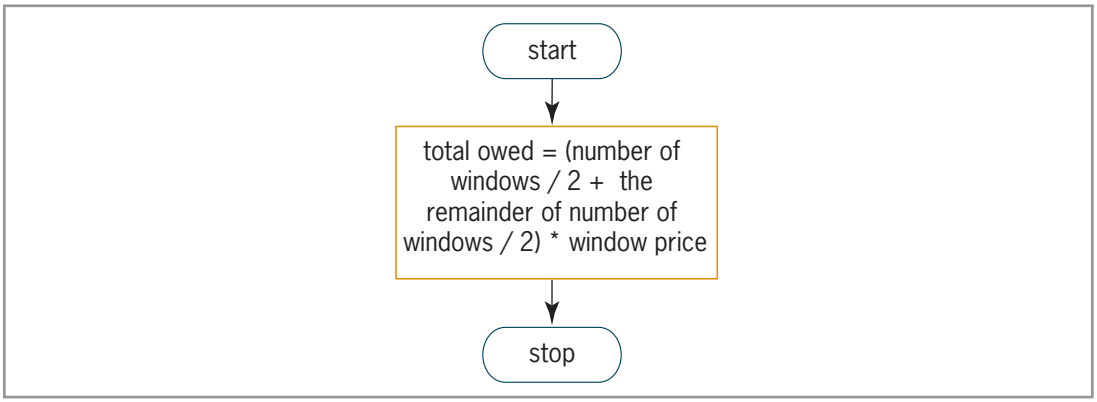


Figure 10-19 IPO charts for the functions in the Lab10-2 program

The third step in the problem-solving process is to desk-check the algorithm. You will desk-check the algorithms twice. For the first desk-check, you will use 1, 11, and 300 as the pricing option, number of windows, and window price, respectively. Using these values, the total owed should be $3300.00, as shown earlier in Figure 10-18. For the second desk-check, you will use 2, 15, and 200 as the pricing option, number of windows, and window price, respectively; the total owed should be $1600.00, as shown earlier in Figure 10-18. Figure 10-20 shows the completed desk-check table.

Note: The names in black indicate items that belong to the `main` function. The names in red indicate items that belong to the `getRegular` function. The names in blue indicate items that belong to the `getBoGo` function.

pricing option	number of windows	window price	~~total owed~~ ~~total owed~~ total owed
~~1~~	~~11~~	~~300.0~~	~~3300.0~~
2	15	200.0	1600.0

~~number of windows~~	~~window price~~
~~11~~	~~300.0~~

~~number of windows~~	~~window price~~
~~15~~	~~200.0~~

Figure 10-20 Completed desk-check table for Lab 10-2's algorithms

The fourth step in the problem-solving process is to code the algorithm into a program. The IPO chart information and C++ instructions for the program are shown in Figure 10-21.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

main function
IPO chart information
Input
pricing option
number of windows
window price

Processing
none

Output
total owed

Algorithm
1. call the displayOptions function to display the pricing options
2. get the pricing option
3. if (the pricing option is either 1 or 2)
    get the number of windows and the window price
    if (the pricing option is 1)
        call the getRegular function to calculate the total owed; pass the number of windows and the window price, as well as the address of a variable to store the total owed
    else
        call the getBoGo function to calculate the total owed; pass the number of windows and the window price, as well as the address of a variable to store the total owed
    end if
    display the total owed
else
    display "Invalid option" message
end if

displayOptions function
IPO chart information
Input
none

Processing
none

main function
C++ instructions

```
int option = 0;
int numOrdered = 0;
double winPrice = 0.0;

double totalOwed = 0.0;

displayOptions();

cout << "Pricing option? ";
cin >> option;
if (option == 1 || option == 2)
{
    cout << "Number of windows: ";
    cin >> numOrdered;
    cout << "Price per window: ";
    cin >> winPrice;
    if (option == 1)
        getRegular(numOrdered,
        winPrice, totalOwed);

    else

        getBoGo(numOrdered,
        winPrice, totalOwed);

    //end if

    cout << "Total owed-----> $"
    << totalOwed << endl << endl;
}
else
cout << "Invalid option" << endl;
//end if
```

displayOptions function
C++ instructions

Figure 10-21 IPO chart information and C++ instructions for Lab 10-2's program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Output
pricing options

Algorithm

1. display pricing options	`cout << "Pricing options:" << endl;`
	`cout << "1   Regular pricing" << endl;`
	`cout << "2   BOGO pricing" << endl;`

getRegular function **IPO chart information**	**getRegular function** **C++ instructions**
Input	
number of windows (formal parameter)	`int windows`
window price (formal parameter)	`double price`
address of a variable to store the total owed (formal parameter)	`double &total`
Processing	
none	
Output	
total owed	stored in the total formal parameter
Algorithm	
1. calculate total owed = number of windows * window price	`total = windows * price;`

getBoGo function **IPO chart information**	**getBoGo function** **C++ instructions**
Input	
number of windows (formal parameter)	`int windows`
window price (formal parameter)	`double price`
address of a variable to store the total owed (formal parameter)	`double &total`
Processing	
none	
Output	
total owed	stored in the total formal parameter
Algorithm	
1. calculate total owed = (number of windows / 2 + the remainder of number of windows / 2) * window price	`total = (windows / 2 +` `windows % 2) * price;`

Figure 10-21 IPO chart information and C++ instructions for Lab 10-2's program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The fifth step in the problem-solving process is to desk-check the program. Figure 10-22 shows the entire program, and Figure 10-23 shows the completed desk-check table.

```
 1 //Lab10-2.cpp - displays total owed
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <iomanip>
 6 using namespace std;
 7
 8 //function prototypes
 9 void displayOptions();
10 void getRegular(int windows, double price, double &total);
11 void getBoGo(int windows, double price, double &total);
12
13 int main()
14 {
15    int option = 0;
16    int numOrdered = 0;
17    double winPrice = 0.0;
18    double totalOwed = 0.0;
19
20    cout << fixed << setprecision(2);
21
22    displayOptions();
23    cout << "Pricing option? ";
24    cin >> option;
25
26    if (option == 1 || option == 2)
27    {
28        cout << "Number of windows: ";
29        cin >> numOrdered;
30        cout << "Price per window: ";
31        cin >> winPrice;
32
33        if (option == 1)
34            getRegular(numOrdered, winPrice, totalOwed);
35        else
36            getBoGo(numOrdered, winPrice, totalOwed);
37        //end if
38
39        cout << "Total owed-----> $" << totalOwed << endl << endl;
40    }
41    else
42        cout << "Invalid option" << endl;
43    //end if
44
45    return 0;
46 }   //end of main function
47
48 //*****function definitions*****
49 void displayOptions()
50 {
51    cout << "Pricing options:" << endl;
52    cout << "1   Regular pricing" << endl;
53    cout << "2   BOGO pricing" << endl;
54 }  //end displayOptions
```

Figure 10-22 Lab 10-2's program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
55
56 void getRegular(int windows, double price, double &total)
57 {
58    total = windows * price;
59 }  //end getRegular function
60
61 void getBoGo(int windows, double price, double &total)
62 {
63    total = (windows / 2 + windows % 2) * price;
64 }  //end getBoGo function
```

Figure 10-22 Lab 10-2's program

Note: The names in black indicate variables that belong to the `main` function. The names in red indicate variables that belong to the `getRegular` function. The names in blue indicate variables that belong to the `getBoGo` function.

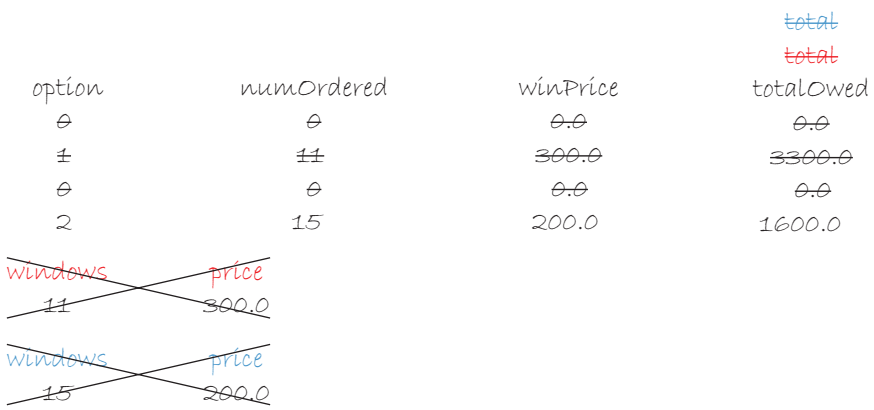


Figure 10-23 Completed desk-check table for Lab 10-2's program

The final step in the problem-solving process is to evaluate the program by entering its instructions into the computer and then using the computer to run (execute) it. While the program is running, you enter the same sample data used when desk-checking the program. If the program is not working correctly, you modify it until it works as intended.

DIRECTIONS

Follow the instructions for starting your C++ development tool. Depending on the development tool you are using, you may need to create a new project; if so, name the project Lab10-2 Project, and save it in the Cpp8\Chap10 folder. Enter the instructions shown in Figure 10-22 in a source file named Lab10-2.cpp. (Do not enter the line numbers.) Save the file in either the project folder or the Cpp8\Chap10 folder. Now, follow the appropriate instructions for running the Lab10-2.cpp file. Test the program using the same data used to desk-check the program. Also test it using your own sample data. If necessary, correct any bugs (errors) in the program.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

LAB 10-3 Modify

If necessary, create a new project named Lab 10-3 Project, and save it in the Cpp8\Chap10 folder. Enter (or copy) the Lab10-2.cpp instructions into a new source file named Lab10-3.cpp. Change Lab10-2.cpp in the first comment to Lab10-3.cpp. Change the `getRegular` and `getBoGo` functions to value-returning functions. Save and then run the program. Test the program appropriately.

LAB 10-4 What's Missing?

The program in this lab should display an employee's raise and new salary, given his or her current salary. Start your C++ development tool, and view the Lab10-4.cpp file, which is contained in either the Cpp8\Chap10\Lab10-4 Project folder or the Cpp8\Chap10 folder. (Depending on your C++ development tool, you may need to open Lab10-4's project/solution file first.) Put the C++ instructions in the proper order, and then determine the one or more missing instructions. Test the program appropriately.

LAB 10-5 Desk-Check

Desk-check the program shown in Figure 10-24. What will the program display?

```
//Lab10-5.cpp - displays a sum
//Created/revised by <your name> on <current date>

#include <iostream>
#include <cmath>
using namespace std;

//function prototype
void getSquare(int num, int &sqAnswer);
void getCube(int num, int &cubeAnswer);

int main()
{
    int sum = 0;

    for (int number = 1; number < 4; number += 1)
    {
        getSquare(number, sum);
        getCube(number, sum);
    }  //end for
    cout << "The sum is: " << sum << endl;
    return 0;
}   //end of main function
```

Figure 10-24 Code for Lab 10-5 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
//*****function definitions*****
void getSquare(int num, int &sqAnswer)
{
    sqAnswer += pow(num, 2);
}  //end getSquare function

void getCube(int num, int &cubeAnswer)
{
    cubeAnswer += pow(num, 3);
}  //end getCube function
```

Figure 10-24 Code for Lab 10-5

LAB 10-6 Debug

Follow the instructions for starting C++ and viewing the Lab10-6.cpp file, which is contained in either the Cpp8\Chap10\Lab10-6 Project folder or the Cpp8\Chap10 folder. (Depending on your C++ development tool, you may need to open Lab10-6's project/solution file first.) Run the program. Enter the following scores: 93, 90, 85, and 100. The program should display 368 as the total points and A as the grade, but it is not working correctly. Debug the program.

Chapter Summary

All functions fall into one of two categories: value-returning or void. A value-returning function returns precisely one value to the statement that called the function. A void function, on the other hand, does not return a value.

Like a value-returning function, a void function is composed of a function header and a function body. However, unlike a value-returning function, the function header for a void function begins with the keyword `void` rather than with a data type. Also unlike a value-returning function, the function body for a void function does not contain a `return` statement.

You call a void function by including its name and actual arguments (if any) in a statement.

Unlike a call to a value-returning function, a call to a void function appears as a statement by itself rather than as part of another statement. When the computer finishes processing a void function's code, it continues program execution with the statement immediately below the one that called the function.

Variables can be passed to functions either *by value* (the default) or *by reference*.

When you pass a variable *by value*, only a copy of the value stored inside of the variable is passed to the receiving function. The receiving function is not given access to a variable passed *by value*, so it cannot change the variable's contents.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

When you pass a variable *by value*, the computer uses the data type and name of the corresponding formal parameter to create a separate memory location in which to store a copy of the value.

When you pass a variable *by reference*, the variable's address in memory is passed to the receiving function, allowing the receiving function to change the variable's contents. Only variables can be passed *by reference*.

When you pass a variable *by reference*, the computer locates the variable in memory and then assigns the name of its corresponding formal parameter to the memory location. As a result, the variable will have two names: one assigned by the calling function and the other assigned by the receiving function.

To pass a variable *by reference* in a C++ program, you include the address-of operator (&) before the name of the corresponding formal parameter in the function header. If the function definition appears below the `main` function in the program, you must also include the address-of operator in the function prototype. The address-of operator tells the computer to pass the variable's address rather than its contents.

Key Terms

&—the address-of operator

Address-of operator—the ampersand; tells the computer to pass a variable's address in memory rather than its contents

Passing by reference—refers to the process of passing a variable's address to a function

Passing by value—refers to the process of passing a copy of a variable's value to a function

Void functions—functions that do not return a value after completing their assigned task

Review Questions

1. Which of the following is false?
   a. A void function does not contain a `return` statement.
   b. A void function call typically appears as its own statement in a C++ program.
   c. A void function cannot receive any items of information when it is called.
   d. A void function header begins with the keyword `void`.

2. Which of the following correctly calls a void function named `displayTotal`, passing it an `int` variable named `total`?
   a. `cout << displayTotal(int total);`
   b. `cout << displayTotal(total);`
   c. `displayTotal(int total);`
   d. `displayTotal(total);`

3. A void function named `getEndBal` is passed the values stored in two `int` variables. Which of the following function prototypes is correct for this function?
   a. `void getEndBal(int, int);`
   b. `void getEndBal(int, int)`
   c. `void getEndBal(int &, int &);`
   d. `int getEndBal(void);`

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

4. A void function named `getInventory` is passed four `int` variables named `beginInv`, `sales`, `purchases`, and `endInv`. The function's task is to calculate the ending inventory, using the beginning inventory, sales, and purchase amounts passed to the function. The function should store the result in the `endInv` variable. Which of the following function headers is correct?

   a. `void getInventory(int b, int s, int p, int &e)`
   b. `void getInventory(int b, int s, int p, int e)`
   c. `void getInventory(int &b, int &s, int &p, int e)`
   d. `void getInventory(&int b, &int s, &int p, &int e)`

5. Which of the following statements calls the `getInventory` function described in Review Question 4?

   a. `getInventory(int, int, int, int);`
   b. `getInventory(beginInv, sales, purchases, &endInv);`
   c. `getInventory(beginInv, sales, purchases, endInv);`
   d. `getInventory(int beginInv, int sales, int purchases, int &endInv);`

6. To determine whether an item is being passed *by value* or *by reference*, you must examine either the _____ or the _____.

   a. function call, function header
   b. function call, function prototype
   c. function header, function prototype
   d. function header, function body

7. Which of the following calls a void function named `displayName`, passing it no actual arguments?

   a. `call displayName();`
   b. `displayName;`
   c. `displayName()`
   d. `displayName();`

8. Which of the following is a correct function prototype for a void function that requires no formal parameters? The function's name is `displayName`.

   a. `displayName();`
   b. `void displayName;`
   c. `void displayName();`
   d. `void displayName(none);`

9. If the function definitions section is located below the `main` function in a program, the program will have one function prototype for each program-defined function.

   a. True
   b. False

10. Which of the following is false?

    a. When you pass a variable *by reference*, the receiving function can change the variable's contents.
    b. When you pass a variable *by value*, the receiving function creates a local variable that it uses to store the value.
    c. Unless specified otherwise, all variables in C++ are passed *by value*.
    d. To pass a variable *by reference* in C++, you place an ampersand (&) before the variable's name in the statement that calls the function.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

11. A program contains a void function named `getNewPrice`. The function receives two `double` variables named `oldPrice` and `newPrice`. The function multiplies the contents of the `oldPrice` variable by 1.1 and then stores the result in the `newPrice` variable. Which of the following is the appropriate function prototype for this function?

    a. `void getNewPrice(double, double);`
    b. `void getNewPrice(double &, double);`
    c. `void getNewPrice(double, double &);`
    d. `void getNewPrice(double &, double &);`

12. Which of the following can be used to call the `getNewPrice` function described in Review Question 11?

    a. `getNewPrice(double oldPrice, double newPrice);`
    b. `getNewPrice(&oldPrice, newPrice);`
    c. `getNewPrice(oldPrice, &newPrice);`
    d. `getNewPrice(oldPrice, newPrice);`

13. Which of the following is false?

    a. The names of the formal parameters in the function header must be identical to the names of the actual arguments in the function call.
    b. When listing the formal parameters in a function header, you include each parameter's data type and name.
    c. The formal parameters should be the same data type as the actual arguments.
    d. If a function call passes an `int` variable first and a `char` variable second, the receiving function should receive an `int` variable followed by a `char` variable.

14. When a variable is passed *by reference*, the computer assigns the name of its corresponding formal parameter to the variable's location in memory.

    a. True b. False

Exercises

Pencil and Paper

TRY THIS

1. Write the C++ code for a function that receives an integer, a `double` number, and the address of a `double` variable from the calling statement. The function should multiply the integer by the `double` number and then store the result in the `double` variable. Name the function `getProduct`. Name the formal parameters `intNum`, `dblNum`, and `answer`. (The answers to TRY THIS Exercises are located at the end of the chapter.)

TRY THIS

2. Write the function prototype for the `getProduct` function from Pencil and Paper Exercise 1. Also write the statement to call the function. Name the actual arguments `firstNum`, `secondNum`, and `product`. (The answers to TRY THIS Exercises are located at the end of the chapter.)

MODIFY THIS

3. Rewrite the code from Pencil and Paper Exercises 1 and 2 so that the `getProduct` function receives two integers, a `double` number, and the address of a `double` variable from the calling statement. The function should add both integers together, multiply the sum by the `double` number, and store the result in the `double` variable. Name the formal parameters `intNum1`, `intNum2`, `dblNum`, and `answer`. Name the actual arguments `firstNum`, `secondNum`, `thirdNum`, and `product`.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

4. Write the C++ code for a void function that receives three `double` variables: the first two *by value* and the last one *by reference*. Name the formal parameters `n1`, `n2`, and `answer`. The function should divide the `n1` variable by the `n2` variable and then store the result in the `answer` variable. Name the function `calcQuotient`. Also write an appropriate function prototype for the `calcQuotient` function. In addition, write a statement that invokes the `calcQuotient` function, passing it the `num1`, `num2`, and `quotient` variables. INTRODUCTORY

5. Write the C++ code for a void function that receives four `int` variables: the first two *by value* and the last two *by reference*. Name the formal parameters `n1`, `n2`, `sum`, and `diff`. The function should calculate the sum of the two variables passed *by value* and then store the result in the first variable passed *by reference*. It should also calculate the difference between the two variables passed *by value* and then store the result in the second variable passed *by reference*. When calculating the difference, always subtract the smaller number from the larger number. Name the function `calcSumAndDiff`. Also write an appropriate function prototype for the `calcSumAndDiff` function. In addition, write a statement that invokes the `calcSumAndDiff` function, passing it the `num1`, `num2`, `numSum`, and `numDiff` variables. INTERMEDIATE

6. Write the C++ code for a function that receives four items of information: three *by value* and one *by reference*. Each item has the `double` data type. Name the formal parameters `num1`, `num2`, `num3`, and `avg`. The function should calculate the average of the three numbers and then assign the result to the `avg` variable. Name the function `calcAverage`. Also write an appropriate function prototype for the function. In addition, write a statement that invokes the function, passing it the following actual arguments: `janAvg`, `febAvg`, `marAvg`, and `quarterAvg`. INTERMEDIATE

7. Desk-check the code shown in Figure 10-25. Show the desk-check table after the first four statements in the `main` function are processed. Also show it after the statement in the `calcEnd` function is processed. Finally, show the desk-check table after the `calcEnd` function ends. INTERMEDIATE

```
void calcEnd(int beg, int pur, int sale, int &ending);

int main()
{
    int begVal = 950;
    int purchase = 400;
    int sale = 700;
    int endVal = 0;

    calcEnd(begVal, purchase, sale, endVal);

    cout << "Ending value: " << endVal << endl;
    return 0;
}  //end of main function

void calcEnd(int beg, int pur, int sale, int &ending)
{
    ending = beg + pur - sale;
}  //end of calcEnd function
```

Figure 10-25

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

SWAT THE BUGS

8. A program's `main` function declares three `double` variables named `sales`, `taxRate`, and `salesTax`. It also contains the following function call: `calcSalesTax (sales, taxRate, salesTax);`. The `calcSalesTax` function is responsible for calculating the sales tax. Its function header looks like this: `void calcSalesTax (double sold, double rate, double tax)`. Correct the function header.

Computer

TRY THIS

9. In this exercise, you will experiment with passing variables *by value* and *by reference*. (The answers to TRY THIS Exercises are located at the end of the chapter.)

   a. Follow the instructions for starting C++ and viewing the TryThis9.cpp file, which is contained in either the Cpp8\Chap10\TryThis9 Project folder or the Cpp8\Chap10 folder. (Depending on your C++ development tool, you may need to open this exercise's project/solution file first.)

   b. Notice that the `main` function passes the `age` variable *by value* to the `getAge` function. Run the program. When prompted to enter your age, type your age and press Enter. The message that appears should contain your age; however, it contains the number 0 instead. This is because the `age` variable is passed *by value* to the `getAge` function.

   c. Modify the program so that it passes the `age` variable *by reference* to the `getAge` function. Save and then run the program. When prompted to enter your age, type your age and press Enter. This time, the message contains your age.

TRY THIS

10. In this exercise, you will modify the program from Lab 9-1 in Chapter 9. Follow the instructions for starting C++ and viewing the TryThis10.cpp file, which is contained in either the Cpp8\Chap10\TryThis10 Project folder or the Cpp8\Chap10 folder. (Depending on your C++ development tool, you may need to open this exercise's project/solution file first.) Modify the program to use void functions to calculate the area and the diameter. Test the application appropriately. (The answers to TRY THIS Exercises are located at the end of the chapter.)

TRY THIS

11. If necessary, create a new project named TryThis11 Project, and save it in the Cpp8\Chap10 folder. Code the IPO charts shown in Figure 10-26. Enter your C++ instructions into a source file named TryThis11.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Display the Celsius temperature in fixed-point notation with no decimal places. Test the program appropriately. (The answers to TRY THIS Exercises are located at the end of the chapter.)

`main` function

Input	Processing	Output
Fahrenheit temperature	Processing items: none Algorithm: 1. enter Fahrenheit temperature 2. call calcCelsius to calculate the Celsius temperature; pass the Fahrenheit temperature and the address of a variable in which to store the Celsius temperature 3. display the Celsius temperature	Celsius temperature

Figure 10-26 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

calcCelsius function

Input	Processing	Output
Fahrenheit temperature address of a variable in which to store the Celsius temperature	Processing items: none Algorithm: 1. Celsius temperature = 5.0 / 9.0 * (Fahrenheit temperature - 32.0)	Celsius temperature

Figure 10-26

12. In this exercise, you will modify the program from TRY THIS Exercise 11. If necessary, create a new project named ModifyThis12 Project, and save it in the Cpp8\Chap10 folder. Copy the instructions from the TryThis11.cpp file into a source file named ModifyThis12.cpp. (Alternatively, you can enter the instructions shown later in Figure 10-32 into the ModifyThis12.cpp file.) Change the filename in the first comment to ModifyThis12.cpp. Add a void function named `calcFahrenheit` to the program. The program should now allow the user to convert the temperature he or she entered to either Celsius or Fahrenheit. Make the necessary modifications to the `main` function. Test the program appropriately. MODIFY THIS

13. In this exercise, you will modify the program from Lab 9-2 in Chapter 9. If necessary, create a new project named ModifyThis13 Project, and save it in the Cpp8\Chap10 folder. Copy the instructions from the Lab9-2.cpp file (which is contained in either the Cpp8\Chap09\Lab9-2 Project folder or the Cpp8\Chap09 folder) into a source file named ModifyThis13.cpp. (Alternatively, you can enter the instructions from Figure 9-34 into the ModifyThis13.cpp file.) Change the filename in the first comment to ModifyThis13.cpp. Change the `getPayment` function to a void function. Test the program appropriately. MODIFY THIS

14. In this exercise, you will modify the program from Lab10-1. Follow the instructions for starting C++ and viewing the ModifyThis14.cpp file, which is contained in either the Cpp8\Chap10\ModifyThis14 Project folder or the Cpp8\Chap10 folder. (Depending on your C++ development tool, you may need to open this exercise's project/solution file first.) Modify the program to allow the user to display a circle's circumference, given its radius. Use a void function named `getCircumference`. Test the program appropriately. MODIFY THIS

15. In this exercise, you will modify the guessing game program from Figure 9-11 in Chapter 9. Follow the instructions for starting C++ and viewing the Introductory15.cpp file, which is contained in either the Cpp8\Chap10\Introductory15 Project folder or the Cpp8\Chap10 folder. (Depending on your C++ development tool, you may need to open this exercise's project/solution file first.) Modify the program so that it uses a void function to determine the random number. The program should ask the user for both the minimum and maximum random numbers that the void function should generate. The function call should pass that information to the void function. Test the application appropriately. INTRODUCTORY

16. In this exercise, you will create a program that displays the gross pay for one or more employees. If necessary, create a new project named Introductory16 Project, and save it in the Cpp8\Chap10 folder. The program should allow the user to enter the number of hours the employee worked and his or her hourly pay rate. Use a negative sentinel value INTRODUCTORY

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

to stop the program. Employees are paid at their regular pay rate for hours worked from 1 through 37. They are paid time and a half for the hours worked from 38 through 50, and double-time for the hours worked over 50. Use a void function to calculate and return the employee's overtime pay, if applicable. Enter your C++ instructions into a source file named Introductory16.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Test the application appropriately. (Hint: If an employee earns $10 per hour and works 37 hours, the gross pay is $370.00. If he or she works 38 hours, the gross pay is $385.00. If he or she works 51 hours, the gross pay is $585.00.)

INTRODUCTORY

17. In this exercise, you will create a program that displays a measurement in either inches or centimeters. If necessary, create a new project named Introductory17 Project, and save it in the Cpp8\Chap10 folder. The program should allow the user the choice of converting a measurement from inches to centimeters or vice versa. Use two void functions: one for each different conversion type. Enter your C++ instructions into a source file named Introductory17.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Test the application appropriately.

INTERMEDIATE

18. In this exercise, you will modify the program that you created in Chapter 6's Lab 6-2. If necessary, create a new project named Intermediate18 Project, and save it in the Cpp8\Chap10 folder. Copy the instructions from the Lab6-2.cpp file (which is contained in either the Cpp8\Chap06\Lab6-2 Project folder or the Cpp8\Chap06 folder) into a source file named Intermediate18.cpp. (Alternatively, you can enter the instructions from Figure 6-29 into the Intermediate18.cpp file.) Change the filename in the first comment. Modify the program so that it uses two void functions: one to calculate the price of a medium pizza and the other to calculate the price of a large pizza. In addition to the $2 coupon on the purchase of a large pizza, Sophia is now e-mailing customers a $1 coupon on the purchase of a medium pizza. Test the program appropriately.

INTERMEDIATE

19. In this exercise, you will modify the savings account program from Figure 9-18 in Chapter 9. Follow the instructions for starting C++ and viewing the Intermediate19.cpp file, which is contained in either the Cpp8\Chap10\Intermediate19 Project folder or the Cpp8\Chap10 folder. (Depending on your C++ development tool, you may need to open this exercise's project/solution file first.) Modify the program to allow the user to enter the minimum and maximum interest rates, as shown in Figure 10-27. Also change the `getBalance` function to a void function. Test the program appropriately.

```
Deposit: 1000
Minimum rate (in decimal form): 0.02
Maximum rate (in decimal form): 0.04
Rate 2%:
     Year 1: $1020.00
     Year 2: $1040.40
     Year 3: $1061.21
Rate 3%:
     Year 1: $1030.00
     Year 2: $1060.90
     Year 3: $1092.73
Rate 4%:
     Year 1: $1040.00
     Year 2: $1081.60
     Year 3: $1124.86
```

Figure 10-27

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

20. In this exercise, you will modify the program from Lab10-2. If necessary, create a new project named Intermediate20 Project, and save it in the Cpp8\Chap10 folder. Copy the instructions from the Lab10-2.cpp file into a source file named Intermediate20.cpp. (Alternatively, you can enter the instructions from Figure 10-22 into the Intermediate20.cpp file.) Change the filename in the first comment to Intermediate20.cpp. Patterson Windows is now offering another pricing option: BOGOHO (buy one, get one half-off). Modify the program to allow for this new pricing option. Use a void function named `getBoGoHo`. Test the program appropriately. INTERMEDIATE

21. A local department store wants a program that displays the number of reward points a customer earns each month. The reward points are based on the customer's membership type and total monthly purchase amount, as shown in Figure 10-28. The program should use a separate void function for each membership type. If necessary, create a new project named Advanced21 Project, and save it in the Cpp8\Chap10 folder. Enter your C++ instructions into a source file named Advanced21.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Display the reward points in fixed-point notation with no decimal places. Test the program appropriately. ADVANCED

Membership type	Total monthly purchase ($)	Reward points
Standard	Less than 75	5% of the total monthly purchase
	75–149.99	7.5% of the total monthly purchase
	150 and over	10% of the total monthly purchase
Plus	Less than 150	6% of the total monthly purchase
	150 and over	13% of the total monthly purchase
Premium	Less than 200	4% of the total monthly purchase
	200 and over	15% of the total monthly purchase

Figure 10-28

22. An online retailer wants a program that displays the total amount a customer owes, including shipping. The user will enter the total amount due before shipping. The amount to charge for shipping is based on the customer's membership status, which can be either Standard or Premium. The appropriate shipping charges are shown in Figure 10-29. The program should use two void functions: one to determine the shipping charge for a Standard member and the other to determine the shipping charge for a Premium member. If necessary, create a new project named Advanced22 Project, and save it in the Cpp8\Chap10 folder. Enter your C++ instructions into a source file named Advanced22.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Display the total due in fixed-point notation with two decimal places. Test the program appropriately. ADVANCED

Membership type	Total due before shipping ($)	Shipping ($)
Standard	0–100	12.99
	Over 100	4.99
Premium	0–49.99	4.99
	Over 49.99	0

Figure 10-29

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

ADVANCED

23. In this exercise, you will create a program that displays the amount of a cable bill. The amount is based on the type of customer, as shown in Figure 10-30. For a residential customer, the user will need to enter the number of premium channels only. For a business customer, the user will need to enter the number of connections and the number of premium channels. Use a separate void function for each customer type. If necessary, create a new project named Advanced23 Project, and save it in the Cpp8\Chap10 folder. Enter your C++ instructions into a source file named Advanced23.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Test the program appropriately.

Residential customers:	
Processing fee:	$4.50
Basic service fee:	$30
Premium channels:	$5 per channel
Business customers:	
Processing fee:	$16.50
Basic service fee:	$80 for the first 5 connections; $4 for each additional connection
Premium channels:	$50 per channel for any number of connections

Figure 10-30

SWAT THE BUGS

24. Follow the instructions for starting C++ and viewing the SwatTheBugs24.cpp file, which is contained in either the Cpp8\Chap10\SwatTheBugs24 Project folder or the Cpp8\Chap10 folder. (Depending on your C++ development tool, you may need to open this exercise's project/solution file first.) The program should calculate and display a bonus amount, but it is not working correctly. Run the program. Enter 1000 and 0.1 as the sales and bonus rate, respectively. Debug the program.

SWAT THE BUGS

25. Follow the instructions for starting C++ and viewing the SwatTheBugs25.cpp file, which is contained in either the Cpp8\Chap10\SwatTheBugs25 Project folder or the Cpp8\Chap10 folder. (Depending on your C++ development tool, you may need to open this exercise's project/solution file first.) The program should calculate and display the sum of two numbers, but it is not working correctly. Debug the program.

Answers to TRY THIS Exercises

Pencil and Paper

1.
```
void getProduct(int intNum, double dblNum, double &answer)
{
   answer = intNum * dblNum;
} //end of getProduct function
```

2. Function prototype: `void getProduct(int intNum, double dblNum, double &answer);` [or `void getProduct(int, double, double &);`]

   Function call: `getProduct(firstNum, secondNum, product);`

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Computer

9. To modify the program, change the function prototype to `void getAge (int &years);` and change the function header to `void getAge(int &years)`.
10. See Figure 10-31. The modifications to the program are shaded in the figure.

```
//TryThis10.cpp - circle calculations
//Created/revised by <your name> on <current date>

#include <iostream>
#include <cmath>
using namespace std;

//function prototypes
void getArea(double rad, double &area);
void getDiameter(double rad, double &diameter);

int main()
{
    int choice = 0;
    double radius = 0.0;
    double answer = 0.0;

    cout << "1   Circle area" << endl;
    cout << "2   Circle diameter" << endl;
    cout << "Enter your choice (1 or 2): ";
    cin >> choice;

    if (choice < 1 || choice > 2)
        cout << "Invalid choice" << endl;
    else
    {
        cout << "Radius: ";
        cin >> radius;
        if (choice == 1)
        {
            getArea(radius, answer);
            cout << "Area: " << answer;
        }
        else
        {
            getDiameter(radius, answer);
            cout << "Diameter: " << answer;
        }  //end if
        cout << endl;
    }  //end if
    return 0;
}   //end of main function

    //*****function definitions*****
void getArea(double rad, double &area)
```

Figure 10-31 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
{
    const double PI = 3.141593;
    area = PI * pow(rad, 2);
}  //end getArea function

void getDiameter(double rad, double &diameter)
{
    diameter = 2 * rad;
}  //end getDiameter function
```

the variable declaration and `return` statements were removed

Figure 10-31

11. See Figure 10-32.

```
//TryThis11.cpp - converts Fahrenheit to Celsius
//Created/revised by <your name> on <current date>

#include <iostream>
#include <iomanip>
using namespace std;

//function prototype
void calcCelsius(double tempF, double &tempC);

int main()
{
    double fahrenheit = 0.0;
    double celsius = 0.0;

    cout << "Enter Fahrenheit temperature: ";
    cin >> fahrenheit;
    calcCelsius(fahrenheit, celsius);

    cout << fixed << setprecision(0);
    cout << "Celsius temperature: " << celsius << endl;
    return 0;
}    //end of main function

//*****function definitions*****
void calcCelsius(double tempF, double &tempC)
{
    tempC = 5.0 / 9.0 * (tempF - 32.0);
}    //end of calcCelsius function
```

Figure 10-32

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

CHAPTER 11

One-Dimensional Arrays

After studying Chapter 11, you should be able to:

- Declare and initialize a one-dimensional array
- Enter data into a one-dimensional array
- Display the contents of a one-dimensional array
- Pass a one-dimensional array to a function
- Calculate the total and average of the values in a one-dimensional array
- Search a one-dimensional array
- Access an individual element in a one-dimensional array
- Find the highest value in a one-dimensional array
- Use parallel one-dimensional arrays
- Explain the bubble sort algorithm

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Arrays

Ch11-Chapter Preview

All of the variables you have used so far have been simple variables. A **simple variable**, also called a **scalar variable**, is one that is unrelated to any other variable in memory. Some programs, however, will require the use of variables that *are* related to each other. In those cases, it is easier and more efficient to treat the related variables as a group.

You already are familiar with the concept of grouping. The clothes in your closet are probably separated into groups, such as coats, sweaters, shirts, and so on. Grouping your clothes in this manner allows you to easily locate your favorite sweater because you only need to look through the sweater group rather than through the entire closet. You may also have the songs on your MP3 player grouped by either music type or artist. If the songs are grouped by artist, it will take only a few seconds to find all of your Katy Perry songs and, depending on the number of Katy Perry songs you own, only a short time after that to locate a particular song.

When you group together related variables that have the same data type, the group is referred to as an array of variables or, more simply, an **array**. You might use an array of 50 variables to store the population of each U.S. state. Or, you might use an array of four variables to store the sales made in each of your company's four sales regions.

Storing data in an array increases the efficiency of a program because data can be both stored in and retrieved from the computer's internal memory much faster than it can be written to and read from a file on a disk. In addition, after the data is entered into an array, which typically is done at the beginning of a program, the program can use the data as many times as necessary without having to enter the data again. Your company's sales program, for example, can use the sales amounts stored in an array to calculate the total company sales and the percentage that each region contributed to the total sales. It can also use the sales amounts in the array either to calculate the average sales amount or to simply display the sales made in a specific region.

It takes longer for the computer to access the information stored in a disk file because the computer must wait for the disk drive to first locate the needed information and then read the information into internal memory.

As you will learn in this chapter, the variables in an array can be used just like any other variables. You can assign values to them, use them in calculations, display their contents, and so on.

The most commonly used arrays in business applications are one-dimensional and two-dimensional. You will learn about one-dimensional arrays in this chapter. Two-dimensional arrays are covered in Chapter 12. Arrays having more than two dimensions are used mostly in scientific and engineering programs and are beyond the scope of this book.

As is true of functions, which you learned about in Chapters 9 and 10, arrays are one of the more challenging topics for beginning programmers. Therefore, it is important for you to read and study each section in this chapter thoroughly before moving on to the next section. For example, be sure you understand the concept of one-dimensional arrays before you continue to the sections pertaining to parallel arrays and the bubble sort. If you still feel overwhelmed by the end of the chapter, try reading the chapter again, paying particular attention to the examples and programs shown in the figures.

One-Dimensional Arrays

A subscript is also called an index.

The variables in an array are stored in consecutive locations in the computer's internal memory. Each variable in an array has the same name and data type. You distinguish one variable in a **one-dimensional array** from another variable in the same array using a unique number. The unique number, which is always an integer, is called a subscript. The **subscript** indicates the variable's position in the array and is assigned by the computer when the array is created in internal memory. The first variable in a one-dimensional array is assigned a subscript of 0, the second a subscript of 1, and so on.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

You refer to each variable in an array by the array's name and the variable's subscript, which is specified in a set of square brackets immediately following the array name. Figure 11-1 illustrates a one-dimensional array named `scientists`. The array contains three variables that store the names Marie Curie, Charles Darwin, and Albert Einstein. You use `scientists[0]`—read "`scientists` sub zero"—to refer to the first variable in the array. You use `scientists[1]` to refer to the second variable in the array, and you use `scientists[2]` to refer to the third (and last) variable in the array. The last subscript in an array is always one number less than the total number of variables in the array. This is because array subscripts in C++ (and in most other programming languages) start at 0.

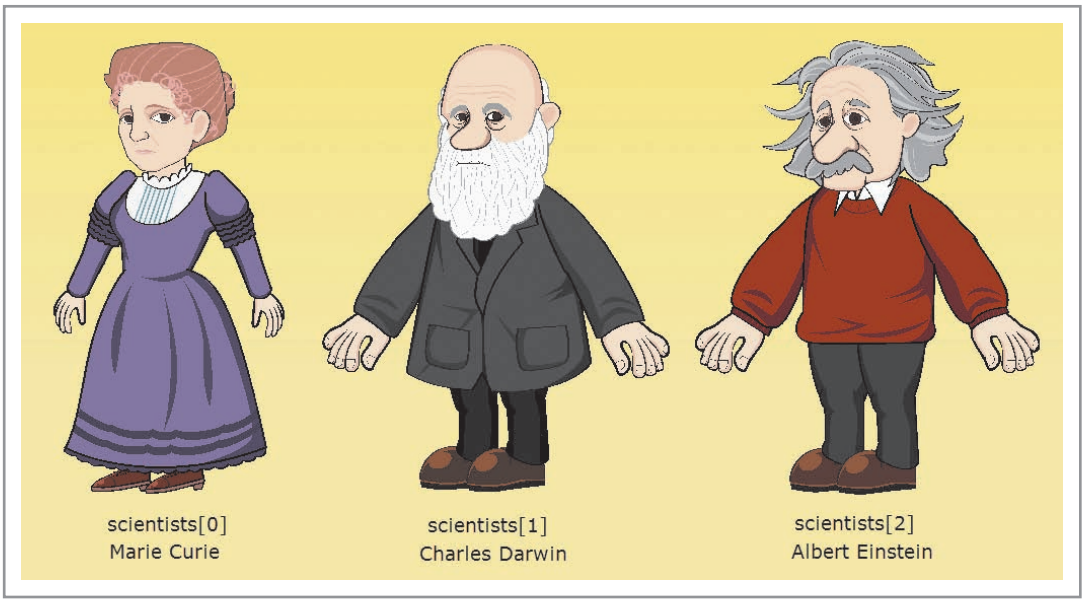


Figure 11-1 Illustration of the naming convention for the one-dimensional `scientists` array
Image by Diane Zak; created with Reallusion CrazyTalk Animator

Declaring and Initializing a One-Dimensional Array

You must declare (create) the array before you can use it in a program. You should also initialize each variable in the array to ensure it will not contain garbage when the program is run. As you learned in Chapter 3, the garbage found in uninitialized variables is the remains of what was last stored at the memory location that the variable now occupies.

Figure 11-2 shows the syntax for declaring and initializing a one-dimensional array and includes examples of using the syntax. In the syntax, *dataType* is the type of data that each of the array variables, referred to as **elements**, will store. *ArrayName* is the name of the array. You use the same rules for naming an array as you do for naming a variable. *NumberOfElements* is an integer that specifies the size of the array—in other words, the number of elements. To declare an array that contains 10 elements, you enter the number 10 as the *numberOfElements*. Notice that the *numberOfElements* is enclosed in square brackets (`[]`).

You can initialize the array elements at the same time you declare the array. You do this by entering one or more values, separated by commas, in the *initialValues* section of the syntax. You enclose the *initialValues* section in braces (`{}`), as shown in Figure 11-2. Assigning initial values to an array is often referred to as **populating the array**. The values used to populate an array should have the same data type as the array variables. If the data types are not the same, the computer will either promote or demote the values to fit the array variables. However,

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

recall from Chapter 3 that the implicit demotion of values can adversely affect a program's output. Therefore, you should always be sure to populate an array using values that have the appropriate data type.

HOW TO Declare and Initialize a One-Dimensional Array

Syntax
dataType arrayName[*numberOfElements*] = {*initialValues*};

Example 1
```
char letters[3] = {'A', 'B', 'C'};
```
The statement declares and initializes a three-element `char` array named `letters`.

Example 2
```
double sales[4] = {0.0, 0.0, 0.0, 0.0};
          or
double sales[4] = {0.0};
```
Both statements declare and initialize a four-element `double` array named `sales`; each element is initialized to 0.0.

Example 3
```
int numbers[6] = {12, 0, 0, 0, 0, 0};
          or
int numbers[6] = {12};
```
Both statements declare and initialize a six-element `int` array named `numbers`. The first element is initialized to 12, and the other elements are initialized to 0.

Note: The = {*initialValues*} portion of the syntax is optional. Typically, optional items are enclosed in square brackets when shown in the syntax. The square brackets were omitted here so as not to confuse them with the square brackets that are required.

Figure 11-2 How to declare and initialize a one-dimensional array

The declaration statement in Example 1 in Figure 11-2 creates a three-element `char` array named `letters`. It initializes the `letters[0]` element to A, the `letters[1]` element to B, and the `letters[2]` element to C.

Many C++ compilers initialize `char`, `string`, and `bool` array elements to a space, the empty string, and the keyword `false`, respectively.

Example 2 shows two statements you can use to declare a four-element `double` array, initializing each element to the `double` number 0.0. The statement `double sales[4] = {0.0, 0.0, 0.0, 0.0};` provides an initial value for each of the four array elements, whereas the statement `double sales[4] = {0.0};` provides only one value. When the array declaration statement does not provide an initial value for each of the elements in a numeric array, many C++ compilers initialize the uninitialized array elements to either 0.0 or 0 (depending on the data type of the array). However, this is done only when you provide at least one value in the *initialValues* section. If you omit the *initialValues* section from the declaration statement—for example, if you use the statement `double sales[4];` to declare the array—the compiler does not automatically initialize the elements, so the array elements may contain garbage.

Example 3 in Figure 11-2 shows two statements you can use to declare a six-element `int` array named `numbers`. The statement `int numbers[6] = {12, 0, 0, 0, 0, 0};` initializes the

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

first array element to the integer 12 and initializes the remaining elements to the integer 0. The same result can be accomplished using the `int numbers[6] = {12};` statement shown in the example.

If you inadvertently provide more values in the *initialValues* section than the number of array elements, most C++ compilers will display the error message "too many initializers" when you attempt to compile the program. However, not all C++ compilers display a message when this error occurs. Rather, some compilers store the extra values in memory locations adjacent to but not reserved for the array. Therefore, you should always be careful to provide no more than the appropriate number of *initialValues*.

Entering Data into a One-Dimensional Array

You can use an assignment statement to enter data into an array element, as shown in the syntax and examples in Figure 11-3. In the syntax, *arrayName*[*subscript*] is the name and subscript of the array variable to which you want the *expression* (data) assigned. The *expression* can include any combination of constants, variables, and operators. The data type of the *expression* must match the data type of the array variable. If both data types do not match, the computer will perform an implicit type conversion, which could result in incorrect output.

HOW TO Use an Assignment Statement to Assign Data to a One-Dimensional Array

Syntax

arrayName[*subscript*] = *expression*;

Example 1

```
letters[1] = 'Y';
```

The assignment statement assigns the letter Y to the second element in the `letters` array.

Example 2

```
int subscript = 0;
while (subscript < 4)
{
    sales[subscript] = 0.0;
    subscript += 1;
}   //end while
```

The `while` loop assigns the `double` number 0.0 to each of the four elements in the `sales` array. The loop provides another means of initializing the array.

Example 3

```
for (int x = 1; x <= 6; x += 1)
    numbers[x - 1] = pow(x, 2);
//end for
```

The `for` loop assigns the squares of the numbers from 1 through 6 to the six-element `numbers` array.

The loops in Examples 2 through 4 provide a convenient way to access each element in a one-dimensional array.

Figure 11-3 How to use an assignment statement to assign data to a one-dimensional array *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Example 4

```
int increase = 0;
cout << "Enter increase amount: ";
cin >> increase;
for (int x = 0; x < 6; x += 1)
    numbers[x] += increase;
//end for
```

The `for` loop assigns a sum to each element in the six-element `numbers` array. Each sum is calculated by adding the value stored in the current element to the value stored in the `increase` variable.

Figure 11-3 How to use an assignment statement to assign data to a one-dimensional array

The examples included in Figure 11-3 show various ways of assigning data to the arrays declared earlier in Figure 11-2. The assignment statement in Example 1 assigns the letter Y to the second element in the `letters` array, replacing the letter B that was stored in the element when the array was declared. The `while` loop in Example 2 assigns the `double` number 0.0 to each of the four elements in the `sales` array and provides another means of initializing the array.

The `for` loop in Example 3 assigns the squares of the numbers from 1 through 6 to the six-element `numbers` array, replacing the array's initial values. The square of the number 1 is assigned to the `numbers[0]` element. The square of the number 2 is assigned to the `numbers[1]` element, and so on. Notice that the `x` variable keeps track of the six numbers to be squared. Also notice that in order to assign the square of each number to its appropriate element in the `numbers` array, the number 1 must be subtracted from the value stored in the `x` variable. This is because the `x` variable's values go from 1 through 6, whereas the corresponding array subscripts go from 0 through 5.

The `for` loop shown in Example 4 in Figure 11-3 updates the contents of each element in the `numbers` array. It does this by adding the value contained in the `increase` variable to the value contained in the current array element and then assigning the sum to the element.

You can also use the extraction operator to store data in an array element, as shown in the syntax and examples in Figure 11-4. (The arrays in Figure 11-4 were declared earlier in Figure 11-2.) The `cin` statement in Example 1 stores the user's entry in the first element in the `letters` array, replacing the element's existing data. Example 2 contains a `for` loop that repeats its instructions four times: once for each element in the `sales` array. The loop instructions prompt the user to enter a sales amount and then store the user's response in the current element. Example 3 contains a `while` loop that repeats its instructions for each of the six elements in the `numbers` array. The loop instructions prompt the user to enter an integer and then store the user's response in the current element.

HOW TO Use the Extraction Operator to Store Data in a One-Dimensional Array

Syntax

```
cin >> arrayName[subscript];
```

Example 1

```
cin >> letters[0];
```

The statement stores the user's entry in the first element in the `letters` array.

Figure 11-4 How to use the extraction operator to store data in a one-dimensional array *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Example 2

```
for (int sub = 0; sub < 4; sub += 1)
{
    cout << "Enter the sales for Region " << sub + 1 << ": ";
    cin >> sales[sub];
}   //end for
```

The `for` loop stores the user's entries in the four-element `sales` array.

Example 3

```
int x = 0;
while (x < 6)
{
    cout << "Enter an integer: ";
    cin >> numbers[x];
    x += 1;
}   //end while
```

The `while` loop stores the user's entries in the six-element `numbers` array.

Figure 11-4 How to use the extraction operator to store data in a one-dimensional array

Displaying the Contents of a One-Dimensional Array

To display the contents of an array, you need to access each of its elements. You do this using a loop along with a counter variable that keeps track of each subscript in the array. Figure 11-5 shows examples of loops you can use to display the contents of the arrays declared earlier in Figure 11-2. Example 1 uses a `while` loop to display the contents of the `letters` array, which contains three elements. Example 2 uses a `for` loop to display the contents of the four-element `sales` array. Notice that the valid subscripts for the `sales` array are 0 through 3, whereas the valid region numbers are 1 through 4. Example 3 uses a `do while` loop to display the contents of the six-element `numbers` array.

HOW TO Display the Contents of a One-Dimensional Array

Example 1

```
int x = 0;
while (x < 3)
{
    cout << letters [x] << endl;
    x += 1;
}   //end while
```

The `while` loop displays the contents of the three-element `letters` array.

Example 2

```
for (int sub = 0; sub < 4; sub += 1)
{
    cout << "Sales for Region " << sub + 1 << ": $";
    cout << sales[sub] << endl;
}   //end for
```

The `for` loop displays the contents of the four-element `sales` array.

Figure 11-5 How to display the contents of a one-dimensional array *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Example 3

```
int x = 0;
do   //begin loop
{
    cout << numbers[x] << endl;
    x += 1;
}  while (x < 6);
```

The `do while` loop displays the contents of the six-element `numbers` array.

Figure 11-5 How to display the contents of a one-dimensional array

The Calories Program

Figure 11-6 shows the problem specification and IPO chart information for the calories program. The program uses a five-element array to store the number of calories consumed for each of five days. The daily calories are entered by the user and then displayed on the computer screen.

Problem specification

Create a program that allows the user to enter the number of calories a person consumes for each of five days. Store the daily calories in a five-element array named `calories`. Then, display the contents of the array on the computer screen.

Input	Processing	Output
calories (for each of 5 days)	Processing items: array (5 elements) subscript (counter: 0 to 4)	calories (for each of 5 days)

Algorithm:

```
1. repeat for (subscript from 0 to 4)
      enter the calories into array[subscript]
   end repeat
2. repeat for (subscript from 0 to 4)
      display the calories stored in array[subscript]
   end repeat
```

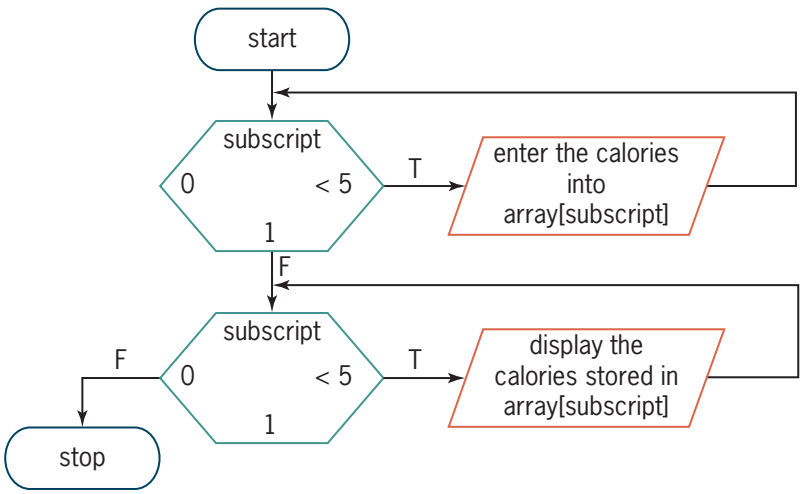


Figure 11-6 Problem specification and IPO chart for the calories program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Figure 11-7 shows the IPO chart information and the corresponding C++ instructions. Figure 11-8 shows the entire program along with a sample run of the program.

IPO chart information	C++ instructions
Input	
calories (for each of 5 days)	the calories will be entered into the array
Processing	
array (5 elements)	`int calories[5] = {0};`
subscript (counter: 0 to 4)	declared and initialized in the for clause
Output	
calories (for each of 5 days)	displayed from the array by the for loop
Algorithm	

1. repeat for (subscript from 0 to 4)
enter the calories into array[subscript]
end repeat
	`for (int sub = 0; sub < 5; sub += 1)`
	`{`
	`  cout << "Calories for day "`
	`  << sub + 1 << ": ";`
	`  cin >> calories[sub];`
	`} //end for`

2. repeat for (subscript from 0 to 4)
display the calories stored in array[subscript]
end repeat
	`for (int sub = 0; sub < 5; sub += 1)`
	`  cout << "Calories for day "`
	`  << sub + 1 << ": "`
	`  << calories[sub] << endl;`
	`//end for`

Figure 11-7 IPO chart information and C++ instructions for the calories program

```
 1 //Calories.cpp - gets and displays daily calories
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 using namespace std;
 6
 7 int main()
 8 {
 9    int calories[5] = {0};
10
11    //store data in the array
12    for (int sub = 0; sub < 5; sub += 1)
13    {
14        cout << "Calories for day " << sub + 1 << ": ";
15        cin >> calories[sub];
16    } //end for
17
18    //display the contents of the array
19    cout << endl << "Array contents:" << endl;
20    for (int sub = 0; sub < 5; sub += 1)
21        cout << "Calories for day " << sub + 1
22            << ": " << calories[sub] << endl;
23    //end for
24
25    return 0;
26 } //end of main function
```

array declaration

stores data in the array

displays the contents of the array

Figure 11-8 Calories program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

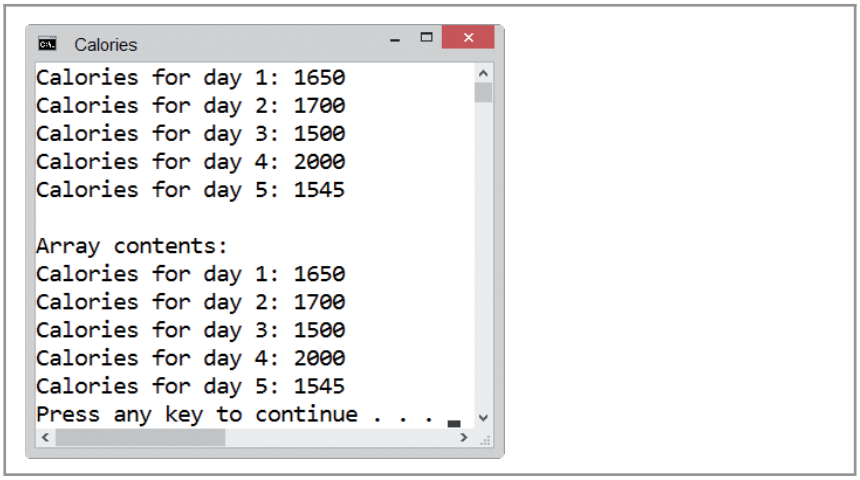


Figure 11-8 Calories program

Desk-checking the code in Figure 11-8 will help you understand how arrays operate in a program. You will desk-check the code using the daily calorie amounts shown in the figure. First, the declaration statement on Line 9 declares and initializes a five-element `int` array named `calories`. Figure 11-9 shows the desk-check table after the declaration statement is processed.

calories[0]	calories[1]	calories[2]	calories[3]	calories[4]
0	0	0	0	0

Figure 11-9 Desk-check table after the array declaration statement is processed

The `for` clause on Line 12 is processed next. The clause's *initialization* argument declares an `int` variable named `sub` and initializes it to the number 0. The `sub` variable is a counter variable that will keep track of the five array subscripts: 0, 1, 2, 3, and 4. As you learned in Chapter 7, a variable declared in a `for` clause is local to the `for` loop and can be used only by the statements within the loop. In this case, the `sub` variable is local to the `for` loop on Lines 12 through 16. The `sub` variable will remain in memory until the `for` loop ends. Figure 11-10 shows the desk-check table after the *initialization* argument on Line 12 has been processed.

local to the `for` loop on Lines 12 through 16

calories[0]	calories[1]	calories[2]	calories[3]	calories[4]	sub
0	0	0	0	0	0

Figure 11-10 Desk-check table after the *initialization* argument on Line 12 is processed

The `for` clause's *condition* argument checks whether the value stored in the `sub` variable is less than 5. It is, so the statements in the body of the `for` loop are processed. The `cout` statement on Line 14 prompts the user to enter the calories for the current day—in this case, day 1. Notice that the current day is determined by adding the number 1 to the value stored in the `sub` variable (0). This is because unlike the array subscripts, which go from 0 through 4, the day numbers go from 1 through 5. The day number is always one number more than the subscript of its corresponding element in the array. In other words, day 1's calories will be stored in the

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

element whose subscript is 0. Likewise, day 2's calories will be stored in the element whose subscript is 1, and so on. The `cin` statement on Line 15 gets day 1's calories from the user and then stores the amount in the first array element (`calories[0]`). Figure 11-11 shows the calories for day 1 (1650) entered in the array.

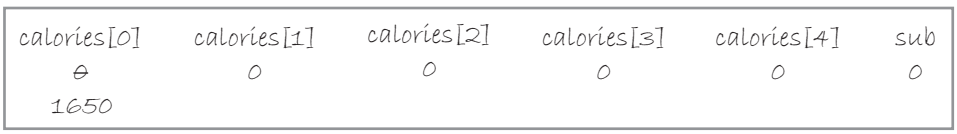


Figure 11-11 Desk-check table after day 1's calories are entered in the array

The `for` clause's *update* argument adds the number 1 to the contents of the `sub` variable, giving 1. The *condition* argument then checks whether the `sub` variable's value is less than 5. It is, so the statements in the body of the `for` loop are processed again. The `cout` statement prompts the user to enter the calories for day 2, and the `cin` statement stores the user's response in the second array element (`calories[1]`). Figure 11-12 shows the calories for day 2 (1700) entered in the array.

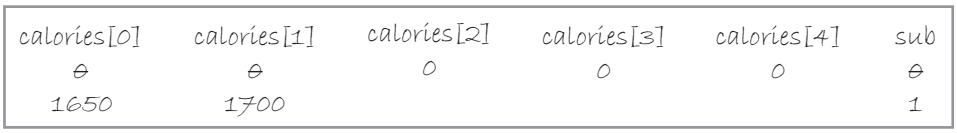


Figure 11-12 Desk-check table after day 2's calories are entered in the array

Next, the `for` clause's *update* argument adds the number 1 to the contents of the `sub` variable, giving 2. The *condition* argument then checks whether the `sub` variable's value is less than 5. It is, so the statements in the body of the `for` loop prompt the user to enter the calories for day 3 and then store the user's response in the third array element (`calories[2]`). Figure 11-13 shows the calories for day 3 (1500) entered in the array.

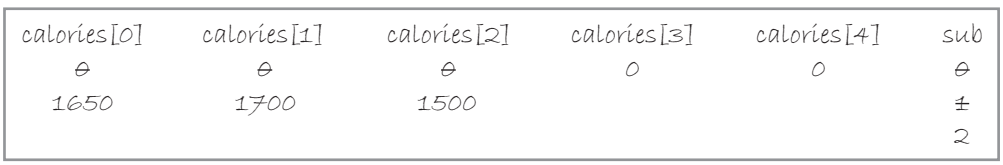


Figure 11-13 Desk-check table after day 3's calories are entered in the array

Once again, the `for` clause's *update* argument adds the number 1 to the contents of the `sub` variable; the result is 3. The *condition* argument then checks whether the `sub` variable's value is less than 5. It is, so the statements in the body of the `for` loop prompt the user to enter day 4's calories and then store the user's response in the fourth array element (`calories[3]`). Figure 11-14 shows the calories for day 4 (2000) entered in the array.

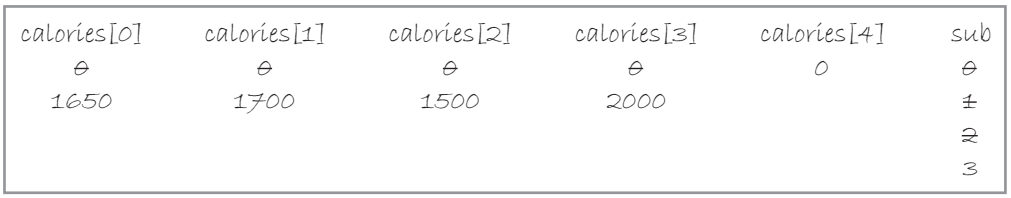


Figure 11-14 Desk-check table after day 4's calories are entered in the array

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Next, the `for` clause's *update* argument increases the value in the `sub` variable by 1, giving 4. The *condition* argument then checks whether the `sub` variable's value is less than 5. It is, so the statements in the body of the `for` loop prompt the user to enter day 5's calories and then store the user's response in the fifth (and last) array element (`calories[4]`). Figure 11-15 shows the calories for day 5 (1545) entered in the array.

calories[0]	calories[1]	calories[2]	calories[3]	calories[4]	sub
0	0	0	0	0	0
1650	1700	1500	2000	1545	1
					2
					3
					4

Figure 11-15 Desk-check table after day 5's calories are entered in the array

Once again, the `for` clause's *update* argument increases the `sub` variable's value by 1; this time, the result is 5. The *condition* argument then checks whether the `sub` variable's value is less than 5. It's not, so the `for` loop on Lines 12 through 16 ends and the computer removes the loop's local `sub` variable from internal memory. Figure 11-16 shows the current status of the desk-check table.

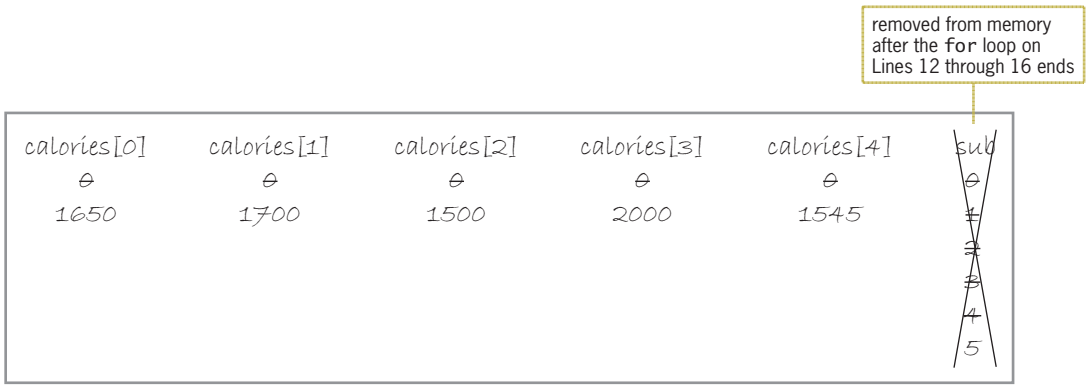


Figure 11-16 Desk-check table after the `for` loop on Lines 12 through 16 ends

The `cout` statement on Line 19 is processed next and displays the "Array contents:" message on the computer screen. Then, the computer processes the `for` clause on Line 20. The clause's *initialization* argument declares and initializes an `int` variable named `sub`. Although the variable's name is the same as the one in the first `for` clause (on Line 12), it is not the same variable. This `sub` variable is local to the `for` loop on Lines 20 through 23. The `sub` variable created by the `for` clause on Line 12 was local to the `for` loop on Lines 12 through 16 and was removed from memory when that loop ended. Figure 11-17 shows the desk-check table after the *initialization* argument on Line 20 is processed.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

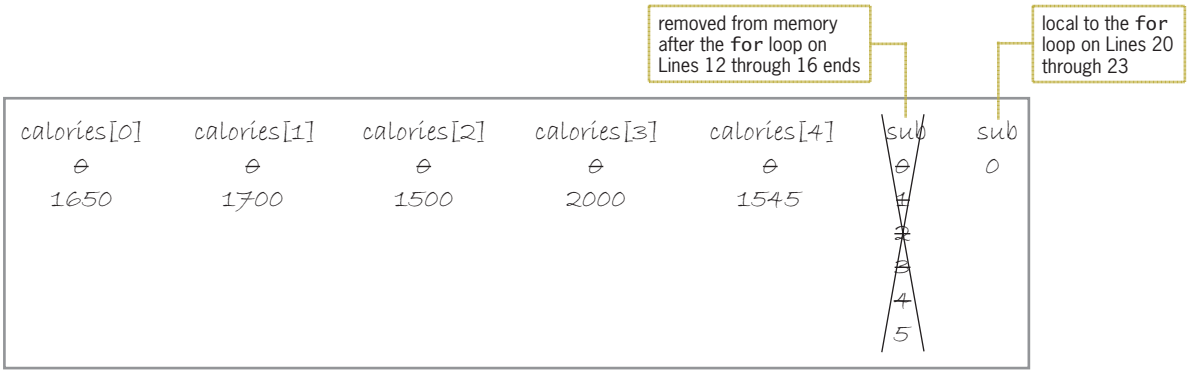


Figure 11-17 Desk-check table after the *initialization* argument on Line 20 is processed

The *condition* argument in the `for` clause on Line 20 checks whether the value in the `sub` variable is less than 5. It is, so the `for` loop displays day 1's calories, which are stored in the `calories[0]` element, on the computer screen. Notice that the day number is one number more than the subscript.

Next, the `for` clause's *update* argument adds the number 1 to the contents of the `sub` variable, giving 1. The *condition* argument then checks whether the value in the `sub` variable is less than 5. It is, so the `for` loop displays day 2's calories from the `calories[1]` element.

Here again, the `for` clause's *update* argument increases the `sub` variable's value by 1; the result is 2. The *condition* argument then checks whether the value in the `sub` variable is less than 5. It is, so the `for` loop displays day 3's calories from the `calories[2]` element.

The `for` clause's *update* argument again adds the number 1 to the contents of the `sub` variable, giving 3. The *condition* argument then checks whether the value in the `sub` variable is less than 5. It is, so the statements in the body of the `for` loop display day 4's calories from the `calories[3]` element.

Next, the `for` clause's *update* argument increases the `sub` variable's value by 1; the result is 4. The *condition* argument then checks whether the value in the `sub` variable is less than 5. It is, so the statements in the body of the `for` loop display day 5's calories from the `calories[4]` element.

Once again, the `for` clause's *update* argument increases the value in the `sub` variable by 1; this time, the result is 5. The *condition* argument then checks whether the `sub` variable's value is less than 5. It's not, so the `for` loop ends and the computer removes the loop's local `sub` variable from internal memory. Figure 11-18 shows the desk-check table after the `for` loop on Lines 20 through 23 ends.

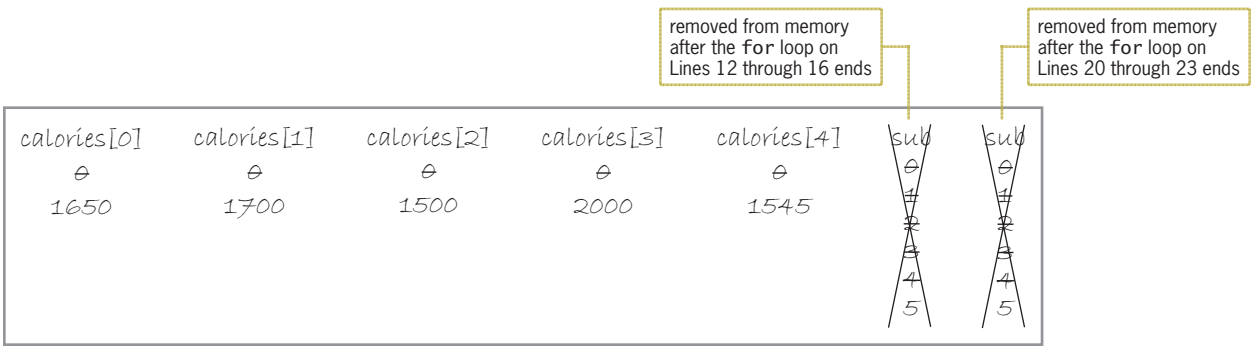


Figure 11-18 Desk-check table after the `for` loop on Lines 20 through 23 ends

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Finally, the computer processes the `return 0;` statement on Line 25. When the program ends, the computer removes the `calories` array from its internal memory.

Passing a One-Dimensional Array to a Function

Figure 11-19 shows a modified version of the calories program. In this version, the `main` function passes the `calories` array to a program-defined void function named `displayArray`, whose task is to display the contents of the array. The changes made to the original code (shown earlier in Figure 11-8) are shaded in Figure 11-19.

