Licensed to: iChapters User

C++ PROGRAMMING

Program Design Including Data Structures

D.S. MALIK

Fourth Edition

Licensed to: iChapters User

C++ Programming:

Program Design Including Data Structures

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

C++ Programming:

Program Design Including Data Structures

Fourth Edition

D.S. Malik

COURSE TECHNOLOGY
CENGAGE Learning™

Australia • Brazil • Japan • Korea • Mexico • Singapore • Spain • United Kingdom • United States

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

COURSE TECHNOLOGY
CENGAGE Learning™

C++ Programming: Program Design Including Data Structures, Fourth Edition
by D.S. Malik

Senior Product Manager: Alyssa Pratt

Acquisitions Editor: Amy Jollymore

Content Product Manager: Jill Braiewa

Marketing Manager: Bryant Chrzan

Editorial Assistant: Patrick Frank

Print Buyer: Julio Esperas

Art Director: Marissa Falco

Cover Designer: Lisa Kuhn, Curio Press, LLC

Compositor: Integra

Validation: Green Pen Quality Assurance

Proofreader: Green Pen Quality Assurance

Indexer: Liz Cunningham

© 2009 Course Technology, Cengage Learning

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 **cengage.com/permissions**
Further permissions questions can be emailed to **permissionrequest@cengage.com**

ISBN-13: 978-1-4239-0222-5

ISBN-10: 1-4239-0222-x

Course Technology
25 Thomson Place
Boston, MA 02210
USA

CENGAGE brain.com

Visual® C++ .NET and PowerPoint® are registered trademarks of the Microsoft Corporation; Pentium® is a registered trademark of Intel Corporation; IBM is a registered trademark of International Business Machines Corporation.

Disclaimer
Course Technology reserves the right to revise this publication and make changes from time to time in its content without notice.

The programs in this book are for instructional purposes only. They have been tested with care, but are not guaranteed for any particular intent beyond educational purposes. The authors and the publisher do not offer any warranties or representations, nor do they accept any liabilities with respect to the programs.

Cengage Learning is a leading provider of customized learning solutions with office locations around the globe, including Singapore, the United Kingdom, Australia, Mexico, Brazil, and Japan. Locate your local office at: **international.cengage.com/region**

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

For your lifelong learning solutions, visit **course.cengage.com**
Visit our corporate website at **cengage.com**

Printed in Canada
1 2 3 4 5 6 7 14 13 12 11 10 09 08

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

TO

My Parents

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

BRIEF CONTENTS

PREFACE xxxii

1. An Overview of Computers and Programming Languages 1
2. Basic Elements of C++ 29
3. Input/Output 115
4. Control Structures I (Selection) 167
5. Control Structures II (Repetition) 231
6. User-Defined Functions I 307
7. User-Defined Functions II 345
8. User-Defined Simple Data Types, Namespaces, and the `string` Type 415
9. Arrays and Strings 469
10. Records (`structs`) 545
11. Classes and Data Abstraction 583
12. Inheritance and Composition 659
13. Pointers, Classes, Virtual Functions, and Abstract Classes 727
14. Overloading and Templates 795
15. Exception Handling 885
16. Recursion 923
17. Linked Lists 959
18. Stacks and Queues 1053
19. Searching and Sorting Algorithms 1161
20. Binary Trees 1251