```
 1 //Modified Calories.cpp - gets and displays daily calories
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 using namespace std;
 6
 7 //function prototype
 8 void displayArray(int cals[], int numElements);
 9
10 int main()
11 {
12    int calories[5] = {0};
13
14    //store data in the array
15    for (int sub = 0; sub < 5; sub += 1)
16    {
17        cout << "Calories for day " << sub + 1 << ": ";
18        cin >> calories[sub];
19    } //end for
20
21    //display the contents of the array
22    displayArray(calories, 5);
23
24    return 0;
25 } //end of main function
26
27 //*****function definitions*****
28 void displayArray(int cals[], int numElements)
29 {
30    cout << endl << "Array contents:" << endl;
31    for (int sub = 0; sub < 5; sub += 1)
32        cout << "Calories for day " << sub + 1
33             << ": " << cals[sub] << endl;
34    //end for
35 }  //end of displayArray function
```

only the name is optional

the name is optional

function prototype

function call

function header

Figure 11-19 Modified calories program

The function call, which appears on Line 22, passes two items of information to the `displayArray` function: the `calories` array and the number of elements in the array. Unlike scalar (simple) variables, which you learned about in the previous chapters, arrays in C++ are passed automatically *by reference* rather than *by value*. This is because it is more efficient to pass arrays in that manner. Since many arrays are large, passing an array *by value* would consume a great deal of the computer's memory and time because the computer would need

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

to duplicate the array in the receiving function's formal parameter. Passing an array *by reference* allows the computer to pass the address of only the first array element. Because array elements are stored in contiguous locations in memory, the computer can use the address to locate the remaining elements in the array.

Given that arrays are passed automatically *by reference,* you do not include the address-of (&) operator before the name of an array's formal parameter in the function header, as you do when passing scalar variables *by reference*. You also do not include the address-of operator in the function prototype. Instead, you indicate that you are passing an array to a function by entering the formal parameter's data type and name, followed by an empty set of square brackets, in the receiving function's header and in its prototype, as shown in Figure 11-19. However, recall that the formal parameter's name is optional in a function's prototype. Therefore, you could also write the function prototype on Line 8 in Figure 11-19 as `void displayArray(int [], int);`.

Figure 11-20 shows the completed desk-check table for the modified calories program. Recall from Chapter 10 that when you pass a variable *by reference* to a function, the computer locates the variable and then assigns the name of the corresponding formal parameter to the variable. The same process occurs with array variables and explains why each array variable in Figure 11-20 has two names: one assigned by the `main` function, and the other assigned by the `displayArray` function. Although both functions can access the memory locations where the array variables reside, each function uses a different name to do so. The `main` function uses the name `calories`, whereas the `displayArray` function uses the name `cals`.

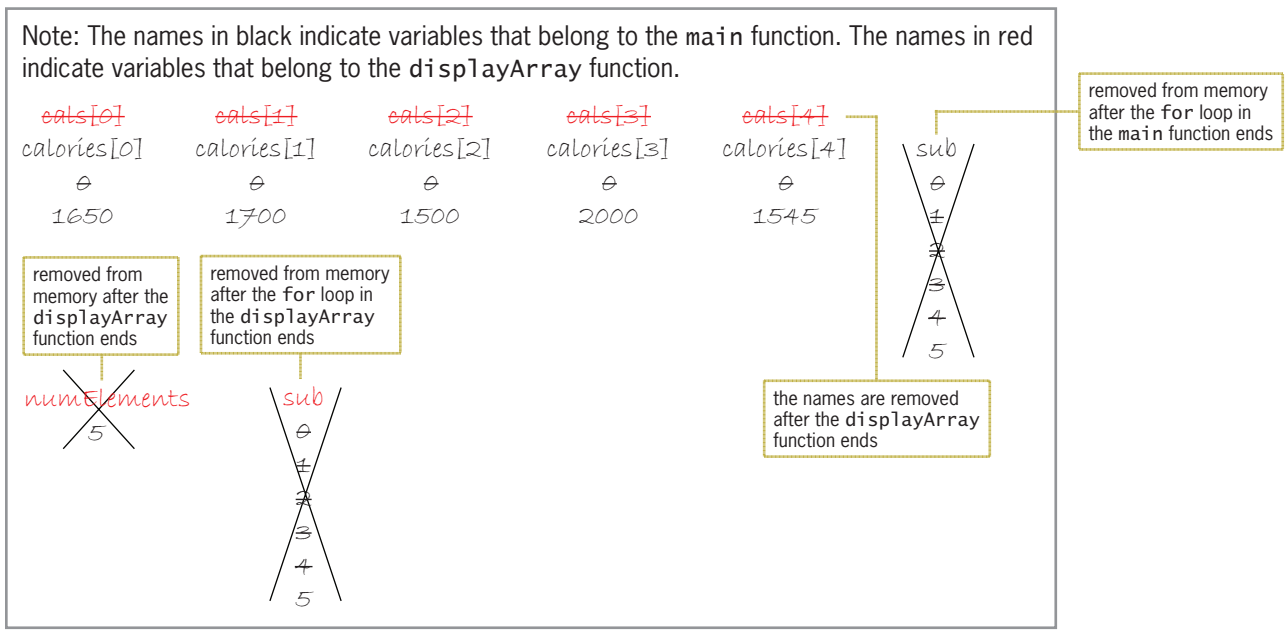


Figure 11-20 Completed desk-check table for the modified calories program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 11-1

1. Which of the following declares a one-dimensional `int` array named `quantities` and initializes each of its 20 elements to the number 0?
   a. `int quantities[20] = {0};`
   b. `int quantities(20) = {0};`
   c. `int quantities{20} = 0;`
   d. none of the above
2. What is the name of the first element in the `quantities` array from Question 1?
3. What is the name of the last element in the `quantities` array from Question 1?
4. Write a C++ statement that assigns the number 7 to the fourth element in the `quantities` array from Question 1.
5. Which of the following calls the value-returning `getTotal` function, passing it the `quantities` array from Question 1 and the number of array elements?
   a. `total = getTotal(quantities[], 20);`
   b. `total = getTotal(quantities[20]);`
   c. `total = getTotal(quantities, 20);`
   d. none of the above

Calculating a Total and an Average

In this section, we will add another program-defined function to the calories program. The additional function will be a value-returning function named `getAverage`. The function will calculate the average number of calories consumed for the five days and then return the result to the `main` function. Figure 11-21 shows the `getAverage` function's code. It also includes the function call added to the `main` function, as well as a sample run of the program. (The `average` variable is a `double` variable declared in the `main` function.)

Function call added to the `main` function

```
average = getAverage(calories, 5);
```

`getAverage` function

```
double getAverage(int cals[], int numElements)
{
    double total = 0.0;    //accumulator

    //accumulate array values
    for (int sub = 0; sub < numElements; sub += 1)
        total += cals[sub];
    //end for

    //calculate and return average
    return static_cast<double>(total) / numElements;
}  //end of getAverage function
```

Figure 11-21 Function call, `getAverage` function, and a sample run *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

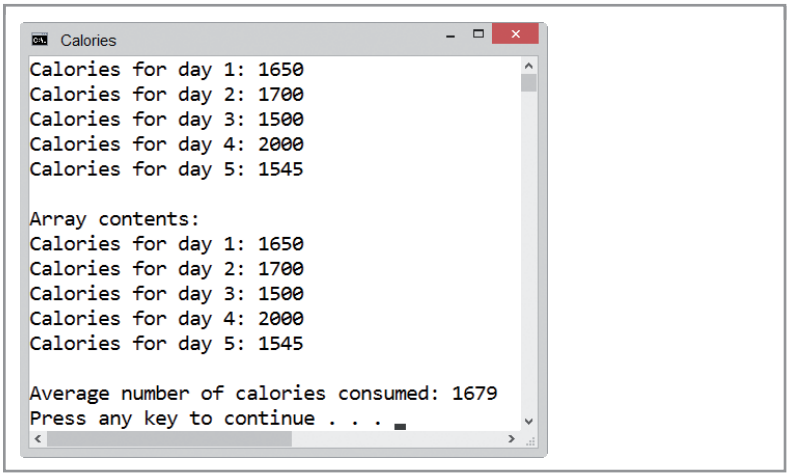


Figure 11-21 Function call, `getAverage` function, and a sample run

The `average = getAverage(calories, 5);` statement in the `main` function invokes the `getAverage` function, passing it the `calories` array and the number of array elements. (Recall that when an array is passed to a function, the computer passes only the address of the first array element.) The computer temporarily leaves the `main` function to process the `getAverage` function's code, beginning with the function header.

When processing the `getAverage` function's header, the computer locates the `calories` array in memory and assigns the formal parameter's name—in this case, `cals`—to each element. As a result, each array element has two names. The first element is called `calories[0]` in the `main` function but `cals[0]` in the `getAverage` function.

After processing the `getAverage` function's header, the computer processes the statements within its function body. The first statement in the function body declares and initializes a `double` variable named `total`. The function then uses a `for` loop and the `total` variable to accumulate (add together) the value stored in each array element. The sum of those values (8395) represents the total number of calories consumed during the five days. The `for` loop ends when its `sub` variable contains the integer 5 because that is the first integer that is not less than 5. When the `for` loop ends, its `sub` variable is removed from memory.

The `getAverage` function's `return` statement is processed next. The statement calculates the average number of calories by dividing the total calories (8395) by the number of array elements (5). It then returns the average (1679) to the assignment statement that invoked the `getAverage` function. That statement, which appears in the `main` function, assigns the return value to the `average` variable.

When the `getAverage` function ends, the computer removes the `cals` name from the array elements. It also removes the `numElements` and `total` variables from internal memory.

The Social Media Program—Searching an Array

Figure 11-22 shows the problem specification, IPO chart information, and C++ instructions for the social media program. The program uses an array to store the results of a poll of 25 people who were asked to estimate the average number of minutes they spend on Facebook each day. The program allows the user to determine the number of people spending more than a specific

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

number of minutes—which is entered by the user—on Facebook. To accomplish this task, the program uses a loop to search the `pollResults` array. The loop contains a selection structure that compares the value in the current array element with the number of minutes entered by the user. If the array element's value is greater than the user's entry, the program adds the number 1 to the `numOver` counter variable. After searching each array element, the program displays the contents of the `numOver` variable on the screen.

Note: In most programs, the array values are either entered by the user at the keyboard or read from a file. However, for convenience, many of the programs in this chapter fill the array with values in its declaration statement.

Problem specification
Create a program that uses an array to store the results from a poll of 25 people. Each person was asked to estimate the amount of time, in minutes, that he or she spends on Facebook each day. The program should allow the user to enter a specific number of minutes and then display the number of people spending more than that length of time.

IPO chart information

Input
array (25 elements)

search for minutes

Processing
subscript (counter: 0 to 24)

Output
number spending more than the search for minutes (counter)

Algorithm
1. enter the search for minutes
2. repeat for (subscript from 0 to 24)
   if (the array[subscript]value is greater than the search for minutes)
      add 1 to the number spending more than the search for minutes
   end if
   end repeat
3. display the number spending more than the search for minutes

C++ instructions

```
int pollResults[25] = {35, 120, 75, 60, 20,
                       25, 15, 90, 85, 35,
                       60, 15, 10, 25, 60,
                       100, 90, 10, 120, 5,
                       40, 70, 30, 25, 5};
int minutes = 0;
```

declared and initialized in the for clause

```
int numOver = 0;

cout << "Search for minutes over: ";
cin >> minutes;
for (int sub = 0; sub < 25; sub += 1)
    if (pollResults[sub] > minutes)

        numOver += 1;

    //end if
//end for
cout << endl << "Number who spend more than "
<< minutes << " minutes" << endl;
cout << "per day on Facebook: " << numOver <<
endl;
```

Figure 11-22 Problem specification, IPO chart information, and C++ instructions for the social media program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

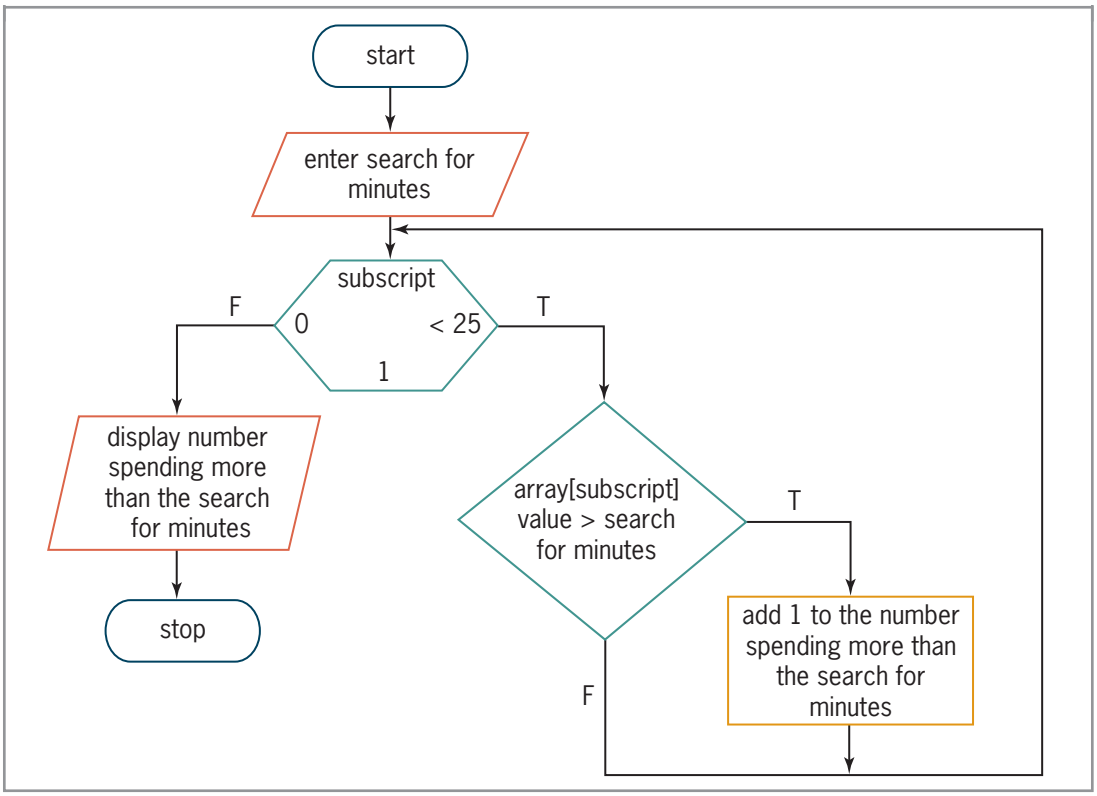


Figure 11-22 Problem specification, IPO chart information, and C++ instructions for the social media program

Figure 11-23 shows the entire social media program along with a sample run of the program. The declaration statement on Lines 10 through 14 creates and initializes the 25-element `pollResults` array. The statements on Lines 15 and 16 declare two `int` variables named `minutes` and `numOver`. The `cout` statement on Line 18 prompts the user to enter the number of minutes to search for, and the `cin` statement on Line 19 stores the user's response in the `minutes` variable. The `for` loop in the program accesses each element in the `pollResults` array, beginning with the element whose subscript is 0 and ending with the element whose subscript is 24. The selection structure in the loop compares the value stored in the current array element with the value stored in the `minutes` variable. If the array element's value is greater than the variable's value, the selection structure's true path adds the number 1 to the contents of the `numOver` variable. When the `for` loop ends, which is when the `sub` variable contains the number 25, the `cout` statements on Lines 28 through 30 display the number of people whose Facebook time exceeds the number of minutes entered by the user.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

```
 1 //Social.cpp - displays the number of people whose
 2 //Facebook time exceeds a specific number of minutes
 3 //Created/revised by <your name> on <current date>
 4
 5 #include <iostream>
 6 using namespace std;
 7
 8 int main()
 9 {
10    int pollResults[25] = {35, 120, 75, 60, 20,
11                           25, 15, 90, 85, 35,
12                           60, 15, 10, 25, 60,
13                           100, 90, 10, 120, 5,
14                           40, 70, 30, 25, 5};
15    int minutes = 0;
16    int numOver = 0;
17
18    cout << "Search for minutes over: ";
19    cin >> minutes;
20
21    //search the array
22    for (int sub = 0; sub < 25; sub += 1)
23        if (pollResults[sub] > minutes)
24            numOver += 1;
25        //end if
26    //end for
27
28    cout << endl << "Number who spend more than " << minutes
29        << " minutes" << endl;
30    cout << "per day on Facebook: " << numOver << endl;
31    return 0;
32 } //end of main function
```

```
Social Media
Search for minutes over: 40

Number who spend more than 40 minutes
per day on Facebook: 11
Press any key to continue . . .
```

Figure 11-23 Social media program

The Currency Converter Program—Accessing an Individual Element

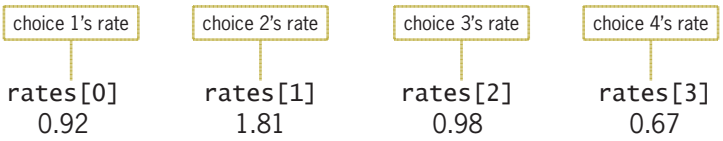
Figure 11-24 shows the problem specification, IPO chart information, and C++ instructions for the currency converter program. The program converts the number of American dollars entered by the user into the equivalent number of Euros, British pounds, German marks, or Swiss francs, depending on the user's selection from a menu. It then displays the name of the selected foreign currency and the converted result on the computer screen. The program uses a four-element array to store the foreign exchange rates.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The flowchart for the currency converter program is contained in the Currency.pdf file.

Problem specification

Create a program that converts the number of American dollars entered by the user into one of the following foreign currencies: euro, British pound, German mark, or Swiss franc. Allow the user to select the foreign currency from a menu. Store the exchange rates in a four-element `double` array named `rates`. The array is illustrated below and includes the exchange rates the program should use. Notice that the menu choice is always one number more than the subscript of its corresponding rate. For example, menu choice 1's rate is stored in the array element whose subscript is 0.



IPO chart information	C++ instructions
Input	
menu choice	`int choice = 0;`
dollars	`int dollars = 0;`
Processing	
array (4 elements)	`double rates[4] = {0.92, 1.81, 0.98, 0.67};`
Output	
converted result	`double result = 0.0;`
Algorithm	

1. display menu	`cout << "Currency Menu" << endl;`
	`cout << "1  Euro" << endl;`
	`cout << "2  British Pound" << endl;`
	`cout << "3  German Mark" << endl;`
	`cout << "4  Swiss Franc" << endl;`
	`cout << "Choice (1 to 4): ";`

2. enter the menu choice	`cin >> choice;`
3. if (the menu choice is greater than 0 and less than 5)	`if (choice > 0 && choice < 5)` `{`

enter dollars	`cout << "Number of American dollars: ";`
	`cin >> dollars;`
	`cout << endl;`

if (menu choice is 1)	`if (choice == 1)`
display "Euros: "	`cout << "Euros: ";`
else if (menu choice is 2)	`else if (choice == 2)`
display "British pounds: "	`cout << "British pounds: ";`
else if (menu choice is 3)	`else if (choice == 3)`
display "German marks: "	`cout << "German marks: ";`
else	`else`
display "Swiss francs: ";	`cout << "Swiss francs: ";`
end if	`//end if`

Figure 11-24 Problem specification, IPO chart information, and C++ instructions for the currency converter program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
    converted result = dollars *
    rates[menu choice - 1]
    display converted result
                                        result = dollars * rates[choice - 1];

                                        cout << result << endl;
                                    }
else                                else
    display "Invalid menu choice"       cout << "Invalid menu choice" << endl;
end if                              //end if
```

Figure 11-24 Problem specification, IPO chart information, and C++ instructions for the currency converter program

Before accessing an array element, you should always verify that the subscript is valid for the array. If the compiler encounters an invalid subscript, it will display an error message and the program will end abruptly.

Figure 11-25 shows the entire currency converter program and includes a sample run of the program. The statements on Lines 10 through 13 create and initialize the four-element `rates` array and the program's three variables. The statements on Lines 15 through 20 display the currency menu and then prompt the user to enter his or her choice. The statement on Line 21 stores the user's response in the `choice` variable. Next, the outer selection structure, which begins on Line 23, determines whether the user's choice is valid. If it isn't valid, the structure's false path displays the "Invalid menu choice" message. If it is valid, on the other hand, the instructions in the structure's true path are processed.

The instructions in the outer selection structure's true path in Figure 11-25 prompt the user to enter the number of American dollars and then store the user's response in the `dollars` variable. The nested selection structure on Lines 28 through 36 uses the value in the `choice` variable to display the name of the selected foreign currency. The statement on Line 38 converts the number of American dollars to the selected currency and then stores the result in the `result` variable. Notice that the statement uses the `choice` variable to access the appropriate element in the `rates` array. The number 1 must be subtracted from the variable's value because it is always one number more than the subscript of its associated exchange rate in the array. The statement on Line 39 displays the contents of the `result` variable on the computer screen. Finally, the `return 0;` statement on Line 44 is processed before the program ends.

```
 1 //Currency.cpp - converts American dollars
 2 //to different currencies
 3 //Created/revised by <your name> on <current date>
 4
 5 #include <iostream>
 6 using namespace std;
 7
 8 int main()
 9 {
10    double rates[4] = {0.92, 1.81, 0.98, 0.67};
11    int choice = 0;
12    int dollars = 0;
13    double result = 0.0;
14
```

Figure 11-25 Currency converter program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
    cout << "Currency Menu" << endl;
    cout << "1  Euro" << endl;
    cout << "2  British Pound" << endl;
    cout << "3  German Mark" << endl;
    cout << "4  Swiss Franc" << endl;
    cout << "Choice (1 to 4): ";
    cin >> choice;

    if (choice > 0 && choice < 5)
    {
        cout << "Number of American dollars: ";
        cin >> dollars;
        cout << endl;
        if (choice == 1)
            cout << "Euros: ";
        else if (choice == 2)
            cout << "British pounds: ";
        else if (choice == 3)
            cout << "German marks: ";
        else
            cout << "Swiss francs: ";
        //end if

        result = dollars * rates[choice - 1];
        cout << result << endl;
    }
    else
        cout << "Invalid menu choice" << endl;
    //end if
    return 0;
} //end of main function
```

```
Currency Converter
Currency Menu
1  Euro
2  British Pound
3  German Mark
4  Swiss Franc
Choice (1 to 4): 4
Number of American dollars: 100

Swiss francs: 67
Press any key to continue . . .
```

Figure 11-25 Currency converter program

The Highest Number Program—Finding the Highest Value

Figure 11-26 shows the problem specification, IPO chart information, and C++ instructions for the highest number program, which displays the highest number stored in a four-element array. The program uses a program-defined value-returning function to determine the highest number.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The flowchart for the highest number program is contained in the Highest.pdf file.

Problem specification
Create a program that displays the highest number stored in a four-element array. The array contains the following numbers: 13, 2, 40, and 25. Use a program-defined value-returning function to determine the highest number.

`main` function

IPO chart information	**C++ instructions**
Input	
none	
Processing	
array (4 elements)	`int numbers[4] = {13, 2, 40, 25};`
Output	
highest number	returned by the getHighest function and displayed by the cout statement
Algorithm	
1. call the getHighest function to determine the highest number; pass the array and the number of elements 2. display the highest number	`cout << "The highest number in the array is "` `<< getHighest(numbers, 4) << "." << endl;`

`getHighest` function

IPO chart information	**C++ instructions**
Input	
array (4 elements) (formal parameter) number of elements (formal parameter)	`int numArray[]` `int numElements`
Processing	
subscript (counter: 1 to one less than the number of elements)	declared and initialized in the for clause
Output	
highest number	`int high = numArray[0];`
Algorithm	

1. assign the first array element's value as the highest number
2. repeat for (subscripts from 1 to one less than the number of elements)
if (the array[subscript] value is greater than the highest number)
assign the array[subscript] value as the highest number
end if
end repeat
3. return the highest number
	assigned in the high variable's declaration statement
	`for (int sub = 1; sub < numElements; sub += 1)`
	`if (numArray[sub] > high)`
	`high = numArray[sub];`
	`//end if`
	`//end for`
	`return high;`

Figure 11-26 Problem specification, IPO chart information, and C++ instructions for the highest number program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Figure 11-27 shows the entire highest number program and includes a sample run of the program.

```
 1 //Highest.cpp - displays the highest number in an array
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 using namespace std;
 6
 7 //function prototype
 8 int getHighest(int numArray[], int numElements);
 9
10 int main()
11 {
12    int numbers[4] = {13, 2, 40, 25};
13
14    cout << "The highest number in the array is "
15         << getHighest(numbers, 4) << "." << endl;
16    return 0;
17 } //end of main function
18
19 //*****function definitions*****
20 int getHighest(int numArray[], int numElements)
21 {
22    //assign first element's value to the high variable
23    int high = numArray[0];
24
25    //begin search with second element
26    for (int sub = 1; sub < numElements; sub += 1)
27        if (numArray[sub] > high)
28            high = numArray[sub];
29        //end if
30    //end for
31    return high;
32 } //end of getHighest function
```

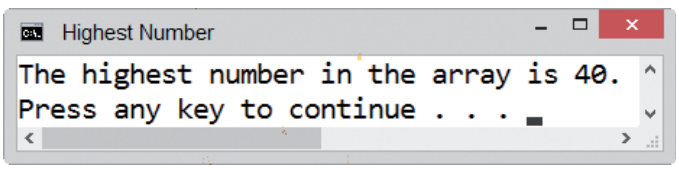


Figure 11-27 Highest number program

The loop in the `getHighest` function searches the second through the last elements in the `numArray` array. It doesn't need to search the first element because that element's value is already stored in the `high` variable by the statement on Line 23.

Desk-checking the program will help you understand how the highest number is determined. First, the statement on Line 12 in the `main` function declares and initializes a four-element array named `numbers`. The `cout` statement on Lines 14 and 15 is processed next. The `getHighest(numbers, 4)` part of the statement invokes the `getHighest` function, passing it the `numbers` array and the number of array elements. At this point, the computer temporarily leaves the `main` function to process the `getHighest` function's code, beginning with the function header on Line 20.

When processing the `getHighest` function's header, the computer locates the `numbers` array in memory and assigns the name of the first formal parameter (`numArray`) to each array element. It also creates the second formal parameter—an `int` variable named `numElements`—and stores the number 4 in it. The code in the function's body is processed next. The first statement (on Line 23) creates and initializes an `int` variable named `high`. The function uses the `high` variable

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

to keep track of the highest value stored in the array. Notice that the variable is initialized to the value stored in the first array element (13). When searching for the highest (or lowest) value in an array, it is a common programming practice to use the first array element to initialize the variable. Figure 11-28 shows the desk-check table after the declaration statement is processed.

Note: The names in black indicate variables that belong to the `main` function. The names in red indicate variables that belong to the `getHighest` function.

numArray[0]	numArray[1]	numArray[2]	numArray[3]
numbers[0]	numbers[1]	numbers[2]	numbers[3]
13	2	40	25

numElements	high
4	13

Figure 11-28 Desk-check table after the declaration statement on Line 23 is processed

The `for` clause on Line 26 creates an `int` variable named `sub` and initializes it to 1. The clause's *condition* argument then checks whether the value in the `sub` variable is less than the value in the `numElements` variable (4). It is, so the `if` statement's condition compares the value stored in the `numArray[1]` element, which is the second element in the array, with the value stored in the `high` variable. (Recall that at this point, the `high` variable contains the same value as the first array element.) The value in the `numArray[1]` element (2) is not greater than the value in the `high` variable (13), so the `if` statement ends.

Next, the `for` clause's *update* argument increases the `sub` variable's value by 1, giving 2. Its *condition* argument then checks whether the variable's value is less than number of array elements. It is, so the `if` statement's condition compares the value stored in the `numArray[2]` element with the value stored in the `high` variable. In this case, the array element's value (40) *is* greater than the number stored in the `high` variable (13), so the instruction in the `if` statement's true path assigns the element's value to the `high` variable, as shown in Figure 11-29, and then the `if` statement ends.

Note: The names in black indicate variables that belong to the `main` function. The names in red indicate variables that belong to the `getHighest` function.

numArray[0]	numArray[1]	numArray[2]	numArray[3]
numbers[0]	numbers[1]	numbers[2]	numbers[3]
13	2	40	25

numElements	high	sub
4	~~13~~	~~1~~
	40	2

Figure 11-29 Desk-check table showing the third element's value assigned to the `high` variable

After the `if` statement ends, the `for` clause's *update* argument increases the `sub` variable's value by 1, giving 3. Its *condition* argument then checks whether the variable's value is less than number of array elements. It is, so the `if` statement's condition compares the value stored in the `numArray[3]` element with the value stored in the `high` variable. The array element's value (25) is not greater than the `high` variable's value (40), so the `if` statement ends.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Once again, the `for` clause's *update* argument adds 1 to the value stored in the `sub` variable; the result is 4. Its *condition* argument then checks whether the variable's value is less than number of array elements. It's not, so the `for` loop ends and the computer processes the `return` statement on Line 31. The statement returns the value stored in the `high` variable to the statement that called the `getHighest` function. That statement is the `cout` statement on Lines 14 and 15 in the `main` function. When the `getHighest` function ends, the computer removes the `numElements` and `high` variables from memory. It also removes the `numArray` name from the appropriate locations in memory. See Figure 11-30.

Note: The names in black indicate variables that belong to the `main` function. The names in red indicate variables that belong to the `getHighest` function.

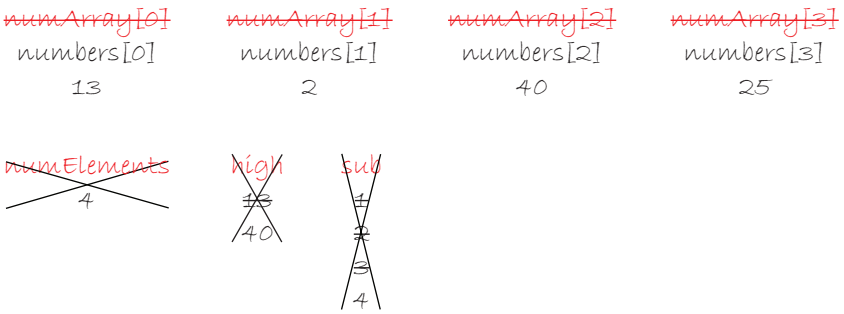


Figure 11-30 Desk-check table after the `getHighest` function ends

When the `cout` statement on Lines 14 and 15 receives the return value from the `getHighest` function, it displays the value on the computer screen. Finally, the computer processes the `return 0;` statement on Line 16. When the program ends, the computer removes the `numbers` array from memory.

Mini-Quiz 11-2

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

1. Which of the following adds the contents of the third element in the `orders` array to the `total` variable?

   a. `orders[2] += total;`
   b. `orders[3] += total;`
   c. `total += orders[2];`
   d. `total += orders[3];`

2. Which of the following determines whether the value stored in the fourth element in the `orders` array is greater than 25?

   a. `if (orders(3) > 25)`
   b. `if (orders{4} > 25)`
   c. `if (orders[3] > 25)`
   d. `if (orders[4] > 25)`

3. Write a C++ statement that multiplies the contents of the first element in the `sales` array by 0.15 and then stores the result in the `bonus` variable. Both the array and the variable have the `double` data type.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

4. Which of the following determines whether the `sub` variable contains a valid subscript for the 10-element `scores` array?

   a. `if (sub > 0 && sub < 10)`
   b. `if (sub >= 0 && sub <= 10)`
   c. `if (sub >= 0 && sub < 10)`
   d. `if (sub > 0 && sub <= 10)`

5. Which of the following tells the computer to process the loop instructions for each of the 20 elements in the `inventory` array? The program uses an `int` variable named `x` to keep track of the array subscripts. The `x` variable is initialized to 0.

   a. `while (x < 20)`
   b. `while (x <= 20)`
   c. `while (x > 0)`
   d. `while (x >= 0)`

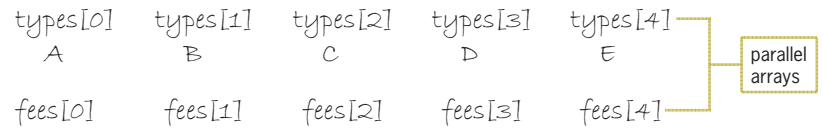
Parallel One-Dimensional Arrays

Figure 11-31 shows the problem specification, IPO chart information, and C++ instructions for the motorcycle club program, which displays the annual fee associated with the membership type entered by the user. The program uses two one-dimensional arrays: a `char` array named `types` and an `int` array named `fees`. The `types` array stores the five membership types, and the `fees` array stores the annual fees associated with those types. The first element in each array pertains to the first membership type (A and 100); the second element pertains to the second membership type (B and 110), and so on. The two arrays are referred to as **parallel arrays** because their elements are related by their position (subscript) in the arrays. Each element in the `types` array corresponds to the element located in the same position in the `fees` array. To determine the annual fee, you simply locate the membership type in the `types` array and then view its corresponding element in the `fees` array.

The flowchart for the motorcycle club program is contained in the Motorcycle.pdf file.

Problem specification
The members of a local motorcycle club are required to pay an annual fee based on their membership type. Create a program that displays a member's annual fee and membership type. The membership types and associated fees are shown here. Use a one-dimensional `char` array named `types` to store the membership types. Use a one-dimensional `int` array named `fees` to store the annual fees.

Type	Annual fee
A	100
B	110
C	125
D	150
E	200



IPO chart information

Input
membership type (A, B, C, D, or E)

C++ instructions

```
char memberType = ' ';
```

Figure 11-31 IPO chart information and C++ instructions for the motorcycle club program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Processing

types array (5 elements)
fees array (5 elements)
subscript (counter: 0 to 4)

```
char types[5] = {'A', 'B', 'C', 'D', 'E'};
int fees[5] = {100, 110, 125, 150, 200};
int sub = 0;
```

Output

fee
membership type

```
from the fees array
from the types array
```

Algorithm

1. enter the membership type

```
cout << "Membership type
(A, B, C, D, or E): ";
cin >> memberType;
memberType = toupper(memberType);
```

2. repeat while (the subscript is less than 5 and the membership type has not been located in the types array)
   add 1 to the subscript
   end repeat

```
while (sub < 5 && types[sub] !=
memberType)

    sub += 1;
//end while
```

3. if (the subscript is less than 5)
   display types[subscript] and fees[subscript]
   else
   display "Invalid membership type"
   end if

```
if (sub < 5)
    cout << "Annual fee for
    membership type "
    << types[sub] << ": $" << fees[sub]
    << endl;
else
    cout << "Invalid membership type"
    << endl;
//end if
```

Figure 11-31 IPO chart information and C++ instructions for the motorcycle club program

Figure 11-32 shows the code for the entire motorcycle club program and includes a sample run of the program.

```
 1 //Motorcycle.cpp - displays the annual membership fee
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 using namespace std;
 6
 7 int main()
 8 {
 9    char types[5] = {'A', 'B', 'C', 'D', 'E'};
10    int fees[5] = {100, 110, 125, 150, 200};
11    char memberType = ' ';
12    int sub = 0;
13
14    //get type to search for
15    cout << "Membership type (A, B, C, D, or E): ";
16    cin >> memberType;
17    memberType = toupper(memberType);
18
```

Figure 11-32 Motorcycle Club program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
19      //locate the position of the membership
20      //type in the types array
21      while (sub < 5 && types[sub] != memberType)
22          sub += 1;
23      //end while
24
25      //if the membership type was located in the
26      //types array, display the membership type
27      //and the corresponding fee
28      if (sub < 5)
29          cout << "Annual fee for membership type "
30               << types[sub] << ": $" << fees[sub] << endl;
31      else
32          cout << "Invalid membership type" << endl;
33      //end if
34      return 0;
35  }   //end of main function
```

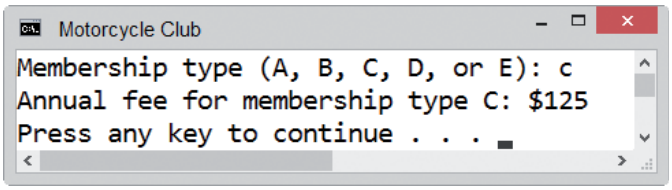


Figure 11-32 Motorcycle Club program

The program declares and initializes two parallel arrays (`types` and `fees`) and two variables (`memberType` and `sub`). The `memberType` variable will store the membership type entered by the user, and the `sub` variable will keep track of the array subscripts. The statements on Lines 15 through 17 prompt the user to enter a membership type, store the user's response in the `memberType` variable, and then convert the contents of the `memberType` variable to uppercase. The `while` loop that begins on Line 21 will continue to increment the `sub` variable's value by 1 as long as the value is less than 5 and (at the same time) the membership type has not been located in the `types` array. The loop will stop when either of the following conditions is true: the `sub` variable contains the number 5 (which indicates that the loop reached the end of the array without finding the membership type) or the membership type is located in the array.

After the loop completes its processing, the `if` statement that begins on Line 28 compares the number stored in the `sub` variable with the number 5. If the `sub` variable contains a number that is less than 5, it indicates that the loop stopped processing because the membership type *was* located in the `types` array. In that case, the `cout` statement on Lines 29 and 30 displays both the membership type from the `types` array and the corresponding annual fee from the `fees` array. However, if the `sub` variable contains a number that is *not* less than 5, it indicates that the loop stopped processing because it reached the end of the `types` array without finding the membership type. In that case, the `cout` statement on Line 32 displays the message "Invalid membership type".

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Sorting the Data Stored in a One-Dimensional Array

In some programs, you might need to arrange the contents of a one-dimensional array in either ascending or descending order. Arranging data in a specific order is called **sorting**. When a one-dimensional array is sorted in ascending order, the first element in the array contains the smallest value and the last element contains the largest value. The reverse is true when a one-dimensional array is sorted in descending order.

Over the years, many different sorting algorithms have been developed; one such algorithm is called the bubble sort. The **bubble sort** provides a quick and easy way to sort the items stored in an array, as long as the number of items is relatively small—for example, fewer than 50. The bubble sort algorithm works by comparing adjacent array elements and interchanging (swapping) the ones that are out of order. The algorithm continues comparing and swapping until the data in the array is sorted.

You can learn about another sorting algorithm in the Selection Sort section in the Ch11WantMore.pdf file.

To demonstrate the logic of a bubble sort, you will manually sort the contents of a three-element array named `nums` in ascending order. The array contains the following numbers: 9, 8, and 7. Figure 11-33 shows the array values before, during, and after the bubble sort.

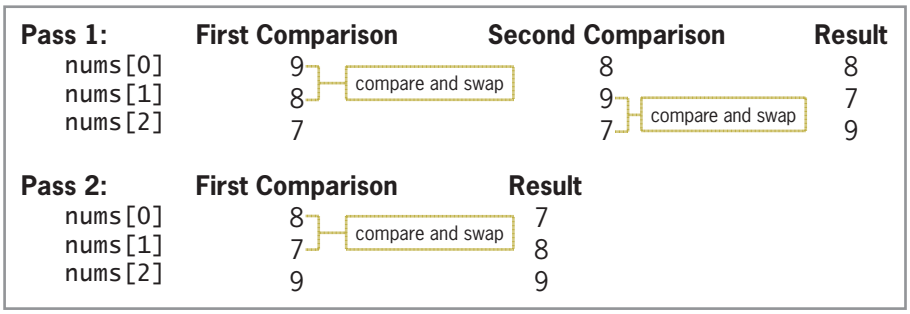


Figure 11-33 Array values before, during, and after the bubble sort

The bubble sort begins by comparing the array's first value with its second value. If the first value is less than or equal to the second value, then no swap is made. However, if the first value is greater than the second value, then both values are interchanged. In this case, the first value (9) is greater than the second value (8), so the values are swapped as shown in the Second Comparison column in Figure 11-33.

After comparing the first value with the second value, the bubble sort compares the second value with the third value. In this case, 9 is greater than 7, so the two values are swapped as shown in the Result column in Figure 11-33. At this point, the bubble sort has completed its first time through the entire array—referred to as a pass. Notice that at the end of the first pass, the largest value (9) is stored in the last element in the array. The bubble sort gets its name from the fact that as the larger values drop to the bottom of the array, the smaller values rise (like bubbles) to the top.

The bubble sort begins its second pass through the array by comparing the array's first value with its second value. In this case, 8 is greater than 7, so the two values are interchanged as shown in the Result column in Figure 11-33. Notice that at this point, the data in the array is sorted.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The program shown in Figure 11-34 uses the bubble sort to sort the contents of a four-element `int` array in ascending order. It then displays the contents of the sorted array on the screen.

```
 1 //Bubble Sort.cpp - uses the bubble sort to sort the
 2 //contents of a one-dimensional array in ascending order
 3 //Created/revised by <your name> on <current date>
 4
 5 #include <iostream>
 6 using namespace std;
 7
 8 int main()
 9 {
10    int numbers[4] = {23, 46, 12, 35};
11    int sub = 0;         //keeps track of subscripts
12    int temp = 0;        //variable used for swapping
13    int maxSub = 3;      //maximum subscript
14    int lastSwap = 0;    //position of last swap
15    char swap = 'Y';     //indicates if a swap was made
16
17    //repeat loop instructions as long as a swap was made
18    while (swap == 'Y')
19    {
20        swap = 'N';     //assume no swaps are necessary
21
22        sub = 0;        //begin comparing with first
23                        //array element
24
25        //compare adjacent array elements to determine
26        //whether a swap is necessary
27        while (sub < maxSub)
28        {
29            if (numbers[sub] > numbers[sub + 1])
30            {
31                //a swap is necessary
32                temp = numbers[sub];
33                numbers[sub] = numbers[sub + 1];
34                numbers[sub + 1] = temp;
35                swap = 'Y';
36                lastSwap = sub;
37            } //end if
38            sub += 1;  //increment subscript
39        } //end while
40
41        maxSub = lastSwap;  //reset maximum subscript
42    } //end while
43
44    //display sorted array
45    for (int x = 0; x < 4; x += 1)
46        cout << numbers[x] << endl;
47    //end for
48    return 0;
49 }   //end of main function
```

Figure 11-34 Bubble sort program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

To help you understand the bubble sort, you will desk-check the program shown in Figure 11-34. (If you already understand the bubble sort program's code, you can skip the remainder of this section.) The statements on Lines 10 through 15 create and initialize the `numbers` array and five variables. Figure 11-35 shows the desk-check table after these statements are processed.

Ch11-Bubble Sort Desk-Check

numbers[0]	numbers[1]	numbers[2]	numbers[3]
23	46	12	35

sub	temp	maxSub	lastSwap	swap
0	0	3	0	Y

Figure 11-35 Desk-check table after the declaration statements on Lines 10 through 15 are processed

The condition in the `while` clause on Line 18 compares the contents of the `swap` variable with the letter Y. The condition evaluates to true; therefore, the computer processes the instructions in the body of the loop. The first two instructions assign the letter N to the `swap` variable and assign the number 0 to the `sub` variable. The `while` clause on Line 27 begins a nested loop that repeats its instructions as long as the value in the `sub` variable is less than the value in the `maxSub` variable. At this point, the `sub` variable contains the number 0, and the `maxSub` variable contains the number 3; therefore, the computer processes the instructions in the nested loop.

The first instruction in the nested loop is an `if` statement. The statement's condition, which appears on Line 29, determines whether the value stored in the `numbers[0]` variable (23) is greater than the value stored in the `numbers[1]` variable (46). The condition evaluates to false, so the instructions in the `if` statement's true path are skipped over and processing continues with the `sub += 1;` statement on Line 38. The statement increases the value in the `sub` variable by 1; the result is 1. Figure 11-36 shows the desk-check table after the nested loop instructions are processed the first time. The new values entered in the table are shaded in the figure.

numbers[0]	numbers[1]	numbers[2]	numbers[3]
23	46	12	35

sub	temp	maxSub	lastSwap	swap
~~0~~	0	3	0	~~Y~~
~~0~~				N
1				

Figure 11-36 Desk-check table after the nested loop is processed the first time

The condition in the `while (sub < maxSub)` clause on Line 27 is evaluated next. The condition evaluates to true because the `sub` variable's value (1) is less than the `maxSub` variable's value (3). As a result, the nested loop instructions are processed again.

The `if` statement's condition in the nested loop determines whether the value stored in the `numbers[1]` variable (46) is greater than the value stored in the `numbers[2]` variable (12). The condition evaluates to true, so the instructions in the statement's true path are processed; the instructions appear on Lines 32 through 36. The first three instructions swap the values stored in the `numbers[1]` and `numbers[2]` variables. The fourth instruction assigns the letter Y to the `swap` variable to indicate that a swap was made. The last instruction in the true path

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

assigns the value stored in the `sub` variable—in this case, the number 1—to the `lastSwap` variable, which keeps track of the position of the last swap in the array. After the `if` statement ends, the `sub += 1;` statement on Line 38 increases the `sub` variable's value by 1, giving 2. Figure 11-37 shows the desk-check table after the nested loop instructions are processed the second time. The new values entered in the table are shaded in the figure.

numbers[0]	numbers[1]	numbers[2]	numbers[3]
23	~~46~~	~~12~~	35
	12	46	

sub	temp	maxSub	lastSwap	swap
~~0~~	~~0~~	3	~~0~~	~~Y~~
~~0~~	46		1	~~N~~
~~1~~				Y
2				

Figure 11-37 Desk-check table after the nested loop is processed the second time

Next, the computer evaluates the condition in the `while (sub < maxSub)` clause on Line 27. The condition evaluates to true because the `sub` variable's value (2) is less than the `maxSub` variable's value (3). Therefore, the computer once again processes the instructions in the nested loop.

The `if` statement's condition in the nested loop determines whether the value stored in the `numbers[2]` variable (46) is greater than the value stored in the `numbers[3]` variable (35). The condition evaluates to true, so the instructions in the statement's true path are processed. The first three instructions swap the values stored in the `numbers[2]` and `numbers[3]` variables. The fourth instruction assigns the letter Y to the `swap` variable to indicate that a swap was made. The last instruction in the true path assigns the value stored in the `sub` variable (2) to the `lastSwap` variable. When the `if` statement ends, the `sub += 1;` statement on Line 38 increases the `sub` variable's value by 1, giving 3. Figure 11-38 shows the desk-check table after the nested loop instructions are processed the third time. The new values entered in the table are shaded in the figure.

numbers[0]	numbers[1]	numbers[2]	numbers[3]
23	~~46~~	~~12~~	~~35~~
	12	~~46~~	46
		35	

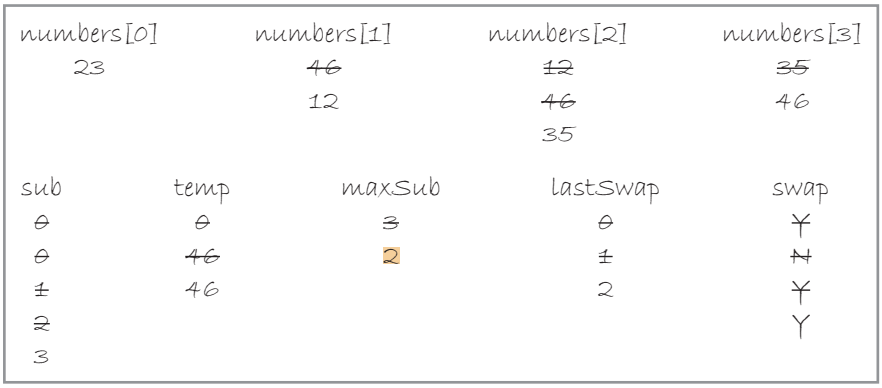
sub	temp	maxSub	lastSwap	swap
~~0~~	~~0~~	3	~~0~~	~~Y~~
~~0~~	~~46~~		~~1~~	~~N~~
~~1~~	46		2	~~Y~~
~~2~~				Y
3				

Figure 11-38 Desk-check table after the nested loop is processed the third time

The computer evaluates the condition in the `while (sub < maxSub)` clause on Line 27 next. The condition evaluates to false because the `sub` variable's value (3) is not less than the `maxSub` variable's value (3). As a result, the nested loop instructions are skipped over and processing continues with the `maxSub = lastSwap;` statement on Line 41. The statement assigns

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

the number 2 to the `maxSub` variable, and then this iteration of the outer loop ends. Figure 11-39 shows the desk-check table after the outer loop instructions are processed the first time. The new value entered in the table is shaded in the figure.

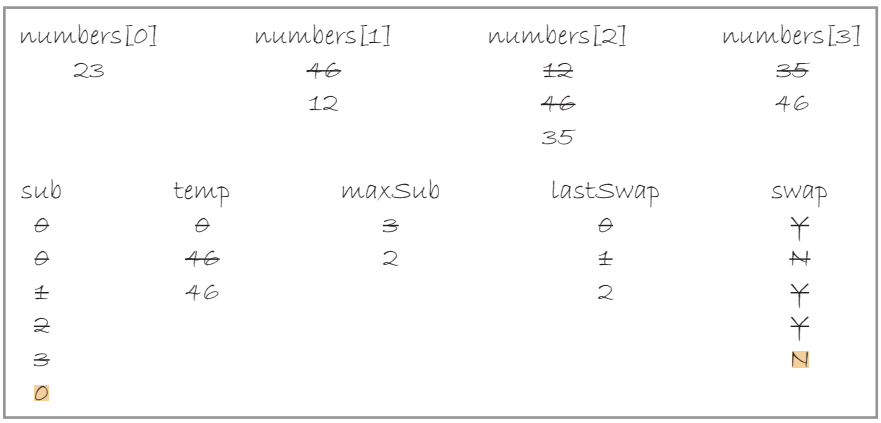


numbers[0]	numbers[1]	numbers[2]	numbers[3]
23	~~46~~	~~12~~	~~35~~
	12	~~46~~	46
		35	

sub	temp	maxSub	lastSwap	swap
~~0~~	~~0~~	~~3~~	~~0~~	~~Y~~
~~0~~	~~46~~	2	~~1~~	~~N~~
~~1~~	46		2	~~Y~~
~~2~~				Y
3				

Figure 11-39 Desk-check table after the outer loop is processed the first time

The condition in the outer loop's `while (swap == 'Y')` clause on Line 18 is processed next. The condition evaluates to true, so the computer processes the outer loop's instructions again. The first two instructions assign the letter N to the `swap` variable and assign the number 0 to the `sub` variable, as shown in Figure 11-40. The new values entered in the table are shaded in the figure.



numbers[0]	numbers[1]	numbers[2]	numbers[3]
23	~~46~~	~~12~~	~~35~~
	12	~~46~~	46
		35	

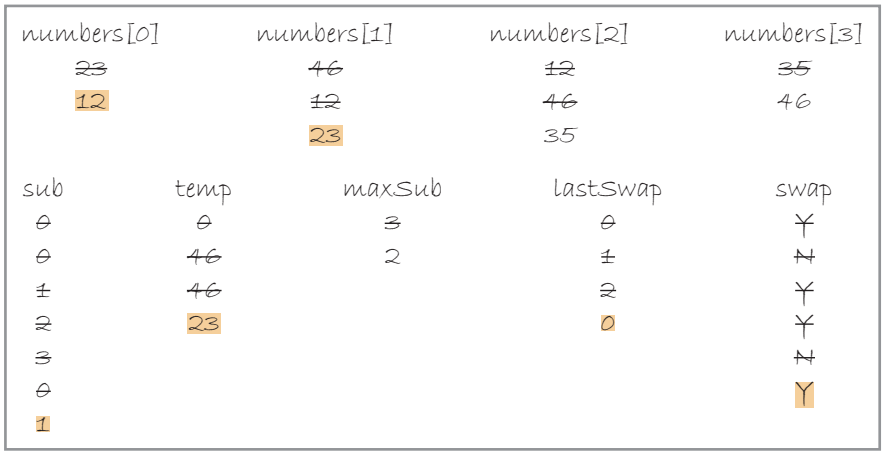
sub	temp	maxSub	lastSwap	swap
~~0~~	~~0~~	~~3~~	~~0~~	~~Y~~
~~0~~	~~46~~	2	~~1~~	~~N~~
~~1~~	46		2	~~Y~~
~~2~~				~~Y~~
~~3~~				N
0				

Figure 11-40 Desk-check table after the instructions on Lines 20 and 22 are processed

Next, the computer evaluates the condition in the nested loop's `while (sub < maxSub)` clause on Line 27. The condition evaluates to true, so the computer processes the instructions in the nested loop.

The `if` statement's condition in the nested loop determines whether the value stored in the `numbers[0]` variable (23) is greater than the value stored in the `numbers[1]` variable (12). The condition evaluates to true, so the instructions in the statement's true path are processed. The first three instructions swap the values stored in the `numbers[0]` and `numbers[1]` variables. The fourth instruction assigns the letter Y to the `swap` variable to indicate that a swap was made. The last instruction in the true path assigns the value stored in the `sub` variable (0) to the `lastSwap` variable. When the `if` statement ends, the `sub += 1;` statement on Line 38 increases the `sub` variable's value by 1, giving 1. Figure 11-41 shows the desk-check table after the instructions in the nested loop are processed. The new values entered in the table are shaded in the figure.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.



numbers[0]	numbers[1]	numbers[2]	numbers[3]
~~23~~	~~46~~	~~12~~	~~35~~
12	~~12~~	~~46~~	46
	23	35	

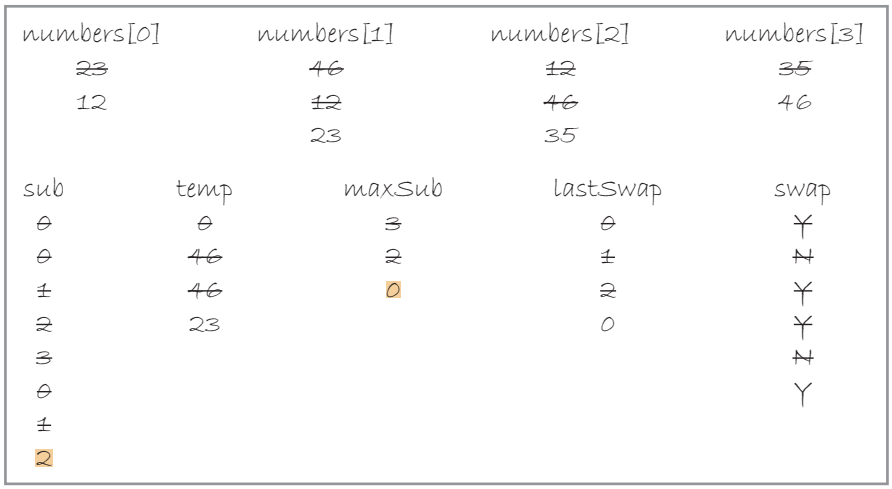
sub	temp	maxSub	lastSwap	swap
~~0~~	~~0~~	~~3~~	~~0~~	~~Y~~
~~0~~	~~46~~	2	~~1~~	~~N~~
~~1~~	~~46~~		~~2~~	~~Y~~
~~2~~	23		0	~~Y~~
~~3~~				~~N~~
~~0~~				Y
1				

Figure 11-41 Desk-check table after the instructions in the nested loop are processed

The computer evaluates the condition in the `while (sub < maxSub)` clause on Line 27 next. The condition evaluates to true because the `sub` variable's value (1) is less than the `maxSub` variable's value (2). Therefore, the computer processes the nested loop instructions once again.

The `if` statement's condition in the nested loop determines whether the value stored in the `numbers[1]` variable (23) is greater than the value stored in the `numbers[2]` variable (35). The condition evaluates to false, so the computer skips over the instructions in the `if` statement's true path. Processing continues with the `sub += 1;` statement on Line 38. The statement increments the `sub` variable's value by 1; the result is 2.

The condition in the `while (sub < maxSub)` clause on Line 27 is processed next. The condition evaluates to false because the `sub` variable's value (2) is not less than the `maxSub` variable's value (2). Because of this, the computer skips over the instructions in the nested loop. Processing continues with the `maxSub = lastSwap;` statement on Line 41. The statement assigns the number 0 to the `maxSub` variable. Figure 11-42 shows the current status of the desk-check table. The new values entered in the table are shaded in the figure.



numbers[0]	numbers[1]	numbers[2]	numbers[3]
~~23~~	~~46~~	~~12~~	~~35~~
12	~~12~~	~~46~~	46
	23	35	

sub	temp	maxSub	lastSwap	swap
~~0~~	~~0~~	~~3~~	~~0~~	~~Y~~
~~0~~	~~46~~	~~2~~	~~1~~	~~N~~
~~1~~	~~46~~	0	~~2~~	~~Y~~
~~2~~	23		0	~~Y~~
~~3~~				~~N~~
~~0~~				Y
~~1~~				
2				

Figure 11-42 Desk-check table after the instructions in the nested loop are processed again

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The computer evaluates the condition in the outer loop's `while (swap == 'Y')` clause on Line 18 next. The condition evaluates to true, so the computer processes the outer loop's instructions again. The first two instructions assign the letter N to the `swap` variable and assign the number 0 to the `sub` variable. Next, the computer evaluates the condition in the nested loop's `while (sub < maxSub)` clause on Line 27. The condition evaluates to false because the `sub` variable's value (0) is not less than the `maxSub` variable's value (0). As a result, the computer skips over the instructions in the nested loop and continues processing with the `maxSub = lastSwap;` statement on Line 41. The statement assigns the number 0 to the `maxSub` variable. Figure 11-43 shows the current status of the desk-check table. The new values entered in the table are shaded in the figure.

numbers[0]	numbers[1]	numbers[2]	numbers[3]
~~23~~	~~46~~	~~12~~	~~35~~
12	~~12~~	~~46~~	46
	23	35	

sub	temp	maxSub	lastSwap	swap
~~0~~	~~0~~	~~3~~	~~0~~	~~Y~~
~~0~~	~~46~~	~~2~~	~~1~~	~~N~~
~~1~~	~~46~~	~~0~~	~~2~~	~~Y~~
~~2~~	23	0	0	~~Y~~
~~3~~				~~N~~
~~0~~				~~Y~~
~~1~~				N
~~2~~				
0				

Figure 11-43 Current status of the desk-check table

The condition in the outer loop's `while (swap == 'Y')` clause on Line 18 is processed next and evaluates to false. Because of this, the computer skips over the instructions in the outer loop. Processing continues with the `for` clause on Line 45. The clause tells the computer to repeat the `cout << numbers[x] << endl;` statement four times: once for each element in the array. Figure 11-44 shows the result of running the bubble sort program.

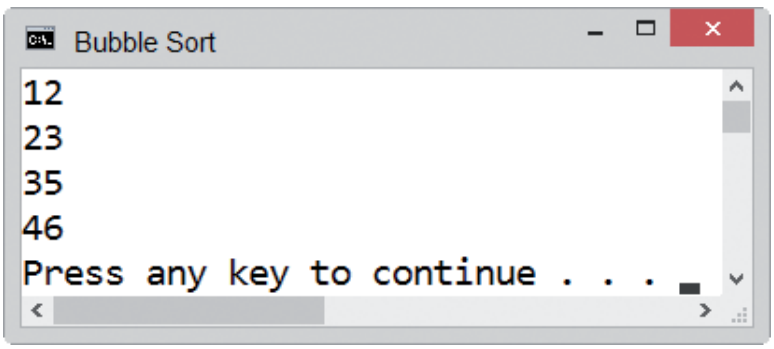


Figure 11-44 Result of running the bubble sort program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 11-3

1. Write a C++ `if` clause that determines whether the value stored in the `prices[x]` variable is less than the value stored in the `lowest` variable. The array and variable have the `double` data type.
2. The process of arranging data in alphabetical or numerical order is called _____.
3. Write a `for` loop that subtracts the number 3 from each of the 10 elements in an `int` array named `orders`. Use a variable named `x` to keep track of the array subscripts. Initialize the `x` variable to 0.

The answers to the labs are contained in the Answers.pdf file.

LAB 11-1 Stop and Analyze

Study the program shown in Figure 11-45, and then answer the questions. The `domestic` array contains the amounts the company sold domestically during the months of January through June. The `international` array contains the amounts the company sold internationally during the same period.

```
 1 //Lab11-1.cpp - calculates the total sales
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 using namespace std;
 6
 7 int main()
 8 {
 9    int domestic[6] = {75000, 30200, 67800,
10                       45000, 60000, 67500};
11    int international[6] = {40000, 75000, 64000,
12                            32600, 47800, 39000};
13    int totalSales = 0;
14
15    for (int x = 0; x < 6; x += 1)
16        totalSales += domestic[x] + international[x];
17    //end for
18    cout << "Total sales: $" << totalSales << endl;
19    return 0;
20 }  //end of main function
```

Figure 11-45 Code for Lab 11-1

QUESTIONS

1. What relationship exists between the `domestic` and `international` arrays?
2. What value is stored in the `domestic[1]` element?
3. How can you calculate the total company sales made in February?
4. What is the highest subscript in the `international` array?

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

5. If you change the `for` clause in Line 15 to `for (int x = 1; x <= 6; x += 1)`, how will the change affect the assignment statement in the `for` loop?
6. Follow the instructions for starting C++ and viewing the Lab11-1.cpp file, which is contained in either the Cpp8\Chap11\Lab11-1 Project folder or the Cpp8\Chap11 folder. (Depending on your C++ development tool, you may need to open Lab11-1's project/solution file first.) Run the program. The total company sales are $643900.
7. Modify the program so that it displays the total domestic sales, total international sales, and total company sales. Save and then run the program.
8. Now modify the program so that it also displays the total sales made in each month. Use month numbers from 1 through 6. Save and then run the program.

LAB 11-2 Plan and Create

In this lab, you will plan and create an algorithm for the problem specification shown in Figure 11-46.

Problem specification
Chris Kaplan runs five 5K races each year. Create a program that allows him to enter his finish time for each race. The program should then display both his average and lowest times. Store the five times in a one-dimensional array named `finishTimes`. Use two value-returning functions named `getAverage` and `getLowest`. Pass the array and the number of elements to each function. Display the average and lowest times with one decimal place.

Figure 11-46 Problem specification for Lab 11-2

Figure 11-47 shows the IPO chart information and corresponding C++ instructions. According to the figure, the `main` function will get the five finish times, storing each in an array element. It will then call the `getAverage` function to calculate and return the average time, which it will store in the `avgTime` variable. Next, the `main` function will call the `getLowest` function to determine the lowest time, and it will store the return value in the `lowestTime` variable. Finally, the `main` function will display the average and lowest times on the computer screen.