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

21. Graphs 1311

22. Standard Template Library (STL) 1351

APPENDIX A Reserved Words 1467

APPENDIX B Operator Precedence 1469

APPENDIX C Character Sets 1471

APPENDIX D Operator Overloading 1475

APPENDIX E Additional C++ Topics 1477

APPENDIX F Header Files 1499

APPENDIX G Memory Size on a System and Random Number Generator 1509

APPENDIX H References 1511

APPENDIX I Answers to Odd-Numbered Exercises 1513

INDEX 1539

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

TABLE OF CONTENTS

Preface xxxii

1 AN OVERVIEW OF COMPUTERS AND PROGRAMMING LANGUAGES 1

Introduction 2

A Brief Overview of the History of Computers 2

Elements of a Computer System 3

Hardware 4

Central Processing Unit 4

Main Memory 5

Secondary Storage 6

Input/Output Devices 6

Software 6

The Language of a Computer 6

The Evolution of Programming Languages 8

A C++ Program 10

Processing a C++ Program 12

Programming with the Problem Analysis–Coding–Execution Cycle 14

Programming Methodologies 22

Structured Programming 22

Object-Oriented Programming 22

ANSI/ISO Standard C++ 24

Quick Review 24

Exercises 26

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

2 **BASIC ELEMENTS OF C++** 29

The Basics of a C++ Program 30

Comments 32

Special Symbols 32

Reserved Words (Keywords) 33

Identifiers 33

Whitespaces 34

Data Types 34

Simple Data Types 35

Floating-Point Data Types 38

Arithmetic Operators and Operator Precedence 40

Order of Precedence 44

Expressions 45

Mixed Expressions 46

Type Conversion (Casting) 48

`string` **Type** 50

Input 51

Allocating Memory with Constants and Variables 52

Putting Data into Variables 54

Assignment Statement 54

Saving and Using the Value of an Expression 58

Declaring and Initializing Variables 59

Input (Read) Statement 60

Variable Initialization 63

Increment and Decrement Operators 67

Output 69

Preprocessor Directives 77

`namespace` and Using `cin` and `cout` in a Program 78

Using the `string` Data Type in a Program 78

Creating a C++ Program 79

Program Style and Form 83

Syntax 83

Use of Blanks 84

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Use of Semicolons, Brackets, and Commas 84
Semantics 84
Naming Identifiers 84
Prompt Lines 85
Documentation 86
Form and Style 86

More on Assignment Statements 88

Programming Example: Convert Length 89

Programming Example: Make Change 93

Quick Review 97

Exercises 99

Programming Exercises 107

3 INPUT/OUTPUT 115

I/O Streams and Standard I/O Devices 116
`cin` and the Extraction Operator >> 117

Using Predefined Functions in a Program 123
`cin` and the `get` Function 125
`cin` and the `ignore` Function 126
The `putback` and `peek` Functions 128
The Dot Notation Between I/O Stream Variables and I/O Functions: A Precaution 130

Input Failure 131
The `clear` Function 133

Output and Formatting Output 135
`setprecision` Manipulator 135
`fixed` Manipulator 136
`showpoint` Manipulator 136
`setw` 138

Additional Output Formatting Tools 141
`setfill` Manipulator 141
`left` and `right` Manipulators 143

Input/Output and the `string` Type 145

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

File Input/Output 146

Programming Example: Movie Ticket Sale and Donation to Charity 150

Programming Example: Student Grade 155

Quick Review 158

Exercises 160

Programming Exercises 163

4 **CONTROL STRUCTURES I (SELECTION)** 167

Control Structures 168

Relational Operators 169

Relational Operators and Simple Data Types 171

Comparing Floating-point Numbers for Equality 171

Comparing Characters 172

Relational Operators and the `string` Type 173

Logical (Boolean) Operators and Logical Expressions 175

Order of Precedence 177

Short-Circuit Evaluation 181

`int` Data Type and Logical (Boolean) Expressions 182

`bool` Data Type and Logical (Boolean) Expressions 183

Selection: `if` and `if...else` 184

One-Way Selection 185

Two-Way Selection 187

Compound (Block of) Statements 191

Multiple Selections: Nested `if` 192

Comparing `if...else` Statements with a Series of `if` Statements 196

Using Pseudocode to Develop, Test, and Debug a Program 196

Input Failure and the `if` Statement 199

Confusion Between the Equality Operator (==) and the Assignment Operator (=) 202

Conditional Operator (`?:`) 203

`switch` Structures 204

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Terminating a Program with the assert Function 211

Programming Example: Cable Company Billing 213

Quick Review 219

Exercises 220

Programming Exercises 225

5 **CONTROL STRUCTURES II (REPETITION)** 231

Why Is Repetition Needed? 232

while Looping (Repetition) Structure 233

Designing while loops 235

Case 1: Counter-Controlled while Loops 236

Case 2: Sentinel-Controlled while Loops 239

Case 3: Flag-Controlled while Loops 243

Case 4: EOF-Controlled while Loops 247

eof Function 248

More on Expressions in while Statements 249

Programming Example: Checking Account Balance 250

Programming Example: Fibonacci Number 259

for Looping (Repetition) Structure 264

Programming Example: Classifying Numbers 270

do...while Looping (Repetition) Structure 274

Choosing the Right Looping Structure 278

break and continue Statements 278

Nested Control Structures 281

Quick Review 288

Exercises 290

Programming Exercises 300

CENGAGE brain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

6 USER-DEFINED FUNCTIONS I 307

Predefined Functions 308

User-Defined Functions 311

Value-Returning Functions 312
- Syntax: Value-Returning Functions 314
- Syntax: Formal Parameter List 314
- Function Call 314
- Syntax: Actual Parameter List 314
- return Statement 315
- Syntax: return Statement 315
- Function Prototype 318
- Syntax: Function Prototype 319
- Flow of Execution 325