`main` function **IPO chart information**	**`main` function** **C++ instructions**
Input	
finish time (5 of them)	`double finishTimes[5] = {0.0};`
Processing	
subscript (counter: 0 to 4)	declared and initialized in the for clause
Output	
average finish time	`double avgTime = 0.0;`
lowest finish time	`double lowestTime = 0.0;`

Figure 11-47 IPO chart information and C++ instructions for Lab 11-2 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Algorithm
1. repeat for (subscript from 0 to 4)
enter finish time
end repeat
	`for (int x = 0; x < 5; x += 1)`
	`{`
	`cout << "Time for race "`
	`<< x + 1 << ": ";`
	`cin >> finishTimes[x];`
	`} //end for`

2. call the getAverage function to calculate the average finish time; pass the array and the number of elements	`avgTime = getAverage(finishTimes, 5);`
3. call the getLowest function to determine the lowest finish time; pass the array and the number of elements	`lowestTime = getLowest(finishTimes, 5);`

4. display the average finish time and the lowest finish time	`cout << "Average 5K finish time: "`
	`<< avgTime << endl;`
	`cout << "Lowest 5K finish time: "`
	`<< lowestTime << endl;`

getAverage function **IPO chart information**	**getAverage function** **C++ instructions**
Input	
array (formal parameter)	`double times[]`
number of elements (formal parameter)	`int numElements`
Processing	
total time	`double total = 0.0;`
subscript (counter: 0 to one less than the number of elements)	declared and initialized in the for clause
Output	
average finish time	calculated in the return statement
Algorithm	

1. repeat for (subscript from 0 to one less than the number of elements)
add array[subscript] time to the total time
end repeat
	`for (int x = 0; x < numElements;`
	`x += 1)`
	`total += times[x];`
	
	`//end for`

2. return total time / number of elements	`return total / numElements;`
getLowest function **IPO chart information**	**getLowest function** **C++ instructions**
Input	
array (formal parameter)	`double times[]`
number of elements (formal parameter)	`int numElements`
Processing	
subscript (counter: 0 to one less than the number of elements)	declared and initialized in the for clause

Figure 11-47 IPO chart information and C++ instructions for Lab 11-2 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Output

lowest finish time

Algorithm

1. repeat for (subscript from 1 to one less than the number of elements)
   if (array[subscript] time is less than lowest finish time)
      assign the array[subscript] time as the lowest finish time
   end if
   end repeat
2. return lowest finish time

```
double lowest = times[0];

for (int x = 1; x < numElements;
x += 1)
    if (times[x] < lowest)

        lowest = times[x];

    //end if
//end for
return lowest;
```

Figure 11-47 IPO chart information and C++ instructions for Lab 11-2

Figure 11-48 shows the code for the entire 5K race program, and Figure 11-49 shows the completed desk-check table for the program, assuming the user enters the following five finish times: 14.5, 15.7, 15.3, 13.1, and 14.2.

```
 1 //Lab11-2.cpp - stores finish times in an array
 2 //and displays the average and lowest times
 3 //Created/revised by <your name> on <current date>
 4
 5 #include <iostream>
 6 #include <iomanip>
 7 using namespace std;
 8
 9 //function prototypes
10 double getAverage(double times[], int numElements);
11 double getLowest(double times[], int numElements);
12
13 int main()
14 {
15    double finishTimes[5] = {0.0};
16    double avgTime = 0.0;
17    double lowestTime = 0.0;
18
19    //enter finish times
20    for (int x = 0; x < 5; x += 1)
21    {
22        cout << "Time for race " << x + 1 << ": ";
23        cin >> finishTimes[x];
24    }  //end for
25
26    avgTime = getAverage(finishTimes, 5);
27    lowestTime = getLowest(finishTimes, 5);
28
```

Figure 11-48 5K race program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
29    cout << fixed << setprecision(1) << endl;
30    cout << "Average 5K finish time: " << avgTime << endl;
31    cout << "Lowest 5K finish time: " << lowestTime << endl;
32    return 0;
33 }   //end of main function
34
35 //*****function definitions*****
36 double getAverage(double times[], int numElements)
37 {
38    double total = 0.0;
39
40    for (int x = 0; x < numElements; x += 1)
41        total += times[x];
42    //end for
43    return total / numElements;
44 }  //end of getAverage function
45
46 double getLowest(double times[], int numElements)
47 {
48    double lowest = times[0];
49    for (int x = 1; x < numElements; x += 1)
50        if (times[x] < lowest)
51            lowest = times[x];
52        //end if
53   //end for
54    return lowest;
55 } //end of getLowest function
```

Figure 11-48 5K race program

Note: The names in black indicate variables that belong to the `main` function. The names in red indicate variables that belong to the `getAverage` function. The names in blue indicate variables that belong to the `getLowest` function.

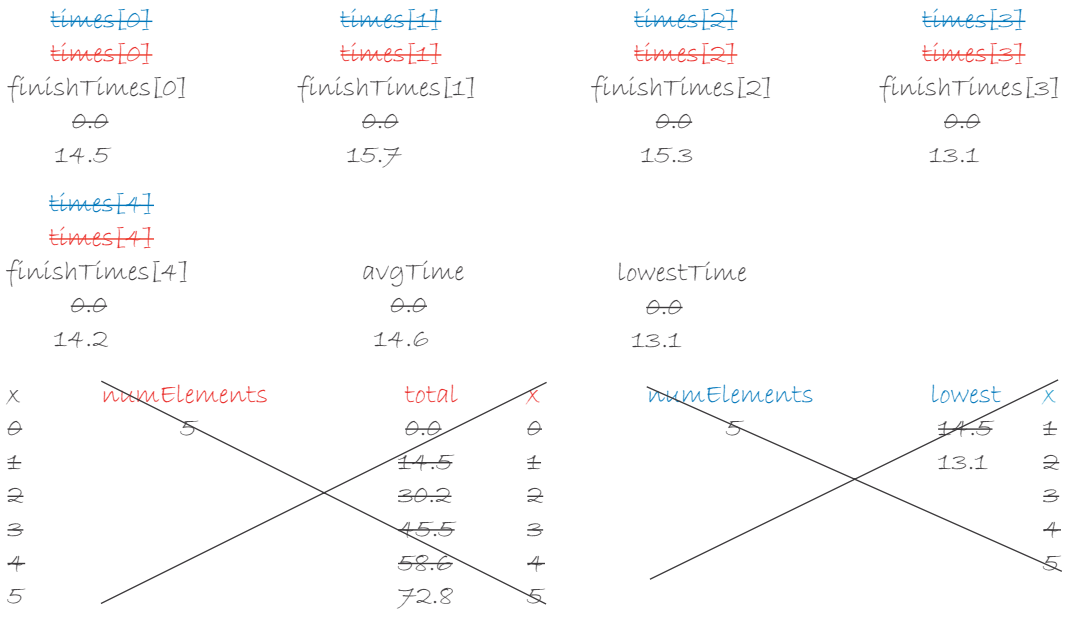


Figure 11-49 Completed desk-check table for the 5K race program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The final step in the problem-solving process is to evaluate and modify (if necessary) the program. Recall that you evaluate a program by entering its instructions into the computer and then using the computer to run (execute) it. While the program is running, you enter the same sample data used when desk-checking the program.

DIRECTIONS

Follow the instructions for starting your C++ development tool. Depending on the development tool you are using, you may need to create a new project; if so, name the project Lab11-2 Project and save it in the Cpp8\Chap11 folder. Enter the instructions shown in Figure 11-48 in a source file named Lab11-2.cpp. (Do not enter the line numbers.) Save the file in either the project folder or the Cpp8\Chap11 folder. Now follow the appropriate instructions for running the Lab11-2.cpp file. Test the program using the same data you used to desk-check the program. (The average and lowest times should be 14.6 and 13.1, respectively.) If necessary, correct any bugs (errors) in the program.

LAB 11-3 Modify

If necessary, create a new project named Lab11-3 Project and save it in the Cpp8\Chap11 folder. Enter (or copy) the Lab11-2.cpp instructions into a new source file named Lab11-3.cpp. Change Lab11-2.cpp in the first comment to Lab11-3.cpp. Change the `getAverage` and `getLowest` functions to void functions. Save and then run the program. Test the program appropriately.

LAB 11-4 What's Missing?

The program in this lab should display the average stock price. Start your C++ development tool, and view the Lab11-4.cpp file, which is contained in either the Cpp8\Chap11\Lab11-4 Project folder or the Cpp8\Chap11 folder. (Depending on your C++ development tool, you may need to open Lab11-4's project/solution file first.) Put the C++ instructions in the proper order, and then determine the one or more missing instructions. Test the program appropriately.

LAB 11-5 Desk-Check

Desk-check the code in Figure 11-50 using the data shown below. What will the `for` loop on Lines 31 through 34 display on the screen?

Student	Midterm	Final
1	90	100
2	88	68
3	77	75
4	85	85
5	45	32

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

```
 1 //Lab11-5.cpp
 2 //Stores averages in a one-dimensional array
 3 //Created/revised by <your name> on <current date>
 4
 5 #include <iostream>
 6 using namespace std;
 7
 8 int main()
 9 {
10    //declare arrays
11    double midterms[5] = {0.0};
12    double finals[5] = {0.0};
13    double averages[5] = {0.0};
14
15    //get exam scores
16    for (int x = 0; x < 5; x += 1)
17    {
18        cout << "Midterm exam score for student "
19             << x + 1 << ": ";
20        cin >> midterms[x];
21        cout << "Final exam score for student "
22             << x + 1 << ": ";
23        cin >> finals[x];
24        cout << endl;
25        //calculate and assign average
26        averages[x] = (midterms[x] + finals[x]) / 2;
27    }  //end for
28
29    //display contents of array
30    cout << endl;
31    for (int y = 0; y < 5; y += 1)
32        cout << "Student " << y + 1 << " average: "
33             << averages[y] << endl;
34    //end for
35    return 0;
36 }  //end of main function
```

Figure 11-50 Code for Lab 11-5

LAB 11-6 Debug

Follow the instructions for starting C++ and viewing the Lab11-6.cpp file, which is contained in either the Cpp8\Chap11\Lab11-6 Project folder or the Cpp8\Chap11 folder. (Depending on your C++ development tool, you may need to open Lab11-6's project/solution file first.) Debug the program.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Chapter Summary

- An array is a group of variables that have the same name and data type and are related in some way. The most commonly used arrays in programs are one-dimensional arrays and two-dimensional arrays.
- Programmers use arrays to temporarily store related data in the internal memory of the computer. By doing so, a programmer can increase the efficiency of a program because data can be both stored in and retrieved from internal memory much faster than it can be written to and read from a file on a disk. In addition, after the data is entered into an array, the program can use the data as many times as it is needed.
- You must declare an array before you can use it. After declaring an array, you can use an assignment statement or the extraction operator to enter data into the array.
- Each of the array elements in a one-dimensional array is assigned a unique number, called a subscript. The first element is assigned a subscript of 0. The second element is assigned a subscript of 1, and so on. Because the first array subscript is 0, the last subscript in a one-dimensional array is always one number less than the number of elements.
- You refer to each element in a one-dimensional array by the array's name and the element's subscript, which is specified in square brackets immediately following the name.
- When searching for either the highest or the lowest element in an array, it is a common practice to assign the first array element's value to the variable that will be used to keep track of the highest or lowest value.
- Parallel arrays are two or more arrays whose elements are related by their corresponding subscript (or position) in the arrays.
- You can use the bubble sort algorithm to sort a small amount of data stored in an array.

Key Terms

Array—a group of related variables that have the same name and data type

Bubble sort—one of many sorting algorithms used to sort small arrays; works by comparing adjacent array elements and swapping the ones that are out of order

Elements—the variables in an array

One-dimensional array—an array whose elements are identified by a unique subscript

Parallel arrays—two or more arrays whose elements are related by their corresponding position (subscript) in the arrays

Populating the array—the process of initializing the elements in an array

Scalar variable—another term for a simple variable

Simple variable—a variable that is unrelated to any other variable in the computer's internal memory; also called a scalar variable

Sorting—the process of arranging data in a specific order

Subscript—a unique number that identifies the position of an element in an array

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Review Questions

1. Which of the following is false?

   a. The elements in an array are related in some way.
   b. All of the elements in an array have the same data type.
   c. All of the elements in a one-dimensional array have the same subscript.
   d. The first element in a one-dimensional array has a subscript of 0 (zero).

2. Which of the following statements declares a five-element array named `population`?

   a. `int population[4] = {0};`
   b. `int population[5] = {0};`
   c. `int population[4] = 0`
   d. `int population[5] = {0}`

   Use the `sales` array to answer Review Questions 3 through 7. The array was declared using the `int sales[5] = {10000, 12000, 900, 500, 20000};` statement.

3. The `sales[3] += 10;` statement will replace the number _____.

   a. 500 with 10
   b. 500 with 510
   c. 900 with 10
   d. 900 with 910

4. The `sales[4] = sales[4 - 2];` statement will replace the number _____.

   a. 20000 with 900
   b. 20000 with 19998
   c. 500 with 12000
   d. 500 with 498

5. The `cout << sales[0] + sales[1] << endl;` statement will _____.

   a. display 22000
   b. display 10000 + 12000
   c. display sales[0] + sales[1]
   d. result in an error

6. Which of the following `if` clauses verifies that the array subscript stored in the `x` variable is valid for the `sales` array?

   a. `if (sales[x] >= 0 && sales[x] < 4)`
   b. `if (sales[x] >= 0 && sales[x] <= 4)`
   c. `if (x >= 0 && x < 4)`
   d. `if (x >= 0 && x <= 4)`

7. Which of the following will correctly add the number 100 to each variable in the `sales` array? The `x` variable was declared using the `int x = 0;` statement.

   a.
   ```
   while (x <= 4)
      x += 100;
   //end while
   ```
   b.
   ```
   while (x <= 4)
   {
      sales = sales + 100;
      x += 1;
   } //end while
   ```

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

c.
```
while (sales < 5)
{
   sales[x] += 100;
} //end while
```

d.
```
while (x <= 4)
{
   sales[x] += 100;
   x += 1;
} //end while
```

Use the `nums` array to answer Review Questions 8 through 12. The array was declared using the `int nums[4] = {10, 5, 7, 2};` statement. The `x` and `total` variables are `int` variables and are initialized to 0. The `avg` variable is a `double` variable and is initialized to 0.0.

8. Which of the following will correctly calculate the average of the elements included in the `nums` array?

a.
```
while (x < 4)
{
   nums[x] = total + total;
   x += 1;
} //end while
avg = static_cast<double>(total) /
    static_cast<double>(x);
```

b.
```
while (x < 4)
{
   total += nums[x];
   x += 1;
} //end while
avg = static_cast<double>(total) /
    static_cast<double>(x);
```

c.
```
while (x < 4)
{
   total += nums[x];
   x += 1;
} //end while
avg = static_cast<double>(total) /
    static_cast<double>(x) - 1;
```

d.
```
while (x < 4)
{
   total += nums[x];
   x += 1;
} //end while
avg = static_cast<double>(total) /
    static_cast<double>(x - 1);
```

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

9. What does the code in Review Question 8's answer a assign to the `avg` variable?

   a. 0.0
   b. 5.0
   c. 6.0
   d. 8.0

10. What does the code in Review Question 8's answer b assign to the `avg` variable?

    a. 0.0
    b. 5.0
    c. 6.0
    d. 8.0

11. What does the code in Review Question 8's answer c assign to the `avg` variable?

    a. 0.0
    b. 5.0
    c. 6.0
    d. 8.0

12. What does the code in Review Question 8's answer d assign to the `avg` variable?

    a. 0.0
    b. 5.0
    c. 6.0
    d. 8.0

13. If the `cities` and `zips` arrays are parallel arrays, which of the following statements will display the city name associated with the zip code stored in the `zips[8]` variable?

    a. `cout << cities[zips[8]] << endl;`
    b. `cout << cities(zips[8]) << endl;`
    c. `cout << cities[8] << endl;`
    d. `cout << cities(8) << endl;`

Exercises

Pencil and Paper

TRY THIS

1. Write the statement to declare and initialize a one-dimensional `int` array named `scores` that has 10 elements. Then write the statement to store the number 12 in the third element in the array. (The answers to TRY THIS Exercises are located at the end of the chapter.)

TRY THIS

2. Write the code to display the contents of the `scores` array from Pencil and Paper Exercise 1. Use the `for` statement with a counter variable named `x`. (The answers to TRY THIS Exercises are located at the end of the chapter.)

MODIFY THIS

3. Rewrite the code from Pencil and Paper Exercise 2 using the `while` statement.

INTRODUCTORY

4. Write the statement to declare and initialize a one-dimensional `double` array named `rates` that has five elements. Use the following numbers to initialize the array: 6.5, 8.3, 4.0, 2.0, and 10.5.

INTRODUCTORY

5. Write the code to display the contents of the `rates` array from Pencil and Paper Exercise 4. Use the `for` statement.

INTRODUCTORY

6. Rewrite the code from Pencil and Paper Exercise 5 using the `do while` statement.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

7. Write the statement to assign the C++ keyword `true` to the variable located in the third element in a one-dimensional `bool` array named `answers`. INTERMEDIATE

8. A program declares and initializes a one-dimensional `int` array named `nums`. Write the code to multiply the first element's value by 2, storing the result in the `numDoubled` variable. INTERMEDIATE

9. A program declares and initializes a one-dimensional `int` array named `nums`. Write the code to add together the numbers stored in the first and second array elements. Display the sum on the computer screen. INTERMEDIATE

10. Write the code to subtract the number 1 from each element in a one-dimensional `int` array named `quantities`. The array has 10 elements. Use the `while` statement. INTERMEDIATE

11. Rewrite the code from Pencil and Paper Exercise 10 using the `for` statement. INTERMEDIATE

12. Write the code to find the square root of the number stored in the second element in a one-dimensional `double` array named `mathNumbers`. Display the result on the computer screen. INTERMEDIATE

13. Write the code to display the smallest number stored in a one-dimensional `int` array named `orders`. The array has five elements. Use the `while` statement. ADVANCED

14. Rewrite the code from Pencil and Paper Exercise 13 using the `for` statement. ADVANCED

15. Draw a flowchart for the bubble sort program shown in Figure 11-34. ADVANCED

16. The `numbers` array is a five-element one-dimensional `int` array. The following statement should display the result of raising the first array element to the second power: `cout << pow(nums[0], 2);`. Correct the statement. SWAT THE BUGS

Computer

17. If necessary, create a new project named TryThis17 Project and save it in the Cpp8\Chap11 folder. Enter the C++ instructions shown earlier in Figure 11-19 into a source file named TryThis17.cpp. Change the filename in the first comment to TryThis17.cpp. Insert a blank line below the first comment, and then enter the following comment: `//Also displays the average number of calories consumed`. Save and then run the program. Test the program using the following calorie amounts: 1650, 1700, 1500, 2000, and 1545. Now, modify the program to include the function call and `getAverage` function shown in Figure 11-21. Display the average as a whole number. Save and then run the program. Test the program using the same data shown here. (The answers to TRY THIS Exercises are located at the end of the chapter.) TRY THIS

18. If necessary, create a new project named TryThis18 Project and save it in the Cpp8\Chap11 folder. Enter the C++ instructions shown earlier in Figure 11-23 into a source file named TryThis18.cpp. Change the filename in the first comment to TryThis18.cpp. Save, run, and test the program using the number 40. Now, modify the program to include a void function named `getSearchResults`. The function will determine the number of people whose Facebook time exceeds the number of minutes entered by the user. Replace the code on Lines 21 through 26 in the `main` function with a call to the `getSearchResults` function. (The answers to TRY THIS Exercises are located at the end of the chapter.) TRY THIS

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

MODIFY THIS

19. In this exercise, you will modify the currency converter program from the chapter. If necessary, create a new project named ModifyThis19 Project and save it in the Cpp8\Chap11 folder. Enter the C++ instructions shown earlier in Figure 11-25 into a source file named ModifyThis19.cpp. Change the filename in the first comment to ModifyThis19.cpp. Modify the program to allow the user to convert American dollars to Mexican pesos. Use 15.24 as the conversion rate. Save and then run the program. Test the program appropriately.

MODIFY THIS

20. In this exercise, you modify the highest number program from the chapter. If necessary, create a new project named ModifyThis20 Project and save it in the Cpp8\Chap11 folder. Enter the C++ instructions shown earlier in Figure 11-27 into a new source file named ModifyThis20.cpp. Change the filename in the first comment to ModifyThis20.cpp. Modify the program to also display the lowest number in the array. Use a value-returning function named `getLowest`. Save and then run the program. Test the program appropriately.

INTRODUCTORY

21. Follow the instructions for starting C++ and viewing the Introductory21.cpp file, which is contained in either the Cpp8\Chap11\Introductory21 Project folder or the Cpp8\Chap11 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) The program should calculate the average stock price stored in the `prices` array. It then should display the average price on the screen. Complete the program using the `for` statement. Save and then run the program.

INTRODUCTORY

22. Follow the instructions for starting C++ and viewing the Introductory22.cpp file, which is contained in either the Cpp8\Chap11\Introductory22 Project folder or the Cpp8\Chap11 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) The program should display the average number of pounds of coffee ordered. The numbers of pounds ordered are stored in the `pounds` array. Complete the program using the `while` statement. Save and then run the program.

INTERMEDIATE

23. Follow the instructions for starting C++ and viewing the Intermediate23.cpp file, which is contained in either the Cpp8\Chap11\Intermediate23 Project folder or the Cpp8\Chap11 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) The program uses an array to store the amount of money a game show contestant won in each of five days. The program should display the total amount won and the average daily amount won. It should also display the day number (1 through 5) corresponding to the highest amount won. Complete the program. Save and then run the program.

INTERMEDIATE

24. If necessary, create a new project named Intermediate24 Project and save it in the Cpp8\Chap11 folder. Enter the instructions shown earlier in Figure 11-23 into a source file named Intermediate24.cpp. Change the filename in the first comment to Intermediate24.cpp. Currently, the program displays the number of people whose Facebook time exceeds the number of minutes entered by the user. Modify the program to also display the average number of minutes these people spend on Facebook. Display the average with one decimal place. (Hint: Three people spend more than 95 minutes on Facebook. The average for these people is 113.3 minutes.) Save and then run the program. Test the program appropriately.

INTERMEDIATE

25. Follow the instructions for starting C++ and viewing the Intermediate25.cpp file, which is contained in either the Cpp8\Chap11\Intermediate25 Project folder or the Cpp8\Chap11 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) Code the program so that it asks the user for a percentage amount by which each price should be increased. The program should

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

increase each price in the array by that amount. For example, when the user enters the number 15, the program should increase each element's value by 15%. After increasing each price, the program should display the contents of the array. Save and then run the program. Test the program by increasing each price by 5%.

26. ADVANCED

If necessary, create a new project named Advanced26 Project and save it in the Cpp8\Chap11 folder. Also create a new source file named Advanced26.cpp. Declare a 12-element `int` array named `days`. Assign the number of days in each month to the array, using 28 for February. Code the program so that it displays the number of days corresponding to the month number entered by the user. For example, when the user enters the number 7, the program should display the number 31. However, if the user enters the number 2, the program should ask the user for the year. The rules for determining whether a year is a leap year are shown in Figure 11-51. If the year is a leap year, the program will need to add 1 to the number of days before displaying the number of days on the screen. The program should also display an appropriate message when the user enters an invalid month number. Use a sentinel value to end the program. Save and then run the program. Test the program using the number 1, and then test it using the numbers 3 through 12. Test it using the number 2 and the year 2015. Then, test it using the number 2 and the year 2016. Also test it using an invalid number, such as 20.

1. If the year number is *not* evenly divisible by 4, it is *not* a leap year.
2. If the year number *is* evenly divisible by 4 and is *not* evenly divisible by 100, then it *is* a leap year.
3. If the year number *is* evenly divisible by both 4 and 100 and *is* also evenly divisible by 400, then it *is* a leap year; otherwise, it is *not* a leap year.

Figure 11-51

27. ADVANCED

Follow the instructions for starting C++ and viewing the Advanced27.cpp file, which is contained in either the Cpp8\Chap11\Advanced27 Project folder or the Cpp8\Chap11 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) Modify the program to include a menu containing the following three options: Highest Domestic Sales, Highest International Sales, and Highest Total Monthly Sales. Also include three program-defined functions named `getDomestic`, `getInternational`, and `getMonthly` in the program. The functions can be either value-returning or void. The `getDomestic` function should determine the highest sales amount in the `domestic` array. The `getInternational` function should determine the highest sales amount in the `international` array. The `getMonthly` function should determine the highest monthly sales, as well as the month in which those sales were made. Depending on the user's menu selection, the program will display the highest domestic sales amount, the highest international sales amount, or the highest monthly total. When displaying the highest monthly total, also display the month (January through June) in which those sales were made. Save and run the program. Test the program appropriately. (Hint: The highest monthly sales are the March sales of $131800.)

28. ADVANCED

If necessary, create a new project named Advanced28 Project and save it in the Cpp8\Chap11 folder. Also create a new source file named Advanced28.cpp. The program should declare a 20-element, one-dimensional `int` array named `commission`. Assign the following 20 numbers to the array: 300, 500, 200, 150, 600, 750, 900, 150, 100, 200, 250, 650, 300, 750, 800, 350, 250, 150, 100, and 300. The program should prompt the user to enter a commission amount from 0 through 1000.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

It then should display the number of salespeople who earned that commission. Use a sentinel value to end the program. Use the application to answer the following questions:

a. How many salespeople earned a commission of 100?
b. How many salespeople earned a commission of 300?
c. How many salespeople earned a commission of 50?
d. How many salespeople earned a commission of 900?

ADVANCED

29. In this exercise, you modify the program from Computer Exercise 28. If necessary, create a new project named Advanced29 Project and save it in the Cpp8\Chap11 folder. Also create a new source file named Advanced29.cpp. Copy the C++ instructions from the Advanced28.cpp file into the Advanced29.cpp file. Change the filename in the first comment to Advanced29.cpp. Modify the program so that it prompts the user to enter a minimum commission amount and a maximum commission amount. Swap the amounts if the minimum amount is greater than the maximum amount. The program should display the number of salespeople who earned a commission within that range. Use a sentinel value to end the program. Save and then run the program. Use the program to answer the following questions.

a. How many salespeople earned a commission from 100 through 300?
b. How many salespeople earned a commission from 700 through 800?
c. How many salespeople earned a commission from 0 through 200?

ADVANCED

30. In this exercise, you create a program that generates and displays six unique random integers for a lottery game. Each lottery number can range from 1 through 54 only. If necessary, create a new project named Advanced30 Project and save it in the Cpp8\Chap11 folder. Also create a new source file named Advanced30.cpp. Create a program that generates six unique random integers from 1 through 54 and then displays the integers on the screen. Save and then run the program.

ADVANCED

31. In this exercise, you create a program that uses two parallel one-dimensional arrays. Ms. Jenkins uses the grade table shown in Figure 11-52 for her Introduction to Programming course. She wants a program that displays the grade after she enters the total points earned. If necessary, create a new project named Advanced31 Project and save it in the Cpp8\Chap11 folder. Also create a new source file named Advanced31.cpp. Store the minimum points in a one-dimensional `int` array. Store the grades in a one-dimensional `char` array. Use a sentinel value to stop the program. Save and then run the program. Test the program using the following amounts: 455, 210, 400, and 349.

Minimum points	Maximum points	Grade
0	299	F
300	349	D
350	399	C
400	449	B
450	500	A

Figure 11-52

ADVANCED

32. In this exercise, you modify the program from Computer Exercise 31. The modified program will allow the user to change the grading scale while the program is running. If necessary, create a new project named Advanced32 Project and save it in the Cpp8\Chap11 folder. Also create a new source file named Advanced32.cpp. Copy the instructions from the Advanced31.cpp file into the Advanced32.cpp file. Change the

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

filename in the first comment. Modify the program so that it allows the user to enter the total number of possible points—in other words, the total number of points a student can earn in the course—when the program is run. Also modify the program so that it uses the grading scale shown in Figure 11-53. For example, when the user enters the number 500 as the total number of possible points, the program should use 450 (which is 90% of 500) as the minimum number of points for an A. When the user enters the number 300 as the total number of possible points, the program should use 270 (which is 90% of 300) as the minimum number of points for an A. Save and then run the program. Test the program using 300 as the total number of possible points and 185 as the number of points earned. The program should display D as the grade. Close the Command Prompt window. Test the program using 500 and 363 as the total number of possible points and the total points earned, respectively. The program should display C as the grade.

Minimum points	Grade
0	F
60% of the possible points	D
70% of the possible points	C
80% of the possible points	B
90% of the possible points	A

Figure 11-53

33. ADVANCED

In this exercise, you create a program that uses two parallel one-dimensional arrays. The program displays a shipping charge that is based on the number of items ordered by a customer. The shipping charge scale is shown in Figure 11-54. If necessary, create a new project named Advanced33 Project and save it in the Cpp8\Chap11 folder. Also create a new source file named Advanced33.cpp. Store the maximum order amounts in a one-dimensional `int` array. Store the shipping charge amounts in a parallel one-dimensional `int` array. The program should allow the user to enter the number of items a customer ordered. It then should display the appropriate shipping charge. Use a sentinel value to stop the program. Save and then run the program. Test the program appropriately.

Minimum order	Maximum order	Shipping charge
1	10	15
11	50	10
51	100	5
101	99999	0

Figure 11-54

34. ADVANCED

In this exercise, you code a program that uses three parallel numeric arrays. The program searches one of the arrays and then displays the corresponding values from the other two arrays. Follow the instructions for starting C++ and viewing the Advanced34.cpp file, which is contained in either the Cpp8\Chap11\Advanced34 Project folder or the Cpp8\Chap11 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) The program should prompt the user to enter a product ID. It then should search for the product ID in the `ids` array and display the corresponding price and quantity from the `prices` and `quantities` arrays. Allow the user to display the price and quantity for as many product IDs as desired without having to execute the program again. Save and then run the program. Test the program appropriately.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

ADVANCED

35. In this exercise, you will create a program that allows the user to enter an unknown number of sales amounts for each of three car dealerships: Dealership 1, Dealership 2, and Dealership 3. Use a three-element `double` array to accumulate each dealership's sales amounts. The program should calculate the total sales and the percentage that each dealership contributed to the total sales. Display the total sales with a dollar sign and two decimal places. Display each percentage with one decimal place and a percent sign. If necessary, create a new project named Advanced35 Project and save it in the Cpp8\Chap11 folder. Enter your C++ instructions in a new source file named Advanced35.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Save and then run the program. Use the following sales amount for Dealership 1: 23000 and 15000. Use the following sales amounts for Dealership 2: 12000, 16000, 34000, and 10000. Use the following sales amounts for Dealership 3: 64000, 12000, and 70000. (Due to rounding, the percentages may not add up to exactly 100%.)

SWAT THE BUGS

36. Follow the instructions for starting C++ and viewing the SwatTheBugs36.cpp file, which is contained in either the Cpp8\Chap11\SwatTheBugs36 Project folder or the Cpp8\Chap11 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) Run the program. The program should display the total amount in inventory, but it is not working correctly. Debug the program.

Answers to TRY THIS Exercises

Pencil and Paper

1.
```
int scores[10] = {0};
scores[2] = 12;
```

2.
```
for (int x = 0; x < 10; x += 1)
   cout << scores[x] << endl;
//end for
```

Computer

17. See Figure 11-55. The changes are shaded in the figure.

```
//TryThis17.cpp - gets and displays daily calories
//Also displays the average number of calories consumed
//Created/revised by <your name> on <current date>

#include <iostream>
#include <iomanip>
using namespace std;

//function prototypes
void displayArray(int cals[], int numElements);
double getAverage(int cals[], int numElements);
```

Figure 11-55 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
int main()
{
    int calories[5] = {0};
    double average = 0.0;

    //store data in the array
    for (int sub = 0; sub < 5; sub += 1)
    {
        cout << "Calories for day " << sub + 1 << ": ";
        cin >> calories[sub];
    } //end for

    //display the contents of the array
    displayArray(calories, 5);

    //get and display the average
    average = getAverage(calories, 5);
    cout << fixed << setprecision(0);
    cout << endl << "Average number of calories consumed: "
        << average << endl;

    return 0;
} //end of main function

//*****function definitions*****
void displayArray(int cals[], int numElements)
{
    cout << endl << "Array contents:" << endl;
    for (int sub = 0; sub < 5; sub += 1)
        cout << "Calories for day " << sub + 1
            << ": " << cals[sub] << endl;
    //end for
}  //end of displayArray function

double getAverage(int cals[], int numElements)
{
    double total = 0.0;    //accumulator

    //accumulate array values
    for (int sub = 0; sub < numElements; sub += 1)
        total += cals[sub];
    //end for

    //calculate and return average
    return static_cast<double>(total) / numElements;
}  //end of getAverage function
```

Figure 11-55

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

18. See Figure 11-56. The changes are shaded in the figure.

```
//TryThis18.cpp - displays the number of people whose
//Facebook time exceeds a specific number of minutes
//Created/revised by <your name> on <current date>

#include <iostream>
using namespace std;

//function prototype
void getSearchResults(int results[], int numElements,
                      int mins, int &over);

int main()
{
    int pollResults[25] = {35, 120, 75, 60, 20,
                           25, 15, 90, 85, 35,
                           60, 15, 10, 25, 60,
                           100, 90, 10, 120, 5,
                           40, 70, 30, 25, 5};
    int minutes = 0;
    int numOver = 0;

    cout << "Search for minutes over: ";
    cin >> minutes;

    //search the array
    getSearchResults(pollResults, 25, minutes, numOver);

    cout << endl << "Number who spend more than " << minutes
        << " minutes" << endl;
    cout << "per day on Facebook: " << numOver << endl;
    return 0;
} //end of main function

//*****function definitions*****
void getSearchResults(int results[], int numElements,
                      int mins, int &over)
{
    for (int sub = 0; sub < numElements; sub += 1)
        if (results[sub] > mins)
            over += 1;
        //end if
    //end for
}  //end of getSearchResults function
```

Figure 11-56

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

CHAPTER 12

Two-Dimensional Arrays

After studying Chapter 12, you should be able to:

- Declare and initialize a two-dimensional array
- Enter data into a two-dimensional array
- Display the contents of a two-dimensional array
- Sum the values in a two-dimensional array
- Search a two-dimensional array
- Pass a two-dimensional array to a function

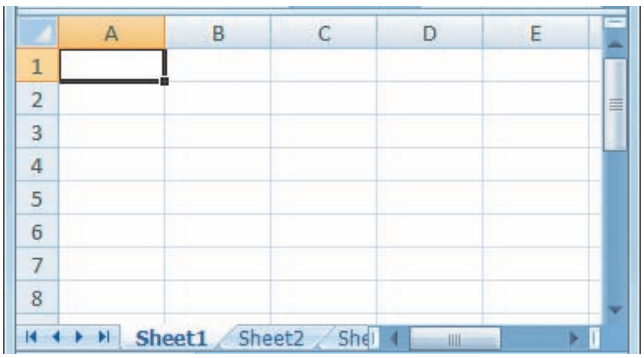
Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Using Two-Dimensional Arrays

Ch12-Chapter Preview

As you learned in Chapter 11, the most commonly used arrays in business applications are one-dimensional and two-dimensional. You can visualize a one-dimensional array as a column of variables in memory. A **two-dimensional array**, on the other hand, resembles a table in that the variables (elements) are in rows and columns, similar to a spreadsheet or a golf scorecard. See Figure 12-1.

Spreadsheet—cells are in rows and columns



Scorecard—scores are in rows and columns

Company Golf Outing Scorecard										
Hole	1	2	3	4	5	6	7	8	9	
Par	4	4	5	4	5	3	4	3	4	36
Player 1	5	4	6	4	7	4	4	5	5	44
Player 2	6	5	6	4	5	4	5	3	6	44
Player 3	4	3	5	3	4	4	5	4	4	36
Player 4	4	6	5	4	6	4	4	4	4	41

Figure 12-1 Examples of items that use the two-dimensional array concept

Recall that a subscript is also called an index.

Each element in a two-dimensional array is identified by a unique combination of two subscripts that the computer assigns to the element when the array is created. The subscripts specify the element's row and column positions in the array. Elements located in the first row in a two-dimensional array are assigned a row subscript of 0, elements in the second row are assigned a row subscript of 1, and so on. Similarly, elements located in the first column in a two-dimensional array are assigned a column subscript of 0, elements in the second column are assigned a column subscript of 1, and so on.

You refer to each element in a two-dimensional array by the array's name and the element's row and column subscripts, with the row subscript listed first and the column subscript listed second. The row subscript is enclosed in a set of square brackets (`[]`) and so is the column subscript. For example, to refer to the element located in the first row, first column in a two-dimensional array named `orders`, you use `orders[0][0]`—read "`orders` sub zero zero." Similarly, to refer to the element located in the second row, third column, you use `orders[1][2]`. Notice that the subscripts are one number less than the row and column in which the element is located. This is because the row and column subscripts start at 0 rather than at 1.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Because the subscripts start at 0, the last row subscript in a two-dimensional array will always be one number less than the number of rows in the array. Likewise, the last column subscript will always be one number less than the number of columns in the array. You can determine the number of elements in a two-dimensional array by multiplying the number of its rows by the number of its columns. An array that has four rows and three columns, for example, contains 12 elements (variables).

Declaring and Initializing a Two-Dimensional Array

You must declare (create) the two-dimensional array before you can use it in a program. You should also initialize the array elements to ensure they will not contain garbage when the program is run. Recall that assigning initial values to an array is often referred to as populating the array. You should populate an array using values that have the same data type as the array.

Figure 12-2 shows the syntax for declaring and initializing a two-dimensional array and includes examples of using the syntax. In the syntax, *arrayName* is the name of the array, and *dataType* is the type of data the array elements will store. Recall that each of the elements in an array has the same data type. The *numberOfRows* and *numberOfColumns* items, each of which is enclosed in its own set of square brackets, are integers that specify the number of rows and columns, respectively, in the array.

You can initialize the elements in a two-dimensional array by entering a separate *initialValues* section, enclosed in braces, for each row in the array. If the array has two rows, then the statement that declares and initializes the array can have a maximum of two *initialValues* sections. If the array has five rows, then the declaration statement can have a maximum of five *initialValues* sections. Within the individual *initialValues* sections, you enter one or more values separated by commas. The maximum number of values you enter corresponds to the number of columns in the array. If the array contains 10 columns, then you can include up to 10 values in each *initialValues* section. In addition to the set of braces that surrounds each individual *initialValues* section, notice in the syntax that a set of braces also surrounds *all* of the *initialValues* sections.

HOW TO Declare and Initialize a Two-Dimensional Array

Syntax

dataType arrayName[*numberOfRows*][*numberOfColumns*] =
{{*initialValues*}, {*initialValues*}, ... {*initialValues*}};

Example 1

```
char grades[3][2] = {{'C', 'A'}, {'B', 'C'}, {'D', 'B'}};
```

declares and initializes a three-row, two-column `char` array named `grades`

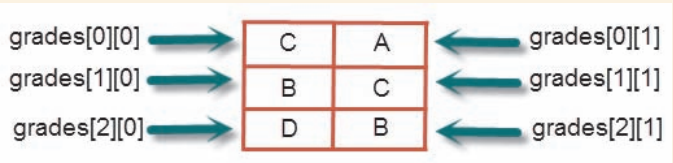


Figure 12-2 How to declare and initialize a two-dimensional array *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Example 2

```
int orders[4][3] = {0};
```

or

```
int orders[4][3] = {{0}, {0}, {0}, {0}};
```

or

```
int orders[4][3] = {{0, 0, 0}, {0, 0, 0}, {0, 0, 0}, {0, 0, 0}};
```

declares and initializes a four-row, three-column `int` array named `orders`; each element is initialized to 0

Example 3

```
double prices[6][5] = {2.0};
```

declares and initializes a six-row, five-column `double` array named `prices`; the `prices[0][0]` is initialized to 2.0; the other elements are initialized to 0.0

Figure 12-2 How to declare and initialize a two-dimensional array

The declaration statement in Example 1 in Figure 12-2 creates a two-dimensional `char` array named `grades`. The `grades` array contains three rows and two columns. The statement initializes the elements in the first row to the grades C and A, the elements in the second row to the grades B and C, and the elements in the third row to the grades D and B, as illustrated in the figure.

You can use any of the three statements shown in Example 2 to declare the two-dimensional `orders` array and initialize the elements in its four rows and three columns to the number 0. When you don't provide an initial value for each of the elements in an `int` array, many C++ compilers initialize the uninitialized elements to the integer 0.

The statement shown in Example 3 declares a two-dimensional `double` array named `prices`; the array contains six rows and five columns. The statement initializes the element located in the first row, first column of the array to 2.0. The remaining elements will be initialized to the `double` number 0.0 by many C++ compilers.

Keep in mind that if you inadvertently provide more *initialValues* sections than the number of rows in the array, or if you provide more values in an *initialValues* section than the number of columns in the array, most C++ compilers will display the error message "too many initializers" when you attempt to compile the program. However, not all C++ compilers display a message when this error occurs. Rather, some compilers store the extra values in memory locations adjacent to, but not reserved for, the array. Therefore, you must be careful to provide the appropriate number of *initialValues* sections and the appropriate number of values in each section.

If you use an invalid row or column subscript when referring to an element in a two-dimensional array, a runtime error will occur and the program will end abruptly.

Entering Data into a Two-Dimensional Array

You can use an assignment statement to enter data into the elements in a two-dimensional array, as shown in the syntax and examples in Figure 12-3. The *arrayName*[*rowSubscript*][*columnSubscript*] section in the syntax represents the name and subscripts of the element to which you want the *expression* (data) assigned. The *expression* can include any combination of constants, variables, and operators. The data type of the *expression* must match the data type of the array. If both data types do not match, the computer will perform an implicit type conversion, which could result in incorrect output.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

HOW TO Use an Assignment Statement to Assign Data to a Two-Dimensional Array

Syntax

arrayName`[`*rowSubscript*`][`*columnSubscript*`] =` *expression*`;`

Example 1

```
grades[1][0] = 'F';
```

assigns the letter F to the element located in the second row, first column in the `grades` array

Example 2

```
for (int row = 0; row < 4; row += 1)
    for (int column = 0; column < 3; column += 1)
        orders[row][column] = 0;
    //end for
//end for
```

assigns the integer 0 to each element in the four-row, three-column `orders` array, row by row; provides another means of initializing the array

Example 3

```
int row = 0;
int column = 0;
double oldPrice = 0.0;
const double INCREASE = 1.15;
while (column < 5)
{
    while (row < 6)
    {
        cout << "Price: ";
        cin >> oldPrice;
        prices[row][column] = oldPrice * INCREASE;
        row += 1;
    } //end while
    column += 1;
    row = 0;
} //end while
```

assigns the new price to each element in the six-row, five-column `prices` array, column by column; the new price is calculated by multiplying the old price by the value stored in the `INCREASE` named constant

You can also use the C++ increment operator (++) to add 1 to a variable. For instance, you can use `row++` and `column++` in Example 2's `for` clauses, and use `row++;` and `column++;` in Example 3.

Figure 12-3 How to use an assignment statement to assign data to a two-dimensional array

The examples included in Figure 12-3 show various ways of assigning data to the arrays declared earlier in Figure 12-2. The assignment statement in Example 1 assigns the letter F to the element located in the second row, first column in the `grades` array, replacing the letter B that was stored in the element when the array was declared.

The code in Example 2 assigns the integer 0 to each of the 12 elements in the `orders` array and provides another means of initializing the array. Notice that the code uses two loops to access each element in the array. One of the loops keeps track of the row subscript, while the other loop keeps track of the column subscript. The code assigns the integer 0 to the array, row by row. In other words, it assigns 0 to each element in the first row before assigning 0 to each element in the second row, and so on.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The code in Example 3 assigns a new price to each of the elements in the `prices` array. The new price is calculated by the assignment statement within the nested loop. The statement multiplies the old price by the contents of the `INCREASE` named constant and then assigns the result to the current array element. Like the code in Example 2, the code in Example 3 uses two loops to access each element in the array. However, unlike the code in Example 2, the code in Example 3 assigns values to the array, column by column, rather than row by row. This means that the code will assign values to each element in the first column before assigning values to each element in the second column, and so on.

You can also use the extraction operator to store data in the elements in a two-dimensional array. Figure 12-4 shows the syntax and examples of doing this using the arrays declared earlier in Figure 12-2.

In Example 2's `for` clauses, you can use `region++` and `month++`. In Example 3, you can use `row++` in the `for` clause and use `column++;` in the `while` loop.

HOW TO Use the Extraction Operator to Store Data in a Two-Dimensional Array

Syntax

```
cin >> arrayName[subscript][subscript];
```

Example 1

```
cin >> grades[2][1];
```

stores the user's entry in the element located in the third row, second column in the `grades` array

Example 2

```
for (int region = 0; region < 4; region += 1)
    for (int month = 0; month < 3; month += 1)
    {
        cout << "Number of orders for Region "
             << region + 1 << ", Month "
             << month + 1 << ": ";
        cin >> orders[region][month];
    } //end for
//end for
```

stores the user's entries in the four-row, three-column `orders` array, region (row) by region (row)

Example 3

```
int column = 0;
while (column < 5)
{
    for (int row = 0; row < 6; row += 1)
    {
        cout << "Price: ";
        cin >> prices[row][column];
    } //end for
    column += 1;
} //end while
```

stores the user's entries in the six-row, five-column `prices` array, column by column

Figure 12-4 How to use the extraction operator to store data in a two-dimensional array

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The `cin` statement in Example 1 in Figure 12-4 stores the user's entry in the element located in the third row, second column in the `grades` array, replacing the element's existing data. The code in Example 2 contains two `for` loops. The instructions in the outer loop will be repeated once for each of the four regions, while the instructions in the nested loop will be repeated once for each of the three months within each region. The `cout` statement in the nested loop prompts the user to enter the number of orders for the current region and month. The `cin` statement stores the user's response in the current element in the `orders` array. The responses will be stored, region (row) by region (row), in the array. In other words, the three monthly sales for Region (row) 1 will be stored before the three monthly sales for Region (row) 2, and so on.

Example 3 contains an outer `while` loop and a nested `for` loop. The `while` loop repeats its instructions for each of the five columns in the array, and the `for` loop repeats its instructions for each of the six rows in the array. The `cout` statement in the `for` loop prompts the user to enter a price, and the `cin` statement stores the user's response in the current element in the `prices` array. The responses will be stored, column by column, in the array. In other words, the six rows in the first column will be filled with prices before the six rows in the second column are filled, and so on.

Displaying the Contents of a Two-Dimensional Array

To display the contents of a two-dimensional array, you need to access each of its elements. You do this using two counter-controlled loops: one to keep track of the row subscript and one to keep track of the column subscript. Figure 12-5 shows examples of loops you can use to display the contents of the arrays declared earlier in Figure 12-2. Example 1 uses two `while` loops to display the contents of the `grades` array, column by column. The `grades` array contains three rows and two columns. Example 2 uses two `for` loops to display the contents of the four-row, three-column `orders` array, region (row) by region (row). Example 3 uses both a `do while` loop and a `for` loop to display the contents of the six-row, five-column `prices` array, row by row.

HOW TO Display the Contents of a Two-Dimensional Array

Example 1

```
int row = 0;
int column = 0;
while (column < 2)
{
    while (row < 3)
    {
        cout << grades[row][column] << endl;
        row += 1;
    } //end while
    column += 1;
    row = 0;
} //end while
```

displays the contents of the three-row, two column `grades` array, column by column

You use one loop to access each element in a one-dimensional array, but two loops to access each element in a two-dimensional array.

You can use `row++;` and `column++;` in Example 1, `region++` and `month++` in Example 2, and `column++` and `row++;` in Example 3.

Figure 12-5 How to display the contents of a two-dimensional array *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Example 2

```
for (int region = 0; region < 4; region += 1)
    for (int month = 0; month < 3; month += 1)
        cout << orders[region][month] << endl;
    //end for
//end for
```

displays the contents of the four-row, three column `orders` array, region (row) by region (row)

Example 3

```
int row = 0;
do     //begin loop
{
    for (int column = 0; column < 5; column += 1)
        cout << prices[row][column] << endl;
    //end for
    row += 1;
} while (row < 6);
```

displays the contents of the six-row, five-column `prices` array, row by row

Figure 12-5 How to display the contents of a two-dimensional array

The Chapton Company Program

Figure 12-6 shows the problem specification and C++ code for the Chapton Company program. The program uses a 12-element, two-dimensional array to store the 12 order amounts entered by the user. It then displays the order amounts by month within each of the company's four regions. The figure also shows a sample run of the program.

Problem specification
Create a program for the Chapton Company. The program should allow the company's sales manager to enter the number of orders received from each of the company's four sales regions during the first three months of the year. Store the order amounts in a two-dimensional `int` array that contains four rows and three columns. Each row in the array represents a region, and each column represents a month. After the sales manager enters the 12 order amounts, the program should display the amounts on the computer screen. The order amounts for Region 1 should be displayed first, followed by Region 2's order amounts, and so on.

```
 1 //Chapton Company.cpp - gets and displays order amounts
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 using namespace std;
 6
 7 int main()
 8 {
 9    int orders[4][3] = {0};
10
```

array declaration

Figure 12-6 Chapton Company problem specification and program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
11    //store data in the array
12    for (int region = 0; region < 4; region += 1)
13        for (int month = 0; month < 3; month += 1)
14        {
15            cout << "Region " << region + 1
16                << ", Month " << month + 1
17                << " orders: ";
18            cin >> orders[region][month];
19        } //end for
20    //end for
21
22    //display the contents of the array
23    cout << endl << "Array contents:" << endl;
24    for (int region = 0; region < 4; region += 1)
25    {
26        cout << "Region " << region + 1
27            << ": " << endl;
28        for (int month = 0; month < 3; month += 1)
29            cout << "     Month " << month + 1
30                << ": " << orders[region][month]
31                << endl;
32        //end for
33    } //end for
34    return 0;
35 } //end of main function
```

stores data in the array

displays the contents of the array

```
Chapton Company
Region 1, Month 1 orders: 23500
Region 1, Month 2 orders: 22000
Region 1, Month 3 orders: 22700
Region 2, Month 1 orders: 34000
Region 2, Month 2 orders: 35600
Region 2, Month 3 orders: 32000
Region 3, Month 1 orders: 56000
Region 3, Month 2 orders: 49600
Region 3, Month 3 orders: 43500
Region 4, Month 1 orders: 15400
Region 4, Month 2 orders: 16900
Region 4, Month 3 orders: 15800

Array contents:
Region 1:
     Month 1: 23500
     Month 2: 22000
     Month 3: 22700
Region 2:
     Month 1: 34000
     Month 2: 35600
     Month 3: 32000
Region 3:
     Month 1: 56000
     Month 2: 49600
     Month 3: 43500
Region 4:
     Month 1: 15400
     Month 2: 16900
     Month 3: 15800
Press any key to continue . . .
```

the array elements are displayed region (row) by region (row)

You can use `region++` in Lines 12 and 24, and use `month++` in Lines 13 and 28.

Figure 12-6 Chapton Company problem specification and program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 12-1

1. Which of the following declares a six-row, three-column `int` array named `quantities` and initializes each of its elements to the number 0?

   a. `int quantities[6][3] = {0};`
   b. `int quantities[3][6] = {0};`
   c. `int quantities{3}{6} = [0];`
   d. `int quantities{6}{3} = [0];`

2. How many elements are contained in the `quantities` array from Question 1?
3. What are the name and subscripts of the first element in the `quantities` array from Question 1?
4. What are the name and subscripts of the last element in the `quantities` array from Question 1?
5. Write a C++ statement that assigns the number 20 to the element located in the first column, second row in the `quantities` array from Question 1.

Accumulating the Values Stored in a Two-Dimensional Array

Figure 12-7 shows the problem specification, IPO chart information, and C++ instructions for the Jenko Booksellers program. The program's flowchart is also shown in the figure. The program uses a two-dimensional array to store the sales made in each of the company's three bookstores. The array contains three rows and two columns. The first column in the array contains the sales amounts for paperback books sold in each of the three stores. The second column contains the sales amounts for hardcover books. The program calculates the total sales by accumulating the amounts stored in the array. It then displays the total sales on the computer screen. Figure 12-8 shows the code for the entire program and includes the result of running the program.

Problem specification
Jenko Booksellers wants a program that calculates and displays the total of its previous month's sales. The program should store the sales amounts, which are shown here, in a two-dimensional `double` array named `sales`. The array should have three rows (one row for each of the three stores) and two columns. The first column should contain the sales amounts for paperback books sold in each of the three stores. The second column should contain the sales amounts for hardcover books sold in each of the three stores.

	Paperback sales ($)	Hardcover sales ($)
Store 1	3567.85	2589.99
Store 2	3239.67	2785.55
Store 3	1530.50	1445.80

Figure 12-7 Problem specification, IPO chart information (including flowchart), and C++ instructions for the Jenko Booksellers program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

IPO chart information	C++ instructions
Input	
store sales (made in each of 3 stores for 2 types of books)	the sales are stored in the array
Processing	

array (3 store rows, 2 book columns)	`double sales[3][2] = {{3567.85, 2589.99},`
	`{3239.67, 2785.55},`
	`{1530.50, 1445.80}};`

store subscript (counter: 0 to 2)	declared and initialized in the for clause
book subscript (counter: 0 to 1)	declared and initialized in the for clause
Output	
total sales (accumulator)	`double total = 0.0;`
Algorithm	

1. repeat for (store subscript from 0 to 2)
repeat for (book subscript from 0 to 1)
add the sales stored in array[store subscript][book subscript] to the total sales
end repeat
end repeat
	`for (int store = 0; store < 3;`
	`store += 1)`
	`for (int book = 0; book < 2;`
	`book += 1)`
	`total += sales[store][book];`
	`//end for`
	`//end for`

2. display the total sales	`cout << "Total sales: $" << total` `<< endl;`

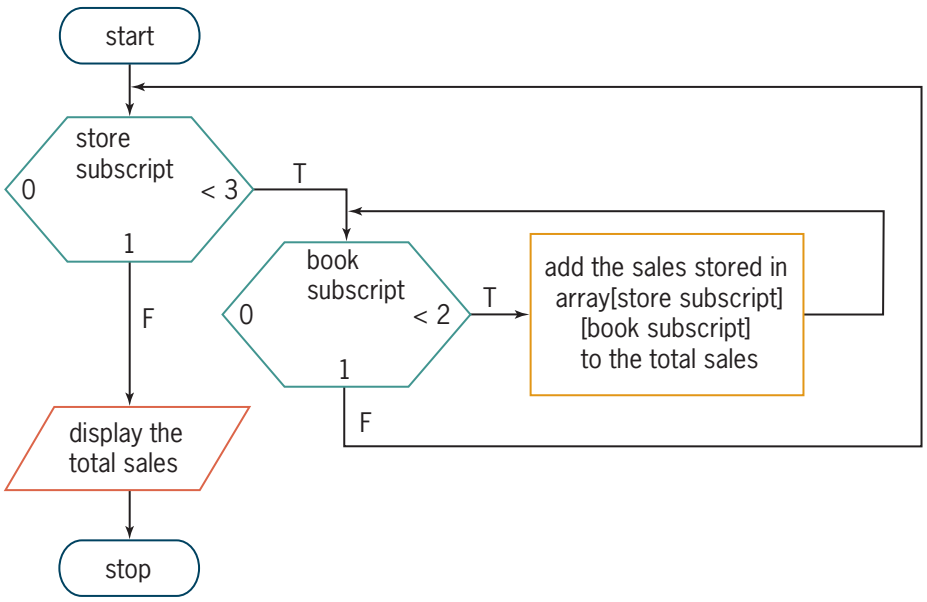


Figure 12-7 Problem specification, IPO chart information (including flowchart), and C++ instructions for the Jenko Booksellers program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

You can use `store++` in Line 16 and use `book++` in Line 17.

```
 1 //Jenko Booksellers.cpp - displays the total sales
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <iomanip>
 6 using namespace std;
 7
 8 int main()
 9 {
10    double sales[3][2] = {{3567.85, 2589.99},
11                          {3239.67, 2785.55},
12                          {1530.50, 1445.80}};
13    double total = 0.0;    //accumulator
14
15    //accumulate sales
16    for (int store = 0; store < 3; store += 1)
17        for (int book = 0; book < 2; book += 1)
18            total += sales[store][book];
19        //end for
20    //end for
21
22    cout << fixed << setprecision(2);
23    cout << "Total sales: $" << total << endl;
24    return 0;
25 }  //end of main function
```

array declaration

accumulates the sales stored in the array

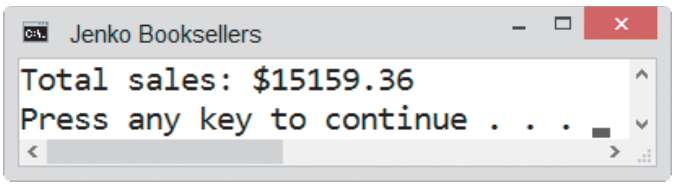


Figure 12-8 Jenko Booksellers program

Searching a Two-Dimensional Array

Figure 12-9 shows the problem specification, IPO chart information, and C++ instructions for the Wilson Company program. The program's flowchart is also shown in the figure. The program uses a four-row, two-column array to store the company's four pay codes and their corresponding pay rates. The pay codes are stored in the first column of each row in the array. The pay rate associated with each code is stored in the same row as its pay code but in the second column. The program gets a pay code from the user and then searches for the pay code in the array's first column. If it finds the pay code, the program displays the corresponding pay rate from the array's second column; otherwise, it displays the "Invalid pay code" message.

Problem specification
Wilson Company wants a program that displays the pay rate corresponding to the pay code entered by the user. The program should store the pay codes and rates, which are listed here, in a two-dimensional `int` array named `codesAndRates`. The array should have four rows (one row for each of the four pay codes) and two columns. The first column should contain the four pay codes, and the second column should contain each code's corresponding rate.

Figure 12-9 Problem specification, IPO chart information, and C++ instructions for the Wilson Company program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Pay code	Pay rate
3	8
6	14
7	18
9	20

IPO chart information

Input

4 pay codes and their pay rates

pay code to search for

Processing

array (4 rows, 2 columns)

row subscript (counter: 0 to 3)

Output

pay rate

Algorithm

1. enter the pay code to search for

2. repeat while (the pay code to search for is greater than or equal to 0)
   assign 0 as the row subscript to begin searching the array with the first row

   repeat while (the row subscript is less than or equal to 3 and the pay code stored in array[row subscript][0] is not the pay code to search for)
   add 1 to the row subscript to continue the search with the next row
   end repeat
   if (the row subscript is less than or equal to 3)
   display the pay code and the pay rate stored in array[row subscript][1]

C++ instructions

```
the pay codes and pay rates are stored in the array
int payCode = 0;

int codesAndRates[4][2] = {{3, 8},
{6, 14}, {7, 18}, {9, 20}};

int row = 0;

displayed from the array

cout << "Pay code (3, 6, 7, or 9).
" << endl;
cout << "Enter a negative number
to end: ";
cin >> payCode;
while (payCode >= 0)
{

    row = 0;

    while (row <= 3 &&
    codesAndRates[row][0]
    != payCode)

        row += 1;

     //end while
     if (row <= 3)

         cout << "Pay rate for pay
         code "
         << payCode << ": $"
         << codesAndRates[row][1]
         << endl << endl;
```

Figure 12-9 Problem specification, IPO chart information, and C++ instructions for the Wilson Company program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
else                                        else
      display "Invalid pay code"                cout << "Invalid pay code"
end if                                          << endl << endl;
                                            //end if
                                            cout << "Pay code (3, 6, 7,
                                            or 9). " << endl;
enter the pay code to search for            cout << "Enter a negative
                                            number to end: ";
                                             cin >> payCode;
end while                                   } //end while
```

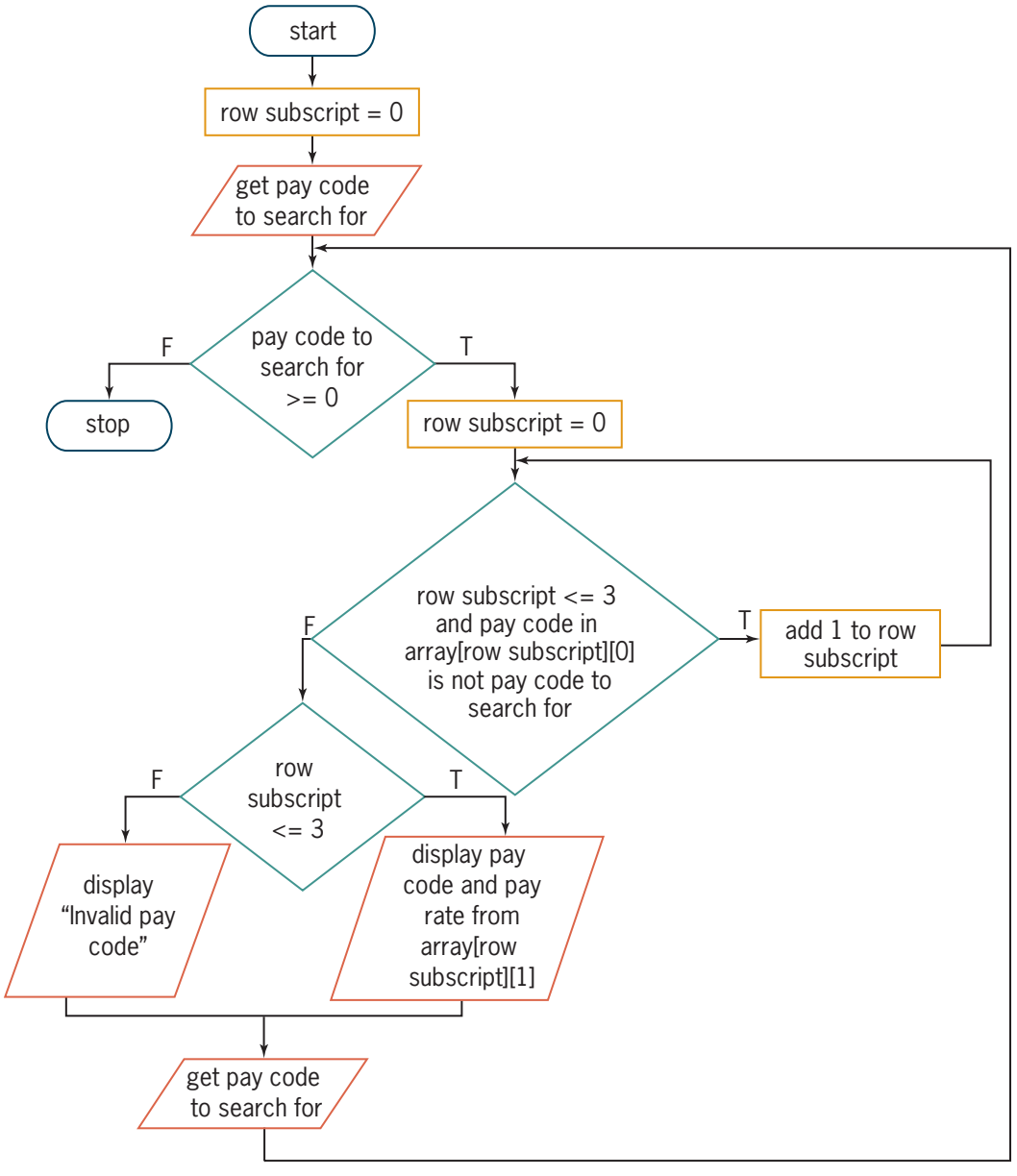


Figure 12-9 Problem specification, IPO chart information, and C++ instructions for the Wilson Company program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Figure 12-10 shows the code for the entire Wilson Company program and includes the result of running the program. As the figure shows, the program displays $14 as the pay rate when the user enters the number 6 as the pay code. This is because the number 6 is contained in the `codesAndRates[1][0]` element, and its corresponding pay rate (14) is contained in the `codesAndRates[1][1]` element. Notice that the pay code and its associated pay rate are contained in the same row but in different columns. As the figure also shows, the program displays the "Invalid pay code" message when the user enters the number 5 as the pay code. This is because the number 5 does not appear in the first column in the array.

```
 1 //Wilson Company.cpp - displays the pay rate
 2 //corresponding to the pay code entered by the user
 3 //Created/revised by <your name> on <current date>
 4
 5 #include <iostream>
 6 using namespace std;
 7
 8 int main()
 9 {
10    //declare array and variables
11    int codesAndRates[4][2] = {{3, 8},
12                               {6, 14},
13                               {7, 18},
14                               {9, 20}};
15    int payCode = 0;
16    int row = 0;
17
18    //get pay code
19    cout << "Pay code (3, 6, 7, or 9). " << endl;
20    cout << "Enter a negative number to end: ";
21    cin >> payCode;
22
23    while (payCode >= 0)
24    {
25        //search each row in the array, looking
26        //for the pay code in the first column
27        //continue the search while there are
28        //array elements to search and the pay
29        //code has not been found
30        row = 0;
31        while (row <= 3
32            && codesAndRates[row][0] != payCode)
33                row += 1;
34        //end while
35
36        //if the pay code was found, display the
37        //pay code and the pay rate stored in the
38        //same row as the pay code but in the
39        //second column of the array
40        if (row <= 3)
41            cout << "Pay rate for pay code "
42                << payCode << ": $"
43                << codesAndRates[row][1]
44                << endl << endl;
```

Figure 12-10 Wilson Company program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
45        else
46            cout << "Invalid pay code" << endl << endl;
47        //end if
48
49        //get pay code
50        cout << "Pay code (3, 6, 7, or 9). " << endl;
51        cout << "Enter a negative number to end: ";
52        cin >> payCode;
53    }    //end while
54    return 0;
55 } //end of main function
```

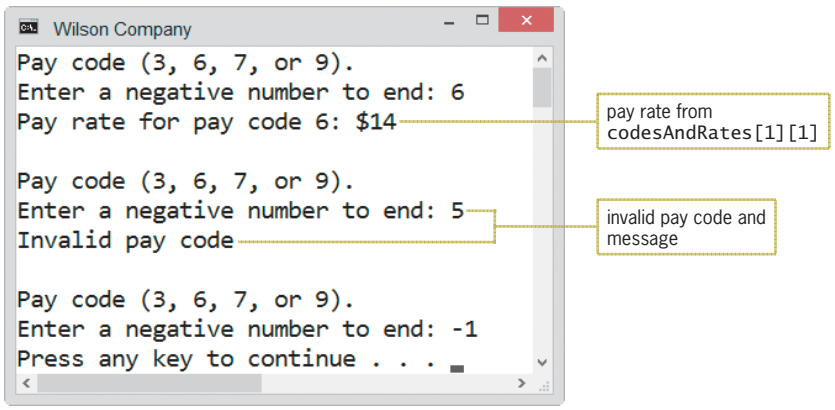


Figure 12-10 Wilson Company program

Ch12-Wilson Desk-Check

You will desk-check the program shown in Figure 12-10 to observe how it searches the array. The declaration statements on Lines 11 through 16 create and initialize the two-dimensional array and two variables. The statements on Lines 19 through 21 prompt the user to enter a pay code and then store the user's response in the **payCode** variable. Figure 12-11 shows the desk-check table after the statements on Lines 11 through 21 are processed, assuming the user enters the number 6 as the pay code.

codesAndRates[0][0]	codesAndRates[0][1]
3	8
codesAndRates[1][0]	codesAndRates[1][1]
6	14
codesAndRates[2][0]	codesAndRates[2][1]
7	18
codesAndRates[3][0]	codesAndRates[3][1]
9	20
payCode	row
0	0
6	

Figure 12-11 Desk-check table after the statements on Lines 11 through 21 are processed

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The `while` clause in the program's outer loop (on Line 23) is processed next. The clause's condition evaluates to true because the value in the `payCode` variable is greater than or equal to the number 0. Therefore, the computer processes the outer loop's instructions.

The instructions on Lines 30 through 34 in the outer loop search for the pay code in the first column of the array; study these instructions closely. The instruction on Line 30 assigns the number 0 to the `row` variable to ensure that the search will begin in the first row. The `while` clause on Lines 31 and 32 marks the beginning of a nested loop. The clause's compound condition evaluates to true because both of its subconditions evaluate to true. The first subcondition evaluates to true because the value in the `row` variable (0) is less than or equal to 3 (the highest row subscript in the array). The second subcondition evaluates to true because the value in the `codesAndRates[0][0]` element (3) is not equal to the value in the `payCode` variable (6). As a result, the `row += 1;` statement on Line 33 adds the number 1 to the contents of the `row` variable, giving 1. Incrementing the `row` variable by 1 allows the computer to search the next row in the array. Figure 12-12 shows the desk-check table after the nested loop is processed the first time.

codesAndRates[0][0]	codesAndRates[0][1]
3	8
codesAndRates[1][0]	codesAndRates[1][1]
6	14
codesAndRates[2][0]	codesAndRates[2][1]
7	18
codesAndRates[3][0]	codesAndRates[3][1]
9	20
payCode	row
0	0
6	0
	1

Figure 12-12 Desk-check table after the nested loop is processed the first time

The compound condition in the nested loop's `while` clause (on Lines 31 and 32) is evaluated again. This time, the compound condition evaluates to false because the value in the `codesAndRates[1][0]` element (6) *is* equal to the value in the `payCode` variable (6). At this point, the nested loop ends and processing continues with the `if` clause on Line 40.

The `if` clause's condition evaluates to true because the value in the `row` variable (1) is less than or equal to 3 (the highest row subscript in the array). Therefore, the computer processes the `cout` statement that appears on Lines 41 through 44. The statement displays a message containing the pay code stored in the `payCode` variable (6) and the pay rate stored in the `codesAndRates[1][1]` element (14), as shown earlier in Figure 12-10. After the message is displayed, the `if` statement ends.

The statements on Lines 50 through 52 prompt the user to enter another pay code and then store the user's response in the `payCode` variable. This time, the user enters the number 5. The computer evaluates the condition in the outer loop's `while` clause (on Line 23) next. The condition evaluates to true because the value in the `payCode` variable (5) is greater than or equal to the number 0. Therefore, the computer processes the outer loop's instructions.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The instruction on Line 30 in the outer loop assigns the number 0 to the `row` variable to ensure that this new search will begin in the first row. The nested `while` clause on Lines 31 and 32 tells the computer to process the `row += 1;` statement as long as the value in the `row` variable is less than or equal to 3 (the highest row subscript in the array) and the current array element does not contain the pay code stored in the `payCode` variable (5). The nested loop will stop when either of the following occurs: the `row` variable contains the number 4 (which indicates that the nested loop reached the end of the array without finding the pay code) or the pay code is located in the array's first column.

Figure 12-13 shows the desk-check table after the nested loop ends. Unlike the nested loop in the previous search, which stopped when the pay code of 6 was located, the nested loop in this search stops when the `row` variable contains the number 4.

codesAndRates[0][0]	codesAndRates[0][1]
3	8
codesAndRates[1][0]	codesAndRates[1][1]
6	14
codesAndRates[2][0]	codesAndRates[2][1]
7	18
codesAndRates[3][0]	codesAndRates[3][1]
9	20
payCode	row
0	0
6	0
5	1
	0
	1
	2
	3
	4

Figure 12-13 Desk-check table after the nested loop ends and the pay code is not located

When the nested loop ends, processing continues with the `if` clause on Line 40. The `if` clause's condition evaluates to false because the value in the `row` variable (4) is not less than or equal to 3. This indicates that the nested loop stopped processing because it reached the end of the array's first column without finding the pay code. Therefore, the `cout` statement on Line 46 displays the "Invalid pay code" message, as shown earlier in Figure 12-10, and then the `if` statement ends.

For more examples of two-dimensional arrays, see the Two-Dimensional Arrays section in the Ch12WantMore.pdf file.

Next, the statements on Lines 50 through 52 prompt the user to enter another pay code and then store the user's response in the `payCode` variable. This time the user enters the number –1. The computer evaluates the condition in the outer loop's `while` clause (on Line 23) next. The condition evaluates to false because the value in the `payCode` variable (–1) is not greater than or equal to the number 0. As a result, the outer loop ends and the computer processes the `return 0;` statement on Line 54. After the `return` statement is processed, the program ends and the computer removes the array and the two scalar (simple) variables from internal memory.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Passing a Two-Dimensional Array to a Function

Figure 12-14 shows a modified version of the Chapton Company program, which you viewed earlier in Figure 12-6. In the modified version, the `main` function passes the `orders` array to a program-defined void function named `displayArray`.

When passing a two-dimensional array, the first set of square brackets in its corresponding formal parameter can be empty, like this: `[]`. This concept is covered in Computer Exercise 17 at the end of the chapter.

You can use `region++` in Lines 15 and 34, and use `month++` in Lines 16 and 38.