Programming Example: Largest Number 326

Programming Example: Cable Company 327

Quick Review 333

Exercises 335

Programming Exercises 340

7 USER-DEFINED FUNCTIONS II 345

Void Functions 346
- Void Functions without Parameters 346
- Void Functions with Parameters 349

Value Parameters 354

Reference Variables as Parameters 356

Value and Reference Parameters and Memory Allocation 360

Reference Parameters and Value-Returning Functions 370

Scope of an Identifier 370

Global Variables, Named Constants, and Side Effects 374

Static and Automatic Variables 376

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Function Overloading: An Introduction 378

Functions with Default Parameters 380

Programming Example: Classify Numbers 383

Programming Example: Data Comparison 388

Quick Review 398

Exercises 400

Programming Exercises 407

8 USER-DEFINED SIMPLE DATA TYPES, NAMESPACES, AND THE `string` TYPE **415**

Enumeration Type 416

Declaring Variables 418

Assignment 418

Operations on Enumeration Types 419

Relational Operators 419

Input/Output of Enumeration Types 420

Functions and Enumeration Types 422

Declaring Variables When Defining the Enumeration Type 424

Anonymous Data Types 424

`typedef` Statement 425

Programming Example: The Game of Rock, Paper, and Scissors 426

Namespaces 437

`string` **Type** 442

Additional `string` Operations 446

`length` Function 446

`size` Function 448

`find` Function 449

`substr` Function 452

`swap` Function 454

Programming Example: Pig Latin Strings 454

Quick Review 460

CENGAGE brain .com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Exercises 463

Programming Exercises 466

9 **ARRAYS AND STRINGS** 469

Arrays 471

Accessing Array Components 472

Processing One-Dimensional Arrays 475

Array Index Out of Bounds 479

Array Initialization During Declaration 480

Partial Initialization of Arrays During Declaration 480

Some Restrictions on Array Processing 481

Arrays as Parameters to Functions 482

Constant Arrays as Formal Parameters 483

Base Address of an Array and Array in Computer Memory 485

Functions Cannot Return a Value of the Type Array 488

Integral Data Type and Array Indices 491

Other Ways to Declare Arrays 491

C-strings (Character Arrays) 492

String Comparison 494

Reading and Writing Strings 496

String Input 496

String Output 497

Specifying Input/Output Files at Execution Time 498

`string` Type and Input/Output Files 498

Parallel Arrays 499

Two- and Multidimensional Arrays 500

Accessing Array Components 502

Two-Dimensional Array Initialization During Declaration 503

Two-Dimensional Arrays and Enumeration Types 503

Initialization 507

Print 507

Input 507

Sum by Row 508

Sum by Column 508

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Largest Element in Each Row and Each Column 508
Reversing Diagonal 509
Passing Two-Dimensional Arrays as Parameters to Functions 511
Arrays of Strings 514
Arrays of Strings and the `string` Type 514
Arrays of Strings and C-Strings (Character Arrays) 515
Another Way to Declare a Two-Dimensional Array 516
Multidimensional Arrays 517

Programming Example: Code Detection 518

Programming Example: Text Processing 525

Quick Review 532

Exercises 534

Programming Exercises 539

10 RECORDS (`struct`S) 545

Records (`struct`s) 546
Accessing `struct` Members 548
Assignment 550
Comparison (Relational Operators) 551
Input/Output 552
`struct` Variables and Functions 552
Arrays versus `struct`s 553
Arrays in `struct`s 554
`struct`s in Arrays 556
`struct`s within a `struct` 559

Programming Example: Sales Data Analysis 563

Quick Review 577

Exercises 577

Programming Exercises 579

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

11 CLASSES AND DATA ABSTRACTION 583

Classes 584

Unified Modeling Language Class Diagrams 587
Variable (Object) Declaration 588
Accessing Class Members 589
Built-in Operations on Classes 590
Assignment Operator and Classes 591
Class Scope 592
Functions and Classes 592
Reference Parameters and Class Objects (Variables) 592
Implementation of Member Functions 593
Accessor and Mutator Functions 598
Order of `public` and `private` Members of a Class 603
Constructors 604
Invoking a Constructor 606
Invoking the Default Constructor 606
Invoking a Constructor with Parameters 607
Constructors and Default Parameters 610
Classes and Constructors: A Precaution 612
Arrays of Class Objects (Variables) and Constructors 613
Destructors 615

Data Abstraction, Classes, and Abstract Data Types 616

A `struct` versus a `class` 618

Information Hiding 619

Executable Code 623

Static Members of a Class 627

Programming Example: Candy Machine 633

Quick Review 648

Exercises 650

Programming Exercises 655

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

12 INHERITANCE AND COMPOSITION 659

Inheritance 660

Redefining (Overriding) Member Functions of the Base Class 663

Constructors of Derived and Base Classes 670

Multiple Inclusions of a Header File 678

C++ Stream Classes 680

Protected Members of a Class 681

Inheritance as `public`, `protected`, or `private` 681

Composition 685

Object-Oriented Design (OOD) and Object-Oriented Programming (OOP) 690

Identifying Classes, Objects, and Operations 692

Programming Example: Grade Report 693

Quick Review 714

Exercises 715

Programming Exercises 721

13 POINTERS, CLASSES, VIRTUAL FUNCTIONS, AND ABSTRACT CLASSES 727

Pointer Data Type and Pointer Variables 728

Declaring Pointer Variables 728

Address of Operator (&) 729

Dereferencing Operator (*) 730

Classes, Structs, and Pointer Variables 736

Initializing Pointer Variables 739

Dynamic Variables 739

Operator `new` 740

Operator `delete` 741

Operations on Pointer Variables 743

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Dynamic Arrays 745
Functions and Pointers 748
Pointers and Function Return Values 748
Dynamic Two-Dimensional Arrays 749

Shallow versus Deep Copy and Pointers 752

Classes and Pointers: Some Peculiarities 754
Destructor 755
Assignment Operator 756
Copy Constructor 758

Inheritance, Pointers, and Virtual Functions 765
Classes and Virtual Destructors 772

Abstract Classes and Pure Virtual Functions 772

Address of Operator and Classes 780

Quick Review 783

Exercises 786

Programming Exercises 793

14 OVERLOADING AND TEMPLATES 795

Why Operator Overloading Is Needed 796

Operator Overloading 797
Syntax for Operator Functions 798
Overloading an Operator: Some Restrictions 798
Pointer **this** 799
Friend Functions of Classes 804
Operator Functions as Member Functions and Nonmember Functions 807
Overloading Binary Operators 810
Overloading the Stream Insertion (<<) and Extraction (>>) Operators 816
Overloading the Assignment Operator (=) 821
Overloading Unary Operators 829
Operator Overloading: Member versus Nonmember 835
Classes and Pointer Member Variables (Revisited) 836
Operator Overloading: One Final Word 836

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Programming Example: `clockType` 836

Programming Example: Complex Numbers 845

Overloading the Array Index (Subscript) Operator ([ ]) 850

Programming Example: `newString` 852

Function Overloading 858

Templates 859

Function Templates 859

Class Templates 861

Quick Review 869

Exercises 871

Programming Exercises 876

15 **EXCEPTION HANDLING** 885

Handling Exceptions within a Program 886

C++ Mechanisms of Exception Handling 890

`try`/`catch` Block 890

Using C++ Exception Classes 897

Creating Your Own Exception Classes 901

Rethrowing and Throwing an Exception 907

Exception Handling Techniques 912

Terminate the Program 912

Fix the Error and Continue 912

Log the Error and Continue 914

Stack Unwinding 914

Quick Review 918

Exercises 920

Programming Exercises 922

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

16 RECURSION 923

Recursive Definitions 924
Direct and Indirect Recursion 927
Infinite Recursion 927