```
 1 //Modified Chapton Company.cpp - gets and displays order amounts
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 using namespace std;
 6
 7 //function prototype
 8 void displayArray(int nums[4][3]);
 9
10 int main()
11 {
12    int orders[4][3] = {0};
13
14    //store data in the array
15    for (int region = 0; region < 4; region += 1)
16        for (int month = 0; month < 3; month += 1)
17        {
18            cout << "Region " << region + 1
19                << ", Month " << month + 1
20                << " orders: ";
21            cin >> orders[region][month];
22        } //end for
23      //end for
24
25    //display the contents of the array
26    displayArray(orders);
27    return 0;
28 } //end of main function
29
30 //*****function definitions*****
31 void displayArray(int nums[4][3])
32 {
33    cout << endl << "Array contents:" << endl;
34    for (int region = 0; region < 4; region += 1)
35    {
36        cout << "Region " << region + 1
37            << ": " << endl;
38        for (int month = 0; month < 3; month += 1)
39            cout << "     Month " << month + 1
40            << ": " << nums[region][month]
41            << endl;
42        //end for
43    } //end for
44 }  //end of displayArray function
```

Figure 12-14 Modified Chapton Company program

Study closely the `displayArray` function prototype, function call, and function header; each is shaded in Figure 12-14. The function call appears on Line 26 and passes one actual argument to the `displayArray` function: the `orders` array. Like one-dimensional arrays, two-dimensional

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

arrays are passed automatically *by reference*. The `displayArray` function prototype and function header appear on Lines 8 and 31, respectively; both contain one formal parameter: `nums[4][3]`. The first set of square brackets that follows the formal parameter's name contains the number of rows in the array; the second set contains the number of columns. Recall that the formal parameter's name is optional in the prototype. Therefore, you could also write the formal parameter in the function prototype as `int [4][3]`.

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 12-2

1. Which of the following increases the `total` variable by the contents of the element located in the third row, second column of the `purchases` array?

   a. `purchases[2][1] += total;`
   b. `purchases[1][2] += total;`
   c. `total += purchases[2][1];`
   d. `total += purchases[1][0];`

2. Which of the following `if` clauses determines whether the value stored in the third column, second row in the `scores` array is greater than 25?

   a. `if (scores[1, 3] > 25)`
   b. `if (scores[2, 1] > 25)`
   c. `if (scores[1][2] > 25)`
   d. `if (scores[2][3] > 25)`

3. Write a C++ statement that multiplies the contents of the element located in the first row, second column in the `sales` array by 0.15 and then stores the result in the `bonus` variable. The `sales` array and `bonus` variable have the `double` data type.

4. Which of the following determines whether the `row` variable contains a valid subscript for an array that has 10 rows and 20 columns.

   a. `if (row >= 0 && row < 10)`
   b. `if (row >= 0 && row <= 10)`
   c. `if (row >= 0 && row < 9)`
   d. `if (row > 0 && row <= 10)`

The answers to the labs are contained in the Answers.pdf file.

LAB 12-1 Stop and Analyze

Study the program shown in Figure 12-15, and then answer the questions. The `company` array contains the amounts the company sold both domestically and internationally during the months of January through June. The first row contains the domestic sales for the six months. The second row contains the international sales during the same period.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

```
 1 //Lab12-1.cpp - calculates the total company sales
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 using namespace std;
 6
 7 int main()
 8 {
 9    int company[2][6] = {{75000, 30200, 67800,
10                          45000, 60000, 67500},
11                         {40000, 75000, 64000,
12                          32600, 47800, 39000}};
13    int companySales = 0;
14
15    for (int location = 0; location < 2; location += 1)
16        for (int month = 0; month < 6; month += 1)
17            companySales += company[location][month];
18        //end for
19    //end for
20    cout << "Company sales: $" << companySales << endl;
21    return 0;
22 }  //end of main function
```

Figure 12-15 Code for Lab 12-1

QUESTIONS

1. What value is stored in the `company[1][5]` element?
2. How can you calculate the total company sales made in February?
3. What is the highest row subscript in the `company` array? What is the highest column subscript in the array?
4. If the January domestic sales are stored in the `company[0][0]` element, where are the January international sales stored?
5. If you change the `for` clause in Line 16 to `for (int month = 1; month <= 6; month += 1)`, how will the change affect the assignment statement in the `for` loop?
6. Follow the instructions for starting C++ and viewing the Lab12-1.cpp file, which is contained in either the Cpp8\Chap12\Lab12-1 Project folder or the Cpp8\Chap12 folder. (Depending on your C++ development tool, you may need to open Lab12-1's project/solution file first.) Run the program. The total company sales are $643900.
7. Modify the program so that it displays the total domestic sales, total international sales, and total company sales. Save and then run the program. (The total domestic sales are $345500.)
8. Next, modify the program so that it also displays the total sales made in each month. Save and then run the program. (The total January sales are $115000.)

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

LAB 12-2 Plan and Create

In this lab, you will plan and create an algorithm for Falcon Incorporated. The problem specification, IPO chart information, and C++ instructions are shown in Figure 12-16. The program displays a shipping charge based on the number of items ordered, which is entered by the user. The problem specification shows the three shipping charges along with their associated minimum and maximum orders. Notice that the maximum order amounts are stored in the first column of the two-dimensional `shipCharges` array, while their corresponding shipping charges are stored in the second column. To find the appropriate shipping charge, you search the first column for the number of items ordered, beginning with the first row. You continue searching the first column in each row as long as there are rows left to search and the number of items ordered is greater than the value in the first column. You stop searching either when there are no more rows to search or when the number of items ordered is less than or equal to the value in the first column. For example, if the number of items ordered is 75, the first value you would look at in the array is 50. The number of items ordered (75) is greater than 50, so you continue searching with the value in the second row (100). The number 100 is greater than the number of items ordered (75), so you stop searching. The appropriate shipping charge is located in the same row—in this case, the second row—but in the second column. The appropriate shipping charge is $10.

Problem specification

Falcon Incorporated wants a program that displays a shipping charge based on the number of items ordered by the customer. The shipping charge information is shown here.

Minimum order	Maximum order	Shipping charge ($)
1	50	20
51	100	10
101	999999	0

IPO chart information	C++ instructions
Input	
number ordered	`int numOrdered = 0;`
maximum orders and shipping charges	stored in the array
Processing	
array (3 rows, 2 columns)	`int shipCharges[3][2] = {{50, 20},` `{100, 10}, {999999, 0}};`
row subscript (counter: 0 to 2)	`int rowSub = 0;`
Output	
shipping charge	displayed from the array
Algorithm	

1. enter the number ordered	`cout << "Number ordered " <<`
	`"(negative number or 0 to end): ";`
	`cin >> numOrdered;`

Figure 12-16 Problem specification, IPO chart information, and C++ instructions for Lab 12-2 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
2. repeat while (the number ordered
   is greater than 0 and less than
   or equal to 999999)
        assign 0 to the row subscript
        to ensure the search begins
        in the first row

        repeat while (the row subscript
        is less than 3 and the
        number ordered is greater than
        the array[row subscript][0] value)

              add 1 to the row subscript
              to continue the search
              in the next row
        end repeat
        display the shipping charge

        enter the number ordered

  end repeat
```

```
while (numOrdered > 0 &&
  numOrdered <= 999999)
{
     rowSub = 0;

     while (rowSub < 3 &&
        numOrdered > shipCharges
        [rowSub][0])

          rowSub += 1;

     //end while
     cout << "Shipping charge for a
     quantity of " << numOrdered
     << " is $"
     << shipCharges[rowSub][1]
     << endl << endl;

     cout << "Number ordered " <<
          "(negative number or
     0 to end): ";
     cin >> numOrdered;
}  //end while
```

Figure 12-16 Problem specification, IPO chart information, and C++ instructions for Lab 12-2

Figure 12-17 shows the code for the entire program, and Figure 12-18 shows the completed desk-check table, assuming the user enters the numbers 75, 200, and –1 as the number of items ordered.

```
 1 //Lab12-2.cpp - displays the shipping charge
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 using namespace std;
 6
 7 int main()
 8 {
 9    int shipCharges[3][2] = {{50, 20},
10                             {100, 10},
11                             {999999, 0}};
12    int numOrdered = 0;
13    int rowSub = 0;
14
15    cout << "Number ordered " <<
16        "(negative number or 0 to end): ";
17    cin >> numOrdered;
```

Figure 12-17 Falcon Incorporated program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

You can use `rowSub++;` in Line 25.

```
18
19     while (numOrdered > 0 && numOrdered <= 999999)
20     {
21         //search array
22         rowSub = 0;
23         while (rowSub < 3 &&
24             numOrdered > shipCharges[rowSub][0])
25                 rowSub += 1;
26         //end while
27
28         cout << "Shipping charge for a quantity of "
29             << numOrdered << " is $"
30             << shipCharges[rowSub][1] << endl << endl;
31
32         cout << "Number ordered " <<
33             "(negative number or 0 to end): ";
34         cin >> numOrdered;
35     }    //end while
36     return 0;
37 } //end of main function
```

Figure 12-17 Falcon Incorporated program

shipCharges[0][0]	shipCharges[0][1]
50	20
shipCharges[1][0]	**shipCharges[1][1]**
100	10
shipCharges[2][0]	**shipCharges[2][1]**
999999	0

numOrdered	rowSub
~~0~~	~~0~~
~~75~~	~~0~~
~~200~~	~~1~~
-1	~~0~~
	~~1~~
	2

Figure 12-18 Completed desk-check table for the Falcon Incorporated program

The final step in the problem-solving process is to evaluate and modify (if necessary) the program. Recall that you evaluate a program by entering its instructions into the computer and then using the computer to run (execute) it. While the program is running, you enter the same sample data used when desk-checking the program.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

DIRECTIONS

Follow the instructions for starting your C++ development tool. Depending on the development tool you are using, you may need to create a new project; if so, name the project Lab12-2 Project and save it in the Cpp8\Chap12 folder. Enter the instructions shown in Figure 12-17 in a source file named Lab12-2.cpp. (Do not enter the line numbers.) Save the file in either the project folder or the Cpp8\Chap12 folder. Now, follow the appropriate instructions for running the Lab12-2.cpp file. Test the program using the same data you used to desk-check the program. If necessary, correct any bugs (errors) in the program.

LAB 12-3 Modify

If necessary, create a new project named Lab12-3 Project and save it in the Cpp8\Chap12 folder. Enter (or copy) the Lab12-2.cpp instructions into a new source file named Lab12-3.cpp. Change Lab12-2.cpp in the first comment to Lab12-3.cpp. Replace the maximum amounts in the `shipCharges` array with the minimum amounts. Then, make the necessary modifications to the program. Save and then run the program. Test the program appropriately.

LAB 12-4 What's Missing?

The program in this lab should display the average stock price. Start your C++ development tool and view the Lab12-4.cpp file, which is contained in either the Cpp8\Chap12\Lab12-4 Project folder or the Cpp8\Chap12 folder. (Depending on your C++ development tool, you may need to open Lab12-4's project/solution file first.) Put the C++ instructions in the proper order, and then determine the one or more missing instructions. Test the program appropriately.

LAB 12-5 Desk-Check

Desk-check the Jenko Booksellers program, which is shown in Figure 12-8 in the chapter.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

LAB 12-6 Debug

Follow the instructions for starting C++ and viewing the Lab12-6.cpp file, which is contained in either the Cpp8\Chap12\Lab12-6 Project folder or the Cpp8\Chap12 folder. (Depending on your C++ development tool, you may need to open Lab12-6's project/solution file first.) Read the program's comments and then run the program. Notice that the program is not working correctly. Debug the program.

Chapter Summary

- A two-dimensional array resembles a table in that the elements are in rows and columns. Each element has the same data type.
- You can determine the number of elements in a two-dimensional array by multiplying the number of its rows by the number of its columns.
- Each element in a two-dimensional array is identified by a unique combination of two subscripts. The first subscript represents the element's row location in the array, and the second subscript represents its column location. You refer to each element in a two-dimensional array by the array's name and the element's subscripts, which are specified in two sets of square brackets immediately following the name.
- In a two-dimensional array, the first row and column subscripts are 0. The last row subscript is always one number less than the number of rows in the array. The last column subscript is always one number less than the number of columns in the array.
- You must declare a two-dimensional array before you can use it. When declaring a two-dimensional array, you must provide the number of rows and the number of columns.
- After declaring a two-dimensional array, you can use an assignment statement or the extraction operator to enter data into the array.
- You need to use two loops to access every element in a two-dimensional array. One of the loops keeps track of the row subscript, and one keeps track of the column subscript.
- To pass a two-dimensional array to a function, you include the array's name in the statement that calls the function. The array's corresponding formal parameter in the function header must specify the formal parameter's data type and name, followed by two sets of square brackets. The first bracket contains the number of rows, and the second bracket contains the number of columns.

Key Term

Two-dimensional array—an array made up of rows and columns; each element has the same data type and is identified by a unique combination of two subscripts: a row subscript and a column subscript

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Review Questions

1. The first element in a two-dimensional array has a row subscript of _____ and a column subscript of _____.

   a. 0, 0
   b. 0, 1
   c. 1, 0
   d. 1, 1

2. Which of the following statements creates a two-dimensional array that contains three rows and four columns?

   a. `int rates[3, 4] = {0};`
   b. `int rates[4, 3] = {0};`
   c. `int rates[3][4] = {0};`
   d. `int rates[4][3] = {0};`

   Use the `sales` array to answer Review Questions 3 through 6. The array was declared using the `int sales[2][5] = {{10000, 12000, 900, 500, 20000}, {350, 600, 700, 800, 100}};` statement.

3. The statement `sales[1][3] += 10;` will replace the number _____.

   a. 900 with 910
   b. 500 with 510
   c. 700 with 710
   d. 800 with 810

4. The statement `sales[0][4] = sales[0][4 - 2];` will replace the number _____.

   a. 20000 with 900
   b. 20000 with 19998
   c. 20000 with 19100
   d. 500 with 12000

5. The statement `cout << sales[0][3] + sales[1][3] << endl;` will _____.

   a. display 1300
   b. display 1600
   c. display `sales[0][3] + sales[1][3]`
   d. result in an error

6. Which of the following verifies that the array subscripts stored in the `row` and `col` variables are valid for the `sales` array?

   a. `if (sales[row][col] >= 0 && sales[row][col] < 5)`
   b. `if (sales[row][col] >= 0 && sales[row][col] <= 5)`
   c. `if (row >= 0 && row < 3 && col >= 0 && col < 6)`
   d. `if (row >= 0 && row <= 1 && col >= 0 && col <= 4)`

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Exercises

Pencil and Paper

TRY THIS

1. Write the code to declare and initialize a two-dimensional `double` array named `balances` that has four rows and six columns. (The answers to TRY THIS Exercises are located at the end of the chapter.)

TRY THIS

2. Write the code to display the contents of the `balances` array from Pencil and Paper Exercise 1. Use two `for` statements to display the array, row by row. (The answers to TRY THIS Exercises are located at the end of the chapter.)

MODIFY THIS

3. Rewrite the code from Pencil and Paper Exercise 2 to display the array, column by column.

INTRODUCTORY

4. Write the code to store the number 100 in each element in the `balances` array from Pencil and Paper Exercise 1. Use two `for` statements.

INTRODUCTORY

5. Rewrite the code from Pencil and Paper Exercise 4 using two `while` statements.

INTRODUCTORY

6. Rewrite the code from Pencil and Paper Exercise 4 using the `do while` statement in the outer loop and the `while` statement in the nested loop.

INTERMEDIATE

7. Write the statement to assign the C++ keyword `true` to the variable located in the third row, first column of a `bool` array named `answers`.

INTERMEDIATE

8. Write the code to display the sum of the numbers stored in the following three elements contained in a two-dimensional `double` array named `sales`: the first row, first column; the second row, third column; and the third row, fourth column.

INTERMEDIATE

9. Write the code to subtract the number 1 from each element in a two-dimensional `int` array named `quantities`. The array has 10 rows and 25 columns. Use two `for` statements.

INTERMEDIATE

10. Rewrite the code from Pencil and Paper Exercise 9 using two `while` statements.

INTERMEDIATE

11. Write the code to find the square root of the number stored in the first row, third column in a two-dimensional `double` array named `mathNumbers`. Display the result on the screen.

INTERMEDIATE

12. Rewrite the code shown in Example 3 in Figure 12-5 so it displays the contents of the `prices` array, column by column. Use an outer `for` loop and a nested `while` loop.

ADVANCED

13. Write the code to display the largest number stored in the first column of a two-dimensional `int` array named `orders`. The array has five rows and two columns. Use the `for` statement.

ADVANCED

14. Rewrite the code from Pencil and Paper Exercise 13 using the `while` statement.

SWAT THE BUGS

15. The `numbers` array is a two-dimensional `int` array that contains three rows and five columns. The following statement should call the void `calcTotal` function, passing it the `numbers` array: `calcTotal(numbers[3][5]);`. Correct the statement.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Computer

16. Follow the instructions for starting C++ and viewing the TryThis16.cpp file, which is contained in either the Cpp8\Chap12\TryThis16 Project folder or the Cpp8\Chap12 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) The file contains the code shown earlier in Figure 12-8. The first column in the array contains the sales amounts for paperback books sold in each of the three stores; the second column contains the sales amounts for hardcover books. Save and then run the program. The total sales are $15159.36. Modify the program to also display the total paperback sales and the total hardcover sales. Save and then run the program. (The answers to TRY THIS Exercises are located at the end of the chapter.) TRY THIS

17. Follow the instructions for starting C++ and viewing the TryThis17.cpp file, which is contained in either the Cpp8\Chap12\TryThis17 Project folder or the Cpp8\Chap12 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) As mentioned in the chapter, when you pass a two-dimensional array to a function, the first set of square brackets in its corresponding formal parameter can be empty. Remove the number 4 from the first formal parameter in the function prototype and function header. The `main` function will now need to pass two actual arguments to the `displayArray` function: the array and the number of rows (regions) in the array. Make the appropriate modifications to the `displayArray` function prototype, function header, and function call. Also modify the outer loop's `for` clause in the `displayArray` function so it uses the number of rows passed to the function rather than the literal constant 4. Save and then run the program. (The answers to TRY THIS Exercises are located at the end of the chapter.) TRY THIS

18. Follow the instructions for starting C++ and viewing the ModifyThis18.cpp file, which is contained in either the Cpp8\Chap12\ModifyThis18 Project folder or the Cpp8\Chap12 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) The file contains the code shown earlier in Figure 12-8. The first column in the array contains the sales amounts for paperback books sold in each of the three stores; the second column contains the sales amounts for hardcover books. Jenko Booksellers has opened another store. The store's sales of paperback and hardcover books are $1650.85 and $1246.85, respectively. Add the new sales information to the array, and then modify the program appropriately. Save and then run the program. MODIFY THIS

19. If necessary, create a new project named ModifyThis19 Project and save it in the Cpp8\Chap12 folder. Enter the C++ instructions shown earlier in Figure 12-10 into a new source file named ModifyThis19.cpp. Change the filename in the first comment. Save and then run the program. Test the program using the following two pay codes: 6 and 5. Enter –1 to stop the program. Add a new pay code and pay rate to the array. The new pay code is 11, and its corresponding pay rate is $23. Make the appropriate modifications to the code. Save and then run the program. Test the program using the following three pay codes: 6, 5, and 11. Enter –1 to stop the program. MODIFY THIS

20. Follow the instructions for starting C++ and viewing the ModifyThis20.cpp file, which is contained in either the Cpp8\Chap12\ModifyThis20 Project folder or the Cpp8\Chap12 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) The file contains the code shown earlier in Figure 12-8. The first column in the array contains the sales amounts for paperback books sold in each of the three stores; the second column contains the sales amounts for hardcover books. Save and then run the program. The total sales are $15159.36. Modify the program to also display each store's total sales. Save and then run the program. MODIFY THIS

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

INTRODUCTORY

21. Follow the instructions for starting C++ and viewing the Introductory21.cpp file, which is contained in either the Cpp8\Chap12\Introductory21 Project folder or the Cpp8\Chap12 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) The program should calculate and display the average of the values stored in the `rates` array. Display the average with two decimal places. Complete the program using the `for` statement. Save and then run the program.

INTRODUCTORY

22. Follow the instructions for starting C++ and viewing the Introductory22.cpp file, which is contained in either the Cpp8\Chap12\Introductory22 Project folder or the Cpp8\Chap12 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) The program should display the contents of the two-dimensional array, column by column and also row by row. Complete the program using a `while` statement in the outer loops and a `for` statement in the nested loops. Save and then run the program.

INTERMEDIATE

23. If necessary, create a new project named Intermediate23 Project and save it in the Cpp8\Chap12 folder. Also create a new source file named Intermediate23.cpp. Declare a seven-row, two-column `int` array named `temperatures`. The program should prompt the user to enter the highest and lowest temperatures for seven days. Store the highest temperatures in the first column in the array. Store the lowest temperatures in the second column. The program should display the average high temperature and the average low temperature. Display the average temperatures with one decimal place. Save and then run the program. Test the program using the data shown in Figure 12-19.

Day	Highest	Lowest
1	95	67
2	98	54
3	86	70
4	99	56
5	83	34
6	75	68
7	80	45

Figure 12-19

INTERMEDIATE

24. In this exercise, you modify the program from Computer Exercise 23. If necessary, create a new project named Intermediate24 Project and save it in the Cpp8\Chap12 folder. Copy the instructions from the Intermediate23.cpp file into a source file named Intermediate24.cpp. Change the filename in the first comment. In addition to displaying the average high temperature and average low temperature, the program should also display the highest temperature stored in the first column in the array and the lowest temperature stored in the second column. Save and then run the program. Test the program using the data shown earlier in Figure 12-19.

ADVANCED

25. Follow the instructions for starting C++ and viewing the Advanced25.cpp file, which is contained in either the Cpp8\Chap12\Advanced25 Project folder or the Cpp8\Chap12 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) Code the program so that it asks the user for a dollar amount by which each price should be increased. The program should increase each price in the array's first column by that amount. For example, when the user enters the number 10, the program should increase each price in the array's first column by $10. Store the

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

updated prices in the second column of the array. After increasing each price, the program should display the contents of the array, row by row. Display the array contents in two columns. Save and then run the program. Increase each price by $10.

26. ADVANCED

In this exercise, you code an application that displays the number of times a value appears in a two-dimensional array. Follow the instructions for starting C++ and viewing the Advanced26.cpp file, which is contained in either the Cpp8\Chap12\Advanced26 Project folder or the Cpp8\Chap12 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) Code the program so that it displays the number of times each of the numbers from 1 through 9 appears in the `numbers` array. Save and then run the program. (Hint: Use a one-dimensional array of counter variables.)

27. ADVANCED

If necessary, create a new project named Advanced27 Project and save it in the Cpp8\Chap12 folder. Also create a new source file named Advanced27.cpp. JM Sales employs 10 salespeople. The sales made by the salespeople during the months of January, February, and March are listed in Figure 12-20. Store the sales amounts in a two-dimensional array. The sales manager wants an application that allows him to enter the current bonus rate. The program should display each sales person's number (1 through 10), total sales amount, and total bonus amount. It also should display the total bonus paid to all salespeople. Display the bonus amounts with two decimal places. Save and then run the program. Test the program using 10% as the bonus rate.

Salesperson	January	February	March
1	2400	3500	2000
2	1500	7000	1000
3	600	450	2100
4	790	240	500
5	1000	1000	1000
6	6300	7000	8000
7	1300	450	700
8	2700	5500	6000
9	4700	4800	4900
10	1200	1300	400

Figure 12-20

28. ADVANCED

Follow the instructions for starting C++ and viewing the Advanced28.cpp file, which is contained in either the Cpp8\Chap12\Advanced28 Project folder or the Cpp8\Chap12 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) Notice that the pay codes in the array are in ascending numerical order. The user will enter the pay code to search for in the array. The program should search for the pay code in the first column of the array, but the search should begin in the middle row, which is row 4. If the pay code the user is searching for is located in the first column of row 4, the program should display the corresponding pay rate from the second column in row 4. If the pay code the user is searching for is *greater* than the pay code in row 4's first column, the search should continue in rows 5, 6, and 7. However, if the pay code the user is searching for is *less* than the pay code in row 4's first column, the search should continue in rows 3, 2, and 1. Save and then run the program. (Hint: To verify that the search works appropriately, use counters to keep track of the number of greater than comparisons made and the number of less than comparisons made.)

29. ADVANCED

In this exercise, you will create a program that allows the user to enter an employee's gross pay amount as well as his or her filing status and number of withholding allowances.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The program should calculate and display the amount of federal withholding tax (FWT) to deduct from the weekly gross pay.

a. The amount of FWT is based on the employee's weekly taxable wages and filing status, which is either single (including head of household) or married. The program will need to calculate the weekly taxable wages by first multiplying the number of withholding allowances by $76.90 (the value of a withholding allowance in 2015) and then subtracting the result from the weekly gross pay. For example, if your weekly gross pay is $400 and you have two withholding allowances, your weekly taxable wages are $246.20, as shown in Figure 12-21.

b. You use the weekly taxable wages, along with the filing status and the appropriate weekly Federal Withholding Tax table, to determine the amount of FWT to withhold. The weekly tax tables for 2015 are shown in Figure 12-22. Each table contains five columns of information. The first two columns list various ranges, also called brackets, of taxable wage amounts. The first column (Over) lists the amount that a taxable wage in that bracket must be over, and the second column (But not over) lists the maximum amount included in the bracket. The remaining three columns (Base amount, Percentage, and Of excess over) tell you how to calculate the tax for each range. For example, assume that you are single and your weekly taxable wages are $246.20. Before you can calculate the amount of your tax, you need to locate your taxable wages in the first two columns of the Single table. Taxable wages of $246.20 fall within the $222 through $764 bracket. After locating the bracket that contains your taxable wages, you then use the remaining three columns in the table to calculate your tax. In this case, you calculate the tax by first subtracting 222 (the amount shown in the Of excess over column) from your taxable wages of 246.20, giving 24.20. You then multiply 24.20 by 15% (the amount shown in the Percentage column), giving 3.63. You then add that amount to the amount shown in the Base amount column (in this case, 17.80), giving $21.43 as your tax. The calculations are shown in Figure 12-21 along with the calculations for a married taxpayer whose weekly taxable wages are $1,659.50.

c. If necessary, create a new project named Advanced29 Project and save it in the Cpp8\Chap12 folder. Enter your C++ instructions in a new source file named Advanced29.cpp. Store each tax table in its own two-dimensional array. Be sure to enter appropriate comments and any additional instructions required by the compiler. Test the program appropriately.

Taxable wage calculation

Gross wages	$ 400.00
Allowances	– 153.80 (2 withholding allowances * 76.90)
Taxable wages	$ 246.20

Single with weekly taxable wages of $246.20		Married with weekly taxable wages of $1,659.50	
Taxable wages	$ 246.20	Taxable wages	$ 1,659.50
Of excess over	– 222.00	Of excess over	– 1,606.00
	24.20		53.50
Percentage	* 0.15	Percentage	* 0.25
	3.63		13.38
Base amount	+ 17.80	Base amount	+ 198.40
Tax	$ 21.43	Tax	$ 211.78

Figure 12-21

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

FWT Tables – Weekly Payroll Period

Single person (including head of household)

If the taxable wages are: The amount of income tax to withhold is:

Over	But not over	Base amount	Percentage	Of excess over
	$ 44	0		
$ 44	$ 222	0	10%	$ 44
$ 222	$ 764	$ 17.80 plus	15%	$ 222
$ 764	$1,789	$ 99.10 plus	25%	$ 764
$1,789	$3,685	$ 355.35 plus	28%	$1,789
$3,685	$7,958	$ 886.23 plus	33%	$3,685
$7,958	$7,990	$2,296.32 plus	35%	$7,958
$7,990		$2,307.52 plus	39.6%	$7,990

Married person

If the taxable wages are: The amount of income tax to withhold is:

Over	But not over	Base amount	Percentage	Of excess over
	$ 165	0		
$ 165	$ 520	0	10%	$ 165
$ 520	$1,606	$ 35.50 plus	15%	$ 520
$1,606	$3,073	$ 198.40 plus	25%	$1,606
$3,073	$4,597	$ 565.15 plus	28%	$3,073
$4,597	$8,079	$ 991.87 plus	33%	$4,597
$8,079	$9,105	$2,140.93 plus	35%	$8,079
$9,105		$2,500.03 plus	39.6%	$9,105

Figure 12-22

30. Follow the instructions for starting C++ and viewing the SwatTheBugs30.cpp file, which is contained in either the Cpp8\Chap12\SwatTheBugs30 Project folder or the Cpp8\Chap12 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) Read the program's comments and then run the program. Notice that the program is not working correctly. Debug the program.

SWAT THE BUGS

Answers to TRY THIS Exercises

Pencil and Paper

1. `double balances[4][6] = {0.0};`
2.
```
for (int row = 0; row < 4; row += 1)
  for (int col = 0; col < 6; col += 1)
       cout << balances[row][col] << endl;
  //end for
//end for
```

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Computer

16. See Figure 12-23.

```
//TryThis16.cpp - displays the total sales
//Created/revised by <your name> on <current date>

#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    double sales[3][2] = {{3567.85, 2589.99},
                          {3239.67, 2785.55},
                          {1530.50, 1445.80}};
    double total = 0.0;
    double paperback = 0.0;  //accumulator
    double hardcover = 0.0;  //accumulator

    //accumulate sales
    for (int store = 0; store < 3; store += 1)
    {
        paperback += sales[store][0];
        hardcover += sales[store][1];
    } //end for
    total = paperback + hardcover;

    cout << fixed << setprecision(2);
    cout << "Total paperback sales: $" << paperback << endl;
    cout << "Total hardcover sales: $" << hardcover << endl;
    cout << "Total sales: $" << total << endl;
    return 0;
}  //end of main function
```

Figure 12-23

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

17. See Figure 12-24. The changes are shaded in the figure.

```
//TryThis17.cpp - gets and displays order amounts
//Created/revised by <your name> on <current date>

#include <iostream>
using namespace std;

//function prototype
void displayArray(int nums[][3], int numRows);

int main()
{
    int orders[4][3] = {0};

    //store data in the array
    for (int region = 0; region < 4; region += 1)
        for (int month = 0; month < 3; month += 1)
        {
            cout << "Region " << region + 1
                << ", Month " << month + 1
                << " orders: ";
            cin >> orders[region][month];
        } //end for
      //end for

    //display the contents of the array
    displayArray(orders, 4);
    return 0;
} //end of main function

//*****function definitions*****
void displayArray(int nums[][3], int numRows)
{
    cout << endl << "Array contents:" << endl;
    for (int region = 0; region < numRows; region += 1)
    {
        cout << "Region " << region + 1
            << ": " << endl;
        for (int month = 0; month < 3; month += 1)
            cout << "     Month " << month + 1
            << ": " << nums[region][month]
            << endl;
        //end for
    } //end for
}  //end of displayArray function
```

Figure 12-24

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

CHAPTER 13

Strings

After studying Chapter 13, you should be able to:

- Declare `string` variables and named constants
- Get string input using the `getline` function
- Ignore characters using the `ignore` function
- Determine the number of characters in a string
- Access the characters in a string
- Search a string
- Remove characters from a string
- Convert a string to a numeric data type
- Replace characters in a string
- Insert characters within a string
- Duplicate characters within a string
- Concatenate strings

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The `string` Data Type

You will learn how to create your own classes and objects in Chapter 15.

The programs created in the previous chapters used `int` and `double` variables and named constants. In this chapter, the programs will also use `string` variables and named constants. As you learned in Chapter 3, the `string` data type is not one of the fundamental data types in C++. Rather, it was added to the C++ language through the use of a class, called the `string` class. Recall that a class is simply a group of instructions that the computer uses to create an object. The instructions for creating a `string` object, which can be either a `string` variable or a `string` named constant, are contained in the string file. Therefore, for a program to use the `string` class, it must contain the `#include <string>` directive. Also included in the string file are functions that you can use to manipulate strings. The functions are called member functions because they are members of a class—in this case, the `string` class. You will explore some of the more commonly used `string` class member functions in this chapter. First, however, you need to learn how to declare a `string` memory location.

Ch13-Chapter Preview

Figure 13-1 shows examples of using the `string` class to create and initialize `string` variables and `string` named constants. Memory locations having the `string` data type are initialized using string literal constants. Recall from Chapter 3 that a string literal constant is zero or more characters enclosed in double quotation marks. The declaration statement in Example 1 creates a `string` variable named `city` and initializes it to the empty string (""), which is two double quotation marks with no space between; most `string` variables are initialized to the empty string. The declaration statement in Example 2 creates a `string` variable named `playAgain` and initializes it to the string "Y". Example 3 creates a `string` named constant called `COMPANY_NAME` and initializes it to "Jacoby Ltd.".

HOW TO Declare and Initialize `string` Variables and Named Constants

Example 1

```
string city = "";
```

declares and initializes a `string` variable named `city`

Example 2

```
string playAgain = "Y";
```

declares and initializes a `string` variable named `playAgain`

Example 3

```
const string COMPANY_NAME = "Jacoby Ltd.";
```

declares and initializes a `string` named constant called `COMPANY_NAME`

Figure 13-1 How to declare and initialize `string` variables and named constants

Getting String Input from the Keyboard

In previous chapters, you used the extraction operator (>>) to get numbers and characters from the user at the keyboard. The extraction operator can also be used to get string input from the keyboard, as shown in the examples in Figure 13-2. However, keep in mind that the extraction operator stops reading characters when it encounters a white-space character in the input. Recall that a white-space character is a blank, a tab, or a newline. You enter a blank character when you press the Spacebar on your keyboard. You enter a tab character when you press the Tab key, and you enter a newline character when you press the Enter key. As a result, if the user inadvertently enters the string "32 101" (rather than "32101") as the ZIP code in Example 1, the

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

extraction operator in the `cin >> zipCode;` statement will store only the string "32" in the `zipCode` variable.

HOW TO Use the Extraction Operator (>>) to Get String Input from the Keyboard

Example 1

```
string zipCode = "";
cout << "Enter your zip code: ";
cin >> zipCode;
```

gets a string from the keyboard and stores it in the `zipCode` variable

Example 2

```
string playAgain = "Y";
cout << "Play the game again? (Y/N): ";
cin >> playAgain;
```

gets a string from the keyboard and stores it in the `playAgain` variable

Figure 13-2 How to use the extraction operator (>>) to get string input from the keyboard

Because many strings entered at the keyboard contain one or more blank characters (such as "San Diego, CA" and "Marie S. Harris"), the `string` class provides a member function called `getline` for accepting that type of input. Figure 13-3 shows the function's syntax and includes examples of using the function. The semicolon that appears as the last character in the syntax indicates that the function is a self-contained statement.

HOW TO Use the `getline` Function to Get String Input from the Keyboard

Syntax

getline(cin, *stringVariableName*[**,** *delimiterCharacter*]**);** — semicolon

Example 1

```
string name = "";
cout << "Enter your name: ";
getline(cin, name);
```

stores the characters entered by the user, up until the newline character, in the `name` variable; consumes the newline character

Example 2

```
string name = "";
cout << "Enter your name: ";
getline(cin, name, '\n');
```

same as Example 1, but specifies the newline delimiter character

Example 3

```
string city = "";
cout << "City: ";
getline(cin, city, '\#');
```

stores the characters entered by the user, up until the # character, in the `city` variable; consumes the # character

Figure 13-3 How to use the `getline` function to get string input from the keyboard

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The `getline` function has three actual arguments, two of which are required. The required `cin` argument refers to the computer keyboard, and the required *stringVariableName* argument is the name of a `string` variable in which to store the input. You can use the optional *delimiterCharacter* argument to indicate the end of the string. The argument represents the character that immediately follows the last character in the string. The **getline function** will continue to read the characters entered at the keyboard until it encounters the delimiter character. If you omit the *delimiterCharacter* argument, the default delimiter character is the newline character. For example, if the user types the words "Good night" and then presses the Enter key, the string will end with the letter t, which is the last character the user typed before pressing the Enter key. When the `getline` function encounters the delimiter character in the input, it discards the character—a process C++ programmers refer to as **consuming the character**.

The `getline` function in Example 1 in Figure 13-3 reads the characters entered at the keyboard and then stores the characters in the `name` variable. The function will stop reading and storing characters when it encounters the newline character, which is when the user presses the Enter key. As mentioned earlier, the newline character is the default delimiter character in the `getline` function. At that point, the function will consume (discard) the newline character.

Like the `getline` function in Example 1, the `getline` function in Example 2 also reads the characters entered at the keyboard and stores them in the `name` variable. Here, too, the function will stop reading and storing characters when it encounters the newline character, which it will consume (discard). The newline character is designated in C++ by a backslash and the letter n, both enclosed in single quotation marks, like this: `'\n'`. Although the newline character consists of two characters, it is treated as one character by the computer. The backslash in the newline character is called an escape character, and it indicates that the character that follows it—in this case, the letter n—has a special meaning. The combination of the backslash and the character that follows it is called an **escape sequence**. An example of another escape sequence is `'\t'`, which represents the Tab key.

The `getline` function in Example 3 in Figure 13-3 reads the characters entered at the keyboard and stores them in the `city` variable. In this case, the function will stop reading and storing characters when it encounters the # character, which it will consume (discard).

The Primrose Auction House Program

Figure 13-4 shows the problem specification and IPO chart for the Primrose Auction House program, which gets two items from the user: a buyer's name and the amount of his or her purchase. It then calculates the buyer's premium and displays the buyer's name and premium amount on the screen.

Problem specification
Primrose Auction House wants a program that calculates the fee a buyer must pay to the auction house when purchasing an item. The fee, which is called the buyer's premium, is a percentage of the purchase price. The Primrose Auction House charges a 10% fee. The program should allow the user to enter the buyer's name and the amount of his or her purchase. It should display a message that contains the buyer's name and the premium amount.

Figure 13-4 Problem specification and IPO chart for the Primrose Auction House program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Input
name
price
rate (10%)

Processing
Processing items: none

Algorithm:
1. enter the name and price
2. calculate the premium by multiplying the price by the rate
3. display the name and premium

Output
name
premium

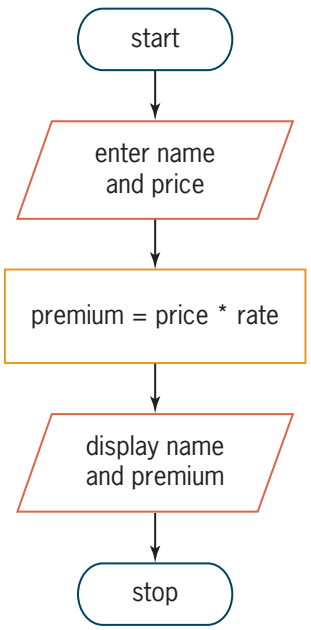


Figure 13-4 Problem specification and IPO chart for the Primrose Auction House program

Figure 13-5 shows the Primrose Auction House program, with the code pertaining to `string` data shaded. The `#include <string>` directive, which is necessary when using `string` memory locations, appears on Line 7. The declaration statement on Line 13 declares a `string` variable called `name` and initializes it to the empty string. The `getline` function on Line 18 waits for the user to respond to the "Buyer's name: " prompt. When the user presses the Enter key, the function stores the characters typed by the user, up until the newline character, in the `name` variable. (Recall that the newline character is the default delimiter character.) The function then consumes the newline character. Figure 13-5 also includes a sample run of the program.

```
1 //Primrose.cpp
2 //displays a buyer's name and premium
3 //Created/revised by <your name> on <current date>
4
```

Figure 13-5 Primrose Auction House program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
 5 #include <iostream>
 6 #include <iomanip>
 7 #include <string>
 8 using namespace std;
 9
10 int main()
11 {
12    const double RATE = 0.1;
13    string name = "";
14    int price = 0;
15    double premium = 0.0;
16
17    cout << "Buyer's name: ";
18    getline(cin, name);
19    cout << "Purchase price: ";
20    cin >> price;
21
22    premium = price * RATE;
23
24    cout << fixed << setprecision(2);
25    cout << "********Auction Summary********" << endl;
26    cout << "Buyer: " << name << endl
27         << "Premium: $" << premium << endl;
28
29    return 0;
30 }  //end of main function
```

```
Primrose Auction House
Buyer's name: Jason Higgins
Purchase price: 2500
********Auction Summary********
Buyer: Jason Higgins
Premium: $250.00
Press any key to continue . . .
```

Figure 13-5 Primrose Auction House program

Now let's make a slight change to the problem specification for the auction house. In addition to entering the buyer's name and the purchase amount, the user should also enter the item number assigned to the purchased item. The program should now display the item number along with the buyer's name and premium amount. Consider how these changes will affect the original program shown in Figure 13-5.

The modified program will need to declare and initialize a `string` variable to store the item number entered by the user. It will also need both a `cout` statement that prompts the user to enter the item number and a `getline` function to get the user's input. A `getline` function is appropriate because the auction house's item numbers may contain one or more spaces. In addition, the program will need to include the item number in the `cout` statement that displays the buyer's name and premium amount. The modifications made to the original program are shaded in Figure 13-6. The figure also contains a sample run of the modified program. Notice that the program does not work correctly: It does not pause to allow the user to enter the item number.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

```
 1 //Modified Primrose.cpp
 2 //displays a buyer's name, premium, and the item number
 3 //Created/revised by <your name> on <current date>
 4
 5 #include <iostream>
 6 #include <iomanip>
 7 #include <string>
 8 using namespace std;
 9
10 int main()
11 {
12    const double RATE = 0.1;
13    string name = "";
14    int price = 0;
15    double premium = 0.0;
16    string itemNum = "";
17
18    cout << "Buyer's name: ";
19    getline(cin, name);
20    cout << "Purchase price: ";
21    cin >> price;
22    cout << "Item number: ";
23    getline(cin, itemNum);
24
25    premium = price * RATE;
26
27    cout << fixed << setprecision(2);
28    cout << "********Auction Summary********" << endl;
29    cout << "Buyer: " << name << endl
30         << "Premium for item " << itemNum
31         << ": $" << premium << endl;
32
33    return 0;
34 }  //end of main function
```

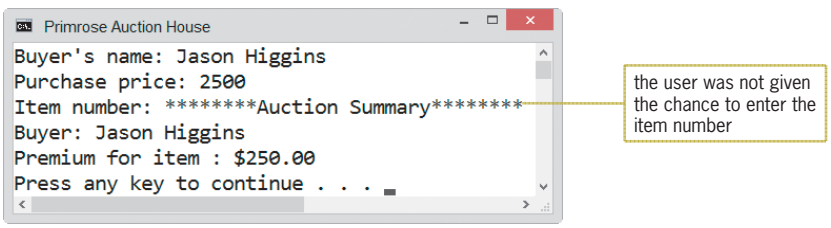


Figure 13-6 Modified Primrose Auction House program

To understand why the modified program is not working correctly, you need to understand how the extraction operator and `getline` function get keyboard input. Toward this end, you will desk-check Lines 18 through 23 from Figure 13-6 using Jason Higgins, 2500, and ABX34 as the buyer's name, purchase amount, and item number, respectively.

The `cout` statement on Line 18 prompts the user to enter the buyer's name. Before allowing the user to enter the name, the `getline` function on Line 19 checks the `cin` object to determine whether it contains any characters. (Recall from Chapter 4 that the `cin` object stores the characters entered at the keyboard.) Because the `cin` object is empty at this point in the program, the `getline` function waits for the user to enter a name. In this case, the user types the string "Jason Higgins" and then presses the Enter key to indicate that he or she is finished entering the name. The computer stores the name and the newline character (`'\n'`) in the `cin`

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

object. It then alerts the `getline` function that the object now contains data. The `getline` function removes both the name and the newline character from the `cin` object. It stores the name in the `name` variable and then consumes the newline character.

Next, the `cout` statement on Line 20 prompts the user to enter the purchase price. Before allowing the user to enter the price, the extraction operator in the `cin >> price;` statement on Line 21 checks the `cin` object to determine whether it contains any characters. The object is empty at this point, so the extraction operator waits for the user to enter the price. In this case, the user types the four numbers 2, 5, 0, and 0 and then presses the Enter key to indicate that he or she has completed the price entry. The computer stores the four numbers and the newline character (`'\n'`) in the `cin` object. It then alerts the extraction operator that the object now contains data. The extraction operator removes the four numbers from the `cin` object and stores them in the `price` variable. However, it leaves the newline character in the object.

Next, the `cout` statement on Line 22 prompts the user to enter the item number. Before allowing the user to respond to the prompt, the `getline` function on Line 23 checks the `cin` object to determine whether it contains any characters. At this point, the object contains the newline character, which the `getline` function interprets as the end of the item number entry. As a result, the `getline` function stores the empty string in the `itemNum` variable and then consumes the newline character. Processing continues with the calculation statement on Line 25. As the desk-check shows, the program is not working correctly because of the newline character that the extraction operator on Line 21 leaves in the `cin` object. You can fix the program by telling the computer to ignore that character.

The `ignore` Function

You can use the **ignore function** to first read and then ignore characters stored in the `cin` object. The function ignores the characters by consuming (discarding) them. Figure 13-7 shows the function's syntax and includes examples of using the function.

You will need to use the `ignore` function whenever the `getline` function is processed after a statement containing the extraction operator.

HOW TO Use the `ignore` Function

Syntax

cin.ignore([*numberOfCharacters*][**,** *delimiterCharacter*]**);** ← semicolon

Example 1

```
cin.ignore();
```

reads and consumes one character; equivalent to `cin.ignore(1);`

Example 2

```
cin.ignore(5);
```

reads and consumes five characters

Example 3

```
cin.ignore(100, '\n');
```

reads and consumes characters until either 100 characters are consumed or the newline character is consumed, whichever occurs first

Figure 13-7 How to use the `ignore` function *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Example 4
`cin.ignore(25, '#');`
reads and consumes characters until either 25 characters are consumed or the # character is consumed, whichever occurs first

Figure 13-7 How to use the `ignore` function

Like the `getline` function, the `ignore` function is a self-contained statement, as the semicolon at the end of the syntax indicates. The function has two actual arguments, both of which are optional. The *numberOfCharacters* argument is an integer that represents the maximum number of characters the function should consume; if omitted, the default number of characters is 1. The *delimiterCharacter* argument is a character that, when consumed, stops the `ignore` function from reading and discarding any additional characters. The `ignore` function stops reading and discarding characters when it either consumes the number of characters specified in the *numberOfCharacters* argument or consumes the *delimiterCharacter*, whichever occurs first.

As indicated in Example 1 in Figure 13-7, you can use either the statement `cin.ignore();` or the statement `cin.ignore(1);` to read and then discard (consume) one character. The `ignore` function in Example 2 reads and consumes five characters. Example 3's `ignore` function reads and consumes characters until either 100 characters are consumed or the newline character is consumed, whichever occurs first. The `ignore` function in Example 4 reads and discards characters until either 25 characters are consumed or the # character is consumed, whichever occurs first.

In the modified auction house program, you will enter the `ignore` function immediately after the `cin >> price;` statement, as shown in Figure 13-8. By doing this, the function will read and then consume the newline character that remains in the `cin` object after the user enters the purchase price. Figure 13-8 also includes a sample run of the modified program.

```
 1 //Modified Primrose.cpp
 2 //displays a buyer's name, premium, and the item number
 3 //Created/revised by <your name> on <current date>
 4
 5 #include <iostream>
 6 #include <iomanip>
 7 #include <string>
 8 using namespace std;
 9
10 int main()
11 {
12    const double RATE = 0.1;
13    string name = "";
14    int price = 0;
15    double premium = 0.0;
16    string itemNum = "";
17
```

Lab 13-6 provides another example of a program that requires the `ignore` function.

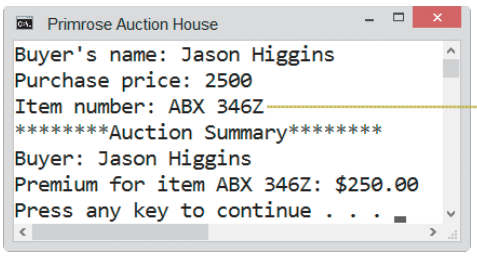
Figure 13-8 The `ignore` function entered in the modified auction house program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
18    cout << "Buyer's name: ";
19    getline(cin, name);
20    cout << "Purchase price: ";
21    cin >> price;
22    cin.ignore(100, '\n');
23    cout << "Item number: ";
24    getline(cin, itemNum);
25
26    premium = price * RATE;
27
28    cout << fixed << setprecision(2);
29    cout << "********Auction Summary********" << endl;
30    cout << "Buyer: " << name << endl
31         << "Premium for item " << itemNum
32         << ": $" << premium << endl;
33
34    return 0;
35 }  //end of main function
```

ignore function



the user can now enter the item number

Figure 13-8 The `ignore` function entered in the modified auction house program

You may be wondering why the modified program in Figure 13-8 uses `cin.ignore(100, '\n');` rather than the simpler `cin.ignore();`. Although both statements will consume the newline character left in the `cin` object after the purchase price is entered, there is an advantage to using the `cin.ignore(100, '\n');` statement in the program. To illustrate, assume that when entering the purchase price, the user types the four numbers 2, 5, 0, and 0, followed inadvertently by the letter w, and then presses the Enter key. The computer stores the four numbers along with the letter w and the newline character in the `cin` object. It then alerts the extraction operator in the `cin >> price;` statement that the object now contains data. The extraction operator removes the four numbers from the `cin` object and stores them in the `price` variable. However, it leaves both the letter w (which cannot be stored in a numeric variable) and the newline character in the object. At this point, the `cin` object contains two characters, as shown in Figure 13-9.

	`cin` object	`price`
Before the >> operator in `cin >> price;` is processed	2500w↵	0
After the >> operator in `cin >> price;` is processed	w↵	2500

Figure 13-9 Contents of `cin` object and `price` variable

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

If the program used the `cin.ignore();` statement, the `ignore` function would consume only the letter w. The newline character would still be in the `cin` object when the `getline(cin, itemNum);` statement on Line 24 is processed. As you learned earlier, the `getline` function will interpret the newline character as the end of the item number entry. The `cin.ignore(100, '\n');` statement, on the other hand, will consume both the letter w and the newline character. This is because the statement tells the computer to read and discard characters until either 100 characters are consumed or the newline character is consumed, whichever occurs first. As a result, the `getline` function on Line 24 will not find any characters in the `cin` object and will wait for the user to enter the item number.

Mini-Quiz 13-1

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

1. Which of the following creates a named constant called `CITY`?

   a. `const string CITY = "Fort Knox"`
   b. `const string CITY = 'Fort Knox';`
   c. `const string CITY = "Fort Knox";`
   d. `constant string CITY = "Fort Knox";`

2. Which of the following declares a variable named `state` and initializes it to the empty string?

   a. `string state = ""`
   b. `string state = "";`
   c. `string state = ' ';`
   d. `string state = '';`

3. Which of the following gets a string of characters from the `cin` object and stores them in the `streetAddress` variable?

   a. `getline(cin, streetAddress, '\n');`
   b. `getline(streetAddress, cin);`
   c. `cin.getline(streetAddress);`
   d. `getline.cin(streetAddress);`

4. Which of the following will stop reading and discarding characters either when 10 characters are consumed or when the user presses the Enter key, whichever occurs first?

   a. `cin.ignore('\n', 10);`
   b. `cin.ignore(10);`
   c. `cin.ignore(10, '\n');`
   d. both b and c

Determining the Number of Characters in a `string` Variable

The `string` class provides the **length function** for determining the number of characters contained in a `string` variable. The function's syntax is shown in Figure 13-10 along with examples of using the function. In the syntax, *string* is the name of the `string` variable whose length you want to determine. The `length` function returns an integer that represents the number of characters contained in the variable.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

HOW TO Use the `length` Function

Syntax
string.**length()**

Example 1
```
string name = "Ariel Chou";
cout << name.length() << endl;
```
displays the number 10 on the screen

Example 2
```
string stateID = "";
cout << "Two-character state ID: ";
cin >> stateID;
if (stateID.length() == 2)
    cout << "You entered two characters.";
else
    cout << "Please enter only two characters.";
//end if
```
compares the number of characters stored in the `stateID` variable with the number 2 and then displays an appropriate message

Example 3
```
string partNum = "";
cout << "Six-character part number: ";
getline(cin, partNum);
while (partNum.length() != 6)
{
    cout << "Six-character part number: ";
    getline(cin, partNum);
}  //end while
```
continues getting a part number until the user enters exactly six characters

Figure 13-10 How to use the `length` function

The ZIP Code program shown in Figure 13-11 uses the `length` function to determine whether the user's entry contains exactly five characters. The function appears in the `if` clause on Line 19 and is shaded in the figure. The figure also contains a sample run of the program.

```
1 //ZIP Code.cpp
2 //displays a message indicating whether a ZIP code
3 //contains the appropriate number of characters
4 //Created/revised by <your name> on <current date>
5
6 #include <iostream>
7 #include <string>
8 using namespace std;
9
```

Figure 13-11 ZIP Code program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
10 int main()
11 {
12    string zipCode = "";
13
14    cout << "Five-character ZIP code (-1 to end): ";
15    cin >> zipCode;
16
17    while (zipCode != "-1")
18    {
19        if (zipCode.length() == 5)
20            cout << "-->Correct number of characters";
21        else
22            cout << "-->Incorrect number of characters";
23        //end if
24        cout << endl << endl;
25
26        cout << "Five-character ZIP code (-1 to end): ";
27        cin >> zipCode;
28    } //end while
29    return 0;
30 }  //end of main function
```

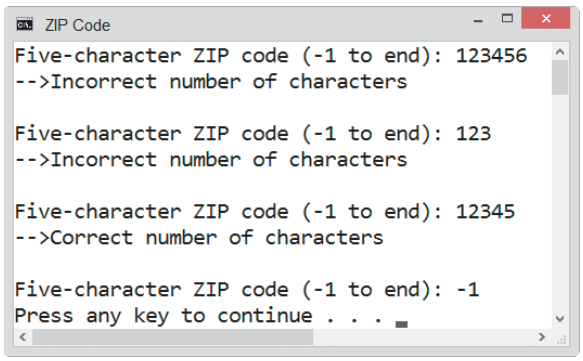


Figure 13-11 ZIP Code program

Accessing the Characters in a `string` Variable

The `string` class's **substr function** allows you to access any number of characters contained in a `string` variable; it then returns those characters. The function's syntax is shown in Figure 13-12. In the syntax, *string* is the name of a `string` variable. The required *subscript* argument represents the subscript of the first character you want to access in the string. You learned about subscripts in Chapter 11, which covered one-dimensional arrays. A string is equivalent to a one-dimensional array of characters, with each character having a unique subscript that indicates its position in the string. The first character in a string has a subscript of 0, the second has a subscript of 1, and so on.

You can use `city[0]` to refer to the first character in a `string` variable named `city`.

The function's *count* argument, which is optional, indicates the number of characters you want to access in the string. To access the first four characters, you use 0 as the *subscript* argument and 4 as the *count* argument. Similarly, to access the tenth through the twelfth characters, you use 9 as the *subscript* argument and 3 as the *count* argument. The `substr` function returns a string that contains *count* number of characters, beginning with the character whose

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

subscript is specified in the *subscript* argument. If you omit the *count* argument, the `substr` function returns all of the characters from the *subscript* position through the end of the *string*. Figure 13-12 also includes examples of using the `substr` function.

HOW TO Use the `substr` Function

Syntax

string.**substr(***subscript*[**,** *count*]**)**

Example 1

```
string name = "Shamika Timkas";
string first = "";
string last = "";
first = name.substr(0, 7);
last = name.substr(8);
```

assigns Shamika to the `first` variable and assigns Timkas to the `last` variable

Example 2

```
string sales = "";
cout << "Enter the sales: ";
getline(cin, sales);
if (sales.substr(0, 1) == "$")
    sales = sales.substr(1);
//end if
```

if the string stored in the `sales` variable begins with the dollar sign, the code assigns the variable's contents, excluding the dollar sign, to the variable

Example 3

```
string rate = "";
cout << "Enter the rate: ";
getline(cin, rate);
if (rate.substr(rate.length() - 1, 1) == "%")
    rate = rate.substr(0, rate.length() - 1);
//end if
```

if the string stored in the `rate` variable ends with the percent sign, the code assigns the variable's contents, excluding the percent sign, to the variable

Figure 13-12 How to use the `substr` function

In Example 1 in Figure 13-12, the first assignment statement assigns the first seven characters stored in the `name` variable (Shamika) to the `first` variable. The second assignment statement in the example assigns all of the characters contained in the `name` variable, beginning with the character whose subscript is 8, to the `last` variable. The character whose subscript is 8 is the letter T; therefore, the statement assigns "Timkas" to the `last` variable.

The `if` clause's condition in Example 2 compares the first character contained in the `sales` variable with the dollar sign. If the condition evaluates to true, the `sales = sales.substr(1);` statement assigns all of the characters from the `sales` variable, beginning with the character whose subscript is 1, to the `sales` variable. In other words, the statement assigns all of the characters *except* the dollar sign to the variable.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

In Example 3 in Figure 13-12, the `if` clause's condition uses the `substr` and `length` functions to determine whether the string stored in the `rate` variable ends with the percent sign. If the condition evaluates to true, the `rate = rate.substr(0, rate.length() - 1);` statement assigns the `rate` variable's contents, excluding the last character (which is the percent sign), to the `rate` variable.

Figure 13-13 shows the C++ code for a modified version of the ZIP Code program from Figure 13-11. The modified program now contains a value-returning function named `verifyNumeric`. The function returns a character—either Y or N—that indicates whether each of the five characters entered by the user is a number. The modifications made to the original `main` function are shaded in the figure, which also contains a sample run of the modified program. The statement containing the `substr` function in the `verifyNumeric` function is also shaded in the figure.

```
 1 //Modified ZIP Code.cpp
 2 //displays a message indicating whether a ZIP code
 3 //contains the appropriate number of characters
 4 //and whether each character is a number
 5 //Created/revised by <your name> on <current date>
 6
 7 #include <iostream>
 8 #include <string>
 9 using namespace std;
10
11 //function prototype
12 char verifyNumeric(string zip);
13
14 int main()
15 {
16    string zipCode = "";
17    char isAllNumbers = ' ';
18
19    cout << "Five-character ZIP code (-1 to end): ";
20    cin >> zipCode;
21
22    while (zipCode != "-1")
23    {
24        if (zipCode.length() == 5)
25        {
26            cout << "-->Correct number of characters";
27            isAllNumbers = verifyNumeric(zipCode);
28            if (isAllNumbers == 'Y')
29                cout << endl << "-->All numbers";
30            else
31                cout << endl << "-->Not all numbers";
32            //end if
33        }
34        else
35            cout << "-->Incorrect number of characters";
36        //end if
37        cout << endl << endl;
38
```

Figure 13-13 Modified ZIP Code program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
39        cout << "Five-character ZIP code (-1 to end): ";
40        cin >> zipCode;
41    } //end while
42    return 0;
43 } //end of main function
44
45 //*****function definitions*****
46 char verifyNumeric(string zip)
47 {
48    //determine whether each character is a number
49    string currentChar = "";
50    int sub = 0;  //character subscript
51    char isANumber = 'Y';  //assume all numbers
52
53    while (sub < 5 && isANumber == 'Y')
54    {
55        currentChar = zip.substr(sub, 1);
56        if (currentChar >= "0" && currentChar <= "9")
57            //character is numeric, so check next character
58            sub += 1;
59        else
60            //character is not a number
61            isANumber = 'N';
62        //end if
63    }  //end while
64    return isANumber;
65 } //end of verifyNumeric function
```

```
Modified Zip Code
Five-character ZIP code (-1 to end): 123456
-->Incorrect number of characters

Five-character ZIP code (-1 to end): 1234
-->Incorrect number of characters

Five-character ZIP code (-1 to end): 12345
-->Correct number of characters
-->All numbers

Five-character ZIP code (-1 to end): 1234x
-->Correct number of characters
-->Not all numbers

Five-character ZIP code (-1 to end): -1
Press any key to continue . . .
```

You use a loop along with the `substr` function to access each character in a `string` variable.