Problem Solving Using Recursion 928
Tower of Hanoi: Analysis 939

Recursion or Iteration? 940

Programming Example: Converting a Number from Binary to Decimal 941

Programming Example: Converting a Number from Decimal to Binary 946

Quick Review 950

Exercises 951

Programming Exercises 954

17 LINKED LISTS 959

Linked Lists 960
Linked Lists: Some Properties 961
Building a Linked List 970

Linked List as an ADT 974
Structure of Linked List Nodes 976
Member Variables of the **class** `linkedListType` 976
Linked List Iterators 976
Print the List 983
Length of a List 983
Retrieve the Data of the First Node 984
Retrieve the Data of the Last Node 984
Begin and End 984
Copy the List 985
Destructor 986
Copy Constructor 986
Overloading the Assignment Operator 987

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Unordered Linked Lists 987
Search the List 988
Insert the First Node 989
Insert the Last Node 990
Header File of the Unordered Linked List 995

Ordered Linked Lists 996
Search the List 998
Insert a Node 998
Insert First and Insert Last 1003
Delete a Node 1004
Header File of the Ordered Linked List 1005

Print a Linked List in Reverse Order (Recursion Revisited) 1008
`printListReverse` 1010

Doubly Linked Lists 1011
Default Constructor 1014
`isEmptyList` 1014
Destroy the List 1014
Initialize the List 1015
Length of the List 1015
Print the List 1015
Reverse Print the List 1015
Search the List 1016
First and Last Elements 1016

Circular Linked Lists 1022

Programming Example: Video Store 1023

Quick Review 1043

Exercises 1043

Programming Exercises 1047

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

18 STACKS AND QUEUES 1053

Stacks 1054

Stack Operations 1056

Implementation of Stacks as Arrays 1058

Initialize Stack 1061

Empty Stack 1062

Full Stack 1062

Push 1062

Return the Top Element 1064

Pop 1064

Copy Stack 1066

Constructor and Destructor 1066

Copy Constructor 1067

Overloading the Assignment Operator (=) 1067

Stack Header File 1068

Programming Example: Highest GPA 1072

Linked Implementation of Stacks 1076

Default Constructor 1079

Empty Stack and Full Stack 1080

Initialize Stack 1080

Push 1081

Return the Top Element 1083

Pop 1083

Copy Stack 1085

Constructors and Destructors 1086

Overloading the Assignment Operator (=) 1086

Stack as Derived from the **class** unorderedLinkedList 1088

Application of Stacks: Postfix Expressions Calculator 1090

Main Algorithm 1095

Function evaluateExpression 1095

Function evaluateOpr 1097

Function discardExp 1099

Function printResult 1099

Removing Recursion: Nonrecursive Algorithm to Print a Linked List Backward 1102

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Queues 1109
Queue Operations 1109
Implementation of Queues as Arrays 1111
Linked Implementation of Queues 1122
Queue Derived from the **class** `unorderedLinkedListType` 1127

Application of Queues: Simulation 1128
Designing a Queuing System 1129
Customer 1130
Server 1133
Server List 1136
Waiting Customers Queue 1141
Main Program 1143

Quick Review 1148

Exercises 1149

Programming Exercises 1155

19 SEARCHING AND SORTING ALGORITHMS 1161

Searching and Sorting Algorithms 1162

Search Algorithms 1162
Sequential Search 1163
Binary Search 1165
Performance of Binary Search 1170
Binary Search Algorithm and the **class** `orderedArrayListType` 1173

Asymptotic Notation: Big-O Notation 1173
Lower Bound on Comparison-Based Search Algorithms 1182

Sorting Algorithms 1182

Sorting a List: Bubble Sort 1183
Analysis: Bubble Sort 1187
Bubble Sort Algorithm and the **class** `unorderedArrayListType` 1188

Selection Sort: Array-Based Lists 1189
Analysis: Selection Sort 1193

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Insertion Sort: Array-Based Lists 1193
Analysis: Insertion Sort 1200

Lower Bound on Comparison-Based Sort Algorithms 1201

Quick Sort: Array-Based Lists 1202
Analysis: Quick Sort 1209

Merge Sort: Linked List-Based Lists 1210
Divide 1212
Merge 1214
Analysis: Merge Sort 1218

Programming Example: Election Results 1220

Quick Review 1242

Exercises 1243

Programming Exercises 1246

20 BINARY TREES **1251**

Binary Trees 1252
Copy Tree 1259
Binary Tree Traversal 1260

Implementing Binary Trees 1264

Binary Search Trees 1273
Binary Search Tree: Analysis 1287

Nonrecursive Binary Tree Traversal Algorithms 1288
Nonrecursive Inorder Traversal 1288
Nonrecursive Preorder Traversal 1290
Nonrecursive Postorder Traversal 1291