Figure 13-13 Modified ZIP Code program

If the user enters exactly five characters, the statement on Line 27 in the `main` function calls the `verifyNumeric` function, passing it a copy of the string stored in the `zipCode` variable. The `verifyNumeric` function, which begins on Line 46, stores the characters it receives in its formal parameter: a `string` variable named `zip`.

The `while` clause on Line 53 tells the computer to repeat the loop instructions as long as both of the following subconditions evaluate to true: First, the value in the `sub` variable, which keeps track of the subscripts in the `zip` variable, must be less than 5 (the number of characters in

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

the ZIP code). Second, the `isANumber` variable, which keeps track of whether a nonnumeric character appears in the `zip` variable, must contain the character 'Y'. If both subconditions evaluate to true, the statement on Line 55 uses the `substr` function and the `sub` variable to access the current character in the `zip` variable; it assigns the character to the `currentChar` variable.

The condition in the `if` statement on Line 56 determines whether the current character is greater than or equal to "0" and less than or equal to "9". If the condition evaluates to true, it means that the character is a number. In that case, the statement on Line 58 increases the `sub` variable's value by 1; doing this allows the loop to look at the next character in the `zip` variable. If the condition evaluates to false, on the other hand, it means that the character is not a number. In that case, the statement on Line 61 assigns the character 'N' to the `isANumber` variable.

Recall that a string is equivalent to a one-dimensional array of characters.

When the `while` loop in the `verifyNumeric` function ends, the `return` statement on Line 64 returns either the character 'Y' or the character 'N' to the `main` function. The 'Y' indicates that the ZIP code contains only numbers, and the 'N' indicates that it contains at least one nonnumeric character. The statement on Line 27 in the `main` function assigns the returned character to the `isAllNumbers` variable.

Next, the `if` statement's condition on Line 28 compares the character stored in the `isAllNumbers` variable with the character 'Y'. If the condition evaluates to true, the statement's true path displays the "-->All numbers" message; otherwise, its false path displays the "-->Not all numbers" message.

Mini-Quiz 13-2

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

1. Which of the following will process the loop instructions as long as the `employee` variable contains more than 20 characters?

   a. `while (employee.length() > 20)`
   b. `while (employee.length() > "20");`
   c. `while (employee.length() > '20');`
   d. `while (length(employee) > 20)`

2. Write a C++ `if` clause that determines whether a `string` variable named `code` contains seven characters.

3. The `cityState` variable contains the string "Los Angeles, CA". Which of the following assigns the state ID ("CA") to a `string` variable named `state`?

   a. `state = cityState.substr(13);`
   b. `state = cityState.substr(13, 2);`
   c. `state = cityState.substr(14, 2);`
   d. both a and b

4. Write a `cout` statement that displays the last character contained in a `string` variable named `college`.

Searching the Contents of a `string` Variable

At times, you may need to search the contents of a `string` variable to determine whether it contains a specific sequence of characters. For example, you may need to determine whether a phone number contains a certain area code or whether a specific street name appears in

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

an address. The `string` class provides the **find function** for performing such searches. In the function's syntax, which is shown in Figure 13-14, *string* is the name of the `string` variable whose contents you want to search, and *searchString* is the string for which you are searching. The *searchString* argument can be a `string` literal constant or the name of either a `string` variable or a `string` named constant. The *subscript* argument specifies the starting position for the search. In other words, it specifies the subscript of the character at which the search should begin.

The `find` function searches for the *searchString* in the *string*, starting with the character whose subscript is specified in the *subscript* argument. The function performs a case-sensitive search, which means that uppercase letters are not equivalent to their lowercase counterparts. When the *searchString* is contained within the *string*, the `find` function returns an integer that indicates the beginning position (subscript) of the *searchString* within the *string*. The function returns the number –1 when the *searchString* is not contained within the *string*. Figure 13-14 also includes examples of using the `find` function.

HOW TO Use the `find` Function

Notice the space after the letter m in the `find` function in Examples 3, 4, and 5.

Syntax

string.**find(***searchString*, *subscript***)**

Example 1

```
int location = 0;
string phone = "(312) 999-9999";
location = phone.find("(312)", 0);
```

searches for the string "(312)" in the `phone` variable, starting with the first character (subscript 0); stores the result (0) in the `location` variable

Example 2

```
int spaceLocation = 0;
string name = "Carol Cho";
spaceLocation = name.find(" ", 1);
```

searches for the space character in the `name` variable, starting with the second character (subscript 1); stores the result (5) in the `spaceLocation` variable

Example 3

```
int location = 0;
string address = "210 Elm Street, Elmwood, NJ";
location = address.find("Elm ", 2);
```

searches for the string "Elm " in the `address` variable, starting with the third character (subscript 2); stores the result (4) in the `location` variable

Example 4

```
int location = 0;
string address = "210 Elm Street, Elmwood, NJ";
location = address.find("elm ", 0);
```

searches for the string "elm " in the `address` variable, starting with the first character (subscript 0); stores the result (–1) in the `location` variable (Recall that the `find` function performs a case-sensitive search.)

Figure 13-14 How to use the `find` function *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Example 5

```
int location = 0;
string address = "210 Elm Street, Elmwood, NJ";
location = address.find("Elm ", 9);
```

searches for the string "Elm " in the `address` variable, starting with the tenth character (subscript 9); stores the result (–1) in the `location` variable

Figure 13-14 How to use the `find` function

The assignment statement in Example 1 in Figure 13-14 searches for the *searchString* "(312)" in the `phone` variable, beginning with the first character in the variable. It then assigns the result—in this case, the number 0—to the `location` variable. The number 0 is assigned because the *searchString* "(312)" begins with the first character in the `phone` variable.

The assignment statement in Example 2 searches for the space character in the `name` variable, starting with the second character. The space character is the sixth character in the `name` variable, which means its subscript is 5. Therefore, the statement assigns the number 5 to the `spaceLocation` variable.

The assignment statement in Example 3 searches the third through the last characters in the `address` variable, looking for the string "Elm " (notice the space after the letter m). The statement assigns the number 4 to the `location` variable because the string "Elm " begins with the character whose subscript is 4 in the `address` variable.

The assignment statement in Example 4 searches for the string "elm " (notice the space after the letter m) in the `address` variable, starting with the first character. The statement assigns the number –1 to the `location` variable because the `address` variable does not contain the string "elm ". (Recall that the `find` function performs a case-sensitive search.)

The assignment statement in Example 5 searches for the string "Elm " (notice the space after the letter m) in the tenth through the last characters in the `address` variable. The statement assigns the number –1 to the `location` variable because the string "Elm " does not appear in the tenth through the last characters. In other words, it doesn't appear in the characters "treet, Elmwood, NJ".

Figure 13-15 shows the C++ code for the Rearrange Name program. The program gets a person's first and last names from the user. It then displays the person's last name followed by a comma, a space, and the person's first name. The figure also includes a sample run of the program.

```
 1 //Rearrange Name.cpp - displays the last name
 2 //followed by a comma, a space, and the first name
 3 //Created/revised by <your name> on <current date>
 4
 5 #include <iostream>
 6 #include <string>
 7 using namespace std;
 8
 9 int main()
10 {
```

Figure 13-15 Rearrange Name program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
11    string firstLast = "";
12    string first = "";
13    string last = "";
14    int spaceLocation = 0;
15
16    //get first and last name
17    cout << "Name (first and last only): ";
18    getline(cin, firstLast);
19
20    //locate space, then pull out first and last names
21    spaceLocation = firstLast.find(" ", 0);
22    first = firstLast.substr(0, spaceLocation);
23    last = firstLast.substr(spaceLocation + 1);
24
25    //display rearranged name
26    cout << last << ", " << first << endl;
27    return 0;
28 } //end of main function
```

```
Rearrange Name
Name (first and last only): Heather Kingsley
Kingsley, Heather
Press any key to continue . . .
```

Figure 13-15 Rearrange Name program

The statement on Line 17 in Figure 13-15 prompts the user to enter a person's first and last names, and the statement on Line 18 stores the user's response in the `firstLast` variable. The statement on Line 21, which is shaded in the figure, uses the `find` function to locate the space between the names stored in the `firstLast` variable; it assigns the function's return value to the `spaceLocation` variable. If the user enters Heather Kingsley as the name, the statement assigns the number 7 to the `spaceLocation` variable. As illustrated in Figure 13-16, all of the characters to the left of the space character represent the first name. Likewise, all of the characters to the right of the space character represent the last name.

Start at 0 when determining a character's subscript. But start at 1 when counting characters.

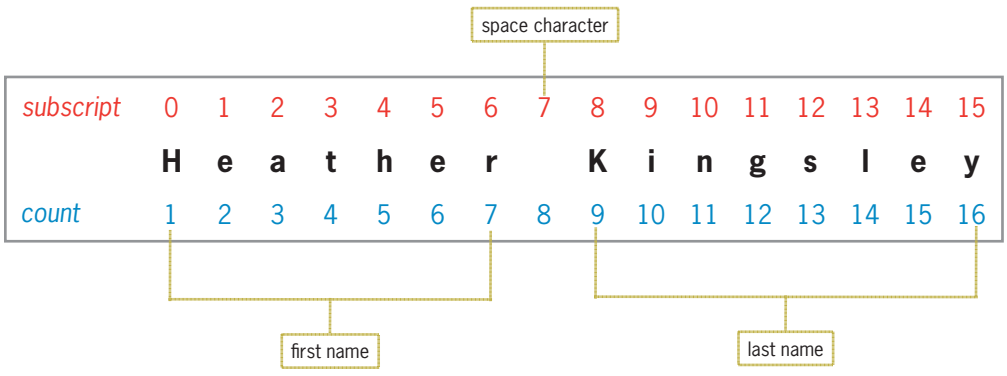


Figure 13-16 Location of the first and last names

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The `first = firstLast.substr(0, spaceLocation);` statement on Line 22 in Figure 13-15 uses the `substr` function to access only the first name from the `firstLast` variable. The first name begins with the first character in the variable. Therefore, the number 0 is used for the `substr` function's *subscript* argument. Recall that the function's second argument, *count*, specifies the number of characters you want to access. In this case, you want to access seven characters. You can use the space character's subscript (7), which is stored in the `spaceLocation` variable, to access the appropriate number of characters. The statement on Line 22 assigns the first name to the `first` variable.

The `last = firstLast.substr(spaceLocation + 1);` statement on Line 23 in Figure 13-15 uses the `substr` function to access only the last name from the `firstLast` variable. The last name begins with the character immediately after the space character in the variable. Therefore, the number 1 is added to the value stored in the `spaceLocation` variable, and the result (8) is used for the `substr` function's *subscript* argument. Because the *count* argument is omitted from the `substr` function, the statement on Line 23 will assign all of the remaining characters, beginning with the one whose subscript is 8, to the `last` variable.

Removing Characters from a `string` Variable

When validating user input, you may need to remove one or more characters from the user's entry, such as a dollar sign from the beginning of a sales amount or a percent sign from the end of a tax rate. The `string` class's **erase function** allows you to remove one or more characters located anywhere in a `string` variable. Figure 13-17 shows the function's syntax and includes examples of using the function. In the syntax, *string* is the name of a `string` variable, and the *subscript* argument is the subscript of the first character you want to remove (erase) from the variable's contents. The optional *count* argument is an integer that specifies the number of characters you want removed. If you omit the *count* argument, the `erase` function removes all characters from the *subscript* position through the end of the string.

HOW TO Use the `erase` Function

Syntax

string`.erase(`*subscript*[`, `*count*]`);`

Example 1

```
string place = "Miami, FL";
place.erase(0, 7);
```

removes the first seven characters from the `place` variable, changing the variable's contents to "FL"

Example 2

```
string place = "Miami, FL";
place.erase(5);
```

removes all of the characters from the `place` variable, beginning with the character whose subscript is 5, changing the variable's contents to "Miami"

Figure 13-17 How to use the `erase` function *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Example 3

```
string name = "Carol";
name.erase(3, 1);
```

removes the fourth character from the `name` variable, changing the variable's contents to "Carl"

Example 4

```
int x = 0;
string sales = "";
string currentChar = "";
cout << "Sales: ";
getline(cin, sales);
while (x < sales.length())
{
   currentChar = sales.substr(x, 1);
   if (currentChar == "$" || currentChar == ",")
       sales.erase(x, 1);
   else
       x += 1;
   //end if
} //end while
```

removes (erases) any dollar signs and commas from the `sales` variable

Figure 13-17 How to use the `erase` function

The `erase` function in Example 1 in Figure 13-17 removes the first seven characters from the string stored in the `place` variable. The first seven characters are the letters M, i, a, m, and i and the comma and space characters. After the function is processed, the `place` variable contains the string "FL".

The `erase` function in Example 2 removes all of the characters from the `place` variable, beginning with the character whose subscript is 5. In this case, the function removes the ", FL" portion of the string from the variable. After the function is processed, the `place` variable contains the string "Miami".

The `erase` function in Example 3 removes one character from the string stored in the `name` variable, beginning with the character whose subscript is 3; that character is the letter o. After the function is processed, the `name` variable contains the string "Carl".

The code in Example 4 contains a loop that looks at each character in the `sales` variable, one character at a time. The condition in the `if` statement within the loop compares the current character to both a dollar sign and a comma. If the current character is either of those characters, the `erase` function in the `if` statement's true path removes the character from the `sales` variable. Otherwise, the statement in its false path increments the `x` variable by 1; doing this allows the loop to look at the next character in the `sales` variable.

Figure 13-18 shows the C++ code for the Bonus program, which calculates a 10% bonus on the sales amount entered by the user. Before calculating the bonus, the program ensures that the sales amount contains only numbers. It does this using the `erase` function to remove any nonnumeric characters from the user's entry. The `erase` function appears on Line 25 and is shaded in the figure.

After removing the unwanted characters, the program calculates the bonus. The bonus calculation statement appears on Line 32 in Figure 13-18. The **stod function** in the statement

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

temporarily converts the string stored in the `sales` variable to the `double` data type. (C++ also provides a function named `stoi` for temporarily converting a string to the `int` data type.) The statement multiplies the result by the `double` number stored in the `RATE` named constant and then assigns that result to the `double bonus` variable. After calculating the bonus, the program displays the sales and bonus amounts on the computer screen. Figure 13-18 also contains a sample run of the program.

stod stands for *string to double*, and *stoi* stands for *string to integer*.

```
 1 //Bonus.cpp - displays sales and bonus amounts
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <iomanip>
 6 #include <string>
 7 using namespace std;
 8
 9 int main()
10 {
11    const double RATE = 0.1;
12    string sales = "";
13    string currentChar = "";
14    int sub = 0;
15    double bonus = 0.0;
16
17    cout << "Sales: ";
18    getline(cin, sales);
19
20    //remove all characters except numbers
21    while (sub < sales.length())
22    {
23        currentChar = sales.substr(sub, 1);
24        if (currentChar < "0" || currentChar > "9")
25            sales.erase(sub, 1);
26        else
27            sub += 1;
28        //end if
29    }    //end while
30
31    //calculate bonus
32    bonus = stod(sales) * RATE;
33
34    //display sales and bonus
35    cout << fixed << setprecision(2) << endl;
36    cout << "Sales amount: " << sales << endl;
37    cout << "Bonus amount: " << bonus << endl;
38    return 0;
39 }  //end of main function
```

temporarily converts a string to `double`

```
Bonus
Sales: $5, 7a9

Sales amount: 579
Bonus amount: 57.90
Press any key to continue . . .
```

Figure 13-18 Bonus program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Replacing Characters in a `string` Variable

Instead of using the `erase` function to code the Bonus program from the previous section, you can use the `string` class's `replace` function. The **replace function** replaces a sequence of characters in a `string` variable with another sequence of characters. For example, you can use the `replace` function to replace area code "800" with area code "877" in a phone number. Or, you can use it to replace the dashes in a Social Security number with the empty string.

Figure 13-19 shows the `replace` function's syntax and includes examples of using the function. In the syntax, *string* is the name of a `string` variable, and the *subscript* argument specifies where to begin replacing characters in the *string*. The *count* argument indicates the number of characters to replace, and the *replacementString* argument contains the replacement characters.

HOW TO Use the `replace` Function

Syntax

string.**replace(***subscript*, *count*, *replacementString*);

Example 1

subscript 2

```
string phone = "1-800-111-0000";
phone.replace(2, 3, "877");
```

beginning with the character whose subscript is 2, replaces three characters in the `phone` variable with "877"; changes the contents of the `phone` variable to "1-877-111-0000"

Example 2

subscript 3

```
string item = "ABCX34";
item.replace(3, 1, "D");
```

beginning with the character whose subscript is 3, replaces one character in the `item` variable with "D"; changes the contents of the `item` variable to "ABCD34"

Example 3

subscript 7

```
string name = "Karena Wilson";
name.replace(7, 6, "Farley");
```

beginning with the character whose subscript is 7, replaces six characters in the `name` variable with "Farley"; changes the contents of the `name` variable to "Karena Farley"

Example 4

```
int x = 0;
string sales = "";
cout << "Sales: ";
getline(cin, sales);
while (x < sales.length())
    if (sales.substr(x, 1) == "$" || sales.substr(x, 1) == ",")
           sales.replace(x, 1, "");
    else
           x += 1;
    //end if
//end while
```

replaces any dollar signs and commas in the `sales` variable with the empty string

Figure 13-19 How to use the `replace` function

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The partial Bonus program in Figure 13-20 shows how you can use the `replace` function instead of the `erase` function. The `replace` function is shaded in the figure.

Note: Lines 1 through 19 are the same as in Figure 13-18.

```
20    //remove all characters except numbers
21    while (sub < sales.length())
22    {
23        currentChar = sales.substr(sub, 1);
24        if (currentChar < "0" || currentChar > "9")
25            sales.replace(sub, 1, "");
26        else
27            sub += 1;
28        //end if
29    }    //end while
```

Note: The remainder of the program is the same as Lines 30 through 39 in Figure 13-18.

Figure 13-20 Partial Bonus program showing the `replace` function

Mini-Quiz 13-3

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

1. Which of the following searches for a comma in the `cityState` variable and then assigns the result to an `int` variable named `location`?

   a. `location = cityState.find(",", 0);`
   b. `location = cityState.find(0, ",");`
   c. `location = cityState.search(",", 0);`
   d. `location = cityState.searchFor(",");`

2. If the `cityState` variable contains the string "Bowling Green, KY", what will the statement from Question 1 assign to the `location` variable?

3. If the `cityState` variable contains the string "Bowling Green, KY", which of the following changes the variable's contents to "Bowling Green"?

   a. `cityState.erase(13);`
   b. `cityState.erase(13, 4);`
   c. `cityState.replace(13, 4, "");`
   d. all of the above

Inserting Characters Within a `string` Variable

The `string` class provides the **insert function** for inserting characters within a `string` variable. The function's syntax and examples of using the function are shown in Figure 13-21. In the syntax, *string* is the name of a `string` variable, and *subscript* specifies where in the *string* you want the *insertString* inserted. To insert the *insertString* at the beginning of the *string*, you use the number 0 as the *subscript*. To insert the *insertString* starting with the second character in the *string*, you use the number 1 as the *subscript*, and so on.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

HOW TO Use the `insert` Function

Syntax
string.**insert(***subscript*, *insertString*);

Example 1
```
string name = "Harold Cruthers";
name.insert(7, "G. ");
```
inserts the letter G, followed by a period and a space, between the first and last names stored in the `name` variable; changes the contents of the `name` variable to "Harold G. Cruthers"

Example 2
```
string phone = "312 050-1111";
phone.insert(0, "(");
phone.insert(4, ")");
```
inserts the opening and closing parentheses at the beginning and end, respectively, of the area code; changes the contents of the `phone` variable to "(312) 050-1111"

Example 3
```
string ssn = "111220000";
ssn.insert(3, "-");
ssn.insert(6, "-");
```
inserts two hyphens in the Social Security number, one after the third number and the other after the fifth number; changes the contents of the `ssn` variable to "111-22-0000"

Figure 13-21 How to use the `insert` function

The `insert` function in Example 1 in Figure 13-21 inserts the *insertString*—in this case, "G. " (the letter G, a period, and a space)—in the `name` variable. The letter G is inserted in subscript 7, which makes it the eighth character in the `name` variable. The period and space are inserted in subscripts 8 and 9, respectively, making them the ninth and tenth characters in the variable. After the function is processed, the `name` variable contains the string "Harold G. Cruthers".

In Example 2, the first `insert` function changes the contents of the `phone` variable from "312 050-1111" to "(312 050-1111". The second `insert` function in the example then changes the variable's contents to "(312) 050-1111". In Example 3, the first `insert` function changes the contents of the `ssn` variable from "111220000" to 111-220000", and the second `insert` function then changes the variable's contents to "111-22-0000".

Figure 13-22 shows the C++ code for the Social Security Number program. The program allows the user to enter a nine-character Social Security number. If the user's entry contains exactly nine characters, the program uses the `insert` function to insert hyphens in the appropriate places in the entry. The `insert` function appears twice in the program; both occurrences are shaded in the figure. After inserting the hyphens, the program displays the result on the screen. If the user's entry does not contain exactly nine characters, the program displays an appropriate message. Figure 13-22 also includes a sample run of the program.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

```
 1 //SSN.cpp - displays the Social Security number with hyphens
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <string>
 6 using namespace std;
 7
 8 int main()
 9 {
10    string ssn = "";
11    cout << "Nine-character Social Security number: ";
12    getline(cin, ssn);
13
14    if (ssn.length() == 9)
15    {
16        //insert hyphens
17        ssn.insert(3, "-");  //xxx-xxxxxx
18        ssn.insert(6, "-");  //xxx-xx-xxxx
19        cout << "Social Security number: " << ssn << endl;
20    }
21    else
22        cout << "The number must contain 9 characters" << endl;
23    //end if
24    return 0;
25 } //end of main function
```

```
Social Security Number
Nine-character Social Security number: 111220000
Social Security number: 111-22-0000
Press any key to continue . . .
```

Figure 13-22 Social Security Number program

Duplicating a Character Within a `string` Variable

You can use the `string` class's **assign function** to duplicate a single character a specified number of times and then assign the resulting string to a `string` variable. Figure 13-23 shows the function's syntax and includes examples of using the function. In the syntax, *string* is the name of a `string` variable that will store the duplicated characters. The *count* argument is an integer that specifies the number of times you want to duplicate the *character*. The *character* argument can be either a character literal constant enclosed in single quotation marks or the name of a `char` memory location. The `assign` function in Example 1 duplicates the asterisk character 10 times and then assigns the resulting string to the `asterisks` variable. The `assign` function in Example 2 duplicates the hyphen character zero or more times, depending on the number of characters in the `companyName` variable. It then assigns the resulting string to the `underline` variable.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

HOW TO Use the `assign` Function

Syntax
string.**assign(***count*, *character***);**

Example 1
```
string asterisks = "";
asterisks.assign(10, '*');
```
assigns 10 asterisks to the `asterisks` variable

Example 2
```
string companyName = "";
string underline = "";
cout << "Company name: ";
getline(cin, companyName);
underline.assign(companyName.length(), '-');
```
assigns zero or more hyphens to the `underline` variable; the number of hyphens depends on the number of characters in the `companyName` variable

Figure 13-23 How to use the `assign` function

Figure 13-24 shows the C++ code for the Company Name program. The program gets a company name from the user and then displays the name with a row of hyphens below it. The `assign` function appears on Line 18 and is shaded in the figure. The figure also includes a sample run of the program.

```
 1 //Company Name.cpp
 2 //displays the company name underlined with hyphens
 3 //Created/revised by <your name> on <current date>
 4
 5 #include <iostream>
 6 #include <string>
 7 using namespace std;
 8
 9 int main()
10 {
11    string companyName = "";
12    string underline   = "";
13
14    cout << "Company name: ";
15    getline(cin, companyName);
16
17    //assign the appropriate number of hyphens
18    underline.assign(companyName.length(), '-');
19
20    //display the company name and row of hyphens
21    cout << endl << companyName
22         << endl << underline << endl;
23    return 0;
24 }  //end of main function
```

Figure 13-24 Company Name program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

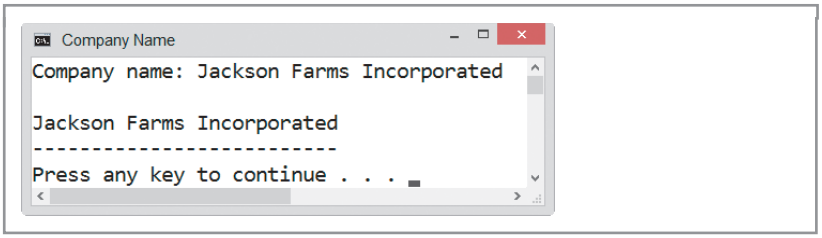


Figure 13-24 Company Name program

Concatenating Strings

The Company Name program, which you viewed in the previous section, used the `assign` function to assign zero or more hyphens to a `string` variable named `underline`. You can accomplish the same result using string concatenation. **String concatenation** refers to the process of connecting (or linking) strings together. You concatenate strings using the **concatenation operator**, which is the + sign in C++. Figure 13-25 shows examples of using the concatenation operator in a C++ statement.

HOW TO Use the Concatenation Operator

Example 1

```
string first = "Perry";
string last = "Lozinsky";
string full = "";
full = first + " " + last;
```

concatenates the contents of the `first` variable, a space, and the contents of the `last` variable and then assigns the result ("Perry Lozinsky") to the `full` variable

Example 2

```
string sentence = "How are you";
sentence = sentence + "?";
```

concatenates the contents of the `sentence` variable and a question mark and then assigns the result ("How are you?") to the `sentence` variable

Example 3

```
string companyName = "";
string underline = "";
cout << "Company name: ";
getline(cin, companyName);
for (int x = 1; x <= companyName.length(); x += 1)
      underline = underline + "-";
//end for
```

concatenates zero or more hyphens within the `underline` variable; the number of hyphens depends on the number of characters in the `companyName` variable (You can also write the assignment statement as `underline += "-";`.)

Figure 13-25 How to use the concatenation operator

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The partial Company Name program in Figure 13-26 shows how you can use string concatenation, instead of the `assign` function, in the Company Name program. The modifications made to the original code from Figure 13-24 are shaded in Figure 13-26. Although you could use the loop shown in Figure 13-26 to assign the appropriate number of hyphens to the `underline` variable, it is much easier to use the `assign` function for this purpose. The figure also shows a sample run of the program.

Line 19 can also be written as `underline += "-";`.

Note: Lines 1 through 16 are the same as in Figure 13-24.

```
17    //assign the appropriate number of hyphens
18    for (int x = 1; x <= companyName.length(); x += 1)
19        underline = underline + "-";
20    //end for
```

Note: The remainder of the program is the same as Lines 19 through 24 in Figure 13-24.

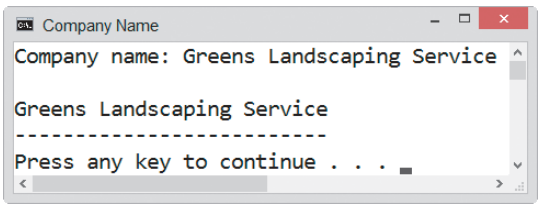


Figure 13-26 Partial Company Name program showing string concatenation

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 13-4

1. Which of the following changes the contents of the `cityState` variable from "Las Vegas Nevada" to "Las Vegas, Nevada"?

   a. `cityState.insert(10, ",");`
   b. `cityState.replace("s N", "s, N");`
   c. `cityState.assign(9, ",");`
   d. none of the above

2. The `temp` and `sentence` variables are `string` variables. Which of the following assigns four exclamation points to the `temp` variable and then concatenates the variable and the `sentence` variable?

   a. `sentence = sentence + temp.assign(4, '!');`
   b. `sentence = sentence + temp.assign(4, "!");`
   c. `sentence = sentence + temp.assign('!', 4);`
   d. none of the above

3. If the `areaCode` variable contains the string "212", which of the following changes the variable's contents to the string "(212)".

   a. `areaCode = "(" + "areaCode" + ")";`
   b. `areaCode = "(" + areaCode + ")";`
   c. `areaCode = '(' & areaCode & ')';`
   d. none of the above

For more examples of manipulating strings, see the Strings section in the Ch13WantMore.pdf file.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

LAB 13-1 Stop and Analyze

Study the program shown in Figure 13-27 and then answer the questions.

The answers to the labs are contained in the Answers.pdf file.

```
 1 //Lab13-1.cpp
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <string>
 6 using namespace std;
 7
 8 int main()
 9 {
10    string sales = "";
11    string currentChar = "";
12    int sub = 0;
13    int numNumbers = 0;
14    int numPeriods = 0;
15    int numOtherChars = 0;
16
17    cout << "Sales amount: ";
18    getline(cin, sales);
19
20    while (sub < sales.length())
21    {
22        currentChar = sales.substr(sub, 1);
23        if (currentChar == ".")
24            numPeriods += 1;
25        else
26            if (currentChar < "0" || currentChar > "9")
27                numOtherChars += 1;
28            else
29                numNumbers += 1;
30            //end if
31        //end if
32        sub += 1;
33    }    //end while
34
35    if (numPeriods > 1 || numOtherChars > 0)
36        cout << "Invalid sales amount" << endl;
37    else
38        cout << "Valid sales amount" << endl;
39    //end if
40    cout << "Numbers: " << numNumbers << endl;
41    cout << "Periods: " << numPeriods << endl;
42    cout << "Other characters: " << numOtherChars << endl;
43
44    return 0;
45 } //end of main function
```

Figure 13-27 Code for Lab 13-1

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

QUESTIONS

1. What is the purpose of the loop on Lines 20 through 33?
2. What is the purpose of the statement on Line 22?
3. What is the purpose of the selection structure on Lines 35 through 39?
4. Why is the statement on Line 32 necessary?
5. Follow the instructions for starting C++ and viewing the Lab13-1.cpp file, which is contained in either the Cpp8\Chap13\Lab13-1 Project folder or the Cpp8\Chap13 folder. (Depending on your C++ development tool, you may need to open Lab13-1's project/solution file first.) Run the program. Test the program using the following four sales amounts: 123.45, $67.5.3, 34 5a6, and 4,500.
6. Make the following two modifications to the program. First, change the `while` statement to a `for` statement. Second, in addition to displaying the number of other characters, display the characters themselves, as shown in Figure 13-28. Save and then run and test the program.

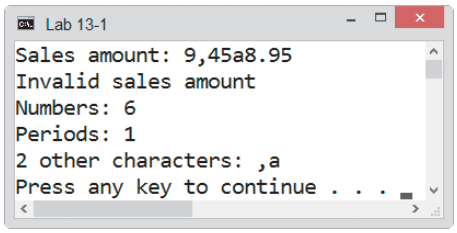


Figure 13-28 Sample run of the modified Lab 13-1 program

LAB 13-2 Plan and Create

In this lab, you will plan and create an algorithm for Mr. Coleman. The problem specification, IPO chart information, and C++ instructions are shown in Figure 13-29.

Problem specification
Mr. Coleman teaches second grade at Hinsbrook School. On days when the weather is bad and the students cannot go outside to play, he spends recess time playing the Guess the Word game with his class. The game requires two people to play. Currently, Mr. Coleman thinks of a word that has five letters. He then draws five dashes on the chalkboard—one for each letter in the word. One student then is chosen to guess the word, letter by letter. When the student guesses a correct letter, Mr. Coleman replaces the appropriate dash(es) with the letter. For example, if the original word is *moose* and the student guesses the letter *o*, Mr. Coleman changes the five dashes on the chalkboard to *-oo--*. The game is over when the student either guesses all of the letters in the word or makes 10 incorrect guesses, whichever occurs first. Mr. Coleman wants a program that allows two students to play the game on the computer.

Figure 13-29 Problem specification, IPO chart information, and C++ instructions for Lab 13-2 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

IPO chart information	C++ instructions
Input	
original word (from player 1) letter (from player 2)	`string origWord = "";` `string letter = "";`
Processing	
variable that keeps track of whether a dash was replaced ('N')	`char dashReplaced = 'N';`
variable that keeps track of whether the game is over ('N')	`char gameOver = 'N';`
number of incorrect guesses	`int numIncorrect = 0;`
Output	
display word (5 dashes when the program begins)	`string displayWord = "-----";`
Algorithm	

1. repeat while (the original word does not contain exactly five characters)
get original word
end while
	`while (origWord.length != 5)`
	`{`
	`cout << "Enter a 5-letter word in uppercase: ";`
	`getline(cin, origWord);`
	`}  //end while`

2. clear the screen	`system("cls");`
3. display the five dashes contained in the display word	`cout << "Guess this word: " << displayWord << endl;`
4. repeat while (the game is not over)	`while (gameOver == 'N')` `{`
get an uppercase letter	`cout << "Enter an uppercase letter: ";` `cin >> letter;`

repeat for (each letter in the original word)
if (the current character in the original word matches the letter)
replace the dash in the display word with the letter
	`for (int x = 0; x < 5; x += 1)`
	`{`
	`if (origWord.substr(x, 1) == letter)`
	`{`
	`displayWord.replace (x, 1,letter);`

assign 'Y' to the variable that keeps track of whether a dash was replaced end if	`dashReplaced = 'Y';` `}  //end if`

Figure 13-29 Problem specification, IPO chart information, and C++ instructions for Lab 13-2 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
    end repeat                                  } //end for
    if (a dash was replaced)                    if (dashReplaced == 'Y')
       if (the display word does not            {
       contain any dashes)                          if (displayWord.find("-", 0)
            assign 'Y' to the variable that         == -1)
            keeps track of whether the              {
            game is over                               gameOver = 'Y';
            display the original word
            display "Great guessing"                   cout << endl << "Yes, the
            message                                    word is "
                                                       << origWord << endl;
                                                       cout << "Great guessing!"
                                                       << endl;
                                                    }
       else                                         else
            display the status of                   {
            the display word                           cout << endl << "Guess
                                                       this word: "
            reset to 'N' the variable that keeps       << displayWord << endl;
            track of whether a dash was
            replaced                                   dashReplaced = 'N';
       end if                                       } //end if
    else                                        }
       add 1 to the number of incorrect         else
       guesses                                  {
       if (the number of incorrect guesses          numIncorrect += 1;
       is 10)
            assign 'Y' to the variable that         if (numIncorrect == 10)
            keeps track of whether the              {
            game is over                               gameOver = 'Y';
            display "Sorry, the word is"
            and the original word                      cout << endl << "Sorry,
       end if                                          the word is
    end if                                             " << origWord << endl;
end repeat
                                                    } //end if
                                                } //end if
                                            } //end while
```

Figure 13-29 Problem specification, IPO chart information, and C++ instructions for Lab 13-2

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Figure 13-30 shows the code for the entire Guess the Word game program, and Figures 13-31 and 13-32 show sample runs of the program using APPLE and HOUSE as the original words.

```
 1 //Lab13-2.cpp - Guess the Word game
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <string>
 6 //#include <cstdlib>
 7 using namespace std;
 8
 9 int main()
10 {
11    string origWord = "";
12    string letter = "";
13    char dashReplaced = 'N';
14    char gameOver = 'N';
15    int numIncorrect = 0;
16    string displayWord = "-----";
17
18    //get original word
19    while (origWord.length() != 5)
20    {
21           cout << "Enter a 5-letter word in uppercase: ";
22           getline(cin, origWord);
23    }  //end while
24
25    system("cls");  //clear the screen
26
27    //start guessing
28    cout << "Guess this word: " <<
29        displayWord << endl;
30    while (gameOver == 'N')
31    {
32        cout << "Enter an uppercase letter: ";
33        cin >> letter;
34
35        //search for the letter in the original word
36        for (int x = 0; x < 5; x += 1)
37        {
38            //if the current character matches
39            //the letter, replace the corresponding
40            //dash in the displayWord variable and then
41            //set the dashReplaced variable to 'Y'
42            if (origWord.substr(x, 1) == letter)
43            {
44                displayWord.replace(x, 1, letter);
45                dashReplaced = 'Y';
46            }  //end if
47        }  //end for
48
49        //if a dash was replaced, check whether the
```

your C++ development tool may require this directive to use the statement on Line 25

Figure 13-30 Guess the Word game program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
50        //displayWord variable contains another dash
51        if (dashReplaced == 'Y')
52        {
53            //if the displayWord variable does not
54            //contain any dashes, the game is over
55              if (displayWord.find("-", 0) == -1)
56              {
57                gameOver = 'Y';
58                cout << endl << "Yes, the word is "
59                    << origWord << endl;
60                cout << "Great guessing!" << endl;
61              }
62              else  //otherwise, continue guessing
63              {
64                 cout << endl << "Guess this word: "
65                    << displayWord << endl;
66                 dashReplaced = 'N';
67              } //end if
68        }
69        else  //processed when dashReplaced contains 'N'
70        {
71            //add 1 to the number of incorrect guesses
72            numIncorrect += 1;
73            //if the number of incorrect guesses is 10,
74            //the game is over
75            if (numIncorrect == 10)
76            {
77                gameOver = 'Y';
78                cout << endl << "Sorry, the word is "
79                    << origWord << endl;
80            } //end if
81        } //end if
82    }  //end while
83    return 0;
84 } //end of main function
```

Figure 13-30 Guess the Word game program

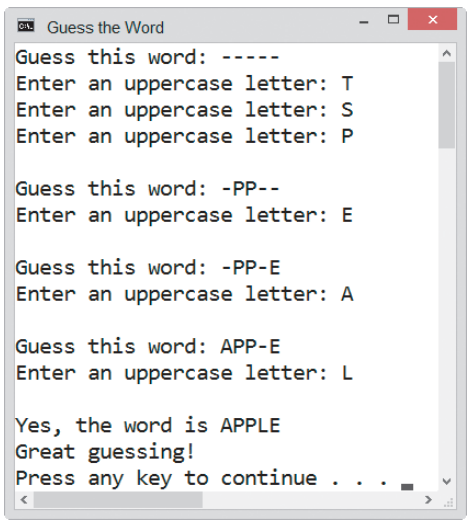


Figure 13-31 Sample run of the Guess the Word game program

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

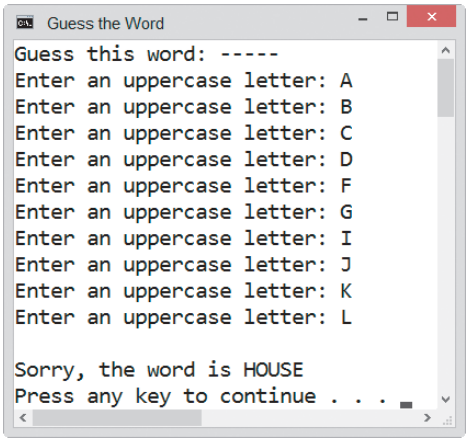


Figure 13-32 Another sample run of the Guess the Word game program

DIRECTIONS

Follow the instructions for starting your C++ development tool. Depending on the development tool you are using, you may need to create a new project; if so, name the project Lab13-2 Project and save it in the Cpp8\Chap13 folder. Enter the instructions shown in Figure 13-30 in a source file named Lab13-2.cpp. (Do not enter the line numbers.) Save the file in either the project folder or the Cpp8\Chap13 folder. Now follow the appropriate instructions for running the Lab13-2.cpp file. Test the program using an original word that does not contain exactly five characters. Also test the program using the words and letters shown in Figures 13-31 and 13-32. If necessary, correct any bugs (errors) in the program.

LAB 13-3 Modify

If necessary, create a new project named Lab13-3 Project and save it in the Cpp8\Chap13 folder. Enter (or copy) the Lab13-2.cpp instructions into a new source file named Lab13-3.cpp. Change Lab13-2.cpp in the first comment to Lab13-3.cpp. Currently, the program allows player 1 to enter only a five-character word. Modify the program so that player 1 can enter a word of any length. Save and then run the program. Test the program appropriately.

LAB 13-4 What's Missing?

The program in this lab should include commas (if necessary) when displaying the output. Start your C++ development tool, and view the Lab13-4.cpp file, which is contained in either the Cpp8\Chap13\Lab13-4 Project folder or the Cpp8\Chap13 folder. (Depending on your C++ development tool, you may need to open Lab13-4's project/solution file first.) Put the C++ instructions in the proper order, and then determine the one or more missing instructions. Test the program appropriately.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

LAB 13-5 Desk-Check

Desk-check the code shown in Figure 13-33. What will the code display on the screen?

```
 1 //Lab13-5.cpp - displays a message
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <string>
 6 using namespace std;
 7
 8 int main()
 9 {
10    string message = "vexprealjik";
11    string subMessage1 = "";
12    string subMessage2 = "";
13
14    message.erase(8, 2);
15    message.insert(7, "da");
16    message.replace(9, 2, "y");
17
18    subMessage1 = message.substr(0, 7);
19    subMessage1.replace(3, 1, "g");
20    subMessage1.erase(2, 1);
21    subMessage1.replace(1, 2, "eag");
22    subMessage1.insert(7, "t");
23    subMessage1.insert(0, "Ha");
24
25    subMessage2.assign(5, '!');
26    subMessage2 = message.substr(7) + subMessage2;
27
28    message = subMessage1 + subMessage2;
29    message.insert(4, " ");
30    message.insert(6, " ");
31    message.insert(12, " ");
32
33    //display message
34    cout << "Message: " << message << endl;
35    return 0;
36 } //end of main function
```

Figure 13-33 Code for Lab 13-5

LAB 13-6 Debug

Follow the instructions for starting C++ and viewing the Lab13-6.cpp file, which is contained in either the Cpp8\Chap13\Lab13-6 Project folder or the Cpp8\Chap13 folder. (Depending on your C++ development tool, you may need to open Lab13-6's project/solution file first.) Run the program. Type Joe and press Enter. Rather than displaying the letters J, o, and e on three separate lines, the program displays Joe, oe, and e. Debug the program.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Chapter Summary

- The `string` data type was added to the C++ language using the `string` class.
- Memory locations (variables and named constants) whose data type is `string` are initialized using string literal constants, which are zero or more characters enclosed in double quotation marks. Most `string` variables are initialized to the empty string.
- You can use the extraction operator to get a string from the user at the keyboard, but only if the string does not contain a white-space character (blank, tab, or newline).
- The `getline` function gets a string of characters entered at the keyboard and stores them in a `string` variable. The string can contain any characters, including white-space characters (blanks, tabs, and newlines). The `getline` function stops reading and storing characters when it encounters the delimiter character in the input. The function's default delimiter character is the newline character. The function reads and then consumes (discards) the delimiter character.
- The computer stores the characters entered at the keyboard in the `cin` object. Both the extraction operator and the `getline` function remove characters from the object. However, unlike the extraction operator, which leaves the newline character in the `cin` object, the `getline` function consumes the newline character.
- The `ignore` function reads and then consumes characters entered at the keyboard. The function stops reading and consuming characters when it consumes either a specified number of characters or the delimiter character, whichever occurs first. The default number of characters to consume is 1.
- You can use the C++ `stod` (*string* to *double*) function to convert a string to a `double` number. C++ also provides the `stoi` (*string* to *int*) function for converting a string to an `int` number.
- Figure 13-34 shows the syntax and purpose of each function covered in the chapter. It also includes the string concatenation operator. The `assign`, `erase`, `insert`, and `replace` functions are self-contained statements that change the value of the *string* variable.

Function/Operator	Syntax	Purpose
+		concatenate strings
`assign` function	*string*.**assign(***count*, *character***);**	duplicate a character within a `string` variable
`erase` function	*string*.**erase(***subscript*[, *count*]**);**	remove one or more characters located anywhere in a `string` variable
`find` function	*string*.**find(***searchString*, *subscript***)**	search a `string` variable to determine whether it contains a specific sequence of characters

Figure 13-34 Summary of `string` functions and the concatenation operator *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

`getline` function	**getline(cin,** *stringVariableName* [**,** *delimiterCharacter*]**);**	get string input from the keyboard
`ignore` function	**cin.ignore(**[*numberOfCharacters*] [**,** *delimiterCharacter*]**);**	read and consume characters entered at the keyboard
`insert` function	*string*.**insert(***subscript***,** *insertString***);**	insert characters within a `string` variable
`length` function	*string*.**length()**	determine the number of characters contained in a `string` variable
`replace` function	*string*.**replace(***subscript***,** *count***,** *replacementString***);**	replace a sequence of characters in a `string` variable with another sequence of characters
`substr` function	*string*.**substr(***subscript* [**,** *count*]**)**	access any number of characters contained in a `string` variable

Figure 13-34 Summary of `string` functions and the concatenation operator

Key Terms

assign function—duplicates a character a specified number of times within a string

Concatenation operator—used to concatenate (connect) strings; the + sign in C++

Consuming the character—another term for discarding the character

erase function—removes (erases) characters from a string

Escape sequence—the combination of the backslash and the character that follows; for example, the escape sequences `'\n'` and `'\t'` represent the Enter key and Tab key, respectively

find function—returns an integer that indicates the beginning position of a string within a `string` variable

getline function—reads characters entered at the keyboard until it encounters the delimiter character, which it consumes

ignore function—tells the computer to first read and then consume (discard) one or more characters

insert function—inserts characters within a string

length function—returns the number of characters contained in a `string` variable

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

replace function—used to replace characters within a string

stod function—converts a string to a `double` number

String concatenation—the process of connecting (or linking) strings together; accomplished with the concatenation operator (+)

substr function—returns the characters you want to access from a `string` variable

Review Questions

1. Which of the following displays the number of characters contained in a `string` variable named `address`?
   a. `cout << address.length() << endl;`
   b. `cout << numChars(address) << endl;`
   c. `cout << length(address) << endl;`
   d. `cout << size.address << endl;`

2. Which of the following should a program use to store the name of any city in a `string` variable named `cityName`?
   a. `cin >> cityName;`
   b. `cin(cityName);`
   c. `getline(cityName, cin);`
   d. `getline(cin, cityName);`

3. If the `amount` variable contains the string "$56.55", which of the following statements will remove the dollar sign from the variable's contents?
   a. `amount.erase("$");`
   b. `amount.erase(0, 1);`
   c. `amount = amount.substr(1);`
   d. both b and c

4. If the `state` variable contains the two letters MI followed by three spaces, which of the following statements will remove the three spaces from the variable's contents?
   a. `state.erase(" ");`
   b. `state.erase(3, "");`
   c. `state.remove(2, 3);`
   d. none of the above

5. What is the subscript of the first character contained in a `string` variable?
   a. 0 (zero)
   b. 1 (one)

6. Which of the following determines whether the string stored in the `part` variable begins with the letter A?
   a. `if (part.begins("A"))`
   b. `if (part.beginswith("A"))`
   c. `if (part.substr(0, 1) == "A")`
   d. `if (part.substr(1) == "A")`

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

7. Which of the following determines whether the string stored in the `part` variable ends with the letter B?

   a. `if (part.ends("B"))`
   b. `if (part.endswith("B")`
   c. `if (part.substr(part.length() - 1, 1) == "B")`
   d. none of the above

8. Which of the following statements assigns the first three characters in the `part` variable to the `code` variable?

   a. `code = part.assign(0, 3);`
   b. `code = part.substr(0, 3);`
   c. `code = part.substr(1, 3);`
   d. `code = part.substring(0, 3);`

9. If the `word` variable contains the string "Bells", which of the following statements will change the contents of the variable to "Bell"?

   a. `word.erase(word.length() - 1, 1);`
   b. `word.replace(word.length() - 1, 1, "");`
   c. `word = word.substr(0, word.length() - 1);`
   d. all of the above

10. Which of the following statements changes the contents of the `word` variable from "men" to "mean"?

    a. `word.addTo(2, "a");`
    b. `word.insert(2, "a");`
    c. `word.insert(3, "a");`
    d. none of the above

11. If the `msg` variable contains the string "Happy holidays", what will the `cout << msg.find("day", 0);` statement display on the screen?

    a. –1
    b. 0
    c. 10
    d. 11

12. If the `msg` variable contains the string "Happy holidays", what will the `location = msg.find("Day", 0);` statement assign to the `location` variable?

    a. –1
    b. 0
    c. 10
    d. 11

13. Which of the following statements assigns the location of the comma in the `amount` variable to an `int` variable named `loc`?

    a. `loc = amount.contains(",");`
    b. `loc = amount.substr(",");`
    c. `loc = amount.find(",", 0);`
    d. none of the above

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

14. Which of the following searches for the string "CA" in a `string` variable named `state` and then assigns the result to an `int` variable named `result`? The search should begin with the character located in subscript 5 in the `state` variable. The `state` variable's contents are uppercase.

    a. `result = find(state, 5, "CA");`
    b. `result = state.find(5, "CA");`
    c. `result = state.find("CA", 5);`
    d. `result = state.find("CA", 5, 2);`

15. If the `state` variable contains the string "San Francisco, CA", what will the correct statement in Question 14 assign to the `result` variable?

    a. –1
    b. 0
    c. 11
    d. 15

16. Which of the following replaces the two characters located in subscripts 4 and 5 in a `string` variable named `code` with the string "AB"?

    a. `code.replace(2, 4, "AB");`
    b. `code.replace(4, 2, "AB");`
    c. `code.replace(4, 5, "AB");`
    d. `replace(code, 4, "AB");`

17. Which of the following assigns five asterisks to a `string` variable named `divider`?

    a. `divider.assign(5, '*');`
    b. `divider.assign(5, "*");`
    c. `divider.assign('*', 5);`
    d. `assign(divider, '*', 5);`

18. Which of the following concatenates the contents of a `string` variable named `city`, a comma, a space, and the contents of a `string` variable named `state` and then assigns the result to a `string` variable named `cityState`?

    a. `cityState = "city" + ", " + "state";`
    b. `cityState = city + ", " + state;`
    c. `cityState = city & ", " & state;`
    d. `cityState = "city, + state";`

19. Which of the following assigns the fifth character in the `word` variable to the `letter` variable?

    a. `letter = word.substr(4);`
    b. `letter = word.substr(4, 1);`
    c. `letter = word(5).substring;`
    d. `letter = substring(word, 5);`

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

20. Which of the following statements tells the computer to consume the next 100 characters?

    a. `cin.ignore(100);`
    b. `cin.ignore('100');`
    c. `ignore(cin, 100);`
    d. none of the above

21. When processed, which of the following can consume the newline character?

    a. >> operator
    b. << operator
    c. `getline` function
    d. both a and c

Exercises

Pencil and Paper

TRY THIS

1. Write a C++ statement that assigns the number of characters contained in the `message` variable to an `int` variable named `numChars`. (The answers to TRY THIS Exercises are located at the end of the chapter.)

TRY THIS

2. Write a C++ statement that uses the `erase` function to remove the first two characters from the `message` variable. (The answers to TRY THIS Exercises are located at the end of the chapter.)

MODIFY THIS

3. Rewrite the code from Pencil and Paper Exercise 2 using the `replace` function.

INTRODUCTORY

4. Write a C++ statement that replaces the first character in a `string` variable named `code` with the letter B.

INTRODUCTORY

5. Write a C++ statement that assigns the first four characters in a `string` variable named `address` to a `string` variable named `streetNum`.

INTRODUCTORY

6. The `part` variable contains the string "ABCD34G". Write a C++ statement that assigns the 34 in the `part` variable to a `string` variable named `code`.

INTRODUCTORY

7. Write a C++ statement to change the contents of the `word` variable from "mend" to "amend".

INTRODUCTORY

8. Write a C++ statement to change the contents of the `word` variable from "mouse" to "mouth".

INTERMEDIATE

9. The `amount` variable contains the string "3,123,560". Write the C++ code to remove the commas from the contents of the variable.

INTERMEDIATE

10. Write the C++ code that uses the `substr` function to determine whether the string stored in the `rate` variable ends with the percent sign. If it does, the code should use the `replace` function to remove the percent sign from the variable's contents.

INTERMEDIATE

11. Write the C++ code to determine whether the `address` variable contains the street name "Grove Street". Begin the search with the fifth character in the `address` variable and assign the result to an `int` variable named `subNum` variable.

INTERMEDIATE

12. Write a C++ statement that searches for the period in a `string` variable named `amount` and then assigns the location of the period to an `int` variable named `location`. Begin the search with the first character in the `amount` variable.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

13. The `total` and `dollars` variables are `string` variables. Write the C++ code that uses the `assign` function to assign 10 asterisks to the `total` variable. The code should then concatenate the contents of the `total` variable and the contents of the `dollars` variable and then assign the resulting string to the `total` variable. INTERMEDIATE

14. A `string` variable named `amount` contains a string that has zero or more commas. Write the C++ code to count the number of commas in the string. Assign the result to an `int` variable named `numCommas`. INTERMEDIATE

15. Correct the following statement, which should change the contents of the `day` variable from "731" to "7/31": `day = day.insert(2, "/");`. SWAT THE BUGS

Computer

16. If necessary, create a new project named TryThis16 Project and save it in the Cpp8\Chap13 folder. Enter the C++ instructions from Figure 13-15 into a source file named TryThis16.cpp. Change the filename in the first comment to TryThis16.cpp. Save and then run the program. Test the program using the data shown in Figure 13-15 in the chapter. Now, modify the program so the user enters the last name followed by a comma, a space, and the first name. The program should display the first name followed by a space and the last name. Be sure to modify the comments that document the program's purpose. Save and then run the program. Test the program appropriately. (The answers to TRY THIS Exercises are located at the end of the chapter.) TRY THIS

17. If necessary, create a new project named TryThis17 Project and save it in the Cpp8\Chap13 folder. Also create a new source file named TryThis17.cpp. Write a program that allows the user to enter a string that represents a date. The date should be entered in the following format: mm/yy. Verify that the user entered exactly five characters and that the third character is the slash character (/). If the user did not enter the required number of characters, or if the third character is not a slash, display a message that indicates the type of entry error made by the user. Otherwise, the program should display the date in the following format: mm/20yy. Use a sentinel value to end the program. Save and then run the program. Test the program by entering the following dates: 6/08, 12/09, 05/10, and 123/4. (The answers to TRY THIS Exercises are located at the end of the chapter.) TRY THIS

18. In this exercise, you will modify the program from TRY THIS Exercise 17. If necessary, create a new project named ModifyThis18 Project and save it in the Cpp8\Chap13 folder. Copy the instructions from the TryThis17.cpp file into a source file named ModifyThis18.cpp. Change the filename in the first comment to ModifyThis18.cpp. Modify the program so that it allows the user to enter the date in the following format: mm/dd/yy. Verify that the user entered exactly eight characters and that the third and sixth characters are slashes (/). If the user did not enter the required number of characters, or if the third and sixth characters are not slashes, display a message that indicates the type of entry error made by the user. Otherwise, the program should display the date in the following format: mm/dd/20yy. Save and then run the program. Test the program appropriately. MODIFY THIS

19. In this exercise, you modify one of the ZIP code programs from the chapter. If necessary, create a new project named ModifyThis19 Project and save it in the Cpp8\Chap13 folder. Enter the C++ instructions from Figure 13-13 into a new source file named ModifyThis19.cpp. Change the filename in the first comment. Save and then run the program. Test the program using the data shown in Figure 13-13 in the chapter. MODIFY THIS

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Now, modify the program so that it allows the user to enter either a five-character ZIP code or a nine-character ZIP code. Pass the number of characters in the ZIP code to the `verifyNumeric` function. Save and then run the program. Test the program appropriately.

INTRODUCTORY

20. If necessary, create a new project named Introductory20 Project and save it in the Cpp8\Chap13 folder. Also create a new source file named Introductory20.cpp. Write a program that displays the appropriate shipping charge based on the region code entered by the user. To be valid, the region code must contain exactly three characters: a letter (either A or B) followed by two numbers. The shipping charge for region A is $25. The shipping charge for region B is $30. Display an appropriate message if the region code is invalid. Use a sentinel value to end the program. Save and then run the program. Test the program using the following region codes: A11, B34, C7, D2A, A3N, C45, and 74TV.

INTRODUCTORY

21. If necessary, create a new project named Introductory21 Project and save it in the Cpp8\Chap13 folder. Also create a new source file named Introductory21.cpp. Write a program that allows the user to enter three separate strings: a city name, a state name, and a ZIP code. The program should use string concatenation to display the city name followed by a comma, a space, the state name, two spaces, and the ZIP code. Use a sentinel value to end the program. Save and then run the program. Test the program.

INTERMEDIATE

22. In this exercise, you modify the program from ModifyThis Exercise 19. If necessary, create a new project named Intermediate22 Project and save it in the Cpp8\Chap13 folder. Enter (or copy) the C++ instructions from the ModifyThis19.cpp file into a new source file named Intermediate22.cpp. Change the filename in the first comment. Modify the program so that it also allows the user to enter a 10-character ZIP code, as long as the sixth character is a hyphen. Remove the hyphen before sending the ZIP code to the `verifyNumeric` function. Save and then run the program. Test the program appropriately.

INTERMEDIATE

23. If necessary, create a new project named Intermediate23 Project and save it in the Cpp8\Chap13 folder. Also create a new source file named Intermediate23.cpp. Write a program that displays the color of the item whose item number is entered by the user. All item numbers contain exactly seven characters. All items are available in four colors: blue, green, red, and white. The fourth character in the item number indicates the item's color, as follows: a B or b indicates Blue, a G or g indicates Green, an R or r indicates Red, and a W or w indicates White. If the item number does not contain exactly seven characters, or if the fourth character is not one of the valid color characters, the program should display an appropriate message. Use a sentinel value to end the program. Save and then run the program. Test the program using the following item numbers: 123B567, 34AG123, 111r222, 111w222, 123, 1234567, and 111k456.

INTERMEDIATE

24. In this exercise, you modify the Social Security Number program from the chapter. If necessary, create a new project named Intermediate24 Project and save it in the Cpp8\Chap13 folder. Enter the instructions from Figure 13-22 into a source file named Intermediate24.cpp. Change the filename in the first comment. Before inserting the missing hyphens, verify that the nine characters entered by the user are numeric. Save and then run and test the program.

INTERMEDIATE

25. If necessary, create a new project named Intermediate25 Project and save it in the Cpp8\Chap13 folder. Also create a new source file named Intermediate25.cpp. Write a program that accepts a string of characters from the user. The program should display the characters in reverse order. In other words, if the user enters the string "Programming", the program should display "gnimmargorP". Save and then run and test the program.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

26. If necessary, create a new project named Intermediate26 Project and save it in the Cpp8\Chap13 folder. Also create a new source file named Intermediate26.cpp. Write a program that allows the user to enter a part number that consists of four or five characters. The second and third characters represent the delivery method, as follows: "MS" represents "Mail – Standard", "MP" represents "Mail – Priority", "FS" represents "FedEx – Standard", "FO" represents "FedEx – Overnight", and "UP" represents "UPS". Display an appropriate message when the part number does not contain either four or five characters. Also display an appropriate message when the second and third characters are not one of the delivery methods. If the part number is valid, the program should display the delivery method. Use a sentinel value to end the program. Save and then run the program. Test the program using the following part numbers: 7MP6, 3fs5, 2UP7, 7mS89, 9FO8, 9fo89, 8ko89, and 1234MS. INTERMEDIATE

27. In this exercise, you modify the program from Lab 13-2. If necessary, create a new project named Intermediate27 Project and save it in the Cpp8\Chap13 folder. Also create a new source file named Intermediate27.cpp. Copy the C++ instructions from the Lab13-2.cpp file into the Intermediate27.cpp file. Change the filename in the first comment. Modify the program so that it displays a message indicating the number of incorrect guesses remaining. Display the message each time the user enters an incorrect guess. INTERMEDIATE

28. If necessary, create a new project named Advanced28 Project and save it in the Cpp8\Chap13 folder. Also create a new source file named Advanced28.cpp. Write a program that determines whether the user entered an item number in the required format: three digits, a hyphen, and two digits. Display an appropriate message indicating whether the format is correct. Use a sentinel value to end the program. Save and then run the program. ADVANCED

29. Follow the instructions for starting C++ and viewing the Advanced29.cpp file, which is contained in either the Cpp8\Chap13\Advanced29 Project folder or the Cpp8\Chap13 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) If necessary, delete the two forward slashes that appear before the `#include <cstdlib>` directive, and then save the program. The program assigns the letters of the alphabet to a `string` variable named `letters`. It also prompts the user to enter a letter. Complete the program by entering instructions to perform the tasks listed in Figure 13-35. Save and then run the program. Test the program appropriately. ADVANCED

1. Generate a random number that can be used to select one of the letters from the `letters` variable. Assign the letter to the `randomLetter` variable.
2. Verify that the user entered exactly one lowercase letter. If the user did not enter exactly one lowercase letter, display an appropriate error message.
3. If the user entered exactly one lowercase letter, compare the letter to the random letter. If the letter entered by the user is the same as the random letter, display the message "You guessed the correct letter." and then end the program. Otherwise, display messages indicating whether the correct letter comes alphabetically before or after the letter entered by the user.
4. Allow the user to enter a letter until he or she guesses the random letter.

Figure 13-35

30. In this exercise, you modify the program from ADVANCED Exercise 29. If necessary, create a new project named Advanced30 Project and save it in the Cpp8\Chap13 folder. Also create a new source file named Advanced30.cpp. Copy the C++ instructions from the Advanced29.cpp file into the Advanced30.cpp file. Research the C++ `compare` function. Modify the program to use the `compare` function. Save and then run the program. Test the program appropriately. ADVANCED

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

ADVANCED

31. In this exercise, you modify the program from Lab 13-2. If necessary, create a new project named Advanced31 Project and save it in the Cpp8\Chap13 folder. Also create a new source file named Advanced31.cpp. Copy the C++ instructions from the Lab13-2.cpp file into the Advanced31.cpp file. Change the filename in the first comment. Modify the program so that it keeps track of the letters guessed by the user. If the user enters a letter that he or she has already entered, display an appropriate message, and do not include the letter in the number of incorrect guesses. Save and then run the program. Test the program appropriately.

ADVANCED

32. In this exercise, you modify the program from ADVANCED Exercise 31. If necessary, create a new project named Advanced32 Project and save it in the Cpp8\Chap13 folder. Also create a new source file named Advanced32.cpp. Copy the instructions from the Advanced31.cpp file into the Advanced32.cpp file. Change the filename in the first comment. Modify the program so that it displays the letters already entered by the user. Display the letters immediately before prompting the user to enter a letter. Save and then run the program. Test the program appropriately.

ADVANCED

33. Some credit card companies assign a special digit, called a check digit, to the end of each customer's credit card number. Many methods for creating the check digit have been developed. One very simple method is to append the second digit in the credit card number to the end of the number. For example, if the first four characters in the credit card number are 1357, you would append the number 3 to the end of the number, making the credit card number 13573. If necessary, create a new project named Advanced33 Project and save it in the Cpp8\Chap13 folder. Also create a new source file named Advanced33.cpp. Write a program that prompts the user to enter a five-digit credit card number, with the fifth digit being the check digit. Verify that the user entered exactly five numbers. If the user entered the required number of numbers, verify that the last number is the check digit. Display appropriate messages indicating whether the credit card number is valid or invalid. Use a sentinel value to end the program. Save and then run the program. Test the program appropriately.

ADVANCED

34. If necessary, create a new project named Advanced34 Project and save it in the Cpp8\Chap13 folder. Also create a new source file named Advanced34.cpp. Create a program that allows the user to enter a password. The program should then create and display a new password using the rules listed in Figure 13-36. Use a sentinel value to end the program.

1. All vowels (A, E, I, O, and U) in the original password should be replaced with the letter X.
2. All numbers in the original password should be replaced with the letter Z.
3. All of the characters in the original password should be reversed.

Figure 13-36

ADVANCED

35. In this exercise, you modify the program from Lab 13-2. If necessary, create a new project named Advanced35 Project and save it in the Cpp8\Chap13 folder. Also create a new source file named Advanced35.cpp. Copy the C++ instructions from the Lab13-2.cpp file into the Advanced35.cpp file. Change the prompt on Line 21 to "Enter a 5-letter word: ". Also change the prompt on Line 32 to "Enter a letter: ". Modify the program so that it converts both the 5-letter word and the letter to uppercase. Save and then run the program. Test the program appropriately. (Hint: Recall that a string is equivalent to a one-dimensional array of characters.)

ADVANCED

36. If necessary, create a new project named Advanced36 Project and save it in the Cpp8\Chap13 folder. Also create a new source file named Advanced36.cpp. Each salesperson at Rembrandt Auto-Mart is assigned an ID number that consists of five characters. The

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

first three characters are numbers. The fourth character is a letter: either the letter N if the salesperson sells new cars or the letter U if the salesperson sells used cars. The fifth character is also a letter: either the letter F if the salesperson is a full-time employee or the letter P if the salesperson is a part-time employee. Create a program that allows the sales manager to enter the ID and the number of cars sold for as many salespeople as needed. Use a sentinel value to stop the program. When the sales manager has finished entering the data, the program should display the total number of cars sold by each of the following four categories of employees: full-time employees, part-time employees, employees selling new cars, and employees selling used cars. Save, run, and test the program.

37. Follow the instructions for starting C++ and viewing the SwatTheBugs37.cpp file, which is contained in either the Cpp8\Chap13\SwatTheBugs37 Project folder or the Cpp8\Chap13 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) The program should calculate and display the total of the prices entered by the user. Run the program. Use your own data to test the program. Notice that the program is not working correctly. Debug the program.

SWAT THE BUGS

Answers to TRY THIS Exercises

Pencil and Paper

1. `numChars = message.length();`
2. `message.erase(0, 2);`

Computer

16. See Figure 13-37.