Binary Tree Traversal and Functions as Parameters 1292

Programming Example: Video Store (Revisited) 1296

Quick Review 1305

Exercises 1307

Programming Exercises 1309

CENGAGE brain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

21 GRAPHS 1311

Introduction 1312

Graph Definitions and Notations 1313

Graph Representation 1316
  Adjacency Matrix 1316
  Adjacency Lists 1317

Operations on Graphs 1318

Graphs as ADTs 1319

Graph Traversals 1323
  Depth First Traversal 1323
  Breadth First Traversal 1325

Shortest Path Algorithm 1327
  Shortest Path 1329

Minimal Spanning Tree 1335

Quick Review 1345

Exercises 1347

Programming Exercises 1349

22 STANDARD TEMPLATE LIBRARY (STL) 1351

Components of the STL 1352
  Container Types 1353
  Sequence Containers 1353
  Sequence Container: `vector` 1353
  Member Functions Common to All Containers 1362
  Member Functions Common to Sequence Containers 1364
  The `copy` Algorithm 1365
  Sequence Container: `deque` 1369
  Sequence Container: `list` 1373

Iterators 1380
  Types of Iterators 1380
  Stream Iterators 1386

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Associative Containers 1386
Associative Containers: set and multiset 1387
Declaring set or multiset Associative Containers 1387
Item Insertion and Deletion from set/multiset 1389

Container Adapters 1393
Stack 1393
Queue 1395

Containers, Associated Header Files, and Iterator Support 1396

Algorithms 1396
STL Algorithm Classification 1398
Function Objects 1400
Insert Iterator 1406
STL Algorithms 1408
The Functions fill and fill_n 1408
The Functions generate and generate_n 1410
The Functions find, find_if, find_end, and find_first_of 1412
The Functions remove, remove_if, remove_copy, and remove_copy_if 1417
The Functions replace, replace_if, replace_copy, and replace_copy_if 1420
The Functions swap, iter_swap, and swap_ranges 1424
The Functions search, search_n, sort, and binary_search 1427
The Functions adjacent_find, merge, and inplace_merge 1431
The Functions reverse, reverse_copy, rotate, and rotate_copy 1435
The Functions count, count_if, max, max_element, min, min_element, and random_shuffle 1438
The Functions for_each and transform 1442
The Functions includes, set_intersection, set_union, set_difference, and set_symmetric_difference 1445
The Functions accumulate, adjacent_difference, inner_product, and partial_sum 1453

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Quick Review 1458

Exercises 1462

Programming Exercises 1464

APPENDIX A: RESERVED WORDS **1467**

APPENDIX B: OPERATOR PRECEDENCE **1469**

APPENDIX C: CHARACTER SETS **1471**

ASCII (American Standard Code for Information Interchange) 1471

EBCDIC (Extended Binary Coded Decimal Interchange Code) 1472

APPENDIX D: OPERATOR OVERLOADING **1475**

APPENDIX E: ADDITIONAL C++ TOPICS **1477**

Binary (Base 2) Representation of a Nonnegative Integer 1477

Converting a Base 10 Number to a Binary Number (Base 2) 1477

Converting a Binary Number (Base 2) to Base 10 1479

Converting a Binary Number (Base 2) to Octal (Base 8) and Hexdecimal (Base 16) 1480

More on File Input/Output 1482

Binary Files 1482

Random File Access 1488

Naming Conventions of Header Files in ANSI/ISO Standard C++ and Standard C++ 1496

APPENDIX F: HEADER FILES **1499**

Header File `cassert` **(**`assert.h`**)** 1499

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Header File `cctype` (`ctype.h`) 1500

Header File `cfloat` (`float.h`) 1501

Header File `climits` (`limits.h`) 1502

Header File `cmath` (`math.h`) 1504

Header File `cstddef` (`stddef.h`) 1505

Header File `cstring` (`string.h`) 1505

APPENDIX G: MEMORY SIZE ON A SYSTEM AND RANDOM NUMBER GENERATOR 1509

Random Number Generator 1510

APPENDIX H: REFERENCES 1511

APPENDIX I: ANSWERS TO ODD-NUMBERED EXERCISES 1513

Chapter 1 1513

Chapter 2 1515

Chapter 3 1517

Chapter 4 1518

Chapter 5 1519

Chapter 6 1520

Chapter 7 1521

Chapter 8 1522

Chapter 9 1522

Chapter 10 1523

Chapter 11 1524

Chapter 12 1526

Chapter 13 1527

CENGAGE brain .com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Chapter 14 1528
Chapter 15 1528
Chapter 16 1529
Chapter 17 1530
Chapter 18 1531
Chapter 19 1533
Chapter 20 1534
Chapter 21 1536
Chapter 22 1537