```
//TryThis16.cpp - displays the first name followed
//by a space and the last name
//Created/revised by <your name> on <current date>

#include <iostream>
#include <string>
using namespace std;

int main()
{
    string firstLast = "";
    string first = "";
    string last = "";
    int commaLocation = 0;

    //get first and last name
    cout << "Name (last, comma, space, first): ";
    getline(cin, firstLast);
```

Figure 13-37 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
    //locate comma, then pull out first and last names
    commaLocation = firstLast.find(",", 0);
    first = firstLast.substr(commaLocation + 2);
    last = firstLast.substr(0, commaLocation);

    //display rearranged name
    cout << first << " " << last << endl;
    return 0;
}    //end of main function
```

Figure 13-37

17. See Figure 13-38.

```
//TryThis17.cpp
//displays a date using the format mm/20yy
//Created/revised by <your name> on <current date>

#include <iostream>
#include <string>
using namespace std;

int main()
{
    string date = "";

    cout << "Enter date (mm/yy). Enter -1 to end. ";
    getline(cin, date);
    while (date != "-1")
    {
        if (date.length() != 5)
            cout << "Invalid length" << endl << endl;
        else
            if (date.substr(2, 1) != "/")
                cout << "Invalid third character"
                << endl << endl;
            else
            {
                date.insert(3, "20");
                cout << date << endl << endl;
            }     //end if
        //end if

        cout << "Enter date (mm/yy). Enter -1 to end. ";
        getline(cin, date);
    }    //end while
    return 0;
}    //end of main function
```

Figure 13-38

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

CHAPTER 14

Sequential Access Files

After studying Chapter 14, you should be able to:

- Create file objects
- Open a sequential access file
- Determine whether a sequential access file was opened successfully
- Write data to a sequential access file
- Read data from a sequential access file
- Test for the end of a sequential access file
- Close a sequential access file

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

File Types

In addition to getting data from the keyboard and sending data to the computer screen, a program can also read data from and write data to a file on a permanent or secondary storage device (such as a flash drive). Files to which data is written are called **output files** because the files store the output produced by a program. Files that are read by the computer are called **input files** because a program uses the data in the files as input.

Ch14-Chapter Preview

Most input and output files are composed of lines of text that are both read and written in consecutive order, one line at a time, beginning with the first line in the file and ending with the last line in the file. Such files are referred to as **sequential access files** because of the manner in which the lines of text are accessed. They are also called **text files** because they are composed of lines of text. Figure 14-1 shows examples of text you might find stored in sequential access files.

Company officers
J.P. Stinson, Chairman of the Board
Heather Chen, CEO
Henry Kasper, President
Sharon Longley, Vice President (Corporate)
Eric Mendez, Vice President (Sales)

Quarterly sales report

	Store 1	Store 2	Store 3
January	80,000	75,000	100,000
February	81,500	73,300	99,500
March	86,000	76,500	107,000
Total	247,500	224,800	306,500

Company memo
To: All employees

From: Keith Herman

The annual company picnic will be held on July 21st at Seminole Park. All employees and their families are invited. Please join us for some food, beverages, and fun!

Figure 14-1 Examples of text stored in sequential access files

You can also create random access and binary access files in C++. The data stored in a random access file can be accessed in either consecutive or random order. The data in a binary access file can be accessed by its byte location in the file. Random access and binary access files are used less often in programs and, therefore, are not covered in this book.

Creating File Objects

In previous chapters, you used stream objects to perform standard input and output operations in a program. The standard input stream object (`cin`) refers to the computer keyboard, and the standard output stream object (`cout`) refers to the computer screen. A program that uses the `cin` and `cout` objects must contain the `#include <iostream>` directive, which tells the compiler to include the contents of the iostream file in the program. The iostream file contains the definitions of the `istream` and `ostream` classes from which the `cin` and `cout` objects, respectively, are created. You do not have to create the `cin` and `cout` objects in a program because C++ creates those objects for you.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Objects are also used to perform file input and output operations in C++. However, unlike the standard `cin` and `cout` objects, the input and output file objects must be created by the programmer. To create a file object in a program, the program must contain the `#include <fstream>` directive, which tells the compiler to include the contents of the fstream file in the program. The fstream file contains the definitions of the `ifstream` (input file stream) and `ofstream` (output file stream) classes, which allow you to create input and output file objects, respectively.

All objects in C++ are created from a class and are referred to as an instance of the class. A `cin` object is an instance of the `istream` class, whereas an input file object is an instance of the `ifstream` class.

Figure 14-2 shows the syntaxes for creating input file objects and output file objects. In the first syntax, `ifstream` is the name of the class from which all input file objects are created. Similarly, `ofstream` in the second syntax is the name of the class from which all output file objects are created. In each syntax, *fileObject* is the name of the file object you want to create. Notice that a semicolon appears at the end of each syntax.

Also included in Figure 14-2 are examples of creating file objects. The statements in Examples 1 and 2 create input file objects named `inFile` and `inEmploy`, respectively. The statements in Examples 3 and 4 create output file objects named `outFile` and `outSales`, respectively. Notice that the names of the input file objects in the examples begin with the two letters `in`, whereas the names of the output file objects begin with the three letters `out`. Although the C++ syntax does not require you to begin file object names with either `in` or `out`, using this naming convention helps to distinguish a program's input file objects from its output file objects.

HOW TO Create Input and Output File Objects

Syntax

To create an input file object: **ifstream** *fileObject*;
To create an output file object: **ofstream** *fileObject*;

Example 1

```
ifstream inFile;
```

creates an input file object named `inFile`

Example 2

```
ifstream inEmploy;
```

creates an input file object named `inEmploy`

Example 3

```
ofstream outFile;
```

creates an output file object named `outFile`

Example 4

```
ofstream outSales;
```

creates an output file object named `outSales`

Figure 14-2 How to create input and output file objects

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Opening a Sequential Access File

The `open` function is defined in the `ifstream` and `ofstream` classes and is referred to as a class member function.

You use a program's input and output file objects, along with the C++ `open` function, to open actual files on your computer's disk. Figure 14-3 shows the `open` function's syntax and describes the modes most commonly used to open a sequential access file. In the syntax, *fileObject* is the name of either an existing `ifstream` file object or an existing `ofstream` file object, and *fileName* is the name of the file (including an optional path) you want to open. The *fileName* argument can be either a string literal constant or a `string` variable. If the *fileName* argument does not contain a path, the computer assumes that the file is located in the same folder as the program file. (See the second TIP on this page.) The **open function** opens the file whose name is specified in the *fileName* argument and associates the file with the *fileObject*. When a subsequent statement in the program needs to refer to the file, it does so using the name of the *fileObject* rather than the *fileName* itself.

In most cases, the program file refers to the .exe file. However, when running a program from the Microsoft Visual C++ IDE, the program file refers to the .cpp file.

The optional *mode* argument in the syntax indicates how the file is to be opened. As Figure 14-3 indicates, you use the `ios::in` mode to open a file for input, which allows the computer to read the data stored in the file. The `ios::out` and `ios::app` modes are used to open output files. Both of these modes allow the computer to write data to the file. You use the `ios::app` (`app` stands for *append*) mode when you want to add data to the end of an existing file. If the file does not exist, the computer creates the file for you. You use the `ios::out` mode to open a new, empty file for output. If the file already exists, the computer erases the contents of the file before writing any data to it. The two colons (`::`) in each mode are called the **scope resolution operator** and indicate that the keywords `in`, `out`, and `app` are defined in the `ios` class.

HOW TO Open a Sequential Access File

Syntax

fileObject.**open(***fileName*[**,** *mode*]**);**

mode	Description
`ios::in`	Used with an `ifstream` object. Opens the file for input, which allows the computer to read the file's contents. This is the default mode for input files.
`ios::out`	Used with an `ofstream` object. Opens the file for output, which creates a new, empty file to which data can be written. If the file already exists, the computer erases the file's contents before the new data is written to it. This is the default mode for output files.
`ios::app`	Used with an `ofstream` object. Opens the file for append, which allows the computer to write new data to the end of the existing data in the file. If the file does not exist, the computer creates the file before writing any data to it.

Figure 14-3 How to open a sequential access file *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Example 1
```
inFile.open("payroll.txt", ios::in);
          or
inFile.open("payroll.txt");
```
opens the payroll.txt file for input

Example 2
```
outFile.open("employ.txt", ios::out);
          or
outFile.open("employ.txt");
```
opens the employ.txt file for output

Example 3
```
outSales.open("F:/FirstQtr/sales.txt", ios::app);
```
opens the sales.txt file for append

Figure 14-3 How to open a sequential access file

Also included in Figure 14-3 are examples of statements that open sequential access files. Although it is not a requirement, many programmers use the three letters *txt* (short for *text*) as the filename extension when naming sequential access files. You can use either of the statements in Example 1 to open the payroll.txt file for input. Because the *fileName* argument in both statements does not contain a path, the computer will look for the payroll.txt file in the same location as the program file. Notice that the *mode* argument is omitted in the second statement in Example 1. Because all files associated with an `ifstream` file object are opened automatically for input, it is not necessary to specify `ios::in` when opening an input file.

Unlike files associated with an `ifstream` object, files associated with an `ofstream` object are opened automatically for output. In other words, `ios::out` is the default mode when opening output files. This explains why you can use either of the statements in Example 2 to open the employ.txt file for output. Here too, because the *fileName* argument in both statements does not contain a path, the computer will look for the employ.txt file in the same location as the program file.

If a program needs to add data to the end of an output file's existing data, you will need to specify the `ios::app` mode in the `open` function, as shown in Example 3. In this case, the `outSales.open("F:/FirstQtr/sales.txt", ios::app);` statement tells the computer to locate the sales.txt file in the FirstQtr folder on the F drive and then open it for append.

The computer uses a file pointer to keep track of the next character either to read from or to write to a file. When you open a file for input, the computer positions the file pointer at the beginning of the file, immediately before the first character. When you open a file for output, the computer also positions the file pointer at the beginning of the file, but recall that the file is empty. (As you learned earlier, when you open a file for output, the computer either creates a new, empty file or erases the contents of an existing file.) However, when you open a file for append, the computer positions the file pointer immediately after the last character in the file. Figure 14-4 illustrates the position of the file pointer when files are opened for input, output, and append.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

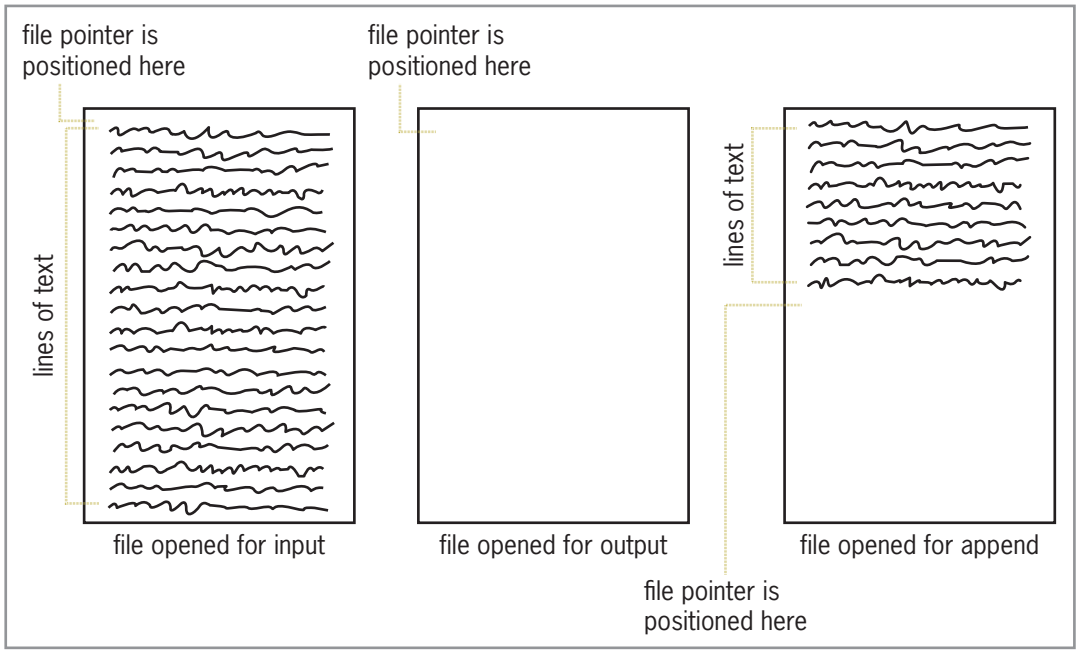


Figure 14-4 Position of the file pointer when files are opened for input, output, and append

Determining Whether a File Was Opened Successfully

The `is_open` function is a class member function in the `ifstream` and `ofstream` classes.

Keep in mind that it is possible for the **open** function to fail when attempting to open a file. For example, the function will not be able to create an output file when either the path specified in the *fileName* argument does not exist or the disk is full. It also will not be able to open an input file that does not exist or one that you don't have permission to open. Therefore, immediately after using the **open** function in a program, you should use the `is_open` function to determine whether the file was opened successfully. The **is_open function** returns the Boolean value true if the **open** function was able to open the file; otherwise, it returns the Boolean value false.

Figure 14-5 shows the `is_open` function's syntax and includes examples of using the function. In the syntax, *fileObject* is the name of an existing file object in the program. Most times, you will use the `is_open` function in an `if` statement's condition, as shown in the examples. (For clarity, an appropriate **open** function is included in each example.)

HOW TO Determine the Success of the **open** Function

Syntax
fileObject.**is_open()**

Example 1
```
inFile.open("payroll.txt");

if (inFile.is_open() == true)
     or
if (inFile.is_open())
```
determines whether the `open` function succeeded in opening the file associated with the `inFile` object

Figure 14-5 How to determine the success of the **open** function *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Example 2

```
outFile.open("employ.txt");

if (outFile.is_open() == false)
      or
if (!outFile.is_open())
```

determines whether the `open` function failed to open the file associated with the `outFile` object

Figure 14-5 How to determine the success of the `open` function

You can use either of the conditions shown in Example 1 to determine whether the `open` function was able to open the file associated with the `inFile` object. The first condition in Example 1, `inFile.is_open() == true`, compares the `is_open` function's return value with the Boolean value true. If the condition evaluates to true, it means that the `open` function was successful in opening the file. If the condition evaluates to false, it means that the `open` function was not able to open the file. As the second condition in Example 1 shows, you can omit the `== true` text from the condition and use `inFile.is_open()` instead.

Unlike the conditions in Example 1, the conditions in Example 2 determine whether the `open` function failed to open the file associated with the `outFile` object. The `outFile.is_open() == false` condition compares the `is_open` function's return value with the Boolean value false. In this case, you can omit the `== false` text by preceding the condition with an exclamation point (!), as shown in the second condition in Example 2. The **!** is the **Not logical operator** in C++, and its purpose is to reverse the truth-value of the condition. If the value of `outFile.is_open()` is true, then the value of `!outFile.is_open()` is false. Likewise, if the value of `outFile.is_open()` is false, then the value of `!outFile.is_open()` is true.

Mini-Quiz 14-1

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

1. Which directive is necessary to create an input file in a program?
   a. `#include <filestream>`
   b. `#include <fstream>`
   c. `#include <instream>`
   d. `#include <iofilestream>`
2. Which mode tells the computer to open a file for input?
   a. `add::ios`
   b. `in::file`
   c. `ios::app`
   d. `ios::in`
3. What value does the `is_open` function return when the `open` function fails?
4. Write the C++ statement to create an output file object named `outAlbums`.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

5. Which of the following statements uses the `outAlbums` file object from Question 4 to open an output file named mine.txt? New information should be written following the existing information in the file.

   a. `outAlbums.open("mine.txt", ios::in);`
   b. `outAlbums.open("mine.txt", ios::out);`
   c. `outAlbums.open("mine.txt", ios::app);`
   d. `outAlbums.open("mine.txt", ios::add);`

Writing Data to a Sequential Access File

Figure 14-6 shows the syntax for writing data to a sequential access file in C++. In the syntax, *fileObject* is the name of an existing `ofstream` object in the program, and *data* is the information you want written to the file. The figure also includes examples of using the syntax.

HOW TO Write Data to a Sequential Access File

Syntax
fileObject **<<** *data*;

Example 1
```
outFile << "Rainbow Boutique" << endl;
```
writes the string "Rainbow Boutique" to the file associated with the `outFile` object and then advances the file pointer to the next line in the file

Example 2
```
outFile << "Gross pay: $";
outFile << gross << endl;
```
writes the string "Gross pay: $" and the contents of the `gross` variable to the file associated with the `outFile` object and then advances the file pointer to the next line in the file

Example 3
```
outSales << custName << endl;
```
writes the contents of the `custName` variable to the file associated with the `outSales` object and then advances the file pointer to the next line in the file

Example 4
```
outEmploy << name << '#' << bonus << endl;
```
writes the contents of the `name` variable, the number sign (#), and the contents of the `bonus` variable to the file associated with the `outEmploy` object and then advances the file pointer to the next line in the file

Figure 14-6 How to write data to a sequential access file

The statement in Example 1 writes the string "Rainbow Boutique" followed by a newline character to the file associated with the `outFile` object. The newline character, which represents the Enter key, advances the file pointer to the next line in the file. The first statement in Example 2 writes the string "Gross pay: $" to the file associated with the `outFile` object,

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

but it leaves the file pointer after the last character written, which is the dollar sign. The second statement in Example 2 writes the contents of the `gross` variable followed by a newline character to the file. If the `gross` variable contains the number 450, the statements in Example 2 write "Gross pay: $450" (without the quotes) to the file before advancing the file pointer.

In many programs, a sequential access file is used to store fields and records. A **field** is a single item of information about a person, place, or thing—such as a name, a salary, a Social Security number, or a price. A **record** is a collection of one or more related fields that contain all of the necessary data about a specific person, place, or thing. The college you are attending keeps a student record on you. Examples of fields contained in your student record include your Social Security number, name, address, phone number, credits earned, and grades earned.

Fields and records are like columns and rows, respectively, in a table.

To distinguish one record from another in a sequential access file, programmers typically write each record on a separate line in the file. You do this by including the `endl` stream manipulator at the end of the statement that writes the record. The `outSales << custName << endl;` statement in Example 3 in Figure 14-6, for instance, writes a record that contains one field (the name stored in the `custName` variable) to the file associated with the `outSales` object. The `endl` stream manipulator writes a newline character at the end of the record, which advances the file pointer to the next line in the file.

When writing a record that contains more than one field, programmers typically separate each field with a character literal constant, such as `'#'` (the number sign or hash mark enclosed in single quotation marks). The `'#'` character appears in the `outEmploy << name << '#' << bonus << endl;` statement in Example 4 in Figure 14-6. The statement writes a record that contains two fields: the name stored in the `name` variable and the bonus amount stored in the `bonus` variable. The statement writes the record on a separate line in the file, with the number sign separating the data in the `name` field from the data in the `bonus` field.

You can also use a string literal constant, such as "#", to separate each field.

You can verify that the information was written correctly to a sequential access file by opening the file in a text editor, such as the text editor in your C++ development tool or in Notepad. Figure 14-7 shows a sequential access file named yearsAndSalaries.txt opened in a text editor. The file contains four records: one for each of the company's four employees. Each record has two fields separated by a number sign (#). The first field in each record represents the number of years the employee has worked for the company; the second field represents the employee's salary. A newline character separates one record from the next. Because the newline character is invisible, you will not see it when you open a sequential access file.

You can use Notepad to create a text file. However, when saving the file, be sure to enclose the filename in quotation marks, like this: "yearsAndSalaries.txt".

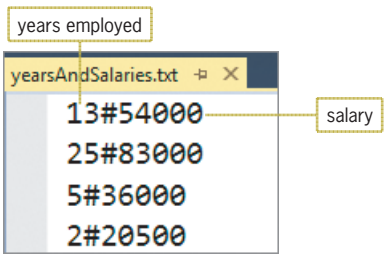


Figure 14-7 The yearsAndSalaries.txt sequential access file opened in a text editor

The OpenTextFile.pdf file contains the instructions for opening a text file in Notepad and in several C++ development tools.

Reading Information from a Sequential Access File

Figure 14-8 shows the syntax for reading numeric and `char` data from a sequential access file in C++. It also includes the syntax for reading `string` data. In each syntax, *fileObject* is the name of an existing `ifstream` object in the program. The *variableName* and *stringVariableName* arguments represent the name of the variable that will store the information read from the file. The figure also shows examples of using each syntax.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

HOW TO Read Data from a Sequential Access File

Syntax

To read numeric and `char` data: *fileObject* **>>** *variableName*;
To read `string` data: **getline(***fileObject*, *stringVariableName*[, *delimiterCharacter*]**);**

Example 1

```
int years = 0;
double salary = 0.0;
inFile >> years;
inFile.ignore(1);
inFile >> salary;
inFile.ignore(1);
```

reads a number from the file associated with the `inFile` object and stores the number in the `years` variable, then ignores (consumes) one character, then reads the next number and stores it in the `salary` variable, and then ignores (consumes) one character; when used to read a record from the yearsAndSalaries.txt file shown earlier in Figure 14-7, the first `ignore` function will consume the # character, and the second `ignore` function will consume the newline character

Example 2

```
char letter = ' ';
inAlphabet >> letter;
```

reads a character from the file associated with the `inAlphabet` object and stores the character in the `letter` variable

Example 3

```
string name = "";
getline(inEmploy, name);
```

reads a string from the file associated with the `inEmploy` object and stores the string in the `name` variable, and then ignores (consumes) the newline character

Example 4

```
string name = "";
double bonus = 0.0;
getline(inEmploy, name, '#');
inEmploy >> bonus;
inEmploy.ignore(1);
```

reads a string from the file associated with the `inEmploy` object and stores the string in the `name` variable, then ignores (consumes) the # character, then reads a number from the file and stores the number in the `bonus` variable, and then ignores (consumes) one character

Figure 14-8 How to read data from a sequential access file

As Figure 14-8 indicates, you use the extraction operator (>>) to read `char` and numeric data from a file. The code in Example 1 shows how you can read the first record from the yearsAndSalaries.txt file (shown earlier in Figure 14-7), which is associated with the `inFile` object. The `inFile >> years;` statement in the example reads the first number from the file (13) and stores the number in the `years` variable. The first `inFile.ignore(1);` statement then consumes (ignores) the number sign (#) that separates the years field from the salary field. Next, the `inFile >> salary;` statement reads the employee's salary (54000) from the file and stores

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

it in the `salary` variable. Finally, the second `inFile.ignore(1);` statement consumes the newline character that separates the first record from the second record.

The code in Example 2 in Figure 14-8 uses the extraction operator to read a character from the file associated with the `inAlphabet` object. The code stores the character in the `letter` variable.

To read `string` data from a sequential access file, you use the `getline` function, which you learned about in Chapter 13. The `getline` function will continue to read characters from the file associated with the *fileObject* until it encounters the *delimiterCharacter*, which it consumes. Recall that consuming a character means to read and discard it. If you omit the *delimiterCharacter* argument in the `getline` function, the default delimiter character is the newline character. The `getline(inEmploy, name);` statement in Example 3 in Figure 14-8 uses the `getline` function to read a string from the file associated with the `inEmploy` object. Because the `getline` function does not specify a *delimiterCharacter*, the function stops reading when it encounters the newline character. The function stores the string in the `name` variable and then consumes the newline character.

The `getline(inEmploy, name, '#');` statement in Example 4 in Figure 14-8 also reads a string from the file associated with the `inEmploy` object. In this case, however, the `getline` function's *delimiterCharacter* argument indicates that the string ends with the character immediately preceding the # character. After storing the string in the `name` variable, the `getline` function consumes the # character. The next statement in the example, `inEmploy >> bonus;`, reads a number from the file and stores the number in the `bonus` variable. The `inEmploy.ignore(1);` statement then consumes the next character in the file.

Mini-Quiz 14-2

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

1. Which of the following statements writes the contents of the `quantity` variable to the inventory.txt file, which is associated with a file object named `outInv`?

   a. `inventory.txt << quantity << endl;`
   b. `ofstream << quantity << endl;`
   c. `outInv << quantity << endl;`
   d. `outInv >> quantity >> endl;`

2. Which of the following statements writes a record to the test.txt file? The file is associated with a file object named `outFile`. The record contains two scores, which are stored in the `score1` and `score2` variables.

   a. `test.txt << score1 << score2 << endl;`
   b. `ofstream << score1 << '#' << score2 << endl;`
   c. `outFile << score1 << score2 << endl;`
   d. `outFile << score1 << '#' << score2 << endl;`

3. Which of the following statements reads a record written by the statement from Question 1 and stores the record in the `number` variable? The inventory.txt file is associated with a file object named `inInv`.

   a. `ifstream >> number;`
   b. `inventory.txt >> number;`
   c. `inInv << number;`
   d. `inInv >> number;`

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Testing for the End of a Sequential Access File

As you learned earlier, the computer uses a file pointer to keep track of the next character to either read from a file or write to a file. When a sequential access file is opened for input, the computer positions the file pointer before the first character in the file. Each time a character is read from the file, the file pointer is moved to the next character. When an entire line from the file is read, the computer moves the file pointer to the beginning of the next line in the file.

Most times, a program will need to read each line contained in a sequential access file, one line at a time, beginning with the first line and ending with the last line. You can accomplish this task by using a loop along with the `eof` (end of file) function. The **eof function** determines whether the last character in a file has been read. In other words, it determines whether the file pointer is located after the last character in the file. If the file pointer is located at the end of the file, the `eof` function returns the Boolean value true; otherwise, it returns the Boolean value false.

The `eof` function is a class member function in the `ifstream` class.

Figure 14-9 shows the `eof` function's syntax and includes examples of using the function. In the syntax, *fileObject* is the name of an existing `ifstream` object in the program. The condition in the `while` clause in Example 1 tells the computer to repeat the loop instructions as long as the end of the file has not been reached. You can also write the condition using the Not logical operator (`!`), as shown in Example 2. As the examples indicate, you should enter the priming read above the `while` clause that contains the `eof` function. (You learned about the priming read in Chapter 7.)

HOW TO Test for the End of a Sequential Access File

Syntax
fileObject.**eof()**

Example 1
priming read instruction
```
while (inFile.eof() == false)
```
tells the computer to repeat the loop instructions as long as (or while) the end of the file associated with the `inFile` object has not been reached

Example 2
priming read instruction
```
while (!inFile.eof())
```
same as Example 1

Figure 14-9 How to test for the end of a sequential access file

Closing a Sequential Access File

The `close` function is a class member function in the `ifstream` and `ofstream` classes.

To prevent the loss of data, you should use the **close function** to close a sequential access file as soon as the program is finished using it. The function's syntax is shown in Figure 14-10 along with examples of using the function. In the syntax, *fileObject* is the name of either an existing `ifstream` object or an existing `ofstream` object in the program. Notice that the `close` function does not require the name of the file you want to close. This is because the computer automatically closes the file whose name is associated with the *fileObject*. (Recall that the `open` function associates the file's name with the *fileObject* when the file is opened.) Because it is so easy to forget to close the files used in a program, you should enter the statement to close the file as soon as possible after entering the one that opens it.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

HOW TO Close a Sequential Access File

Syntax
fileObject.**close()**

Example 1
`inFile.close()`
closes the file associated with the `inFile` object

Example 2
`outFile.close()`
closes the file associated with the `outFile` object

After a file has been read, the only way to access the first record again is to close the file and then reopen it.

Figure 14-10 How to close a sequential access file

The eBook Collection Program

The eBook Collection program will use a sequential access file to save two items of information for each eBook: the title and the author's name. Figure 14-11 shows the IPO chart information and C++ instructions for the program. In addition to the `main` function, the program uses two void functions named `saveInfo` and `displayInfo`. The `saveInfo` function saves the user's input in the eBooks.txt file, and the `displayInfo` function displays the file's contents on the computer screen.

The flowcharts for the eBook Collection program are contained in the eBook.pdf file.

`main` function
IPO chart information

Input
none

Processing
none

Output
none

Algorithm
call the saveInfo function to get and save the eBook information

call the displayInfo function to display the eBook information

`main` function
C++ instructions

```
saveInfo();

displayInfo();
```

`saveInfo` function
IPO chart information

Input
title
author

`saveInfo` function
C++ instructions

```
string title = "";
string author = "";
```

Figure 14-11 IPO chart information and C++ instructions for the eBook Collection program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Processing

none

Output

file (sequential access file containing title and author)

```
ofstream outFile;
```

Algorithm

1. open the file for append
2. if (the file was opened successfully)

```
outFile.open("eBooks.txt", ios::app);
if (outFile.is_open())
{
```

   enter the title

```
    cout << "Title (-1 to stop): ";
    getline(cin, title);
```

   repeat while (the title is not "-1")

```
    while (title != "-1")
    {
```

      enter the author

```
        cout << "Author: ";
        getline(cin, author);
```

      write the title and author to the file

```
        outFile << title << '#' << author
        << endl;
```

      get another title

```
        cout << "Title (-1 to stop): ";
        getline(cin, title);
```

   end repeat

```
    } //end while
```

   close the file

```
    outFile.close();
}
```

else

   display a message indicating that the file could not be opened

```
else
    cout << "eBooks.txt file could not
    be opened" << endl;
```

end if

```
//end if
```

displayInfo function
IPO chart information

Input

file (sequential access file containing title and author)

displayInfo function
C++ instructions

```
ifstream inFile;
```

Processing

none

Output

title

```
string title = "";
```

author

```
string author = "";
```

Figure 14-11 IPO chart information and C++ instructions for the eBook Collection program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Algorithm

```
1. open the file for input
2. if (the file was opened successfully)
     display heading
     read the title and author from the file
     repeat while (it's not the end of the file)
       display the title and author
       read the title and author from the file
     end repeat
     close the file
   else
     display a message indicating that
     the file could not be opened
end if
```

```
inFile.open("eBooks.txt", ios::in);
if (inFile.is_open())
{
    cout << endl << endl <<
    "eBook Collection" << endl;
    cout << "----------------" << endl;
    getline(inFile, title, '#');
    getline(inFile, author);

    while (!inFile.eof())
    {
        cout << title << " by "
        << author << endl;
        getline(inFile, title, '#');
        getline(inFile, author);
    } //end while
    inFile.close();
}
else
    cout << "eBooks.txt file could
    not be opened" << endl;
//end if
```

Figure 14-11 IPO chart information and C++ instructions for the eBook Collection program

The eBook Collection program is shown in Figure 14-12. The instructions pertaining to sequential access files are shaded in the figure. Figure 14-13 shows both a sample run of the program and the eBooks.txt file opened in a text editor.

```
 1 //eBook Collection.cpp - gets and displays the
 2 //items in an eBook collection
 3 //Created/revised by <your name> on <current date>
 4
 5 #include <iostream>
 6 #include <string>
 7 #include <fstream>
 8 using namespace std;
 9
10 //function prototypes
11 void saveInfo();
12 void displayInfo();
13
14 int main()
15 {
16    saveInfo();
17    displayInfo();
18    return 0;
19 } //end of main function
20
```

Figure 14-12 eBook Collection program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
21 //*****function definitions*****
22 void saveInfo()
23 {
24    //writes records to a sequential access file
25    string title = "";
26    string author = "";
27
28    //create file object and open the file
29    ofstream outFile;
30    outFile.open("eBooks.txt", ios::app);
31
32    //determine whether the file was opened
33    if (outFile.is_open())
34    {
35        cout << "Title (-1 to stop): ";
36        getline(cin, title);
37        while (title != "-1")
38        {
39            cout << "Author: ";
40            getline(cin, author);
41            //write the record
42            outFile << title << '#' << author << endl;
43
44            cout << "Title (-1 to stop): ";
45            getline(cin, title);
46        } //end while
47        outFile.close();
48    }
49    else
50        cout << "eBooks.txt file could not be opened"
51            << endl;
52    //end if
53 }  //end of saveInfo function
54
55 void displayInfo()
56 {
57    //displays the records stored in the file
58    string title = "";
59    string author = "";
60
61    //create file object and open the file
62    ifstream inFile;
63    inFile.open("eBooks.txt", ios::in);
64
65    //determine whether the file was opened
66    if (inFile.is_open())
67    {
```

Figure 14-12 eBook Collection program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
68          cout << endl << endl << "eBook Collection" << endl;
69          cout << "----------------" << endl;
70          //read a record
71          getline(inFile, title, '#');
72          getline(inFile, author);
73
74          while (!inFile.eof())
75          {
76              //display the record
77              cout << title << " by " << author << endl;
78              //read another record
79              getline(inFile, title, '#');
80              getline(inFile, author);
81          } //end while
82          inFile.close();
83      }
84      else
85          cout << "eBooks.txt file could not be opened"
86              << endl;
87      //end if
88  }  //end of displayInfo function
```

Figure 14-12 eBook Collection program

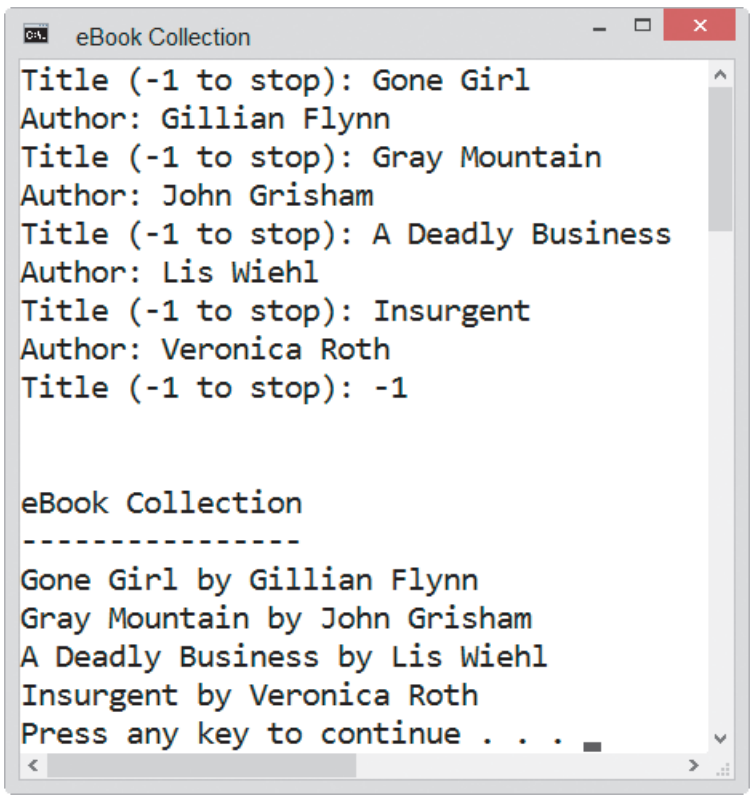


Figure 14-13 Sample run of the program and the eBooks.txt file *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

eBooks.txt

```
Gone Girl#Gillian Flynn
Gray Mountain#John Grisham
A Deadly Business#Lis Wiehl
Insurgent#Veronica Roth
```

Figure 14-13 Sample run of the program and the eBooks.txt file

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 14-3

1. Which of the following tells the computer to continue reading the inventory.txt file until the end of the file is reached? The file object is named `inInv`.
   a. `while (inventory.txt.end())`
   b. `while (inInv.end())`
   c. `while (!inInv.eof())`
   d. `while (!inventory.txt.eof())`
2. What value does the `eof` function return when the file pointer is not at the end of the file?
3. Write the statement to close the inventory.txt file, which is associated with a file object named `outInv`.

The answers to the labs are contained in the Answers.pdf file.

LAB 14-1 Stop and Analyze

Study the program shown in Figure 14-14 and then answer the questions.

```
 1 //Lab14-1.cpp - saves movie titles and release
 2 //years in a sequential access file
 3 //Created/revised by <your name> on <current date>
 4
 5 #include <iostream>
 6 #include <string>
 7 #include <fstream>
 8
 9 using namespace std;
10
11 int main()
12 {
13    string title = "";
14    string year = "";
15    ofstream outFile;
16
17    outFile.open("movies.txt", ios::out);
18
```

Figure 14-14 Code for Lab 14-1 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
19    if (outFile.is_open())
20    {
21        cout << "Movie title (-1 to stop): ";
22        getline(cin, title);
23        while (title != "-1")
24        {
25            cout << "Year released: ";
26            getline(cin, year);
27            outFile << title << '#' << year << endl;
28
29            cout << "Movie title (-1 to stop): ";
30            getline(cin, title);
31        }  //end while
32        outFile.close();
33    }
34    else
35        cout << "The movies.txt file could not be opened."
36             << endl;
37
38    return 0;
39 }  //end of main function
```

Figure 14-14 Code for Lab 14-1

QUESTIONS

1. Why are the instructions in Lines 5, 6, and 7 necessary?
2. The program writes records to a sequential access file. How many fields are in each record, and what are they?
3. Suppose you run the program twice, entering three records the first time and two records the second time. If you open the movies.txt file in a text editor, how many records will the file contain and why?
4. How can you modify the program so that the existing records in the movies.txt file are not erased when the program is run?
5. What is another way of writing the `if` clause in Line 19?
6. What is the purpose of the # character in Line 27?
7. Why is the statement in Line 32 necessary?
8. Follow the instructions for starting C++ and viewing the Lab14-1.cpp file, which is contained in either the Cpp8\Chap14\Lab14-1 Project folder or the Cpp8\Chap14 folder. (Depending on your C++ development tool, you may need to open Lab14-1's project/solution file first.) Run the program. When you are prompted to enter a movie title, type The Avengers and press Enter. When you are prompted to enter the release year, type 2012 and press Enter. Next, enter Maleficent as the movie title and 2014 as the release year. Finally, enter –1 as the movie title.
9. Use a text editor to open the movies.txt file. The file contains two records. Close the movies.txt file.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

10. Run the program again. Enter Frozen and 2013 as the movie title and release year, respectively, and then enter –1 as the movie title. Use a text editor to open the movies.txt file. The file contains one record. Close the movies.txt file.
11. Modify the program so that the existing records in the movies.txt file are not erased when the program is run.
12. Save and then run the program. Enter the following four movie titles and release years, followed by the sentinel value: The Avengers, 2012, Maleficent, 2014, Casablanca, 1942, Chicago, 2002, –1.
13. Use a text editor to open the movies.txt file. The file contains five records. Close the movies.txt file.

LAB 14-2 Plan and Create

In this lab, you will plan and create an algorithm for Cheryl Liu, the owner of a candy shop named Sweets-4-You. The problem specification is shown in Figure 14-15.

Problem specification
Cheryl Liu is the owner of a candy shop named Sweets-4-You. She wants a program that displays the following menu:

Menu Options
1 Add Records
2 Display Total Sales
3 Exit

If Cheryl selects option 1, the program should call a function that prompts her to enter each salesperson's name and sales amount. The function should save Cheryl's entries in a sequential access file named sales.txt. If Cheryl selects option 2, the program should call a function that calculates and displays the total of the sales amounts stored in the sales.txt file. The program should end only when Cheryl selects option 3.

Figure 14-15 Problem specification for Lab 14-2

The Sweets-4-You program will use four functions: `main`, `getChoice`, `addRecords`, and `displayTotal`. Figure 14-16 shows the IPO chart information (including flowcharts) and C++ instructions for the `main` and `getChoice` functions.

`main` function

IPO chart information	**C++ instructions**
Input	
menu choice	`int choice = 0;`
Processing	
none	

Figure 14-16 IPO chart information and C++ instructions for the `main` and `getChoice` functions *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Output

none

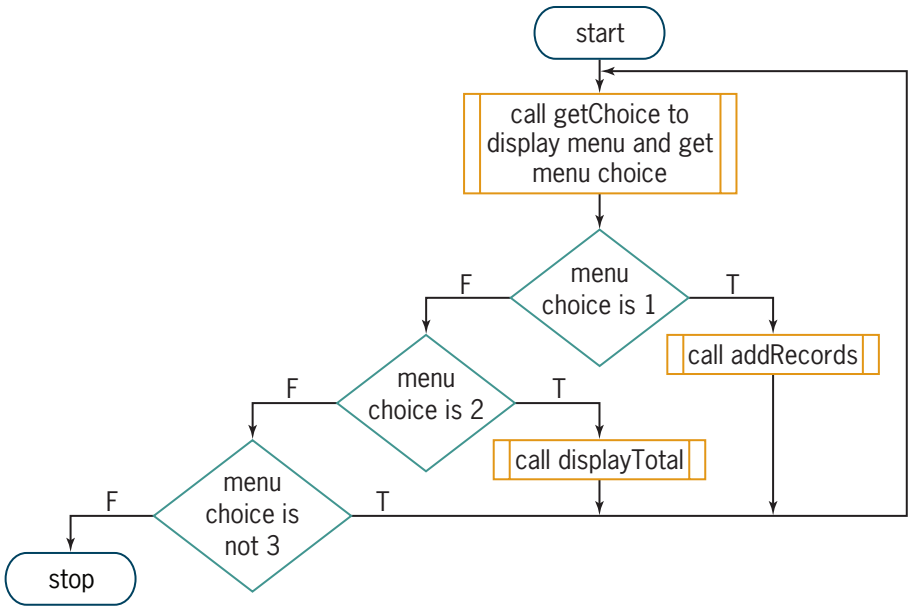
Algorithm

```
repeat
    call the getChoice function to
    display the menu and get the
    menu choice

    if (menu choice is 1)
       call the addRecords function
    else if (menu choice is 2)
       call the displayTotal function
    end if
end repeat while (menu choice is not 3)
```

```
do
{
   choice = getChoice();

   if (choice == 1)
       addRecords();
   else if (choice == 2)
       displayTotal();
   //end if
} while (choice != 3);
```



getChoice function
IPO chart information

Input

none

Processing

none

getChoice function
C++ instructions

Figure 14-16 IPO chart information and C++ instructions for the `main` and `getChoice` functions *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

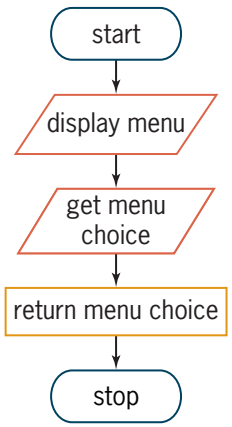
(continued)

Output

menu choice

Algorithm

1. display menu
2. get menu choice
3. return menu choice




```
int menuChoice = 0;

cout << endl << "Menu Options" << endl;
cout << "1  Add Records" << endl;
cout << "2  Display Total Sales" << endl;
cout << "3  Exit" << endl;
cout << "Choice (1, 2, or 3)? ";
cin >> menuChoice;
cin.ignore(100, '\n');
cout << endl;
return menuChoice;
```

Figure 14-16 IPO chart information and C++ instructions for the `main` and `getChoice` functions

The `main` function declares and initializes an `int` variable named `choice`. It then calls the `getChoice` function to display the menu, which contains three options. After displaying the menu, the `getChoice` function prompts the user to enter her choice of menu options: 1 to add records, 2 to display the total sales, or 3 to exit the program. The `getChoice` function returns the user's response to the `main` function, which assigns the value to the `choice` variable. The selection structure in the `main` function uses the value stored in the `choice` variable to determine whether either the `addRecords` function or the `displayTotal` function needs to be called.

The `while` clause in the `main` function compares the value stored in the `choice` variable with the number 3. If the variable does not contain the number 3, the `main` function calls the `getChoice` function to display the menu again and get another choice from the user. The program ends only when the `choice` variable contains the number 3.

Figure 14-17 shows the IPO chart information and C++ instructions for the `addRecords` function; it also includes the function's flowchart. The function creates an output file object named `outFile` and then uses the object along with the `open` function to open the sales.txt file for append. The condition in the `if` clause determines whether the sales.txt file was opened successfully. If the condition evaluates to false, it means that the `open` function failed to open the file. In that case, the `addRecords` function displays an appropriate error message and then the function ends. If the condition evaluates to true, on the other hand, it means that the `open` function was successful in opening the sales.txt file. As a result, the instructions in the `if` statement's true path are processed.

The first two statements in the true path in Figure 14-17 prompt the user to enter the salesperson's name and then store the user's response in the `name` variable. The `while` clause in the true path

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

indicates that the loop body instructions should be repeated as long as the `name` variable does not contain either the letter X or the letter x. The first two statements in the loop body prompt the user to enter the sales amount and then store the user's response in the `sales` variable. The `cin.ignore(100, '\n');` statement instructs the computer to consume the newline character that remains in the `cin` object after the sales amount is entered. The `outFile << name << '#' << sales << endl;` statement then writes a record, followed by a newline character, to the file. The record contains the contents of the `name` variable, the `#` character, and the contents of the `sales` variable. The last two statements in the loop body prompt the user to enter another salesperson's name and then store the user's response in the `name` variable. The loop will end when the `name` variable contains either the string "X" or the string "x". When the loop ends, the `outFile.close();` statement closes the sales.txt file before the `addRecords` function ends.

addRecords function
IPO chart information

Input
salesperson's name
sales amount

Processing
none

Output
sales.txt file (sequential access)

Algorithm
1. open the sales.txt file for append
2. if (the sales.txt file was opened successfully)
   - enter the salesperson's name
   - repeat while (the salesperson's name is not "X" or "x")
     - enter the sales amount
     - write the salesperson's name and sales amount to the sales.txt file
     - enter the salesperson's name
   - end repeat
   - close the sales.txt file
   - else
     - display the "sales.txt file could not be opened" message
   - end if

addRecords function
C++ instructions

```
string name = "";
int sales = 0;

ofstream outFile;

outFile.open("sales.txt", ios::app);
if (outFile.is_open())
{
    cout << "Salesperson's
    name (X to stop): ";
    getline(cin, name);
    while (name != "X" && name != "x")
    {
        cout << "Sales: ";
        cin >> sales;
        cin.ignore(100, '\n');
        outFile << name << '#' <<
        sales << endl;

        cout << "Salesperson's
        name " << "(X to stop): ";
        getline(cin, name);
    } //end while
    outFile.close();
}
else
    cout << "sales.txt file could
    not be opened" << endl;
//end if
```

Figure 14-17 IPO chart information and C++ instructions for the `addRecords` function *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

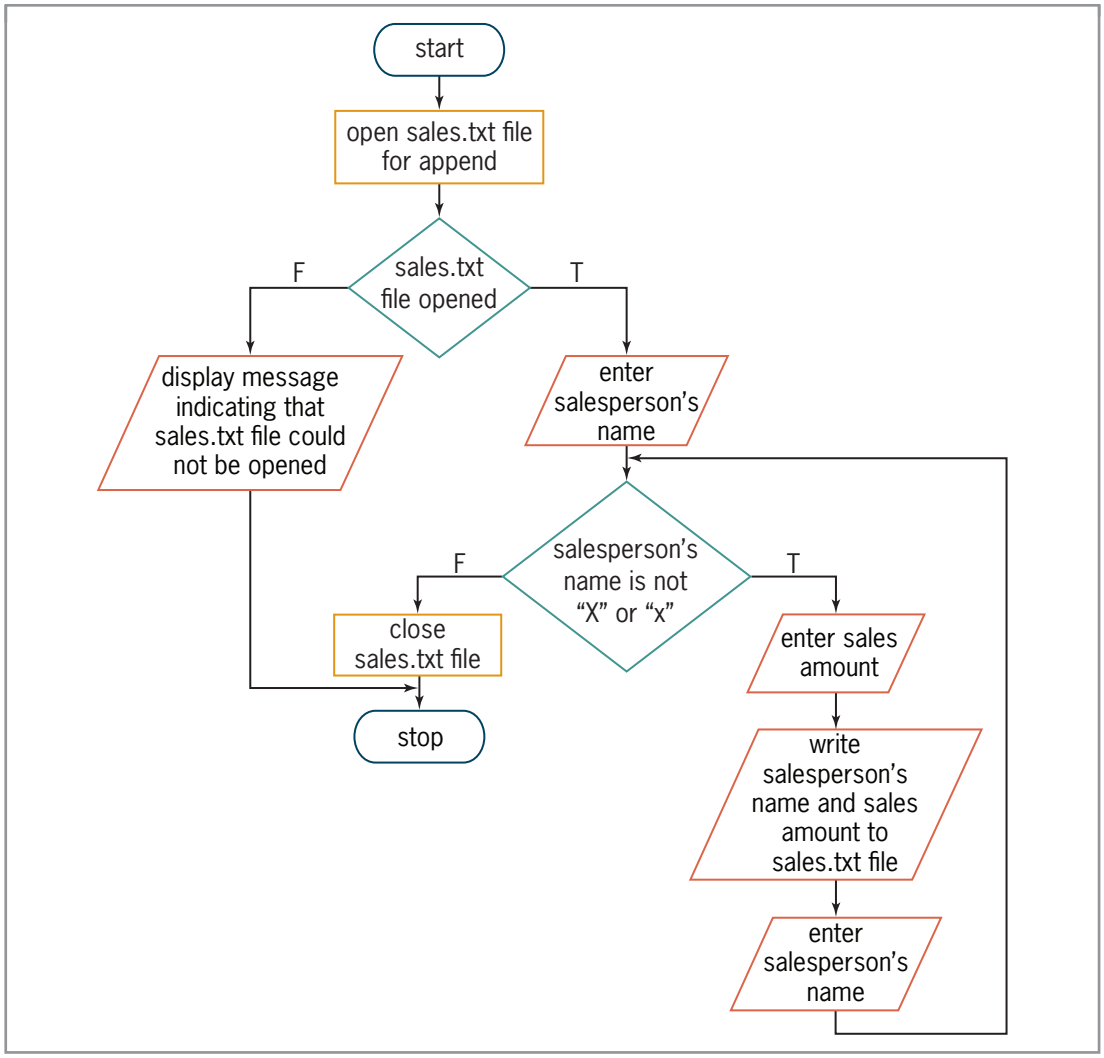


Figure 14-17 IPO chart information and C++ instructions for the `addRecords` function

Finally, Figure 14-18 shows the IPO chart information and C++ instructions for the `displayTotal` function; it also contains the function's flowchart. The function creates an input file object named `inFile` and then uses the object along with the `open` function to open the sales.txt file for input. The condition in the `if` clause determines whether the sales.txt file was opened successfully. If the condition evaluates to false, the `displayTotal` function displays an appropriate error message and then the function ends. If the condition evaluates to true, on the other hand, the instructions in the `if` statement's true path are processed.

The instructions in the true path in Figure 14-18 read a record from the sales.txt file, assigning the name to the `name` variable and assigning the sales to the `sales` variable. The `while` clause in the true path tells the computer to repeat the loop body instructions as long as the file pointer is not at the end of the file. The first statement in the loop body adds the sales amount to the accumulator variable, which is named `total`. The remaining instructions in the loop body read another record from the file. When the loop ends, which occurs when the file pointer is at the

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

end of the sales.txt file, the last two statements in the `if` statement's true path close the file and then display the total sales amount on the screen. After displaying the total sales amount, the `displayTotal` function ends.

displayTotal function
IPO chart information

Input

sales.txt file (sequential access)

Processing

salesperson's name
sales amount

Output

total sales amount (accumulator)

Algorithm

1. open the sales.txt file for input
2. if (the sales.txt file was opened successfully)
   - read the salesperson's name and sales amount from the sales.txt file
   - repeat while (it's not the end of the sales.txt file)
     - add the sales amount to the total sales amount
     - read the salesperson's name and sales amount from the sales.txt file
   - end repeat
   - close the sales.txt file
   - display the total sales amount
   - else
     - display the "sales.txt file could not be opened." message
   - end if

displayTotal function
C++ instructions

```
ifstream inFile;

string name = "";
int sales = 0;

int total = 0;

inFile.open("sales.txt");
if (inFile.is_open())
{
    getline(inFile, name, '#');
    inFile >> sales;
    inFile.ignore();

    while (!inFile.eof())
    {
        total += sales;

        getline(inFile, name, '#');
        inFile >> sales;
        inFile.ignore();

    } //end while
    inFile.close();
    cout << "Total sales $" <<
    total << endl << endl;
}
else
    cout << "sales.txt file could
    not be opened" << endl;
//end if
```

Figure 14-18 IPO chart information and C++ instructions for the `displayTotal` function *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

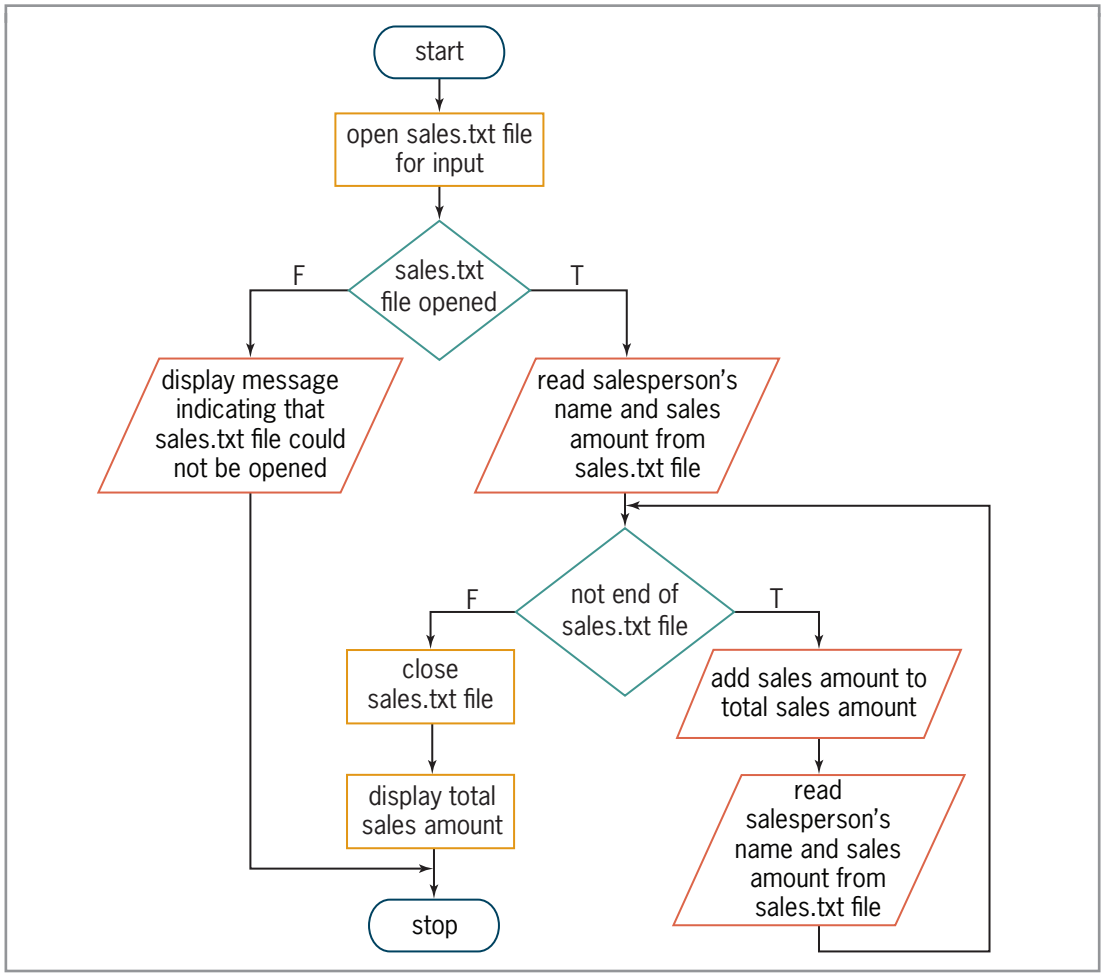


Figure 14-18 IPO chart information and C++ instructions for the `displayTotal` function

Figure 14-19 shows the Sweets-4-You program and includes a sample run of the program.

```
 1 //Lab14-2.cpp - saves records to a sequential access
 2 //file and also calculates and displays the total
 3 //of the sales amounts stored in the file
 4 //Created/revised by <your name> on <current date>
 5
 6 #include <iostream>
 7 #include <string>
 8 #include <fstream>
 9 using namespace std;
10
11 //function prototypes
12 int getChoice();
13 void addRecords();
14 void displayTotal();
15
```

Figure 14-19 Sweets-4-You program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

The instructions on Lines 19 through 28 are a posttest loop.

```
16 int main()
17 {
18   int choice = 0;
19    do
20    {
21       //get user's menu choice
22       choice = getChoice();
23       if (choice == 1)
24           addRecords();
25       else if (choice == 2)
26           displayTotal();
27       //end if
28    } while (choice != 3);
29    return 0;
30 }    //end of main function
31
32 //*****function definitions*****
33 int getChoice()
34 {
35    //displays menu and returns choice
36    int menuChoice = 0;
37    cout << endl << "Menu Options" << endl;
38    cout << "1  Add Records" << endl;
39    cout << "2  Display Total Sales" << endl;
40    cout << "3  Exit" << endl;
41    cout << "Choice (1, 2, or 3)? ";
42    cin >> menuChoice;
43    cin.ignore(100, '\n');
44    cout << endl;
45    return menuChoice;
46 } //end of getChoice function
47
48 void addRecords()
49 {
50    //saves records to a sequential access file
51    string name = "";
52    int sales = 0;
53    ofstream outFile;
54
55    //open file for append
56    outFile.open("sales.txt", ios::app);
57
58    //if the open was successful, get the
59    //salesperson's name and sales amount and
60    //then write the information to the file;
61    //otherwise, display an error message
62    if (outFile.is_open())
63    {
64        cout << "Salesperson's name (X to stop): ";
65        getline(cin, name);
66        while (name != "X" && name != "x")
67        {
68            cout << "Sales: ";
69            cin >> sales;
70            cin.ignore(100, '\n');
```

Figure 14-19 Sweets-4-You program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
 71
 72            outFile << name << '#' << sales << endl;
 73
 74            cout << "Salesperson's name "
 75                 << "(X to stop): ";
 76            getline(cin, name);
 77        } //end while
 78        outFile.close();
 79    }
 80    else
 81        cout << "sales.txt file could not be opened"
 82             << endl;
 83    //end if
 84 }  //end of addRecords function
 85
 86 void displayTotal()
 87 {
 88    //calculates and displays the total sales
 89    string name = "";
 90    int sales = 0;
 91    int total = 0;
 92    ifstream inFile;
 93
 94    //open file for input
 95    inFile.open("sales.txt");
 96
 97    //if the open was successful, read the
 98    //salesperson's name and sales amount, then add
 99    //the sales amount to the accumulator, and then
100    //display the accumulator; otherwise, display
101    //an error message
102    if (inFile.is_open())
103    {
104        getline(inFile, name, '#');
105        inFile >> sales;
106        inFile.ignore();
107
108        while (!inFile.eof())
109        {
110            total += sales;
111            getline(inFile, name, '#');
112            inFile >> sales;
113            inFile.ignore();
114        } //end while
115        inFile.close();
116        cout << "Total sales $" << total
117            << endl << endl;
118    }
119    else
120        cout << "sales.txt file could not be opened"
121             << endl;
122    //end if
123 } //end of displayTotal function
```

Figure 14-19 Sweets-4-You program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

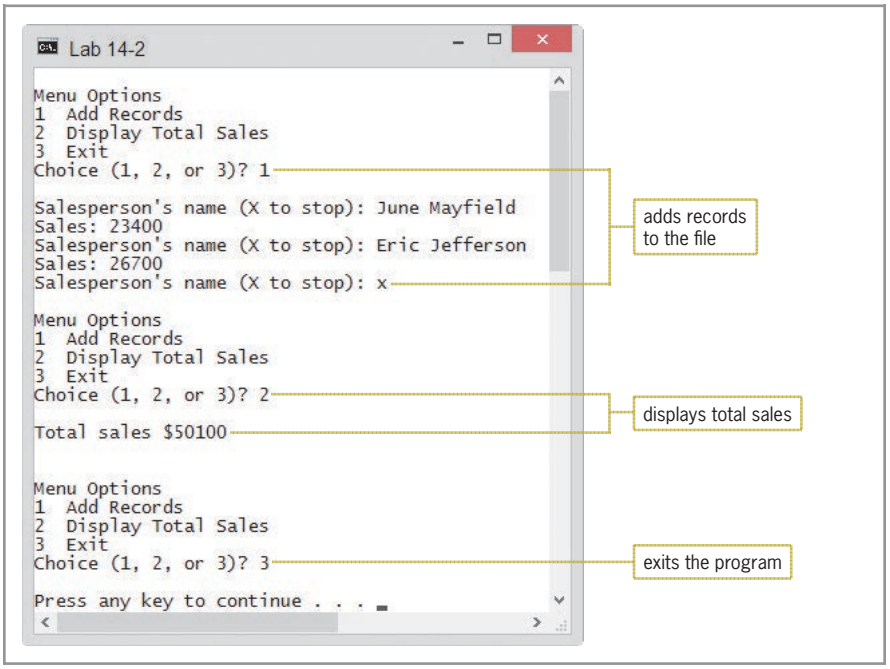


Figure 14-19 Sweets-4-You program

DIRECTIONS

Follow the instructions for starting your C++ development tool. Depending on the development tool you are using, you may need to create a new project; if so, name the project Lab14-2 Project and save it in the Cpp8\Chap14 folder. Enter the instructions shown in Figure 14-19 in a source file named Lab14-2.cpp. (Do not enter the line numbers.) Save the file in either the project folder or the Cpp8\Chap14 folder. Now, follow the appropriate instructions for running the Lab14-2.cpp file. Test the program using the data shown in Figure 14-19. If necessary, correct any bugs (errors) in the program.

LAB 14-3 Modify

If necessary, create a new project named Lab14-3 Project and save it in the Cpp8\Chap14 folder. Enter (or copy) the Lab14-2.cpp instructions into a source file named Lab14-3.cpp. Change Lab14-2.cpp in the first comment to Lab14-3.cpp. Modify the menu so that it contains five options: Add Records, Display Records, Display Total Sales, Display Average Sales, and Exit. When the user selects the Display Records option, the program should call a function to display the contents of the sales.txt file on the screen. When the user selects the Display Average Sales option, the program should call a function to calculate and display the average sales amount stored in the file. Save and then run the program. Test each menu option.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

LAB 14-4 What's Missing?

The program in this lab should read five records from and write five records to a sequential access file. Start your C++ development tool and view the Lab14-4.cpp file, which is contained in either the Cpp8\Chap14\Lab14-4 Project folder or the Cpp8\Chap14 folder. (Depending on your C++ development tool, you may need to open Lab14-4's project/solution file first.) Put the C++ instructions in the proper order, and then determine the one or more missing instructions. Test the program appropriately.

LAB 14-5 Desk-Check

Figure 14-20 shows the code entered in the Lab14-5.cpp file. It also shows the Lab14-5.txt file opened in a text editor. Desk-check the code using the data contained in the text file. What will the code display on the screen?

Lab14-5.txt

```
15400#23600
12345#25350
18450#30900
16750#31000
```

```
//Lab14-5.cpp - displays each region's total sales
//Created/revised by <your name> on <current date>

#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    int store1Sales = 0;
    int store2Sales = 0;
    int store1Total = 0;
    int store2Total = 0;

    ifstream inFile;
    inFile.open("Lab14-5.txt");

    if (inFile.is_open())
    {
        inFile >> store1Sales;
        inFile.ignore();
```

Figure 14-20 Information for Lab 14-5 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
        while (!inFile.eof())
        {
            inFile >> store2Sales;
            inFile.ignore();
            store1Total += store1Sales;
            store2Total += store2Sales;
            inFile >> store1Sales;
            inFile.ignore();
        } //end while
        inFile.close();

        cout << "Store 1's total sales: $"
            << store1Total << endl;
        cout << "Store 2's total sales: $"
            << store2Total << endl;
    }
    else
        cout << "Can't open Lab14-5.txt file." << endl;
    //end if
    return 0;
}    //end of main function
```

Figure 14-20 Information for Lab 14-5

LAB 14-6 Debug

Follow the instructions for starting C++ and viewing the Lab14-6.cpp file, which is contained in either the Cpp8\Chap14\Lab14-6 Project folder or the Cpp8\Chap14 folder. (Depending on your C++ development tool, you may need to open Lab14-6's project/solution file first.) The program should write records consisting of a name and two numbers to a sequential access file named records.txt. Run the program. Type your name and press Enter. Type 5000 and press Enter, and then type 2000 and press Enter. Notice that the program is not working correctly. Debug the program. After debugging the program, be sure to open the records.txt file to verify that it contains the information you entered.

Chapter Summary

- Sequential access files can be either input files or output files. Input files are files whose contents are read by a program. Output files are files to which a program writes data.
- To create a file object in a program, the program must contain the `#include <fstream>` directive.
- You use the `ifstream` and `ofstream` classes, which are defined in the fstream file, to create input and output file objects, respectively. The file objects are used to represent the actual files stored on your computer's disk.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

- After creating a file object, you then use the `open` function (which is a member function in the `ifstream` and `ofstream` classes) to open the file for input, output, or append.
- You can use the `is_open` function (which is a member function in the `ifstream` and `ofstream` classes) to determine whether the `open` function either succeeded or failed to open a sequential access file. The `is_open` function returns the Boolean value true if the `open` function was able to open the file. It returns the Boolean value false if the `open` function could not open the file.
- To distinguish one record from another in a sequential access file, programmers usually write each record on a separate line in the file. You do this by including the `endl` stream manipulator at the end of the statement that writes the record to the file. If the record contains more than one field, programmers use a character (such as `'#'`) to separate the data in one field from the data in another field. You can also use a string (such as `"#"`).
- When reading data from a file, you use the `eof` function (which is a member function in the `ifstream` class) to determine whether the file pointer is at the end of the file. If the file pointer is located after the last character in the file, the `eof` function returns the Boolean value true; otherwise, it returns the Boolean value false.
- When a program is finished with a file, you should close the file by using the `close` function, which is a member function in the `ifstream` and `ofstream` classes. Failing to close an open file can result in the loss of data. A program cannot reopen a file without closing it first.

Key Terms

`!`—the Not logical operator

close function—closes a sequential access file in a program

eof function—determines whether an entire sequential access file has been read; it returns true when the file pointer is located after the last character in the file; otherwise, it returns false

Field—a single item of information about a person, place, or thing

Input files—files that contain information used as input by a program

is_open function—used in a program to determine whether a sequential access file was opened successfully; returns true when the open operation succeeded; otherwise, returns false

Not logical operator—an exclamation point (`!`); reverses the truth-value of a condition

open function—used to open input and output files in a program

Output files—files that store the output produced by a program

Record—a collection of one or more related fields that contain all of the necessary data about a person, place, or thing

Scope resolution operator—two colons (`::`); indicates that the keyword to the right of the operator is a member of the class whose name appears to the left of the operator

Sequential access files—files composed of lines of text; also referred to as text files

Text files—another name for sequential access files

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Review Questions

1. A _____ is a single item of information about a person, place, or thing.
   a. field
   b. file
   c. record
   d. none of the above
2. A group of related fields that contain all of the data about a specific person, place, or thing is called a _____.
   a. field
   b. file
   c. record
   d. none of the above
3. For a program to create a file object, it must include the _____ file.
   a. fileStream
   b. fstream
   c. outFile
   d. sequential
4. You use the _____ class to instantiate an output file object.
   a. `cout`
   b. `fstream`
   c. `ofstream`
   d. `outstream`
5. Which of the following creates an object named `outPayroll` that represents an output file in the program?
   a. `fstream outPayroll;`
   b. `ofstream outPayroll;`
   c. `outPayroll as ofstream;`
   d. `outPayroll as outstream;`
6. Which of the following opens the payroll.txt file for output? The file is associated with the `outPayroll` object.
   a. `outPayroll.open("payroll.txt");`
   b. `outPayroll.open("payroll.txt", ios::out);`
   c. `outPayroll.open("payroll.txt", ios::output);`
   d. both a and b
7. Which *mode* is used in the `open` function to add records to the end of an existing output file?
   a. `add`
   b. `ios::add`
   c. `ios::app`
   d. `ios::out`
8. Which function closes a sequential access file?
   a. `close`
   b. `end`
   c. `exit`
   d. `finish`
9. Which function determines whether the `open` function was successful?
   a. `is_open`
   b. `isopen`
   c. `isFileOpen`
   d. `is_FileOpen`

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

10. Which of the following writes the contents of the `city` variable to an output file named address.txt? The file is associated with the `outFile` object.

    a. `address.txt << city << endl;`
    b. `ofstream << city << endl;`
    c. `outFile << city << endl;`
    d. `outFile >> city >> endl;`

11. Which of the following reads a number from an input file named managers.txt and stores the number in the `salary` variable? The file is associated with the `inFile` object.

    a. `managers.dat << salary;`
    b. `ifstream << salary;`
    c. `inFile << salary;`
    d. none of the above

12. Which of the following writes the contents of the `city` and `state` variables to an output file named address.txt? The file is associated with the `outFile` object.

    a. `address.txt << city << state << endl;`
    b. `ofstream << city << state << endl;`
    c. `outFile >> city >> state >> endl;`
    d. `outFile << city << '#' << state << endl;`

13. Which of the following tells the computer to repeat the loop instructions until the end of the file is reached? The file is associated with the `inFile` object.

    a. `while (inFile.eof())`
    b. `while (!ifstream.eof())`
    c. `while (!inFile.eof())`
    d. `while (!ifstream.fail())`

14. Which of the following creates an object named `inPayroll` that represents an input file in the program?

    a. `instream inPayroll;`
    b. `ifstream inPayroll;`
    c. `inPayroll ifstream;`
    d. `inPayroll as ifstream;`

15. Which of the following opens the payroll.txt file for input? The file is associated with the `inFile` object.

    a. `inFile.open("payroll.txt", ios::app);`
    b. `inFile.open("payroll.txt");`
    c. `inFile.open("payroll.txt", ios::in);`
    d. both b and c

Exercises

Pencil and Paper

TRY THIS

1. Write the statement to declare an input file object named `inSales`. (The answers to TRY THIS Exercises are located at the end of the chapter.)

TRY THIS

2. Write the statement to open a sequential access file named janSales.txt for output. The file is associated with the `outJan` object. (The answers to TRY THIS Exercises are located at the end of the chapter.)

MODIFY THIS

3. Rewrite the statement from Pencil and Paper Exercise 2 so it opens the janSales.txt file for append.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

4. Write the statement to open a sequential access file named inventory.txt for input. The file is associated with the `inInventory` object. INTRODUCTORY

5. Write the statement to open a sequential access file named firstQtr.txt for append. The file is associated with the `outSales` object. INTRODUCTORY

6. Write the statement to open a sequential access file named febSales.txt for output. The file is associated with the `outFeb` object. INTRODUCTORY

7. Write an `if` clause that determines whether an output file was opened successfully. The file is associated with the `outSales` object. INTRODUCTORY

8. Write the statement to read a string from the sequential access file associated with the `inFile` object. Assign the string to the `textLine` variable. INTRODUCTORY

9. Write the statement to read a number from the sequential access file associated with the `inFile` object. Assign the number to the `number` variable. INTRODUCTORY

10. Write the statement to close the janSales.txt file, which is associated with the `outFile` object. INTRODUCTORY

11. A program needs to write the string "Employee" and the string "Name" to the sequential access file associated with the `outFile` object. Each string should appear on a separate line in the file. Write the code to accomplish this task. INTRODUCTORY

12. A program needs to write the contents of a `string` variable named `capital` and the newline character to the sequential access file associated with the `outFile` object. Write the code to accomplish this task. INTRODUCTORY

13. Write a `while` clause that tells the computer to stop processing the loop instructions when the end of the file has been reached. The file is associated with the `inFile` object. INTRODUCTORY

14. A program needs to read a sequential access file, line by line, and display each line on the computer screen. The file, which was opened successfully, is associated with the `inFile` object. Write the code to read and then close the file. INTERMEDIATE

15. Correct the condition in the following `if` clause, which should determine whether the `open` function was able to open the file associated with the `outFile` object: `if (outFile.open())`. SWAT THE BUGS

Computer

16. If necessary, create a new project named TryThis16 Project and save it in the Cpp8\Chap14 folder. Also create a new source file named TryThis16.cpp. Write a program that allows the user to enter the first 10 letters of the alphabet, one at a time. The program should save each letter, in uppercase, on a separate line in a sequential access file named TryThis16.txt. Save and then run the program. Test the program by entering the lowercase letters a through j. Verify that the program worked correctly by opening the TryThis16.txt file in a text editor. Close the TryThis16.txt file. (The answers to TRY THIS Exercises are located at the end of the chapter.) TRY THIS

17. If necessary, create a new project named TryThis17 Project and save it in the Cpp8\Chap14 folder. Also create a new source file named TryThis17.cpp. Write a program that saves records to a sequential access file named TryThis17.txt. Each record should appear on a separate line and contain two fields separated by the number sign (#). The first field should contain numbers from 10 through 25. The second field should contain the square of the number in the first field. For example, the first record TRY THIS

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

will contain 10#100 followed by the newline character. Display the message "Numbers saved in file." if the program was able to save the numbers. Save and then run the program. Verify that the program worked correctly by opening the TryThis17.txt file in a text editor. Close the TryThis17.txt file. (The answers to TRY THIS Exercises are located at the end of the chapter.)

MODIFY THIS

18. In this exercise, you will modify the program from TRY THIS Exercise 17. If necessary, create a new project named ModifyThis18 Project and save it in the Cpp8\Chap14 folder. Copy the instructions from the TryThis17.cpp file into a new source file named ModifyThis18.cpp. Change the filename in the first comment to ModifyThis18.cpp. Also change the name of the sequential access file in both the `open` function and the `cout` statement to ModifyThis18.txt. Modify the program so that each record contains an additional field: the cube of the number in the first field. For example, the first record will contain 10#100#1000 followed by the newline character. Save and then run the program. Verify that the program worked correctly by opening the ModifyThis18.txt file in a text editor. Close the ModifyThis18.txt file.

INTRODUCTORY

19. If necessary, create a new project named Introductory19 Project and save it in the Cpp8\Chap14 folder. Also create a new source file named Introductory19.cpp. Create a program that saves a company's payroll amounts in a sequential access file. Save the amounts in fixed-point notation with two decimal places. Name the sequential access file Introductory19.txt and open the file for append. Use a negative number as the sentinel value. Save and then run the program. Enter the following payroll amounts and sentinel value: 45678.99, 67000.56, and –1. Now, run the program again. This time, enter the following payroll amounts and sentinel value: 25000.89, 35600.55, and –1. Open the Introductory19.txt file in a text editor. The file should contain four payroll amounts, with each amount appearing on a separate line in the file. Close the Introductory19.txt file.

INTRODUCTORY

20. If necessary, create a new project named Introductory20 Project and save it in the Cpp8\Chap14 folder. Also create a new source file named Introductory20.cpp. Create a program that saves prices in a sequential access file. Save the prices in fixed-point notation with two decimal places. Name the sequential access file Introductory20.txt and open the file for append. Use a negative number as the sentinel value. Save and then run the program. Enter the following prices and sentinel value: 10.50, 15.99, and –1. Now, run the program again. This time, enter the following prices and sentinel value: 20, 76.54, 17.34, and –1. Open the Introductory20.txt file in a text editor. The file should contain five prices, with each price appearing on a separate line in the file. Close the Introductory20.txt file.

INTRODUCTORY

21. If necessary, create a new project named Introductory21 Project and save it in the Cpp8\Chap14 folder. Also create a new source file named Introductory21.cpp. If you are using Microsoft Visual C++, copy the Introductory21.txt file from the Cpp8\Chap14 folder to the Cpp8\Chap14\Introductory21 Project folder. Use a text editor to open the Introductory21.txt file, which contains 10 uppercase letters of the alphabet. Close the Introductory21.txt file. Create a program that counts the number of letters stored in the file. The program should display the number of letters on the computer screen. Save and then run the program.

INTERMEDIATE

22. If necessary, create a new project named Intermediate22 Project and save it in the Cpp8\Chap14 folder. Also create a new source file named Intermediate22.cpp. If you are using Microsoft Visual C++, copy the Intermediate22.txt file from the Cpp8\Chap14 folder to the Cpp8\Chap14\Intermediate22 Project folder. Use a text editor to open the Intermediate22.txt file, which contains payroll amounts. Close the Intermediate22.txt file. Create a program that calculates and displays the total of the

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

payroll amounts stored in the file. Display the total with a dollar sign and two decimal places. Save and then run the program.

23. If necessary, create a new project named Intermediate23 Project and save it in the Cpp8\Chap14 folder. Also create a new source file named Intermediate23.cpp. If you are using Microsoft Visual C++, copy the Intermediate23.txt file from the Cpp8\Chap14 folder to the Cpp8\Chap14\Intermediate23 Project folder. Use a text editor to open the Intermediate23.txt file, which contains prices. Close the Intermediate23.txt file. Create a program that calculates and displays the average price stored in the file. Display the average with a dollar sign and two decimal places. Save and then run the program. INTERMEDIATE

24. If necessary, create a new project named Intermediate24 Project and save it in the Cpp8\Chap14 folder. Also create a new source file named Intermediate24.cpp. If you are using Microsoft Visual C++, copy the Intermediate24.txt file from the Cpp8\Chap14 folder to the Cpp8\Chap14\Intermediate24 Project folder. Use a text editor to open the Intermediate24.txt file, which contains payroll codes and salaries. Close the Intermediate24.txt file. Create a program that allows the user to enter a payroll code. The program should search for the payroll code in the file and then display the appropriate salary. If the payroll code is not in the file, the program should display an appropriate message. The program should allow the user to display as many salaries as needed without having to run the program again. Save and then run the program. Test the program by entering the following payroll codes: 10, 24, 55, 32, and 6. Stop the program. INTERMEDIATE

25. If necessary, create a new project named Advanced25 Project and save it in the Cpp8\Chap14 folder. Also create a new source file named Advanced25.cpp. If you are using Microsoft Visual C++, copy the Advanced25.txt file from the Cpp8\Chap14 folder to the Cpp8\Chap14\Advanced25 Project folder. Use a text editor to open the Advanced25.txt file, which contains the names of the items in inventory, as well as each item's quantity and price. Close the Advanced25.txt file. Write a program that displays the contents of the file in three columns titled "Name", "Quantity", and "Price". The program should also display a fourth column that contains the result of multiplying each item's quantity by its price. Use "Value" as the column's title. (Hint: You can align the columns using `'\t'`, which is the escape sequence for the Tab key.) In addition, the program should calculate and display the total value of the items in inventory. Display the price, value, and total value with two decimal places. Save and then run the program. ADVANCED

26. If necessary, create a new project named Advanced26 Project and save it in the Cpp8\Chap14 folder. Also create a new source file named Advanced26.cpp. Write a program that allows the user to record the names of cities and their corresponding ZIP codes in a sequential access file named Advanced26.txt. The program should also allow the user to look up a ZIP code in the file and display the name of its corresponding city. If the ZIP code is not in the file, the program should display an appropriate message. Save and then run the program. Enter any five ZIP codes and their corresponding city names. Then, test the program by entering each valid ZIP code. Also enter one or more invalid ZIP codes. ADVANCED

27. If necessary, create a new project named Advanced27 Project and save it in the Cpp8\Chap14 folder. Also create a new source file named Advanced27.cpp. If you are using Microsoft Visual C++, copy the Advanced27.txt file from the Cpp8\Chap14 folder to the Cpp8\Chap14\Advanced27 Project folder. Each salesperson at BobCat Motors is assigned a code that consists of two characters. The first character is either the letter F (which indicates a full-time employee) or the letter P (which indicates a part-time employee). ADVANCED

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The second character is either a 1 (indicating the salesperson sells new cars) or a 2 (indicating the salesperson sells used cars). Use a text editor to open the Advanced27.txt file, which contains the names of BobCat's salespeople along with each salesperson's code, and then close the file. Write a program that prompts the user to enter the code (F1, F2, P1, or P2). The program should search the Advanced27.txt file for the code and then display only the names of the salespeople assigned that code. Display an appropriate message if the user enters an invalid code. Save and then run the program. Test the program by entering F2 as the code. The program should display three records: Mary Jones, Joel Adkari, and Janice Paulo. Now, test the program using codes of F1, P1, P2, and S3.

ADVANCED

28. If necessary, create a new project named Advanced28 Project and save it in the Cpp8\Chap14 folder. Also create a new source file named Advanced28.cpp. If you are using Microsoft Visual C++, copy the Advanced28.txt file from the Cpp8\Chap14 folder to the Cpp8\Chap14\Advanced28 Project folder. Use a text editor to open the Advanced28.txt file, which contains 20 numbers. Close the Advanced28.txt file. Write a program that performs the following for each number in the Advanced28.txt file: read the number, add 1 to the number, and write the new number to another sequential access file named UpdatedAdvanced28.txt. Save and then run the program. Use a text editor to open the UpdatedAdvanced28.txt file. Each number in the file should be one greater than its corresponding number in the Advanced28.txt file. Close the UpdatedAdvanced28.txt file.

ADVANCED

29. If necessary, create a new project named Advanced29 Project and save it in the Cpp8\Chap14 folder. Also create a new source file named Advanced29.cpp. If you are using Microsoft Visual C++, copy the Advanced29.txt file from the Cpp8\Chap14 folder to the Cpp8\Chap14\Advanced29 Project folder. Use a text editor to open the Advanced29.txt file, which contains 12 numbers. Close the Advanced29.txt file. Write a program that reads the numbers contained in the Advanced29.txt file and writes only the even numbers to a new sequential access file named EvenAdvanced29.txt. Save and then run the program. Use a text editor to open the EvenAdvanced29.txt file, which should contain only the even numbers. Close the EvenAdvanced29.txt file.

SWAT THE BUGS

30. Follow the instructions for starting C++ and viewing the SwatTheBugs30.cpp file, which is contained in either the Cpp8\Chap14\SwatTheBugs30 Project folder or the Cpp8\Chap14 folder. (Depending on your C++ development tool, you may need to open the project/solution file first.) The program should display the contents of the SwatTheBugs30.txt file, but it is not working correctly. Run the program. Debug the program.

Answers to TRY THIS Exercises

Pencil and Paper

1. `ifstream inSales;`
2. `outJan.open("janSales.txt");` or
   `outJan.open("janSales.txt", ios::out);`

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Computer

16. See Figure 14-21.

```
//TryThis16.cpp - writes 10 letters (in uppercase)
//to a sequential access file
//Created/revised by <your name> on <current date>

#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    char letter = ' ';

    ofstream outFile;
    outFile.open("TryThis16.txt");

    if (outFile.is_open())
    {
        for (int x = 1; x < 11; x += 1)
        {
            cout << "Enter letter " << x << ": ";
            cin >> letter;
            letter = toupper(letter);
            outFile << letter << endl;
        }  //end for
        outFile.close();
    }
    else
        cout << "Can't open the TryThis16.txt file ."
             << endl;
    //end if
    return 0;
}   //end of main function
```

TryThis16.txt

```
A
B
C
D
E
F
G
H
I
J
```

Figure 14-21

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

17. See Figure 14-22.

```
//TryThis17.cpp - saves numbers from 10 through
//25, along with the square of each number,
//in a sequential access file
//Created/revised by <your name> on <current date>

#include <iostream>
#include <cmath>
#include <fstream>
using namespace std;

int main()
{
    ofstream outNumbers;
    outNumbers.open("TryThis17.txt");

    if (outNumbers.is_open())
    {
        for (int x = 10; x < 26; x += 1)
            outNumbers << x << '#'
                << pow(x, 2.0) << endl;
        //end for
        outNumbers.close();
        cout << "Numbers saved in file." << endl;
    }
    else
        cout << "Can't open the TryThis17.txt file."
            << endl;
    //end if
    return 0;
}   //end of main function
```

TryThis17.txt

```
10#100
11#121
12#144
13#169
14#196
15#225
16#256
17#289
18#324
19#361
20#400
21#441
22#484
23#529
24#576
25#625
```

Figure 14-22

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

CHAPTER 15

Classes and Objects

After studying Chapter 15, you should be able to:

- Differentiate between procedure-oriented and object-oriented programming
- Define the terms used in object-oriented programming
- Create a class definition
- Instantiate an object from a class that you define
- Create a default constructor
- Create a parameterized constructor
- Include methods other than constructors in a class
- Overload the methods in a class

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Ch15-Chapter Preview

Object-Oriented Terminology

In Chapter 1, you learned that some programs are procedure oriented and some are object oriented. The programs you created in the previous chapters were procedure oriented. Recall that when writing a procedure-oriented program, the programmer concentrates on the major tasks that the program must perform to accomplish its goal. A payroll program, for example, typically performs several major tasks, such as inputting the employee data, calculating the gross pay, calculating the taxes, calculating the net pay, and outputting a paycheck. The programmer usually assigns each major task to a function, which is the primary component in a procedure-oriented program.

The primary component in an object-oriented program, on the other hand, is an object. An **object** is anything that can be seen, touched, or used. In other words, an object is nearly any *thing*. When writing an object-oriented program, the programmer focuses not only on the tasks the program must perform, but also on the objects that the program can use to perform those tasks. The objects can take on many different forms. Programs written for the Windows environment typically use objects such as check boxes, list boxes, and buttons. A payroll program, on the other hand, might utilize objects found in real life, such as a time card object, an employee object, and a paycheck object. Because each object is viewed as an independent unit, an object can be used in more than one program, usually with little or no modification. A check object used in a payroll program, for example, can also be used in a sales revenue program (which receives checks from customers) and an accounts payable program (which issues checks to creditors). The ability to use an object for more than one purpose enables code reuse, which saves programming time and money—advantages that contribute to the popularity of object-oriented programming.

Every object in an object-oriented program is created from a **class**, which is a pattern or blueprint that the computer uses to create the object. Using object-oriented programming (**OOP**) terminology, objects are **instantiated** (created) from a class, and each object is referred to as an **instance** of the class. A `string` object (variable or named constant), for example, is an instance of the `string` class and is instantiated when its declaration statement is processed in a program. Similarly, the input and output file objects discussed in Chapter 14 are instances of the `ifstream` and `ofstream` classes, respectively. Keep in mind that the class itself is not an object; only an instance of the class is an object.

Every object has **attributes**, which are the characteristics that describe the object. When you tell someone that your wristwatch is a Valenti Model VI, you are describing the watch (an object) in terms of some of its attributes—in this case, its maker and model number. A watch also has many other attributes, such as a crown, a dial, an hour hand, a minute hand, and a movement.

In addition to attributes, most objects also have behaviors. An object's **behaviors** fall into two categories: actions that the object is capable of performing and actions to which the object can respond. A watch, for example, can keep track of the time and date. Some watches can also illuminate their dials when a button on the watch is pushed. A class contains—or, in OOP terms, **encapsulates**—all of the attributes and behaviors of the object it instantiates. The term *encapsulate* means to enclose in a capsule. In the context of OOP, the "capsule" is a class.

Abstraction is another term used in OOP discussions. **Abstraction** refers to the hiding of the internal details of an object from the user. Hiding the internal details helps prevent the user from making inadvertent changes to the object. The internal mechanism of a watch, for example, is enclosed (**hidden**) in a case to protect the mechanism from damage. Attributes and behaviors that are not hidden are said to be **exposed** to the user. Exposed on a Valenti Model VI watch are the crown used to set the hour and minute hands and the button used to illuminate the dial. The idea behind abstraction is to expose to the user only those attributes and behaviors that are necessary to use the object and to hide everything else.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Another OOP term, **inheritance**, refers to the fact that you can create one class from another class. The new class, called the **derived class**, inherits the attributes and behaviors of the original class, called the **base class**. For example, the Valenti company might create a blueprint of the Model VII watch from the blueprint of the Model VI watch. The Model VII blueprint (the derived class) will inherit all of the attributes and behaviors of the Model VI blueprint (the base class), but it can then be modified to include an additional feature, such as an alarm.

Finally, you will also hear the term *polymorphism* in OOP discussions. **Polymorphism** is the object-oriented feature that allows the same instruction to be carried out differently, depending on the object. For example, you open a door, but you also open an envelope, a jar, and your eyes. Similarly, you can set the time, the date, and the alarm on a Valenti watch. Although the meaning of the verbs *open* and *set* are different in each case, you can understand each instruction because the combination of the verb and the object makes the instruction clear.

You can use the acronym APIE (Abstraction, Polymorphism, Inheritance, and Encapsulation) to help you remember some of the OOP terms.

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 15-1

1. OOP is the acronym for _____.
2. A class is an object.
   a. True
   b. False
3. An object created from a class is called _____.
   a. an attribute
   b. an instance of the class
   c. the base class
   d. the derived class
4. The actions that an object can perform or to which an object can respond are called the object's _____.
   a. attributes
   b. behaviors
   c. qualities
   d. traits

Defining a Class in C++

In previous chapters, you instantiated objects using existing classes, such as the `string` and `ofstream` classes. You used the instantiated objects in a variety of ways in many different programs. In some programs, you used a `string` object (variable or named constant) to store a name, and in others you used it to store a phone number. Similarly, one of the programs in Chapter 14 used an output file object to save eBook information. Another program in the same chapter used an output file object to save a store's sales information.

You can also define your own classes in C++ and then create instances (objects) from those classes. As do the `string` and `ofstream` classes, your classes must specify the attributes and behaviors of the objects they create. You specify the attributes and behaviors using a **class definition**. Figure 15-1 shows the syntax used in this book to define a class. The figure also includes an example of defining a class named `FormattedDate`.

The creation of a good class, which is one whose objects can be used in a variety of ways by many different programs, requires a lot of planning.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The `class` statement groups related items into one unit.

HOW TO Define a Class

Syntax

```
//declaration section
class className
{
public:                                              colon
    public attributes (data members)
    public behaviors (member methods)
private:                                             colon
    private attributes (data members)
    private behaviors (member methods)
};                                                   semicolon

[//implementation section
member method definitions]
```

Example

```
//declaration section
class FormattedDate
{
public:
     FormattedDate();                                  method prototypes
     FormattedDate(string, string, string);
     void setDate(string, string, string);
     string getFormattedDate();
private:
     string month;                                     variable declarations
     string day;
     string year;
};

//implementation section
FormattedDate::FormattedDate()
{
     //initializes the private variables
     month = "0";
     day = "0";
     year = "0";
}  //end of default constructor

FormattedDate::FormattedDate(string m, string d, string y)
{
     //initializes the private variables
     //using the values provided by the program
     month = m;
     day = d;
     year = y;
}  //end of default constructor


void FormattedDate::setDate(string m, string d, string y)
{
```

Figure 15-1 How to define a class *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
     //assigns program values to the private variables
     month = m;
     day = d;
     year = y;
}  //end of setDate method

string FormattedDate::getFormattedDate()
{
     //formats and returns values stored in the private variables
     return month + "/" + day + "/" + year;
}  //end of getFormattedDate method
```

Figure 15-1 How to define a class

Notice that the syntax contains two sections: a declaration section and an optional implementation section. The **declaration section** contains the C++ **class statement**, which begins with the keyword `class` followed by the name of the class; the statement ends with a semicolon. Although it is not a requirement, the convention is to enter the class name using **Pascal case**, which means you capitalize the first letter in the name and the first letter in any subsequent words in the name. Examples of class names that follow this naming convention include `Check`, `FormattedDate`, and `TimeCard`.

Some C++ programmers refer to the methods in a class as member functions.

Within the `class` statement, you list the attributes and behaviors of the objects that the class will create. You enclose the attributes and behaviors in a set of braces. In most cases, the attributes (called data members) are represented by variable declarations, and the behaviors (called member methods) are represented by method prototypes. A **method** is simply a function that is defined in a class definition. You enter the method definitions in the **implementation section** of a class definition. The implementation section will contain one definition for each prototype listed in the declaration section. If no method prototypes appear in the declaration section, the implementation section is not needed.

As Figure 15-1 indicates, a class can contain both public members and private members. You record the public members below the keyword `public` in the `class` statement. The private members are recorded below the keyword `private`. When you use a class to instantiate (create) an object in a program, only the public members of the class are exposed (made available) to the program; the private members are hidden. In most cases, you will want to expose the member methods and hide the data members. You expose the member methods to allow the program to use the service each method provides. You hide the variables (data members) to protect their contents from being changed inadvertently by the program. Therefore, in most class definitions, you will list the method prototypes below the keyword `public` in the `class` statement, and you will list the variable declarations below the keyword `private`, as shown in the `FormattedDate` class definition in Figure 15-1.

When a program needs to assign data to a private variable, it must use a public member method to do so. For example, a program would need to use the `setDate` method in Figure 15-1 to assign data to a FormattedDate object's `month`, `day`, and `year` variables. It is the public member method's responsibility to validate the data, if necessary, and then either assign the data to the private data member (if the data is valid) or reject the data (if the data is not valid). Keep in mind that a program does not have direct access to the private members of a class. Rather, it must access the private members indirectly, through a public member method.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Instantiating an Object and Referring to a Public Member

Figure 15-2 shows the syntax for using a class to instantiate an object in a C++ program. The figure also includes examples of instantiating a `FormattedDate` object.

HOW TO Instantiate an Object

Syntax
className objectName[(*argumentList*)]; — semicolon

Examples
```
FormattedDate reportDate;
FormattedDate reportDate(bdayMonth, bdayDay, bdayYear);
```

Figure 15-2 How to instantiate an object

After an object has been instantiated in a program, the program can refer to a public member of the class using the syntax shown in Figure 15-3. In the syntax, *objectName* and *publicMember* are the names of the object and public member, respectively. The figure also includes examples of referring to the `reportDate` object's `getFormattedDate` and `setDate` methods. Both methods are public members of the `FormattedDate` class used to instantiate a `reportDate` object.

HOW TO Refer to a Public Member of an Object's Class

Syntax
objectName.*publicMember*

Example 1
```
cout << reportDate.getFormattedDate();
```
refers to the `reportDate` object's `getFormattedDate` method, which is a public method of the `FormattedDate` class shown earlier in Figure 15-1

Example 2
```
reportDate.setDate(monthNum, dayNum, yearNum);
```
refers to the `reportDate` object's `setDate` method, which is a public method of the `FormattedDate` class shown earlier in Figure 15-1

Figure 15-3 How to refer to a public member of an object's class

In the remainder of this chapter, you will view examples of class definitions and also examples of code in which objects are instantiated and used.

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 15-2

1. A program cannot access a class's public member method directly.
   a. True
   b. False
2. In C++, you enter the `class` statement in the _____ section of a class definition, and you enter the method definitions in the _____ section.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

3. Typically, the data members (attributes) of a class are represented by _____ in a class definition.
   a. constant declarations
   b. method prototypes
   c. method definitions
   d. variable declarations
4. A class's private data member can be accessed directly by a public member method within the class.
   a. True
   b. False
5. Write the C++ statement to instantiate a `Check` object named `payCheck`.
6. Which of the following refers to the `payCheck` object's `getCheck` method?
   a. `payCheck.getCheck()`
   b. `payCheck::getCheck()`
   c. `getCheck()`
   d. `getCheck().payCheck`

Example 1—A Class That Contains a Private Data Member and Public Member Methods

Figure 15-4 shows the class definition for the `Square` class, which a program can use to instantiate a `Square` object. A `Square` object has one attribute: the length of one of its sides. The attribute is represented within the `class` statement by a private data member: a `double` variable named `side`. When a variable is declared below the `private` keyword in a `class` statement, it can be used only by the code entered in the class definition. In this case, the code uses the `side` variable to both store and retrieve the side measurement of a `Square` object.

A `Square` object has four behaviors: It can initialize its side measurement when it is created; it can assign a value to its side measurement after it has been created; it can retrieve its side measurement value; and it can calculate and return its area. In the class definition shown in Figure 15-4, these behaviors are represented by four public member methods named `Square`, `setSide`, `getSide`, and `calcArea`. The method prototypes for these methods appear below the `public` keyword in the `class` statement. The definitions of the methods appear in the implementation section of the class definition.

```
//declaration section
class Square
{
public:
    Square();
    void setSide(double);
    double getSide();
    double calcArea();
private:
    double side;
};
```

Figure 15-4 `Square` class definition *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
//implementation section
Square::Square()
{
    side = 0.0;
}  //end of default constructor

void Square::setSide(double sideValue)
{
    if (sideValue > 0.0)
        side = sideValue;
    else
        side = 0.0;
    //end if
}  //end of setSide method

double Square::getSide()
{
    return side;
}  //end of getSide method

double Square::calcArea()
{
    return side * side;
}  //end of calcArea method
```

Figure 15-4 `Square` class definition

In the `Square` class definition in Figure 15-4, the first method prototype (in the declaration section) and the first method definition (in the implementation section) pertain to the default constructor. A **constructor** is a class method whose instructions the computer automatically processes each time an object is instantiated from the class. The sole purpose of a constructor is to initialize the class's private variables.

Every class should have at least one constructor. Each of a class's constructors must have the same name as the class, but its formal parameters (if any) must be different from any other constructor in the class. A constructor that has no formal parameters is called the **default constructor**. A class can have only one default constructor.

Because a constructor does not return a value, its prototype and definition do not begin with a data type. However, notice that its definition begins with the name of the class followed by the scope resolution operator (`::`), the name of the constructor, and a set of empty parentheses—in this case, `Square::Square()`. The scope resolution operator indicates that the `Square` method is a member of (or is contained in) the `Square` class. The `Square` method's definition in Figure 15-4 contains the code to initialize the `Square` class's private `side` variable to the number 0.0.

As you learned earlier, a program does not have direct access to a private variable in a class. Rather, it must use a public method to access the private variable indirectly. A program that instantiates a `Square` object, for instance, can use the public `setSide` method in Figure 15-4 to assign a value to the private `side` variable. In this case, the `setSide` method receives the value from the program that invokes it and then stores the value in its formal parameter: a `double` variable named `sideValue`. The code contained in the `setSide` method's definition verifies that the value received from the program is greater than the number 0.0. If it is, the code assigns the value to the private `side` variable; otherwise, it assigns the number 0.0 to the variable.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Notice that the `setSide` method's prototype and definition begin with the keyword `void`, which indicates that the method does not return a value.

A program that instantiates a `Square` object can use the public `getSide` method in Figure 15-4 to retrieve the value stored in the private `side` variable. Unlike the void `setSide` method, the `getSide` method is a value-returning method. It returns the `double` number stored in the object's `side` variable.

The last method in the `Square` class, `calcArea`, is also a value-returning method. The method first calculates the area of the `Square` object by multiplying the value stored in its private `side` variable by itself. It then returns the area as a `double` number.

Figure 15-5 shows the patio area program, which uses the `Square` class to instantiate a `Square` object that represents a square patio. The program uses the `Square` object to calculate and display the patio's area. The class definition appears on Lines 8 through 43. The code pertaining to the `Square` object in the `main` function is shaded in the figure. Figure 15-5 also includes a sample run of the program.

```
 1 //Patio Area.cpp
 2 //Displays the area of a square patio
 3 //Created/revised by <your name> on <current date>
 4
 5 #include <iostream>
 6 using namespace std;
 7
 8 //declaration section
 9 class Square
10 {
11 public:
12    Square();
13    void setSide(double);
14    double getSide();
15    double calcArea();
16 private:
17    double side;
18 };
19
20 //implementation section
21 Square::Square()
22 {
23    side = 0.0;
24 }  //end of default constructor
25
26 void Square::setSide(double sideValue)
27 {
28    if (sideValue > 0.0)
29        side = sideValue;
30    else
31        side = 0.0;
32    //end if
33 }  //end of setSide method
34
35 double Square::getSide()
36 {
```

Figure 15-5 Patio area program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
37    return side;
38 }  //end of getSide method
39
40 double Square::calcArea()
41 {
42    return side * side;
43 }  //end of calcArea method
44
45 int main()
46 {
47    //instantiate a Square object
48    Square patio;
49    //declare variables
50    double sideMeasurement = 0.0;
51    double area = 0.0;
52
53    //get side measurement
54    cout << "Side measurement (feet): ";
55    cin >> sideMeasurement;
56    //assign side measurement to Square object
57    patio.setSide(sideMeasurement);
58
59    //calculate and display area
60    area = patio.calcArea();
61    cout <<
62        "The area of a square patio "
63        << "with a side measurement of " << endl
64        << patio.getSide() << " feet is "
65        << area << " square feet." << endl;
66    return 0;
67 }  //end of main function
```

```
Patio Area
Side measurement (feet): 11.5
The area of a square patio with a side measurement of
11.5 feet is 132.25 square feet.
Press any key to continue . . .
```

Figure 15-5 Patio area program

The `Square patio;` statement on Line 48 instantiates a `Square` object named `patio`. When the object is created, the default constructor is automatically called to initialize the private data member (the `side` variable) to the number 0.0. The `patio.setSide(sideMeasurement);` statement on Line 57 calls the `Square` object's `setSide` method, passing it the side measurement value entered by the user. Recall that the `setSide` method is a public member of the `Square` class. The `setSide` method verifies that the value passed to it is greater than the number 0.0. If it is, the method assigns the value to the `Square` object's private `side` variable; otherwise, it assigns the number 0.0 to the variable.

Next, the `area = patio.calcArea();` statement on Line 60 calls the `Square` object's `calcArea` method to calculate and return the `Square` object's area. The statement assigns the method's return value to the program's `area` variable. The `cout` statement on Lines 61 through 65 displays a message on the computer screen. The message contains the `Square` object's side measurement and area. When processing the `cout` statement, the `patio.getSide()` code on Line 64 calls the `Square` object's `getSide` method, which simply retrieves the value stored in the private `side` variable.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Header Files

Although you can enter a class definition in the program that uses the class, as shown earlier in Figure 15-5, most programmers enter a class definition in a separate text file called a **header file**. Figure 15-6 shows the definition of the `Square` class entered in a header file named Square.h. Unlike program filenames, which end with .cpp, header filenames end with .h. You will learn how to add a header file to a solution in Lab 15-2.

```
 1 //Square.h
 2 //Created/revised by <your name> on <current date>
 3
 4 //declaration section
 5 class Square
 6 {
 7 public:
 8    Square();
 9    void setSide(double);
10    double getSide();
11    double calcArea();
12 private:
13    double side;
14 };
15
16 //implementation section
17 Square::Square()
18 {
19    side = 0.0;
20 }  //end of default constructor
21
22 void Square::setSide(double sideValue)
23 {
24    if (sideValue > 0.0)
25        side = sideValue;
26    else
27        side = 0.0;
28    //end if
29 }  //end of setSide method
30
31 double Square::getSide()
32 {
33    return side;
34 }  //end of getSide method
35
36 double Square::calcArea()
37 {
38    return side * side;
39 }  //end of calcArea method
```

Figure 15-6 `Square` class definition entered in the Square.h header file

Figure 15-7 shows a modified version of the patio area program. Unlike the original program, the modified program does not contain the `Square` class definition. Instead, it uses the class definition contained in the Square.h header file from Figure 15-6. You can store a header file in the same location as the program file that employs the class. In this case, for example, the Square.h file would be stored in the same location as the Modified Patio Area.cpp file. The programmer uses a `#include` directive to tell the compiler to include the contents of the header

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

The angle brackets (<>) in a directive indicate that the file is located in the same folder as the C++ Standard Library header files.

file in the program. In the modified patio area program, the `#include "Square.h"` directive (which is shaded in Figure 15-7) tells the compiler to merge the contents of the Square.h file with the contents of the current program. The quotation marks before and after the header filename indicate that the header file is located in the same location as the program file.

```
 1 //Modified Patio Area.cpp
 2 //Displays the area of a square patio
 3 //Created/revised by <your name> on <current date>
 4
 5 #include <iostream>
 6 #include "Square.h"
 7 using namespace std;
 8
 9 int main()
10 {
11    //instantiate a Square object
12    Square patio;
13    //declare variables
14    double sideMeasurement = 0.0;
15    double area = 0.0;
16
17    //get side measurement
18    cout << "Side measurement (feet): ";
19    cin >> sideMeasurement;
20    //assign side measurement to Square object
21    patio.setSide(sideMeasurement);
22
23    //calculate and display area
24    area = patio.calcArea();
25    cout <<
26        "The area of a square patio "
27        << "with a side measurement of " << endl
28        << patio.getSide() << " feet is "
29        << area << " square feet." << endl;
30    return 0;
31 }  //end of main function
```

Figure 15-7 Modified patio area program

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

Mini-Quiz 15-3

1. The `::` operator is called the _____.
2. Write the default constructor's prototype for a class named `Item`.
3. The `Item` class in Question 2 contains two private data members: a `char` variable named `code` and an `int` variable named `price`. Write the definition for the default constructor.

Example 2—A Class That Contains a Parameterized Constructor

Figure 15-8 shows a modified version of the `Square` class from Example 1. The modifications made to the original class from Figure 15-6 are shaded in the figure. Notice that this version of the `Square` class contains an additional constructor. The additional constructor has one

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

parameter: a `double` variable named `num`. Constructors that contain at least one parameter are called **parameterized constructors**. In this case, the parameterized constructor allows you to specify the `side` variable's initial value in the statement that instantiates a `Square` object in a program. You specify the value by enclosing it in a set of parentheses after the object's name. For example, either of the following program statements will invoke the parameterized constructor shown in Figure 15-8: `Square picture(9.5);` or `Square picture(sideLength);`. When the parameterized constructor is invoked, it calls the `setSide` method, passing it the value it receives from the program. The `setSide` method determines whether the value is greater than 0.0 and then assigns either the value or the number 0.0 to the private `side` variable.

```
 1 //Modified Square.h
 2 //Created/revised by <your name> on <current date>
 3
 4 //declaration section
 5 class Square
 6 {
 7 public:
 8    Square();
 9    Square(double);
10    void setSide(double);
11    double getSide();
12    double calcArea();
13 private:
14    double side;
15 };
16
17 //implementation section
18 Square::Square()
19 {
20    side = 0.0;
21 }  //end of default constructor
22
23 Square::Square(double num)
24 {
25    setSide(num);
26 }  //end of constructor
27
28 void Square::setSide(double sideValue)
29 {
30    if (sideValue > 0.0)
31        side = sideValue;
32    else
33        side = 0.0;
34    //end if
35 }  //end of setSide method
36
37 double Square::getSide()
38 {
39    return side;
40 }  //end of getSide method
41
42 double Square::calcArea()
43 {
44    return side * side;
45 }  //end of calcArea method
```

default constructor

parameterized constructor

If the `setSide` method did not contain validation code, you could replace the statement in the parameterized constructor with `side = num;`.

Figure 15-8 Modified `Square` class definition entered in the Modified Square.h header file

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

A method's name combined with its optional *parameterList* is called the method's **signature**. When a program statement instantiates an object, the computer compares the statement with the signature of each of the class constructors; it stops comparing when it finds a match. Put another way, the computer determines the appropriate class constructor by matching the quantity, data type, and position (order) of the arguments in the statement that instantiates the object with the quantity, data type, and position (order) of the parameters listed in each constructor's *parameterList*. In this case, the computer will invoke the default constructor when you use the `Square picture;` statement to instantiate a `Square` object. However, as mentioned earlier, it will use the parameterized constructor when you instantiate a `Square` object using statements such as `Square picture(9.5);` or `Square picture(sideLength);`. Figure 15-9 shows how you could use the parameterized constructor in the modified patio area program.

```
 1 //Modified Patio Area.cpp
 2 //Displays the area of a square patio
 3 //Created/revised by <your name> on <current date>
 4
 5 #include <iostream>
 6 #include "Modified Square.h"
 7 using namespace std;
 8
 9 int main()
10 {
11    //declare variables
12    double sideMeasurement = 0.0;
13    double area = 0.0;
14
15    //get side measurement
16    cout << "Side measurement (feet): ";
17    cin >> sideMeasurement;
18
19    //instantiate a Square object
20    Square patio(sideMeasurement);
21
22    //calculate and display area
23    area = patio.calcArea();
24    cout <<
25        "The area of a square patio "
26        << "with a side measurement of " << endl
27        << patio.getSide() << " feet is "
28        << area << " square feet." << endl;
29    return 0;
30 }  //end of main function
```

instantiates a Square object

Figure 15-9 Modified patio area program using the parameterized constructor from Figure 15-8

Compare the code shown in Figure 15-9 with the code shown earlier in Figure 15-7. Notice that the statement to instantiate a `Square` object now appears after the `cin` statement that gets the side measurement, and it now contains an argument (the `sideMeasurement` variable). In Figure 15-7, the instantiation code (on Line 12) appears before the variable declaration statements, and it doesn't contain any arguments. Also notice that the code in Figure 15-9 does not call the `setSide` method, as the code on Line 21 in Figure 15-7 does. The `setSide` method is not necessary in Figure 15-9's code because the parameterized constructor will set the private `side` variable's value when the `Square` object is instantiated.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Example 3—Reusing a Class

In Examples 1 and 2, you used the `Square` class to create an object that represented a square patio. In this example, you will use the `Square` class to create objects that represent a square pizza and a square pizza slice. As mentioned earlier, the ability to use an object for more than one purpose saves programming time and money, which contributes to the popularity of object-oriented programming.

Figure 15-10 shows the pizza slices program, which calculates and displays the number of square pizza slices that can be cut from a square pizza. The figure also contains a sample run of the program.

```
 1 //Pizza Slices.cpp
 2 //Displays the number of square slices
 3 //that can be cut from a square pizza
 4 //Created/revised by <your name> on <current date>
 5
 6 #include <iostream>
 7 #include "Modified Square.h"
 8 using namespace std;
 9
10 int main()
11 {
12    //instantiate Square objects
13    Square wholePizza;
14    Square pizzaSlice;
15    //declare variables
16    double wholeSide = 0.0;
17    double sliceSide = 0.0;
18    double wholeArea = 0.0;
19    double sliceArea = 0.0;
20    double numSlices = 0.0;
21
22    //get side measurements
23    cout << "Whole pizza side measurement (inches): ";
24    cin >> wholeSide;
25    cout << "Pizza slice side measurement (inches): ";
26    cin >> sliceSide;
27
28    //assign side measurements to Square objects
29    wholePizza.setSide(wholeSide);
30    pizzaSlice.setSide(sliceSide);
31
32    //calculate areas
33    wholeArea = wholePizza.calcArea();
34    sliceArea = pizzaSlice.calcArea();
35
36    //calculate number of slices
37    if (sliceArea > 0.0)
38        numSlices = wholeArea / sliceArea;
39    //end if
40    cout << "Number of slices: " << numSlices << endl;
41    return 0;
42 }  //end of main function
```

Figure 15-10 Pizza slices program showing another use for the `Square` class *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Figure 15-10 Pizza slices program showing another use for the `Square` class

The pizza slices program instantiates two `Square` objects named `wholePizza` and `pizzaSlice` to represent the whole pizza and a pizza slice, respectively. The program then declares five `double` variables to store the side measurement of the whole pizza, the side measurement of a pizza slice, the area of the whole pizza, the area of a pizza slice, and the number of slices.

The `cout` and `cin` statements on Lines 23 through 26 prompt the user for the side measurements of the whole pizza and a pizza slice and store the user's responses in the `wholeSide` and `sliceSide` variables, respectively. The statement on Line 29 calls the `wholePizza` object's `setSide` method, passing it the side measurement of the whole pizza. The statement on Line 30 calls the `pizzaSlice` object's `setSide` method, passing it the side measurement of a pizza slice. Each time the `setSide` method is invoked, it validates the value passed to it and then assigns either the value or the number 0.0 to the appropriate object's `side` variable.

Next, the assignment statement on Line 33 in Figure 15-10 calls the `wholePizza` object's `calcArea` method to calculate and return the area of the whole pizza. Similarly, the assignment statement on Line 34 calls the `pizzaSlice` object's `calcArea` method to calculate and return the area of a pizza slice. The condition in the `if` clause on Line 37 then checks whether the pizza slice area is greater than 0.0. This determination is necessary because the pizza slice area is used as the divisor in the statement on Line 38. That statement calculates the number of slices by dividing the area of the whole pizza by the area of a pizza slice. Finally, the `cout` statement on Line 40 displays a message that contains the number of slices.

Example 4—A Class That Contains Overloaded Methods

Figure 15-11 shows a different version of the `Square` class used in the previous examples. This version contains two (rather than one) `calcArea` methods. Although both methods have the same name, notice that their *parameterLists* differ. The *parameterList* in the first `calcArea` method is empty, as it was in the previous examples. The *parameterList* in the second `calcArea` method, however, contains one formal parameter: a `double` variable named `sideValueFromProgram`. (The second method's prototype and definition are shaded in the figure.) When two or more methods have the same name but different *parameterLists*, the methods are referred to as **overloaded methods**.

The two constructors in Figure 15-11 are overloaded methods because both have the same name but a different *parameterList*.

```
1 //Overloaded Square.h
2 //Created/revised by <your name> on <current date>
3
4 //declaration section
5 class Square
6 {
7 public:
8    Square();
9    Square(double);
```

Figure 15-11 `Square` class definition entered in the Overloaded Square.h file *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
10    void setSide(double);
11    double getSide();
12    double calcArea();
13    double calcArea(double);
14 private:
15    double side;
16 };
17
18 //implementation section
19 Square::Square()
20 {
21    side = 0.0;
22 }  //end of default constructor
23
24 Square::Square(double num)
25 {
26    setSide(num);
27 }  //end of constructor
28
29 void Square::setSide(double sideValue)
30 {
31    if (sideValue > 0.0)
32        side = sideValue;
33    else
34        side = 0.0;
35    //end if
36 }  //end of setSide method
37
38 double Square::getSide()
39 {
40    return side;
41 }  //end of getSide method
42
43 double Square::calcArea()
44 {
45    return side * side;
46 }  //end of calcArea method
47
48 double Square::calcArea(double sideValueFromProgram)
49 {
50    setSide(sideValueFromProgram);
51    return side * side;
52 }  //end of calcArea method
```

Figure 15-11 `Square` class definition entered in the Overloaded Square.h file

Overloading is useful when two or more methods require different parameters to perform essentially the same task. Both overloaded `calcArea` methods in the `Square` class, for example, calculate and return the area of a `Square` object. However, the first `calcArea` method does not require a program to pass it any information. The second `calcArea` method, on the other hand, requires a program to pass it one item of information: the side measurement of the `Square` object. Like the parameterized constructor, the parameterized `calcArea` method calls the `setSide` method to validate the side measurement provided by the program. After the `setSide` method assigns the appropriate value to the object's private `side` variable, the `calcArea` method calculates and returns the `Square` object's area.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Overloading is an example of polymorphism.

Figure 15-12 shows a modified version of the pizza slices program from Example 3, with the modifications shaded in the figure. The modified version uses the parameterized `calcArea` method. The figure also contains a sample run of the program.

```
 1 //Modified Pizza Slices.cpp
 2 //Displays the number of square slices
 3 //that can be cut from a square pizza
 4 //Created/revised by <your name> on <current date>
 5
 6 #include <iostream>
 7 #include "Overloaded Square.h"
 8 using namespace std;
 9
10 int main()
11 {
12    //instantiate Square objects
13    Square wholePizza;
14    Square pizzaSlice;
15    //declare variables
16    double wholeSide = 0.0;
17    double sliceSide = 0.0;
18    double wholeArea = 0.0;
19    double sliceArea = 0.0;
20    double numSlices = 0.0;
21
22    //get side measurements
23    cout << "Whole pizza side measurement (inches): ";
24    cin >> wholeSide;
25    cout << "Pizza slice side measurement (inches): ";
26    cin >> sliceSide;
27
28    //calculate areas
29    wholeArea = wholePizza.calcArea(wholeSide);
30    sliceArea = pizzaSlice.calcArea(sliceSide);
31
32    //calculate number of slices
33    if (sliceArea > 0.0)
34        numSlices = wholeArea / sliceArea;
35    //end if
36    cout << "Number of slices: " << numSlices << endl;
37    return 0;
38 }  //end of main function
```

```
Pizza Slices
Whole pizza side measurement (inches): 8
Pizza slice side measurement (inches): 2.5
Number of slices: 10.24
Press any key to continue . . .
```

Figure 15-12 Modified pizza slices program using overloaded methods

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Compare the code shown in Figure 15-12 with the code shown earlier in Figure 15-10. Unlike the code in Figure 15-10 (on Lines 29 and 30), the code in Figure 15-12 does not call the `setSide` method to assign a value to each `Square` object's `side` variable. The `setSide` method is not necessary in Figure 15-12's code because each object's parameterized `calcArea` method calls the `setSide` method before it calculates and returns the area.

Mini-Quiz 15-4

1. A method's name along with its optional *parameterList* is called the method's _____.
2. Write the prototype for a parameterized constructor in the `Item` class. The constructor has one formal parameter that has the `int` data type.
3. If a class contains two methods that have the same name but different *parameterLists*, the methods are referred to as _____ methods.

The answers to Mini-Quiz questions are contained in the Answers.pdf file.

LAB 15-1 Stop and Analyze

Study the program shown in Figure 15-13 and then answer the questions.

The answers to the labs are contained in the Answers.pdf file.

```
 1 //Lab15-1.cpp - displays an increased price
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <string>
 6 #include <iomanip>
 7 using namespace std;
 8
 9 //declaration section
10 class Item
11 {
12 public:
13    Item();
14    void setData(string, double);
15    double getIncreasedPrice(double);
16 private:
17    string id;
18    double price;
19 };
20
21 //implementation section
22 Item::Item()
23 {
24    id    = "";
25    price = 0.0;
26 } //end of default constructor
27
```

Figure 15-13 Code for Lab 15-1 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
28 void Item::setData(string idNum, double p)
29 {
30    id = idNum;
31    price = p;
32 }  //end of setData method
33
34 double Item::getIncreasedPrice(double rate)
35 {
36    if (rate > 1.0)
37        rate /= 100;
38    //end if
39    return price + price * rate;
40 }  //end of getIncreasedPrice method
41
42
43 int main()
44 {
45    //instantiate an Item object
46    Item computer;
47
48    //declare variables
49    string computerId = "";
50    double computerPrice = 0.0;
51    double incRate = 0.0;
52
53    cout << fixed << setprecision(2);
54
55    //get computer ID
56    cout << "Computer ID (X to end): ";
57    getline(cin, computerId);
58    while (computerId != "X" && computerId != "x")
59    {
60        //get price and increase rate
61        cout << "Enter the price: ";
62        cin >> computerPrice;
63        cin.ignore(100, '\n');
64        cout << "Increase rate in decimal form: ";
65        cin >> incRate;
66        cin.ignore(100, '\n');
67
68        //assign the ID and price
69
70
71        //display the increased price
72        cout << "The new price of computer "
73             << computerId << " is $"
74
75             << endl;
76
77        //get computer ID
78        cout << endl << "Computer ID (X to end): ";
79        getline(cin, computerId);
80    }   //end while
81    return 0;
82 }  //end of main function
```

Figure 15-13 Code for Lab 15-1

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

QUESTIONS

1. What are the names of the private data members in the `Item` class?
2. What are the name and purpose of the default constructor?
3. What is the purpose of the `setData` method?
4. What is the purpose of the `getIncreasedPrice` method?
5. What is the purpose of the code on Lines 36 and 37?
6. What statement is missing from Line 69?
7. Line 74 should display the increased price. What code is missing from that line?
8. Follow the instructions for starting C++ and viewing the Lab15-1.cpp file, which is contained in either the Cpp8\Chap15\Lab15-1 Project folder or the Cpp8\Chap15 folder. (Depending on your C++ development tool, you may need to open Lab15-1's project/solution file first.)
9. Enter the missing statement and code from Steps 6 and 7. Save and then run the program. Enter ABX-12 as the computer ID, 2500 as the price, and 0.1 as the increase rate. The program displays the message "The new price of computer ABX-12 is $2750.00." Now, enter PYZ-43 as the computer ID, 1900 as the price, and 10 as the increase rate. The program displays the message "The new price of computer PYZ-43 is $2090.00."
10. Test the program using different computer IDs, prices, and rates. When you are finished testing the program, enter the letter x as the computer ID.

LAB 15-2 Plan and Create

Figure 15-14 shows the problem specification for Lab 15-2.

Problem specification
Sharon Terney of Terney Landscaping wants a program that estimates the cost of laying sod on a rectangular piece of land. Jack Sysmanski, the owner of All-Around Fence Company, wants a program that calculates the cost of installing a fence around a rectangular yard. You will create the Terney Landscaping program in this lab and then create the All-Around Fence Company program in Computer Exercise 14 at the end of the chapter.

While analyzing both problems, you notice that each involves a rectangular shape. In the Terney Landscaping program, you need to find the area of the rectangle on which the sod is to be laid. In the All-Around Fence Company program, on the other hand, you need to find the perimeter of the rectangle around which a fence is to be constructed. To save time, you can create a `Rectangle` class that contains the attributes and behaviors of a rectangle and then use the class to instantiate a `Rectangle` object in both programs.

Figure 15-14 Problem specification for Lab 15-2

Recall that a class defines an object's attributes and behaviors. When determining the attributes, it is helpful to consider how you would describe the object. Rectangles are typically described in terms of two dimensions: length and width. Therefore, the length and width dimensions are the attributes of a `Rectangle` object. You will include both attributes as private data members in the `Rectangle` class, using the `double` variables `length` and `width`.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Next, you determine the object's behaviors. To be useful in both the Terney Landscaping and All-Around Fence Company programs, a `Rectangle` object must be capable of performing the four tasks shown in Figure 15-15.

A `Rectangle` object should be able to:

1. initialize its private data members (default constructor)
2. assign values (received from a program) to its private data members
3. calculate and return its area
4. calculate and return its perimeter

Figure 15-15 Tasks a `Rectangle` object should be capable of performing

As Figure 15-15 indicates, a `Rectangle` object will need to initialize its private data members. You will provide a default constructor for this purpose. A `Rectangle` object will also need to provide a means for the program to assign values to the private data members. This task will be handled by a void member method named `setDimensions`. You will use two value-returning member methods named `calcArea` and `calcPerimeter` to perform the third and fourth tasks listed in Figure 15-15. Figure 15-16 shows the completed class definition for the `Rectangle` class.

```
 1 //Lab15-2 Rectangle.h
 2 //Created/revised by <your name> on <current date>
 3
 4 //declaration section
 5 class Rectangle
 6 {
 7 public:
 8    Rectangle();
 9    void setDimensions(double, double);
10    double calcArea();
11    double calcPerimeter();
12 private:
13    double length;
14    double width;
15 };
16
17 //implementation section
18 Rectangle::Rectangle()
19 {
20    length = 0.0;
21    width  = 0.0;
22 }  //end of default constructor
23
24 void Rectangle::setDimensions(double len, double wid)
25 {
26    //assigns dimensions to private data members
27    if (len > 0.0 && wid > 0.0)
28    {
29        length = len;
30        width  = wid;
31    }  //end if
32 }  //end of setDimensions method
33
```

Figure 15-16 `Rectangle` class definition *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
34 double Rectangle::calcArea()
35 {
36    return length * width;
37 }  //end of calcArea method
38
39 double Rectangle::calcPerimeter()
40 {
41    return (length + width) * 2;
42 }  //end of calcPerimeter method
```

Figure 15-16 `Rectangle` class definition

Now that you have defined the `Rectangle` class, you can begin creating the Terney Landscaping program, which will use the class to create a `Rectangle` object. Figure 15-17 shows the IPO chart information and C++ instructions for the program. According to the IPO chart, the output is the area (in square yards) and the total price. The input is the length and width of the rectangle (both in feet) and the price of a square yard of sod. Notice that a `Rectangle` object is used as a processing item in the program.

IPO chart information	**C++ instructions**
Input	
length (in feet)	`double lawnLength = 0.0;`
width (in feet)	`double lawnWidth  = 0.0;`
sod price (per square yard)	`double priceSqYd  = 0.0;`
Processing	
Rectangle object	`Rectangle lawn;`
Output	
area (in square yards)	`double lawnArea   = 0.0;`
total price	`double totalPrice = 0.0;`
Algorithm	

1. enter length, width, and sod price	`cout << "Length (in feet): ";`
	`cin >> lawnLength;`
	`cout << "Width (in feet): ";`
	`cin >> lawnWidth;`
	`cout << "Sod price (per square yard): ";`
	`cin >> priceSqYd;`

2. use the Rectangle object's setDimensions method to assign the length and width to the Rectangle object; pass the method the length and width measurements	`lawn.setDimensions(lawnLength, lawnWidth);`
3. use the Rectangle object's calcArea method to calculate the area in square feet, then divide the result by 9 to get the area in square yards	`lawnArea = lawn.calcArea() / 9;`

Figure 15-17 IPO chart information and C++ instructions for the Terney Landscaping program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

4. calculate the total price by multiplying the area by the sod price	`totalPrice = lawnArea * priceSqYd;`
5. display the area and the total price	`cout << "Square yards: " << lawnArea << endl;` `cout << "Total price: $" << totalPrice << endl;`

Figure 15-17 IPO chart information and C++ instructions for the Terney Landscaping program

As Figure 15-17 indicates, the program first gets the length, width, and sod price information from the user. The program passes the length and width information to the `Rectangle` object's `setDimensions` method, which assigns the values (assuming that both are greater than 0.0) to the `Rectangle` object's private data members.

Next, the program calculates the area of the `Rectangle` object in square yards. It does this by first calling the `Rectangle` object's `calcArea` method to calculate the area in square feet. It then converts the value returned by the `calcArea` method from square feet to square yards by dividing the return value by the number 9, which is the number of square feet in a square yard.

After calculating the area in square yards, the program calculates the total price by multiplying the number of square yards by the price per square yard of sod. Finally, the program displays the area (in square yards) and the total price on the screen. Although the `Rectangle` object is also capable of calculating its perimeter, the current program does not require the object to perform that task.

Figure 15-18 shows the code for the entire Terney Landscaping program and includes a sample run of the program.

```
 1 //Lab15-2.cpp - displays the cost of laying sod
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <iomanip>
 6 #include "Lab15-2 Rectangle.h"
 7 using namespace std;
 8
 9 int main()
10 {
11    //instantiate a Rectangle object
12    Rectangle lawn;
13
14    //declare variables
15    double lawnLength = 0.0;
16    double lawnWidth  = 0.0;
17    double priceSqYd  = 0.0;
18    double lawnArea   = 0.0;
19    double totalPrice = 0.0;
20
21    //get length, width, and sod price
22    cout << "Length (in feet): ";
23    cin >> lawnLength;
24    cout << "Width (in feet): ";
25    cin >> lawnWidth;
```

Figure 15-18 Terney Landscaping program *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
26    cout << "Sod price (per square yard): ";
27    cin >> priceSqYd;
28
29    //assign dimensions to Rectangle object
30    lawn.setDimensions(lawnLength, lawnWidth);
31
32    //calculate area and total price
33    lawnArea = lawn.calcArea() / 9;
34    totalPrice = lawnArea * priceSqYd;
35
36    //display area and total price
37    cout << fixed << setprecision(2) << endl;
38    cout << "Square yards: " << lawnArea << endl;
39    cout << "Total price: $" << totalPrice << endl;
40    return 0;
41 }  //end of main function
```

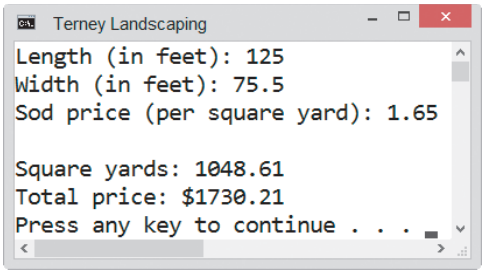


Figure 15-18 Terney Landscaping program

DIRECTIONS

Follow the instructions for starting your C++ development tool. Depending on the development tool you are using, you may need to create a new project; if so, name the project Lab15-2 Project and save it in the Cpp8\Chap15 folder. Enter the instructions shown in Figure 15-18 in a source file named Lab15-2.cpp. (Do not enter the line numbers.) Save the file in either the project folder or the Cpp8\Chap15 folder.

Next, you will add a header file to either the project folder (if you are using Microsoft Visual C++) or the Cpp8\Chap15 folder (if you are using Dev-C++ or Code::Blocks). The instructions for doing this are shown in Figure 15-19. (If you are using a different C++ development tool, you will need to ask your instructor how and where to add a header file.)

If you are using Microsoft Visual C++, click Project on the menu bar, and then click Add New Item. If necessary, click Visual C++. Click Header File (.h) in the Add New Item dialog box, type Lab15-2 Rectangle in the Name box, and then click the Add button. If necessary, delete the `#pragma once` directive.

If you are using Dev-C++, click File on the menu bar, point to New, and then click Source File. Click File, and then click Save As. Type Lab15-2 Rectangle.h in the File name box and then click the Save button.

If you are using CODE::BLOCKS, click File on the menu bar, point to New, and then click Source File. Click File, and then click Save As. Type Lab15-2 Rectangle.h in the File name box and then click the Save button.

Figure 15-19 Instructions for adding a header file

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

In the header file, enter the `Rectangle` class definition shown in Figure 15-16. Save the program. Now, follow the appropriate instructions for running the Lab15-2.cpp file. Test the program using the data shown in Figure 15-18. If necessary, correct any bugs (errors) in the program.

LAB 15-3 Modify

If necessary, create a new project named Lab15-3 Project and save it in the Cpp8\Chap15 folder. Enter (or copy) the Lab15-2.cpp instructions into a new source file named Lab15-3.cpp. Change Lab15-2.cpp in the first comment to Lab15-3.cpp. Also change "Lab15-2 Rectangle.h" in the #include directive to "Lab15-3 Rectangle.h".

Next, enter (or copy) the Lab15-2 Rectangle.h instructions into a new header file named Lab15-3 Rectangle.h. Change Lab15-2 Rectangle.h in the first comment to Lab15-3 Rectangle.h. Add a second `setDimensions` method to the `Rectangle` class. The method should accept two integers rather than two `double` numbers. Now, modify the program so that it uses integers (rather than `double` numbers) for the length and width measurements. Save and then run the program. Test the program using 10 as the length, 15 as the width, and 1.95 as the sod price. The number of square yards and total price are 16.67 and $32.50, respectively.

LAB 15-4 What's Missing?

The program in this lab should display the area of a parking lot. Start your C++ development tool, and view the Lab15-4.cpp and Parallelogram.h files, which are contained in either the Cpp8\Chap15\Lab15-4 Project folder or the Cpp8\Chap15 folder. (Depending on your C++ development tool, you may need to open Lab15-4's project/solution file first.) Put the C++ instructions in the proper order, and then determine the one or more missing instructions. Test the program appropriately.

LAB 15-5 Desk-Check

Desk-check the code shown in Figure 15-20 using Carla Rensen and 12456.75 as the salesperson's name and sales amount, respectively. Then use X to stop the program. What will the code display on the computer screen?

```
1 //Lab15-5.cpp - displays a bonus amount
2 //Created/revised by <your name> on <current date>
3
4 #include <iostream>
5 #include <string>
```

Figure 15-20 Code for Lab 15-5 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
 6 #include <iomanip>
 7 using namespace std;
 8
 9 //declaration section
10 class Bonus
11 {
12 public:
13    Bonus();
14    void setSales(double);
15    double getBonus(double);
16 private:
17    double sales;
18 };
19
20 //implementation section
21 Bonus::Bonus()
22 {
23    sales = 0.0;
24 }  //end of default constructor
25
26 void Bonus::setSales(double s)
27 {
28    if (s > 0.0)
29        sales = s;
30    else
31        sales = 0.0;
32    //end if
33 }  //end of setSales method
34
35 double Bonus::getBonus(double r)
36 {
37    return sales * r;
38 }  //end of getBonus method
39
40 int main()
41 {
42    const double BONUS_RATE = 0.05;
43    string name = "";
44    double dollars = 0.0;
45    double bonus = 0.0;
46    Bonus dollarAmt;
47
48    cout << fixed << setprecision(2);
49    cout << "Salesperson's name (X to exit): ";
50    getline(cin, name);
51    while (name != "X" && name != "x")
52    {
53        cout << "Sales amount: ";
54        cin >> dollars;
55        cin.ignore(100, '\n');
56        dollarAmt.setSales(dollars);
57
```

Figure 15-20 Code for Lab 15-5 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
58         bonus = dollarAmt.getBonus(BONUS_RATE);
59         cout << name << " bonus:  $"  << bonus << endl;
60
61         cout << endl << "Salesperson's name (X to exit): ";
62         getline(cin, name);
63     }   //end while
64     return 0;
65 }  //end of main function
```

Figure 15-20 Code for Lab 15-5

LAB 15-6 Debug

Follow the instructions for starting C++ and viewing the Lab15-6.cpp file, which is contained in either the Cpp8\Chap15\Lab15-6 Project folder or the Cpp8\Chap15 folder. (Depending on your C++ development tool, you may need to open Lab15-6's project/solution file first.) The program should display the item number and inventory quantity entered by the user. Run the program. Notice that the program is not working correctly. Debug the program.

Chapter Summary

- A class is a pattern for creating one or more instances of the class. Each instance is considered an object.
- A class encapsulates all of an object's attributes and behaviors. An object's attributes are the characteristics that describe the object. Its behaviors are the actions that the object can perform or to which the object can respond.
- The OOP term *abstraction* refers to the hiding of an object's internal details from the user. Hiding the internal details prevents the user from making inadvertent changes to the object.
- The idea behind abstraction is to expose to the user only the attributes and behaviors that are necessary to use the object and to hide everything else. In most classes, you expose an object's behaviors (member methods) and you hide its attributes (data members).
- Polymorphism is the object-oriented feature that allows the same instruction to be carried out differently depending on the object.
- You use a class definition to create a class. The class definition contains two sections: declaration and implementation. The declaration section contains the `class` statement. The implementation section contains the method definitions.
- You instantiate (create) an object using the syntax *className objectName*[**(***argumentList***)**]**;**.
- You refer to a public member of a class using the syntax *objectName***.***publicMember*.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

- Most C++ programmers enter class definitions in header files. Header filenames end with .h.
- You can use a constructor to initialize the data members in a class when an object is instantiated. A class can have more than one constructor, but only one can be the default constructor. The default constructor has no formal parameters.
- Each constructor in a class has the same name, but its formal parameters (if any) must be different from any other constructor in the class. A constructor that has one or more formal parameters is called a parameterized constructor.
- A constructor does not have a data type because it cannot return a value.
- You can overload the methods in a class. Doing this allows you to use the same name for methods that require different information to perform the same task. The computer uses the method's signature to determine which overloaded method to process.

Key Terms

Abstraction—the OOP term that refers to the hiding of the internal details of an object from the user

Attributes—the characteristics that describe an object

Base class—the class from which a derived class is created

Behaviors—the actions that an object is capable of performing or to which the object can respond

Class—a pattern or blueprint used to instantiate an object in a program

Class definition—used to specify the attributes and behaviors of an object

class statement—the statement used to create a class in C++

Constructor—a class method whose instructions the computer automatically processes each time an object is instantiated from the class

Declaration section—the section that contains the `class` statement in a class definition

Default constructor—a constructor that has no formal parameters

Derived class—a class that inherits the attributes and behaviors of a base class

Encapsulates—the OOP term that refers to the grouping together of the attributes and behaviors of an object within a class

Exposed—the OOP term that refers to the attributes and behaviors that a program can access

Header file—a file that contains a class definition; header filenames end with .h

Hidden—the OOP term that refers to the attributes and behaviors that a program cannot access

Implementation section—the section that contains the method definitions in a class definition

Inheritance—the OOP term that refers to the fact that you can create one class (the derived class) from another class (the base class); the derived class inherits the attributes and behaviors of the base class

Instance—in OOP terminology, an object instantiated (created) from a class

Instantiated—the OOP term that refers to objects being created from a class

Method—a function that is defined in a class definition

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Object—anything that can be seen, touched, or used

OOP—an acronym for object-oriented programming

Overloaded methods—two or more class methods that share the same name but have different *parameterLists*

Parameterized constructors—constructors that have one or more formal parameters

Pascal case—the practice of capitalizing the first letter in a name and the first letter in any subsequent words in the name

Polymorphism—the object-oriented feature that allows the same instruction to be carried out differently depending on the object

Signature—the combination of a method's name with its optional *parameterList*

Review Questions

1. A blueprint for creating an object in C++ is called _____.
   a. a class
   b. an instance
   c. a map
   d. a pattern

2. Which of the following statements is false?
   a. An example of an attribute is the `minutes` variable in a `Time` class.
   b. An example of a behavior is the `setTime` method in a `Time` class.
   c. An object created from a class is referred to as an instance of the class.
   d. A class is considered an object.

3. You hide a member of a class by recording the member below the keyword _____ in the `class` statement.
   a. `confidential`
   b. `hidden`
   c. `private`
   d. `restricted`

4. You expose a member of a class by recording the member below the _____ keyword in the `class` statement.
   a. `common`
   b. `exposed`
   c. `public`
   d. `unrestricted`

5. A program can access the private members of a class _____.
   a. directly
   b. only through the public members of the class
   c. only through other private members of the class
   d. none of the above because the program cannot access the private members of a class in any way

6. In most classes, you expose the _____ and hide the _____.
   a. attributes, data members
   b. data members, member methods
   c. member methods, data members
   d. variables, member methods

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

7. The method definitions for a class are entered in the _____ section in the class definition.

   a. declaration
   b. implementation
   c. method
   d. program-defined

8. Which of the following is the scope resolution operator?

   a. `::` (two colons)
   b. `*` (asterisk)
   c. `.` (period)
   d. `->` (hyphen and greater than symbol)

9. The name of the constructor for a class named `Animal` is _____.

   a. `Animal`
   b. `AnimalConstructor`
   c. `ConstAnimal`
   d. Any of the above could be used as the name of the constructor.

10. Which of the following statements is false?

    a. You typically use a public member method to change the value stored in a private data member.
    b. Because a constructor does not return a value, you place the keyword `void` before the constructor's name.
    c. The public member methods in a class can be accessed by any program that uses an object created from the class.
    d. An instance of a class is considered an object.

11. Which of the following creates an `Animal` object named `dog`?

    a. `Animal dog;`
    b. `Animal "dog";`
    c. `dog = "Animal";`
    d. `dog Animal();`

12. A program creates an `Animal` object named `dog`. Which of the following calls the `displayBreed` method, which is a public member method contained in the `Animal` class?

    a. `Animal::displayBreed();`
    b. `displayBreed();`
    c. `dog::displayBreed();`
    d. `dog.displayBreed();`

Exercises

Pencil and Paper

TRY THIS

1. Write the class definition for a class named `Employee`. The class should include private data members for an `Employee` object's name and salary. The salary may contain a decimal place. The class should contain two constructors: the default constructor and a constructor that allows a program to assign initial values to the data members. (The answers to TRY THIS Exercises are located at the end of the chapter.)

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

TRY THIS

2. Include an additional public method in the `Employee` class from Pencil and Paper Exercise 1. The method should allow a program to assign values to the data members after an `Employee` object has been instantiated. (The answers to TRY THIS Exercises are located at the end of the chapter.)

MODIFY THIS

3. Include two additional public methods in the `Employee` class from Pencil and Paper Exercise 2. One method should allow a program to view the contents of the salary data member. The other method should allow a program to view the contents of the employee name data member.

MODIFY THIS

4. Include another public method in the `Employee` class from Pencil and Paper Exercise 3. The method should calculate an `Employee` object's new salary, which is based on the raise percentage provided by the program using the object. Before making the calculation, the method should verify that the raise percentage is greater than or equal to 0.0. If the raise percentage is less than 0.0, the method should assign the number 0.0 as the new salary.

INTRODUCTORY

5. Write the code for two overloaded methods named `getArea`. The methods belong to the `Square` class. The first `getArea` method should accept two integers. The second `getArea` method should accept two `double` numbers. Both methods should calculate the area by multiplying the first number by the second number. Each should then return the calculated value.

SWAT THE BUGS

6. Correct the errors in the `Item` class shown in Figure 15-21.

```
//declaration section
class Item
{
private:
    item();
    void assignItem(string, double);
public:
    string name;
    double price;
}

//implementation section
Item()
{
    name = "";
    price = 0.0;
}   //end of default constructor

void assignItem(string n, double p)
{
    name = n;
    price = p;
}   //end of assignItem method
```

Figure 15-21

Computer

TRY THIS

7. In this exercise, you use the `Employee` class from Pencil and Paper Exercise 4 to create an `Employee` object. Follow the instructions for starting C++ and viewing the TryThis7.cpp file, which is contained in either the Cpp8\Chap15\TryThis7 Project

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

folder or the Cpp8\Chap15 folder. (Depending on your C++ development tool, you may need to open this exercise's project/solution file first.) In the TryThis7 Employee.h header file, enter the `#include <string>` and `using namespace std;` directives. Then enter the class definition you created in Pencil and Paper Exercise 4. Next, complete the TryThis7.cpp file by entering the appropriate instructions. Use the comments as a guide. Save and then run the program. Test the program by entering your name, a current salary amount of 35000, and a raise rate of 0.05. The program should display your name, current salary ($35000), and new salay ($36750). (The answers to TRY THIS Exercises are located at the end of the chapter.)

8. In this exercise, you complete a program that uses the `FormattedDate` class shown in Figure 15-1 in the chapter. Follow the instructions for starting C++ and viewing the TryThis8.cpp file, which is contained in either the Cpp8\Chap15\TryThis8 Project folder or the Cpp8\Chap15 folder. (Depending on your C++ development tool, you may need to open this exercise's project/solution file first.) In the TryThis8 FormattedDate.h header file, enter the `#include <string>` and `using namespace std;` directives. Then enter the class definition from Figure 15-1. Next, complete the TryThis8.cpp file by entering the appropriate instructions. Use the comments as a guide. Test the program appropriately. (The answers to TRY THIS Exercises are located at the end of the chapter.)

TRY THIS

9. In this exercise, you modify the program from TRY THIS Exercise 8. If necessary, create a new project named ModifyThis9 Project and save it in the Cpp8\Chap15 folder. Copy the instructions from the TryThis8.cpp file into a source file named ModifyThis9.cpp. Change the filename in the first comment. Also copy the instructions from the TryThis8 FormattedDate.h file into a header file named ModifyThis9 FormattedDate.h file. If necessary, change the filename in the first comment. Also change the filename in the #include directive. Modify the program so that it asks the user whether he or she wants to display the formatted date using either slashes (/) or hyphens (-). Save and then run the program. Test the program by entering 4 as the month, 9 as the day, 2017 as the year, and a - (hyphen) as the separator. The program displays 4-9-2017 on the computer screen. Run the program again. Enter 12 as the month, 21 as the day, 2016 as the year, and a / (slash) as the separator. The program displays 12/21/2016 on the computer screen. (Hint: The `getFormattedDate` method should receive a string that indicates whether the user wants slashes or hyphens in the date.)

MODIFY THIS

10. In this exercise, you modify the pizza slices program shown in Figure 15-10 in the chapter. If necessary, create a new project named ModifyThis10 Project and save it in the Cpp8\Chap15 folder. Enter the C++ instructions from the figure into a source file named ModifyThis10.cpp. Change the filename in the first comment. Enter the instructions shown in Figure 15-8 in the chapter in a header file named ModifyThis10 Square.h. Change the filename in the first comment. Modify the pizza slices program so it uses the parameterized constructor in the ModifyThis10 Square.h file. Test the program appropriately.

MODIFY THIS

11. In this exercise, you modify the patio area program shown in Figure 15-7 in the chapter. If necessary, create a new project named ModifyThis11 Project and save it in the Cpp8\Chap15 folder. Enter the C++ instructions from the figure into a source file named ModifyThis11.cpp. Change the filename in the first comment. Enter the instructions shown in Figure 15-11 in the chapter in a header file named ModifyThis11 Square.h. Change the filename in the first comment. Modify the patio area program so it uses the parameterized `calcArea` method. Test the program appropriately.

MODIFY THIS

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

MODIFY THIS

12. In this exercise, you modify the program from Lab 15-4. If necessary, create a new project named ModifyThis12 Project and save it in the Cpp8\Chap15 folder. Copy the instructions from the Lab15-4.cpp file into a new source file named ModifyThis12.cpp. Change the filename in the first comment. Also copy the instructions from the Parallelogram.h file into a new header file named ModifyThis12.h. Change the filename in the first comment. Modify the program so it also asks the user to enter the paving cost per square yard. The program should now also display the cost of paving the parking lot. Test the program using 900 feet as the length, 650 feet as the height, and $10 as the cost per square yard. (Hint: The cost is $650000.)

INTRODUCTORY

13. In this exercise, you complete a program that uses the `Square` class shown in Figure 15-6 in the chapter. Follow the instructions for starting C++ and viewing the Introductory13.cpp file, which is contained in either the Cpp8\Chap15\Introductory13 Project folder or the Cpp8\Chap15 folder. (Depending on your C++ development tool, you may need to open this exercise's project/solution file first.) Enter the `Square` class definition from Figure 15-6 in the Introductory13 Square.h file. Next, complete the Introductory13.cpp file by entering the appropriate instructions. Use the comments as a guide. Test the program appropriately.

INTERMEDIATE

14. In this exercise, you use the `Rectangle` class from Lab 15-2 to instantiate a `Rectangle` object in the All-Around Fence Company program. If necessary, create a new project named Intermediate14 Project. Copy the instructions from the Lab15-2 Rectangle.h file (which is located in either the Cpp8\Chap15\Lab15-2 Project folder or the Cpp8\Chap15 folder) into a header file named Intermediate14 Rectangle.h. Change the filename in the first comment. The owner of All-Around Fence Company wants a program that calculates the cost of installing a fence. Use the IPO chart shown in Figure 15-22 to code the program. Enter your C++ instructions into a source file named Intermediate14.cpp. Display the perimeter as an integer. Display the total price with a dollar sign and two decimal places. Also enter appropriate comments and any additional instructions required by the compiler. Save and then run the program. Test the program using 120 as the length, 75 as the width, and 10 as the cost per linear foot. The program should display 390 linear feet as the perimeter and $3900.00 as the total price.

Input	Processing	Output
length (in feet) width (in feet) fence cost (per linear foot)	Processing items: Rectangle object Algorithm: 1. enter length, width, and fence cost 2. use the Rectangle object's setDimensions method to assign the length and width to the Rectangle object; pass the method the length and width 3. use the Rectangle object's calcPerimeter method to calculate and return the perimeter 4. calculate the total price by multiplying the perimeter by the fence cost 5. display the perimeter and total price	perimeter total price

Figure 15-22

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

15. In this exercise, you modify the `Rectangle` class from Lab 15-2 so that it allows a program to view the contents of the `length` and `width` data members. You also modify the Terney Landscaping program so that it displays the length and width measurements. INTERMEDIATE

    a. If necessary, create a new project named Intermediate15 Project and save it in the Cpp8\Chap15 folder. Copy the instructions from the Lab15-2.cpp file into a source file named Intermediate15.cpp. Change the filename in the first comment.

    b. Copy the instructions from the Lab15-2 Rectangle.h file (which is located in either the Cpp8\Chap15\Lab15-2 Project folder or the Cpp8\Chap15 folder) into a header file named Intermediate15 Rectangle.h file. Change the filename in the first comment.

    c. Add two value-returning methods to the `Rectangle` class. Each method should return the value of one of the private variables.

    d. Modify the Terney Landscaping program so that it uses the methods to display the length and width of the `Rectangle` object. (The program should also display the area and total price.) Test the program appropriately.

16. In this exercise, you modify the `Rectangle` class from Lab 15-2 so that its `setDimensions` method returns a value. You also modify the Terney Landscaping program. INTERMEDIATE

    a. If necessary, create a new project named Intermediate16 Project and save it in the Cpp8\Chap15 folder. Copy the instructions from the Lab15-2.cpp file into a source file named Intermediate16.cpp. Change the filename in the first comment.

    b. Copy the instructions from the Lab15-2 Rectangle.h file (which is located in either the Cpp8\Chap15\Lab15-2 Project folder or the Cpp8\Chap15 folder) into a header file named Intermediate16 Rectangle.h file. Change the filename in the first comment.

    c. Modify the `setDimensions` method so that it returns a value that indicates whether the length and width dimensions passed to the method are valid. To be valid, each dimension must be greater than 0.0. If the `setDimensions` method indicates that the length and width dimensions are valid, the program should calculate and display both the area and the total price; otherwise, it should display an error message. Modify the program appropriately.

    d. Save and then run the program. Test the program using 120 feet as the length, 75 feet as the width, and 1.55 as the price. The program should display 1000.00 as the area in square yards and $1550.00 as the total price. Now, run the program again. Enter –5 as the length, 6 as the width, and 3 as the price. The program should display an error message because the length dimension is less than 0.0.

17. In this exercise, you create a `Triangle` class and a program that uses the `Triangle` class to instantiate a `Triangle` object. INTERMEDIATE

    a. If necessary, create a new project named Intermediate17 Project and save it in the Cpp8\Chap15 folder. Create a `Triangle` class. Enter the class definition in a header file named Intermediate17 Triangle.h. The class should include a void method that allows the program to set the triangle's dimensions. The method should verify that all of the dimensions are greater than 0.0 before assigning the values to the private data members. The class also should include two value-returning methods. One value-returning method should calculate the area of a triangle, and the other should calculate the perimeter of a triangle. The formula for calculating the area of a triangle is 1/2 * *b* * *h*, where *b* is the base and *h* is the height. The formula for calculating the

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

perimeter of a triangle is $a + b + c$, where a, b, and c are the lengths of the sides. Determine the appropriate variables to include in the class. Be sure to include a default constructor that initializes the variables.

b. Create a program that prompts the user for the triangle's dimensions and then displays the triangle's area and perimeter amounts. Enter your C++ instructions in a source file named Intermediate17.cpp. Display the amounts with zero decimal places. Also enter appropriate comments and any additional instructions required by the compiler. Save and then run the program. Test the program appropriately.

ADVANCED

18. In this exercise, you modify an existing header file.

a. Follow the instructions for starting C++ and viewing the Advanced18.cpp and Advanced18 MyDate.h files, which are contained in either the Cpp8\Chap15\Advanced18 Project folder or the Cpp8\Chap15 folder. The program uses the `MyDate` class to create an object named `today`. Notice that the program prompts the user to enter the month, day, and year. It then uses the `MyDate` class's public methods (`setDate` and `displayDate`) to set and display the date entered by the user. The program also uses a public method named `updateDate` to increase the day number by 1. It then displays the new date on the screen.

b. Run the program. Enter 3 as the month, 15 as the day, and 2016 as the year. The computer screen shows that today is 3/15/2016 and tomorrow is 3/16/2016, which is correct.

c. Run the program again. This time, enter 3 as the month, 31 as the day, and 2016 as the year. The computer screen shows that today is 3/31/2016 and tomorrow is 3/32/2016, which is incorrect.

d. Modify the `updateDate` method so that it updates the date correctly. For example, if today is 3/31/2016, then tomorrow is 4/1/2016. If today is 12/31/2017, then tomorrow is 1/1/2018. You do not have to worry about leap years; treat February as though it always has 28 days. Save and then run the program. Test the program four times, using the following dates: 3/15/2016, 4/30/2017, 2/28/2017, and 12/31/2016.

ADVANCED

19. In this exercise, you modify the Terney Landscaping program from Lab 15-2 so that it passes an object to a function.

a. If necessary, create a new project named Advanced19 Project and save it in the Cpp8\Chap15 folder. Copy the instructions from the Lab15-2.cpp file into a source file named Advanced19.cpp. Change the filename in the first comment. Copy the instructions from the Lab15-2 Rectangle.h file (which is located in either the Cpp8\Chap15\Lab15-2 Project folder or the Cpp8\Chap15 folder) into a header file named Advanced19 Rectangle.h file. Change the filename in the first comment.

b. Modify the program so that it uses a function named `calcAndDisplay` to calculate and display the area and the total price. Pass the `Rectangle` object and the price per square yard to the function. Save and then run the program. Test the program using 120 feet as the length, 75 feet as the width, and 1.55 as the price. The program should display 1000.00 as the area in square yards and $1550.00 as the total price.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

ADVANCED

20. In this exercise, you modify the `Rectangle` class from Lab 15-2 so that it includes an overloaded method.

    a. If necessary, create a new project named Advanced20 Project and save it in the Cpp8\Chap15 folder. Copy the instructions from the Lab15-2 Rectangle.h file (which is located in either the Cpp8\Chap15\Lab15-2 Project folder or the Cpp8\Chap15 folder) into a header file named Advanced20 Rectangle.h. Change the filename in the first comment.

    b. Pool-Time, a company that sells in-ground pools, wants a program that its salespeople can use to determine the number of gallons of water required to fill an in-ground pool. To calculate the number of gallons, you need to find the volume of the pool. The volume formula is *length * width * depth*. Modify the `Rectangle` class so that it can be used to represent a pool. You will need to include an additional private variable to store the depth value, as well as an additional public method to calculate and return the volume. You also will need to modify the default constructor and also overload the `setDimensions` method. Be sure to verify that the depth value is greater than 0.0 before assigning the value to the private variable.

    c. Use the IPO chart shown in Figure 15-23 to code the program. Enter your C++ instructions in a source file named Advanced20.cpp. Also enter appropriate comments and any additional instructions required by the compiler. Display the volume and number of gallons with two decimal places.

    d. Save and then run the program. Use 25 feet as the length, 15 feet as the width, and 6.5 feet as the depth. The program should display 2437.50 as the volume and 18233.84 as the number of gallons of water.

Input	**Processing**	**Output**
length (in feet) width (in feet) depth (in feet)	Processing items: Rectangle object Algorithm: 1. enter length, width, and depth 2. use the Rectangle object's setDimensions method to assign the length, width, and depth to the Rectangle object; pass the method the length, width, and depth 3. use the Rectangle object's calcVolume method to calculate and return the volume 4. calculate the gallons of water by dividing the volume by 0.13368 5. display the volume and gallons of water	volume (in cubic feet) gallons of water

Figure 15-23

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Answers to TRY THIS Exercises

Pencil and Paper

1. See Figure 15-24.

```
//declaration section
class Employee
{
public:
    Employee();
    Employee(string, double);
private:
    string name;
    double salary;
};

//implementation section
Employee::Employee()
{
    name = "";
    salary = 0.0;
} //end of default constructor

Employee::Employee(string n, double s)
{
    name = n;
    salary = s;
} //end of constructor
```

Figure 15-24

2. See Figure 15-25.

```
//declaration section
class Employee
{
public:
    Employee();
    Employee(string, double);
    void setEmployee(string, double);
private:
    string name;
    double salary;
};

//implementation section
Employee::Employee()
{
    name = "";
    salary = 0.0;
} //end of default constructor
```

Figure 15-25 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
Employee::Employee(string n, double s)
{
    name = n;
    salary = s;
} //end of constructor

void Employee::setEmployee(string n, double s)
{
    name = n;
    salary = s;
} //end of setEmployee method
```

Figure 15-25

Computer

7. See Figures 15-26 and 15-27.

```
 1 //TryThis7.cpp
 2 //Calculates and displays a new salary
 3 //Created/revised by <your name> on <current date>
 4
 5 #include <iostream>
 6 #include <string>
 7 #include "TryThis7 Employee.h"
 8 using namespace std;
 9
10 int main()
11 {
12    //instantiate Employee object
13    Employee myEmployee;
14
15    //declare variables
16    string name = "";
17    double pay  = 0;
18    double rate = 0.0;
19
20    //get name, salary, and raise percentage
21    cout << "Employee's name: ";
22    getline(cin, name);
23    cout << "Employee's current salary: ";
24    cin >> pay;
25    cin.ignore(100, '\n');
26    cout << "Raise rate: ";
27    cin >> rate;
28    cin.ignore(100, '\n');
29
30    //assign name and salary to the Employee object
31    myEmployee.setEmployee(name, pay);
32
```

Figure 15-26 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
33    //use the Employee object to display the
34    //name and current salary
35    cout << "Name: " << myEmployee.getName() << endl;
36    cout<< "Current salary: $" << myEmployee.getSalary()
37        << endl;
38
39    //use the Employee object to calculate the new salary
40    myEmployee.calcNewSalary(rate);
41
42    //use the Employee object to display the new salary
43    cout << "New salary: $" << myEmployee.getSalary()
44         << endl;
45    return 0;
46 }  //end of main function
```

Figure 15-26

```
 1 //TryThis7 Employee.h
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <string>
 5 using namespace std;
 6
 7 //declaration section
 8 class Employee
 9 {
10 public:
11    Employee();
12    Employee(string, double);
13    void setEmployee(string, double);
14    double getSalary();
15    string getName();
16    void calcNewSalary(double);
17 private:
18    string name;
19    double salary;
20 };
21
22 //implementation section
23 Employee::Employee()
24 {
25    name   = "";
26    salary = 0.0;
27 }  //end of default constructor
28
29 Employee::Employee(string n, double s)
30 {
31    name   = n;
32    salary = s;
33 }  //end of constructor
34
```

Figure 15-27 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
35 void Employee::setEmployee(string n, double s)
36 {
37    name   = n;
38    salary = s;
39 }  //end of setEmployee method
40
41 double Employee::getSalary()
42 {
43    return salary;
44 }  //end of getSalary method
45
46 string Employee::getName()
47 {
48    return name;
49 }  //end of getName method
50
51 void Employee::calcNewSalary(double r)
52 {
53    if (r >= 0.0)
54        salary = salary * r + salary;
55    else
56        salary = 0.0;
57    //end if
58 }  //end of calcNewSalary method
```

Figure 15-27

8. See Figures 15-28 and 15-29.

```
 1 //TryThis8.cpp - displays a formatted date
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <string>
 6 #include "TryThis8 FormattedDate.h"
 7 using namespace std;
 8
 9 int main()
10 {
11    //create a FormattedDate object
12    FormattedDate hireDate;
13
14    //declare variables
15    string hireMonth = "";
16    string hireDay   = "";
17    string hireYear  = "";
18
19    //get month, day, and year
20    cout << "Enter the month number: ";
21    cin >> hireMonth;
22    cout << "Enter the day number: ";
23    cin >> hireDay;
24    cout << "Enter the year number: ";
25    cin >> hireYear;
```

Figure 15-28 *(continues)*

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

```
26
27    //use the FormattedDate object to set the date
28    hireDate.setDate(hireMonth, hireDay, hireYear);
29
30    //display the formatted date
31    cout << "Employee hire date: "
32         << hireDate.getFormattedDate() << endl;
33    return 0;
34 }  //end of main function
```

Figure 15-28

```
 1 //TryThis8 FormattedDate.h
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <string>
 5 using namespace std;
 6
 7 //declaration section
 8 class FormattedDate
 9 {
10 public:
11    FormattedDate();
12    void setDate(string, string, string);
13    string getFormattedDate();
14 private:
15    string month;
16    string day;
17    string year;
18 };
19
20 //implementation section
21 FormattedDate::FormattedDate()
22 {
23    //initializes the private variables
24    month = "0";
25    day   = "0";
26    year  = "0";
27 }  //end of default constructor
28
29 void FormattedDate::setDate(string m , string d, string y)
30 {
31    //assigns program values to the private variables
32    month = m;
33    day = d;
34    year = y;
35 }  //end of setDate method
36
37 string FormattedDate::getFormattedDate()
38 {
39    //formats and returns values stored in the private variables
40    return month + "/" + day + "/" + year;
41 }  //end of getFormattedDate method
```

Figure 15-29

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

APPENDIX A

C++ Keywords

```
abstract
and
and_eq
array
asm
auto
bitand
bitor
bool
break
case
catch
char
class
compl
const
const_cast
continue
default
delegate
delete
do
double
dynamic_cast
else
enum
event
explicit
export
extern
false
finally
float
for
friend
gcnew
generic
goto
if
initonly
inline
int
interface
interior_ptr
literal
long
mutable
namespace
new
not
not_eq
nullptr
operator
or
or_eq
private
property
protected
public
register
reinterpret_cast
return
safe_cast
sealed
short
signed
sizeof
static
static_cast
struct
switch
template
this
throw
true
try
typedef
typeid
typename
union
unsigned
using
virtual
void
volatile
wchar_t
while
xor
xor_eq
```

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

APPENDIX B

ASCII Codes

Character	ASCII	Binary	Character	ASCII	Binary	Character	ASCII	Binary
SPACE	32	00100000	:	58	00111010	T	84	01010100
!	33	00100001	;	59	00111011	U	85	01010101
“	34	00100010	<	60	00111100	V	86	01010110
#	35	00100011	=	61	00111101	W	87	01010111
$	36	00100100	>	62	00111110	X	88	01011000
%	37	00100101	?	63	00111111	Y	89	01011001
&	38	00100110	@	64	01000000	Z	90	01011010
'	39	00100111	A	65	01000001	[	91	01011011
(	40	00101000	B	66	01000010	\	92	01011100
)	41	00101001	C	67	01000011	]	93	01011101
*	42	00101010	D	68	01000100	^	94	01011110
+	43	00101011	E	69	01000101	_	95	01011111
‘	44	00101100	F	70	01000110	`	96	01100000
–	45	00101101	G	71	01000111	a	97	01100001
.	46	00101110	H	72	01001000	b	98	01100010
/	47	00101111	I	73	01001001	c	99	01100011
0	48	00110000	J	74	01001010	d	100	01100100
1	49	00110001	K	75	01001011	e	101	01100101
2	50	00110010	L	76	01001100	f	102	01100110
3	51	00110011	M	77	01001101	g	103	01100111
4	52	00110100	N	78	01001110	h	104	01101000
5	53	00110101	O	79	01001111	i	105	01101001
6	54	00110110	P	80	01010000	j	106	01101010
7	55	00110111	Q	81	01010001	k	107	01101011
8	56	00111000	R	82	01010010	l	108	01101100
9	57	00111001	S	83	01010011	m	109	01101101

(continues)

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

Character	ASCII	Binary	Character	ASCII	Binary	Character	ASCII	Binary
n	110	01101110	t	116	01110100	z	122	01111010
o	111	01101111	u	117	01110101	{	123	01111011
p	112	01110000	v	118	01110110	|	124	01111100
q	113	01110001	w	119	01110111	}	125	01111101
r	114	01110010	x	120	01111000	~	126	01111110
s	115	01110011	y	121	01111001	DELETE	127	01111111

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

APPENDIX C

Common Syntax Errors

1. A statement that does not end with a semicolon
2. A missing `#include` directive (such as `#include <iostream>`, `#include <string>`, `#include <fstream>`, `#include <cmath>`, `#include <ctime>`, `#include <iomanip>`, or `#include <cstdlib>`)
3. No `using namespace std;` statement
4. Unmatched braces in a function, selection structure, or loop (for example, an opening brace that does not have a corresponding closing brace or vice versa)
5. A statement block (function, multi-statement selection structure, multi-statement loop) not enclosed in a set of braces
6. Braces facing the wrong way (for example, using `}` as the opening brace or `{` as the closing brace)
7. A reserved word used as the name of a variable or named constant
8. `cin` used with the insertion operator (<<) rather than with the extraction operator (>>)
9. `cout` used with the extraction operator (>>) rather than with the insertion operator (<<)
10. The extraction operator entered as > rather than as >>
11. The insertion operator entered as < rather than as <<
12. A variable, named constant, class, object, keyword, function, or method whose name is either misspelled or entered in the wrong case
13. A space entered between two characters in an arithmetic assignment operator (for example, using + = rather than +=)
14. An arithmetic assignment operator whose characters are reversed (for example, using =* rather than *=)
15. A space entered between two characters in a comparison operator (for example, using > = rather than >=)
16. A comparison operator whose characters are reversed (for example, using => rather than >=)
17. A comparison made using one equal sign (=) rather than two equal signs (==)
18. In an `if` statement, an `else` clause that does not have a matching `if` clause
19. A `do while` statement that does not end with a semicolon

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

20. A `string` enclosed in single quotation marks rather than in double quotation marks (for example, using `'Mary'` rather than `"Mary"`)
21. A `char` character enclosed in double quotation marks rather than in single quotation marks (for example, using `"A"` rather than `'A'`)
22. A `char` memory location initialized using two single quotation marks (`''`) rather than two single quotation marks separated by a space character (`' '`)
23. An array declared using parentheses rather than square brackets
24. Accessing an element that is beyond the memory allocated to an array
25. A `for` clause that contains commas rather than semicolons, or one that does not contain two semicolons
26. In a `switch` statement, a missing colon (`:`) in a `case` clause
27. In a `switch` statement, using semicolons in a `case` clause
28. A missing `break;` statement in a `switch` statement
29. A value-returning function that does not contain a `return` statement
30. A missing function prototype for a function that is defined below the `main` function
31. A missing `#include` directive for a header file that contains a class definition

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

APPENDIX D

How To Boxes

How To	Figure	Page(s)
Create a Computer Solution to a Problem	2-3	25
Name a Memory Location in C++	3-2	54
Use the Decimal (*Base 10*) Number System	3-7	57
Use the Binary (*Base 2*) Number System	3-8	58
Initialize Variables	3-10	61
Declare a Variable in C++	3-12	62
Declare a Named Constant in C++	3-14	63
Use `cin` and >> to Get Numeric or Character Data	4-3	78
Use the `cout` Object	4-5	79
Use the `static_cast` Operator	4-10	83–84
Write an Assignment Statement	4-11	85–86
Use an Arithmetic Assignment Operator	4-13	87
Use the `if` Statement	5-5	119–120
Use Comparison Operators in an `if` Statement's Condition	5-6	120
Use Logical Operators in an `if` Statement's Condition	5-14	127–128
Use the `toupper` and `tolower` Functions	5-20	134
Use the `fixed` and `scientific` Stream Manipulators	5-21	135
Use the `setprecision` Stream Manipulator	5-22	136
Use the `switch` Statement	6-22	175
Use the `while` Statement	7-10	209
Update Counters and Accumulators	7-14	213
Use the `for` Statement	7-23	219
Use the `do while` Statement	8-4	252
Use the `pow` Function	9-2	282

(continues)

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

How To	Figure	Page(s)
Use the `sqrt` Function	9-3	283
Use the `rand` Function	9-7	286
Generate Random Integers within a Specific Range	9-8	286–287
Use the `srand` Function	9-9	288
Create a Program-Defined Value-Returning Function	9-12	291–292
Call (Invoke) a Value-Returning Function	9-13	294
Write a Function Prototype	9-17	298
Create a Program-Defined Void Function	10-2	331
Call (Invoke) a Void Function	10-3	332
Declare and Initialize a One-Dimensional Array	11-2	372
Use an Assignment Statement to Assign Data to a One-Dimensional Array	11-3	373–374
Use the Extraction Operator to Store Data in a One-Dimensional Array	11-4	374–375
Display the Contents of a One-Dimensional Array	11-5	375–376
Declare and Initialize a Two-Dimensional Array	12-2	427–428
Use an Assignment Statement to Assign Data to a Two-Dimensional Array	12-3	429
Use the Extraction Operator to Store Data in a Two-Dimensional Array	12-4	430
Display the Contents of a Two-Dimensional Array	12-5	431–432
Declare and Initialize `string` Variables and Named Constants	13-1	462
Use the Extraction Operator (>>) to Get String Input from the Keyboard	13-2	463
Use the `getline` Function to Get String Input from the Keyboard	13-3	463
Use the `ignore` Function	13-7	468–469
Use the `length` Function	13-10	472
Use the `substr` Function	13-12	474
Use the `find` Function	13-14	478–479
Use the `erase` Function	13-17	481–482
Use the `replace` Function	13-19	484
Use the `insert` Function	13-21	486
Use the `assign` Function	13-23	488
Use the Concatenation Operator	13-25	489
Create Input and Output File Objects	14-2	513
Open a Sequential Access File	14-3	514–515

(continues)

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

(continued)

How To	Figure	Page(s)
Determine the Success of the `open` Function	14-5	516–517
Write Data to a Sequential Access File	14-6	518
Read Data from a Sequential Access File	14-8	520
Test for the End of a Sequential Access File	14-9	522
Close a Sequential Access File	14-10	523
Define a Class	15-1	554–555
Instantiate an Object	15-2	556
Refer to a Public Member of an Object's Class	15-3	556

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Index

Note: Page numbers in **boldface** type indicate where key terms are defined.

Special Characters
& (ampersand), 127–128, 132, 341
* (asterisk), 81, 87
\ (backslash), 464
: (colon), 514–515, 558
{}(curly braces), 118, 208–209, 220, 293, 371
" (double quotation mark), 462
= (equal sign), 85, 87, 120, 121, 132
! (exclamation point), 120, 121, 517, 522
/ (forward slash), 81, 87, 92, 132
< (left angle bracket), 78–79, 120, 132, 562
- (minus sign), 81, 87, 132
(number sign), 519
() (parentheses), 81, 93, 132
% (percent sign), 81, 87, 132
| (pipe symbol), 127–129, 132
+ (plus sign), 81, 87, 132, 429, 489
> (right angle bracket), 77, 120, 132, 374–375, 430, 462–463, 520, 562
' (single quotation mark), 60
[] (square brackets), 371, 443

A

abstraction, **552**
accumulating values stored in two-dimensional arrays, 434–435
accumulators, **211,** 211–217
actual argument, **281,** 281–282
Addison O'Reilly problem, 26–27, 28–34, 78
calculation statements, 86
`cout` object, 79–80
data types, 45
desk-checking, 31–34, 54, 88–89
initial values for variables, 62
IPO charts, 26, 28, 54, 76
output statement, 80
problem specification, 26–27, 28, 30, 54, 76
prompts, 80
revising, 54–55
source code, 92
variable declaration statements, 63, 76
addition assignment operator (+=), 87
addition operator (+), 81, 132
address-of operator (&), **341**
algorithms, **5**
basketball through hoop problem, 158–160
clock program, 256
coding, 52, 76–80
daily rental fee program, 165–170
daily specials problem, 254–255
desk-checking, 31–42, 88–90. *See also* desk-checking
Dr. N problem, 115
Pete's Pizzeria problem, 117
planning, 28–31
Sherri problem, 248, 249
voter eligibility program, 162–163
American Standard Code for Information Interchange (ASCII), **58,** 58–59
codes, 595–596
ampersand (&)
address-of operator, **341**
And operator (&&), 127–128, 132
And operator (&&), 127–128, 132
area or circumference program, 124–126, 130–132
flowchart, 125
argument(s), **134**
actual, **281,** 281–282
argument(s) (listed by name)
argumentList argument, 336
character argument, 487
count argument, 473, 481, 487
delimiterCharacter argument, 464, 469, 521
fileName argument, 514, 515, 516
insertString argument, 485, 486
mode argument, 514
numberOfCharacters argument, 469
parameterList argument, 292–293, 304, 335, 564, 566
replacementString argument, 484
returnDataType argument, 292, 293
searchString argument, 478, 479
stringVariableName argument, 464, 519
subscript argument, 473, 478, 481, 484
variableName argument, 519
argumentList argument, 336
arithmetic assignment statements, **87,** 87–88
arithmetic operators, 81–84, 132–133
arrays, **370**
one-dimensional. *See* one-dimensional arrays
populating, 371
two-dimensional. *See* two-dimensional arrays
arrays (listed by name)
`calories` array, 376–383, 385
`fees` array, 398
`grades` array, 428, 429, 431
`letters` array, 375
`numbers` array, 372–373
`nums` array, 399
`orders` array, 426
`pollResults` array, 386–388
`prices` array, 428, 431–432
`rates` array, 389–391
`scientists` array, 371
`types` array, 396–398
ASCII (American Standard Code for Information Interchange), **58,** 58–59
codes, 595–596
assemblers, **3,** 3–4
assembly languages, **3,** 3–4
`assign` function, **487**
assignment operator (=), **85**
assignment statements, **84,** 84–88
assigning data to one-dimensional arrays, 373–374
asterisk (*)
multiplication assignment operator (*=), 87, 132
multiplication operator, 81
attributes, **552**
average, calculating, 384–385

B

backslash (\), newline character (\n), 464
base class, **553**
basketball through hoop problem, 158–160
problem specification and algorithm, 158–160
behaviors, **552**
bill-paying problem, 24–25
bonus program, 483, 485

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

`bool` data type, 56, 60
Boolean operators, **126,** 126–129
`break` statement, **176,** 176–177
bubble sort, **399,** 399–401
bugs, **90**
built-in functions, **280**

C

C++ instructions
  calories program, 377
  commission program, 225
  eBook Collection program, 523–525
  guessing game program, 289
  hypotenuse program, 283–285
  Jenko Booksellers program, 434–435
  motorcycle club program, 396–397
  searching an array, 386–387
  stock price program, 213–214, 217–218
  tip program, 223
  total payroll program, 221
  Wilson Company program, 436–438
`calcArea` method, 557, 559, 566, 567
calculation statements, Addison O'Reilly problem, 86
calling functions, 293–297
`calories` array, 376–383, 385
calories program, 376–382
  C++ instructions, 377
  desk-checking, 377–382
  problem specification and IPO chart, 376
camel case, **53**
car depreciation program, 258–261
  flowchart, 261
  problem specification, 258–260
case
  converting to uppercase or lowercase, 133–134
  sensitivity of memory location names, 53
`case` clauses, 175–177
Chapton Company program, problem specification, 432–433
`char` data type, 56
character(s), **56**
  converting to uppercase or lowercase, 133–134
character argument, 487
character literal constants, **60**
`cin` object, **77,** 77–78, 512, 513
`cin` statement, 300, 342
  one-dimensional arrays, 379, 387
  passing variables by reference, 342
  `string` variables, 467
  two-dimensional arrays, 431
class(es), **552**
  base, **553**
  containing overloaded methods, 566–569
  containing parameterized constructors, 562–564
  containing private data members and public member methods, 557–562
  defining, 553–555
  derived, **553**
  reusing, 565–566
class definitions, **553,** 553–555
`class` statement, **555,** 557
clock program, 255–258
  algorithm, code, and sample run, 256
  desk-checking, 257
`close` function, **522,** 522–523
closing sequential access files, 522–523
coding, **2**
  algorithms, 52, 76–80
  nested selection structures, 163–165
  selection structures, 118–120
colon (:), scope resolution operator (::), 514–515, 558
comments, **92,** 92–93, 118, 209
commission program, 207–208, 224–226
  C++ instructions, 225
  flowchart, 207
  IPO chart information and C++ instructions, 210
  posttest loop, 253
  problem specification and IPO chart, 204–205, 225
Company Name program, 488–489, 490
company ratings program, 338–341
  desk-checking, 340
comparison, unnecessary, in conditions, 170–171
comparison operators, **120,** 120–121, 132–133
compilers, **4**
compound conditions, using instead of nested selection structure, 167–168
computer programs, **2**
concatenating strings, 489–490
concatenation operator (+), **489**
constants
  literal, **60,** 60–61
  named. *See* named constants
constructors, **558**
  default, **558**
  parameterized, 562–564, **563**
consuming the character, **464**
control structures, **5,** 5–9. *See also* repetition structures; selection structures; sequence structure
count argument, 473, 481, 487
counter(s), **211,** 211–217
counter-controlled loops, **217**
`cout` object, **78,** 78–80, 512, 513
`cout` statement, 300, 303, 342
  nested loops, 257
  one-dimensional arrays, 378–379, 387, 393, 395, 398, 405
  passing variables by reference, 342
  `string` variables, 467
  two-dimensional arrays, 431
curly braces ()
  function body, 293
  initializing arrays, 371
  statement blocks, 118, 208–209, 220
currency converter program, 388–391
  problem specification, IPO chart information, and C++ instructions, 389–390

D

daily rental fee program
  algorithm, 165–170
  problem specification, 166
daily specials problem, 254–255
  problem specification and algorithm, 254–255
data types
  Addison O'Reilly problem, 45
  fundamental, **55**
  selecting for memory locations, 55–60
  user-defined, **55**
data types (listed by name)
  `bool` data type, 56, 60
  `char` data type, 56
  `double` data type, 56–57, 60, 291
  `float` data type, 56–57, 60
  `int` data type, 56, 60
  `short` data type, 56, 60
  `string` data type, 55, 56, 60–61, 462
data validation, **129**
debugging, **90**
decision structure. *See* selection structures
decision symbol, **116**
declaration section, **555**
declaration statements, 62
  Addison O'Reilly problem, 76
declaring
  memory locations, 62–64
  named constants, 63–64
  one-dimensional arrays, 371–373
  `string` variables, 462
  two-dimensional arrays, 427–428
  variables, 62–63
decrementing, **212**
default constructors, **558**
delimiterCharacter argument, 464, 469, 521
demoted values, **61**
derived classes, **553**
desk-checking, **31,** 31–42
  Addison O'Reilly problem, 31–34, 54, 88–89

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

calories program, 377–382
clock program, 257
company ratings program, 340
gas mileage problem, 35
highest number program, 394–395
savings account program, 300–303
stock price program, 215–216, 219
tip program, 224
tips program, 342–344
total payroll program, 222
Wilson Company program, 440–442
directives, 93
displayArray function, 382, 383, 443–444
displayCompanyInfo function, 336
displayInfo function, 523
displaying
area or circumference of a circle, 124–125, 130–132
contents of one-dimensional arrays, 375–382
contents of two-dimensional arrays, 431–433
displayLine function, 336
displayRating function, 338–341
displayTotalSales function, 336
division assignment operator (/=), 87
division operator (/), 81, 132
do while loops, displaying contents of one-dimensional arrays, 375–376
do while statement, **252,** 252–254
double data type, 56–57, 60, 291
double quotation marks (“), empty strings, 462
Dr. N problem, 114–116
problem specification, 116
dual-alternative selection structures, **115**
duplicating characters within string variables, 487–489

E

eBook Collection program, 523–528
IPO chart information and C++ instructions, 523–525
elements, **371**
empty strings (“”), **61,** 462
encapsulation, **552**
endl stream manipulator, **79**
endless loops, **210**
eof function, **522**
equal sign (=)
arithmetic assignment operators, 87
assignment operator, **85**
equal to operator (==), 120, 121, 132
not equal to operator (!=), 120, 121, 132
equal to operator (==), 120, 121, 132
erase function, **481,** 481–482
errors
logic. *See* logic errors
runtime, 90, 428
syntax. *See* syntax errors
escape sequence, **464**
even integers program, 226–228
problem specification, 226–227
exclamation point (!)
not equal to operator (!=), 120, 132
Not logical operator, 517, 522
executable files, **91**
explicit type conversion, **83,** 83–84
exponentiation, **281**
exposed details, **552**
extended selection structures. *See* multiple-alternative selection structures
extraction operator (>>), **77**
getting string input from keyboard, 462–463
reading char and numeric data from files, 520
storing data in one-dimensional arrays, 374–375
storing data in two-dimensional arrays, 430

F

false path, **115**
fees array, 398
fields, **519**
file(s)
executable, **91**
header, **561**
input, **512**
object, **91**
output, **512**
sequential access. *See* sequential access files
source, **91**
text, **512**
file objects, creating, 512–513
fileName argument, 514, 515, 516
find function, **478,** 478–481
fixed stream manipulator, **135**
float data type, 56–57, 60
flowchart(s), **28,** 28–29
area or circumference program, 125
car depreciation program, 261
commission program, 207
savings account program, 297
stock price program, 215, 218
swapping numeric values program, 123
flowcharting
nested selection structures, 161–163
posttest loops, 250–252
pretest loops, 206–208
flowlines, **29**
for clause, 300
nested loops, 257
one-dimensional arrays, 378–382
for loops
displaying contents of one-dimensional arrays, 375
one-dimensional arrays, 374, 385, 387, 395
for statement, **219,** 219–229
formal parameters, **292**
formatting numeric output, 134–137
forward slash (/)
comments, 92
division assignment operator (/=), 87
division operator, 81, 132
function(s), **93,** 279–317
arguments, **134**
built-in, **280**
calling, 293–297
passing one-dimensional arrays to, 382–384
passing two-dimensional arrays to, 443–444
program-defined, **280**
value-returning. *See* value-returning functions
function(s) (listed by name)
assign function, **487**
close function, **522,** 522–523
displayArray function, 382, 383, 443–444
displayCompanyInfo function, 336
displayInfo function, 523
displayLine function, 336
displayRating function, 338–341, 339–340
displayTotalSales function, 336
eof function, **522**
erase function, **481,** 481–482
find function, **478,** 478–481
getAverage function, 384–385
getBalance function, 298–303, 304
getHighest function, 392–395
getline function, 463–464, **464,** 465–468, 521
getRandomNumber function, 291–292, 294
getRectangleArea function, 291, 292–293, 294–295
getSubtotal function, 291, 292, 295
getTips function, 341–344
ignore function, **468,** 468–471
insert function, **485,** 485–487
is_open function, **516,** 516–517
length function, **471,** 471–473, 475
main function. *See* main function
open function, **514,** 514–518
pow function, **281,** 281–282, 285
rand function, **285,** 287–287, 289
replace function, **484,** 484–485
saveInfo function, 523

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

sqrt function, **282,** 282–283, 285
srand function, **287,** 287–288
stod function, **482,** 482–483
stoi function, 483
substr function, **473,** 473–477, 477
time function, 288, **288**
tolower function, **133,** 133–134, 281
toupper function, **133,** 133–134, 281
verifyNumeric function, 475–477
function body, **93,** 293
function headers, **93**
function prototypes, **298,** 298–303
fundamental data types, **55**

G

game programs, problem specification, 212
gas mileage problem, 34–35
getAverage function, 384–385
getBalance function, 298–303, 304
getFormattedDate method, 556
getHighest function, 392–395
getline function, 463–464, **464,** 465–468, 521
getRandomNumber function, 291–292, 294
getRectangleArea function, 291, 292–293, 294–295
getSide method, 557, 559
getSubtotal function, 291, 292, 293, 295
getTips function, 341–344
global variables, **304**
grades array, 428, 429, 431
greater than operator (>), 120, 132
greater than or equal to operator (>=), 120, 132
gross pay program
problem specifications, 129
truth tables, 129–130
guessing game program, 289–290
problem specification, IPO chart information, and C++ instructions, 289

H

hand-tracing. *See* desk-checking
header files, **561**
hidden details, **552**
highest number program, 391–396
desk-checking, 394–395
problem specification, IPO chart information, and C++ instructions, 392
high-level languages, **4**
How To boxes, list, 599–601
hypotenuse program, 283–285
problem specification, IPO chart information; and C++ instructions, 283–285

I

IDE (Integrated Drive Environment), **91**
if clause, strings, 475
if statement
comments to mark end, 118
comparison operators, 120–121
how to use, 119–120
logical operators, 127–128
one-dimensional arrays, 401, 402
ifstream class, 512, 513
ifstream file object, 515, 522
ignore function, **468,** 468–471
implementation section, **555**
implicit type conversion, **61**
in arithmetic expressions, 82–83
inAlphabet object, 521
#include directive, 561–562
#include directives, **93,** 512
INCREASE named constant, 430
increment operator (++), 429
incrementing, **212**
indexes. *See* subscript(s)
inFile object, 520–521
infinite loops, **210**
inheritance, **553**
initializing, **60**
counters and accumulators, **211**
one-dimensional arrays, 371–373
string variables, 462
two-dimensional arrays, 427–428
variables, 60–62
input, **26**
input files, **512**
input statement, Addison O'Reilly problem, 78
input/output symbol, **29**
insert function, **485,** 485–487
inserting characters within string variables, 485–487
insertion operator (<<), **78,** 78–79
insertString argument, 485, 486
instances, **552**
instantiating, **552**
objects, 556–557
int data type, 56, 60
integers, **56**
internal memory, 52–53
data storage, 57–60
interpreters, **4**
invalid data, **34**
IPO, **26**
IPO (Input, Processing, and Output) charts, **26,** 52
Addison O'Reilly problem, 26, 28, 54, 76
calories program, 376
commission program, 204–205, 225
eBook Collection program, 523–525
gas mileage problem, 34
guessing game program, 289
hypotenuse program, 283–285
Jenko Booksellers program, 434–435
Martin Sports program, 332–334
motorcycle club program, 396–397
Primrose Auction House program, 464–465
searching an array, 386–387
Snowboard Shop problem, 173
stock price program, 213–214, 217–218
tip program, 223
total payroll program, 221
Wilson Company program, 436–438
is_open function, **516,** 516–517
istream class, 512
iterations, **9**

J

Jenko Booksellers program, 434–435
problem specification, IPO chart information, and C++ instructions, 434–435

K

keyboard, getting string input from, 462–464
keywords, **53**
keywords (listed by name)
list, 593
private keyword, 557
void keyword, 331

L

left angle bracket (<)
directives, 562
insertion operator (<<), 78–79
less than operator, 120, 132
less than or equal to operator, 120, 132
length function, **471,** 471–473, 475
less than operator (<), 120, 132
less than or equal to operator (<=), 120, 132
letters array, 375
lifetime, variables, **304**
linkers, **91**
literal constants, **60,** 60–61
local variables, **122,** 122–123, **304**
logic errors, **90,** 90–91
selection structures, 165–172
logic structures. *See* control structures; repetition structures; selection structures; sequence structure
logical operators, **126,** 126–129, 132–133
truth tables, 128
loop(s), **9.** *See also* repetition structures
endless (infinite), **210**
posttest. *See* posttest loops
pretest. *See* pretest loops
loop body, **203**

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

loop exit condition, **202**
looping condition, **202**

M

machine code, **3**
machine languages, **3**
`main` function, 280, 298, 300, 302, 304
  one-dimensional arrays, 393
  passing variables by value, 339
  strings, 476
Martin Sports program, 332–336
  problem specification, IPO chart information, and C++ instructions, 332–334
memory, internal, 52–53
memory locations
  declaring, 62–64
  selecting data types, 55–60
  selecting initial values, 60–62
  selecting names, 53–55
methods, **555**
  overloaded. *See* overloaded methods
  signature, **564**
methods (listed by name)
  `calcArea` method, 557, 559, 566, 567
  `getFormattedDate` method, 556
  `getSide` method, 557, 559
  `setDate` method, 555, 556
  `setSide` method, 557, 558–559, 560, 563, 566
  `Square` method, 557
minus sign (-)
  negation operator, 81, 132
  subtraction assignment operator (-=), 87
  subtraction operator, 81, 132
mnemonics, **3**
mode argument, 514
modulus assignment operator (%=), 87
modulus operator (%), **81,** 132
motorcycle club program, 396–398
  problem specification, IPO chart information, and C++ instructions, 396–397
multiple-alternative selection structures, **173,** 173–177
  `switch` statement, **174,** 174–177
multiplication assignment operator (*=), 87
multiplication operator (*), 81, 87, 132

N

named constants, **53**
  declaring, 63–64
negation operator (-), 81, 132
nested loops, **254,** 254–258
nested selection structures, **158,** 158–165
  coding, 163–165
  flowcharting, 161–163
  reversing outer and nested decisions, 169
newline character (\n), 464
not equal to operator (!=), 120, 121, 132
Not logical operator (!), **517,** 522
number(s)
  integers, **56**
  random, 285–288
  real, **56,** 56–57
number sign (#), separating fields, 519
numberOfCharacters argument, 469
`numbers` array, 372–373
numeric literal constants, **60**
numeric output, formatting, 134–137
numeric values, swapping, 121–123
`nums` array, 399

O

object(s), **552**
  instantiating, 556–557
object code, **91**
object file, **91**
object-oriented program(s), **4,** 4–5
object-oriented programming (OOP), **552**
`ofstream` class, 512, 513
`ofstream` file object, 515, 518, 522
one-dimensional arrays, 369–413, **370**
  accessing individual elements, 388–391
  calculating totals and averages, 384–385
  data entry, 373–375
  declaring and initializing, 371–373
  displaying contents, 375–382
  elements, 371
  finding highest value, 391–396
  parallel, 396–398
  passing to a function, 382–384
  searching, 385–388
  sorting data, 399–406
  storing data, 376–382
  subscripts, 370–371
`open` function, **514,** 514–518
opening sequential access files, 514–518
Or operator (||), 127–129, 132
order of precedence, 81, 132
`orders` array, 426
`ostream` class, 512
`outFile` object, 518–519
output, **26**
output files, **512**
output statement, Addison O'Reilly problem, 80
`outSales` object, **519**
overloaded methods, **566**
  classes containing, 566–569

P

parallel arrays, **396**
parameterized constructors, 562–564, **563**
  classes containing, 562–564
parameterList argument, 564, 566
  value-returning functions, 292–293, 304
  void functions, 336
parentheses (())
  functions, 93
  overriding order of precedence, 81, 132
Pascal case, **555**
passing
  one-dimensional arrays to functions, 382–384
  by reference, **295, 337,** 341–344
  two-dimensional arrays to a function, 443–444
  by value, **295, 337,** 338–341
percent sign (%)
  modulus assignment operator (%=), 87
  remainder operator, 81, 132
Pete's Pizzeria problem, problem specification and algorithms, 117
pipe symbol, Or operator (||), 127–129, 132
pizza slices program, 565–569
plus sign (+)
  addition assignment operator (+=), 87
  addition operator, 81, 132
  concatenation operator, **489**
  increment operator (++), 429
`pollResults` array, 386–388
polymorphism, **553**
populating the array, **371**
posttest loops, **204,** 248–254
  `do while` statement, 252–254
  flowcharting, 250–252
`pow` function, **281,** 281–282, 285
precedence, order of, 81, 132
pretest loops, **204,** 204–208
  counter-controlled, 217–219
  flowcharting, 206–208
  `for` statement, 219–229
  `while` statement, 208–211
`prices` array, 428, 431–432
priming read, **206**
Primrose Auction House program, 464–471
  problem specification and IPO chart, 464
private data, 555, 557–562
`private` keyword, 557
problem specifications
  Addison O'Reilly problem, 26–27, 28, 30, 54, 76
  basketball through hoop problem, 158–160
  calories program, 376
  car depreciation program, 258–260
  Chapton Company program, 432–433
  commission program, 204–205, 225
  daily rental fee program, 166
  daily specials problem, 254–255
  Dr. N problem, 116
  even integers program, 226–227
  game programs, 212

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

gas mileage problem, 34
gross pay program, 129
guessing game program, 289
hypotenuse program, 283–285
Jenko Booksellers program, 434–435
Martin Sports program, 332–334
motorcycle club program, 396–397
Pete's Pizzeria problem, 117
Primrose Auction House program, 464–465
searching an array, 386–387
Sherri problem, 248, 249
stock price program, 213–214, 217–218
tip program, 223
total payroll program, 221
voter eligibility program, 162–163
problem-solving process, 23–42, 75–101
coding the algorithm, 52, 76–80
desk-checking the algorithm, 31–42, 88–90. *See also* desk-checking
evaluating and modifying the program, 90–94
everyday problems, 24–25
planning the algorithm, 28–31
problem analysis, 26–27
steps, 26
procedure-oriented programs, **4**
process symbol, **29**
processing items, **29,** 29–30
program(s), **2**
object-oriented, **4,** 4–5
procedure-oriented, **4**
program-defined functions, **280**
program-defined value-returning functions, 291–293
program-defined void functions, 331–336
programmers, **2**
employment opportunities, 2–3
job, 2
programming, **2,** 2–3
programming languages, **2**
assembly, **3,** 3–4
high-level, **4,** 4–5
history, 3–5
machine, **3**
promoted values, **61**
prompts, **78**
Addison O'Reilly problem, 80
pseudocode, **28**
pseudo-random number generators, **285**
public members, 555, 557–562
referring to, 556–557
Pythagorean theorem, 283–285

R

`rand` function, **285,** 287–287, 289
`RAND_MAX` constant, **285,** 286
random numbers, 285–288
`RATE` names constant, 483
`rates` array, 389–391
reading from sequential access files, 519–522
real numbers, **56,** 56–57
Rearrange Name program, 479–481
records, **519**
reference, passing variables by, **295, 337,** 341–344
remainder operator (%), 81, 132
removing characters from `string` variables, 481–483
repetition structures, **8,** 8–9, 201–236, **202,** 247–268. *See also* loop(s); posttest loops; pretest loops
counters and accumulators, 211–217
`replace` function, **484,** 484–485
replacementString argument, 484
replacing characters in `string` variables, 484–485
`return` statement, **293,** 301, 303, 331
returnDataType argument, 292, 293
reusing classes, 565–566
right angle bracket (>)
directives, 562
extraction operator (>>), 77, 374–375, 430, 462–463, 520
greater than operator, 120, 132
greater than or equal to operator, 120, 132
runtime, **53**
runtime errors, 90, 428

S

Sahirah problem, problem specification, 202–203
`saveInfo` function, 523
savings account program, 295–297, 298–303
desk-checking, 300–303
flowcharts, 297
problem specification, IPO chart information, and C++ instructions, 296–297
scalar variables, **370**
`scientific` stream manipulator, **135**
`scientists` array, 371
scope, variables, **304**
scope resolution operator (::), **514,** 514–515, 558
searching
contents of `string` variables, 477–481
two-dimensional arrays, 436–442
searching an array, 385–388
problem specification, IPO chart information, and C++ instructions, 386–387
searchString argument, 478, 479
selection structures, **6,** 6–8, 113–146, **114,** 157–184
coding, 118–120
comparison operators, 120–121
converting to uppercase or lowercase, 133–134
displaying area or circumference, 124–125, 130–132
dual-alternative, **115**
flowcharting, 116–118
formatting numeric output, 134–137
logic errors, 165–172
logical operators, 126–129
multiple-alternative (extended), **173,** 173–177
nested. *See* nested selection structures
single-alternative, **115**
summary of operators, 132–133
swapping numeric values, 121–123
truth tables, 129–130
unnecessary, 169–170
sentinel values, **205**
sequence structure, 5–6, **6**
sequential access files, 511–542, **512**
closing, 522–523
opening, 514–518
reading information form, 519–522
testing for end, 522
writing data to, 518–519
`setDate` method, 555, 556
setprecision stream manipulator, **136**
`setSide` method, 557, 558–559, 560, 563, 566
Sherri problem, problem specification, illustrations, and algorithms, 248
`short` data type, 56, 60
short-circuit evaluation, **129**
signatures, methods, **564**
simple variables, **370**
single quotation mark ('), character literal constants, 60
single-alternative selection structures, **115**
Snowboard Shop problem, 173–177
IPO chart, 173
social media program, 385–388
Social Security Number program, 486–487
sorting, **399**
data stored in one-dimensional array, 399–406
source code, **91**
Addison O'Reilly problem, 92
source file, **91**
`sqrt` function, **282,** 282–283, 285
square brackets ([]), declaring arrays, 371, 443
`Square` class, 557, 558–562, 563, 565–566, 567
`Square` method, 557

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Square objects, 557, 567
srand function, **287,** 287–288
start/stop symbol, **29**
statement(s), **62**
  arithmetic assignment, **87,** 87–88
  assignment. *See* assignment statements
  calculation, Addison O'Reilly problem, 86
  declaration, 62, 76
  input, Addison O'Reilly problem, 78
  output, Addison O'Reilly problem, 80
statement(s) (listed by name)
  break statement, **176,** 176–177
  cin statement. *See* cin statement
  class statement, **555,** 557
  cout statement. *See* cout statement
  do while statement, **252,** 252–254
  if statement. *See* if statement
  return statement, **293,** 301, 303, 331
  for statement, **219,** 219–229
  switch statement, **174,** 174–177
  while statement. *See* while statement
statement blocks, **118,** 208–209, 220
static_cast operator, **83,** 83–84
stock price program, 213–216
  desk-checking, 215–216, 219
  flowchart, 215, 218
  problem specification, IPO chart information, and C++ instructions, 213–214, 217–218
stod function, **482,** 482–483
stoi function, 483
stream(s), **76**
stream manipulators, **79**
stream objects, **76,** 76–77
string(s), 461–500
string concatenation, **489,** 489–490
string data type, 55, 56, 60–61, 462
string literal constants, **60,** 60–61
string variables
  accessing characters, 473–477
  concatenating strings, 489–490
  declaring and initializing, 462
  determining number of characters, 471–473
  duplicating characters within, 487–489
  inserting characters within, 485–487
  Primrose Auction House program, 465–471
  removing characters, 481–483
  replacing characters, 484–485
  searching contents, 477–481
stringVariableName argument, 464, 519
sub variable, one-dimensional arrays, 378, 379–381
subscript(s), **370**
  one-dimensional arrays, 370–371
  two-dimensional arrays, 426–427
subscript argument, 473, 478, 481, 484
substr function, **473,** 473–477, 477
subtraction assignment operator (-=), 87
subtraction operator (-), 81, 132
swapping numeric values program, 121–123
  flowchart, 123
switch statement, **174,** 174–177
syntax, **62**
syntax errors, **90**
  common, list, 597–598

T

temp variable, 304
testing for end of sequential access files, 522
text, **52**
text files, **512**
time function, 288, **288**
tip program, 223–224
  desk-checking, 224
  problem specification, IPO chart information, and C++ instructions, 223
tips program, 341–344
  desk-checking, 342–344
tolower function, **133,** 133–134, 281
total(s), calculating, 384–385
total payroll program, 220–223
  desk-checking, 222
  problem specification, IPO chart information, and C++ instructions, 221
toupper function, **133,** 133–134, 281
true path, **115**
truth tables, **128,** 129–130
  logical operators, 128
two-dimensional arrays, 425–450, **426**
  accumulating values stored in, 434–435
  data entry, 428–431
  declaring and initializing, 427–428
  displaying contents, 431–433
  passing to a function, 443–444
  searching, 436–442
type casts, **83,** 83–84
type conversion, in arithmetic expressions, 82–83
types array, 396–398

U

update read, **206**
updating, **212,** 213
user-defined data types, **55**
using directives, **93**

V

valid data, **34**
value, passing variables by, **295, 337,** 338–341
value-returning functions, **281,** 281–285
  program-defined, 291–293
variable(s), **53**
  declaring, 62–63
  global, **304**
  initializing, 60–62
  local, **122,** 122–123, **304**
  passing by reference, **295, 337,** 341–344
  passing by value, **295, 337,** 338–341
  scalar, **370**
  scope and lifetime, 304
  simple, **370**
  string. *See* string variables
variableName argument, 519
verifyNumeric function, 475–477
void functions, **281,** 329–358, **330**
  passing variables to, 337–344
  program-defined, 331–336
void keyword, 331
voter eligibility program, 161–163
  problem specification and algorithm, 162–163

W

while clause
  one-dimensional arrays, 401, 402–405
  strings, 476–477
  two-dimensional arrays, 441–442
while loops
  one-dimensional arrays, 374–375, 398
  two-dimensional arrays, 431
while statement, **208,** 208–211, 339
  comments to mark end, 209
  passing variables by value, 339, 340
white-space character, **77**
Wilson Company program, 436–440
  desk-checking, 440–442
  problem specification, IPO chart information, and C++ instructions, 436–438
writing to sequential access files, 518–519

Z

ZIP Code program, 472–473, 475–477

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Copyright 2016 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).
Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.