INDEX 1539

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

PREFACE

WELCOME TO THE FOURTH EDITION OF *C++ Programming: Program Design Including Data Structures*. Designed for a two semester (CS1 and CS2) C++ course, this text will provide a breath of fresh air to you and your students. The CS1 and CS2 courses serve as the cornerstone of the Computer Science curriculum. My primary goal is to motivate and excite all introductory programming students, regardless of their level. Motivation breeds excitement for learning. Motivation and excitement are critical factors that lead to the success of the programming student. This text is a culmination and development of my classroom notes throughout more than fifty semesters of teaching successful programming to Computer Science students.

C++ Programming: Program Design Including Data Structures started as a collection of brief examples, exercises, and lengthy programming examples to supplement the books that were in use at our university. It soon turned into a collection large enough to develop into a text. *The approach taken in this book is, in fact, driven by the students' demand for clarity and readability.* The material was written and rewritten until the students felt comfortable with it. Most of the examples in this book resulted from student interaction in the classroom.

CENGAGE brain

As with any profession, practice is essential. Cooking students practice their recipes. Budding violinists practice their scales. New programmers must practice solving problems and writing code. This is not a C++ cookbook. We do not simply list the C++ syntax followed by an example; we dissect the "why" behind all the concepts. The crucial question of "why?" is answered for every topic when first introduced. This technique offers a bridge to learning C++. Students must understand the "why?" in order to be motivated to learn.

Traditionally, a C++ programming neophyte needed a working knowledge of another programming language. This book assumes no prior programming experience. However, some adequate mathematics background such as college algebra is required.

Changes in the Fourth Edition

The fourth edition contains more than 20 new programming exercises in Chapters 2 to 12, and 14. Certain programming examples and programming exercises require input from a file. In the earlier editions the input file was assumed to be stored on the floppy disk in drive A. However, newer computers label drives differently. So in this edition, we assume that the input file is in the same directory (subdirectory) as the project containing the source code file. Furthermore, some parts of Chapters 1, 2, 4, and 5 are rewritten and updated. When a programming assignment is given, typically, students are required to include the author of the program and a brief explanation describing the purpose of the program. To emphasize this requirement, Programming Examples in each chapter are modified by including comments showing the author(s) of the programs and a brief explanation describing the purpose of the program.

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Approach

The programming language C++, which evolved from C, is no longer considered an industry-only language. Numerous colleges and universities use C++ for their first programming language course. C++ is a combination of structured programming and object-oriented programming, and this book addresses both types.

This book is intended for a two-semester course, CS1 and CS2, in Computer Science. The first 11 or 12 chapters can be covered in the first course and the remaining in the second course.

In July 1998, ANSI/ISO Standard C++ was officially approved. This book focuses on ANSI/ISO Standard C++. Even though the syntax of Standard C++ and ANSI/ISO Standard C++ is very similar, Chapter 8 discusses some of the features of ANSI/ISO Standard C++ that are not available in Standard C++.

Chapter 1 briefly reviews the history of computers and programming languages. The reader can quickly skim through this chapter and become familiar with some of the hardware components and the software parts of the computer. This chapter contains a section on processing a C++ program. This chapter also describes structured and object-oriented programming.

Chapter 2 discusses the basic elements of C++. After completing this chapter, students become familiar with the basics of C++ and are ready to write programs that are complicated enough to do some computations. Input/output is fundamental to any programming language. It is introduced early, in Chapter 3, and is covered in detail.

CENGAGEbrain.com

Chapters 4 and 5 introduce control structures to alter the sequential flow of execution. Chapters 6 and 7 study user-defined functions. It is recommended that readers with no prior programming background spend extra time on Chapters 6 and 7. Several examples are provided to help readers understand the concepts of parameter passing and the scope of an identifier.

Chapter 8 discusses the user-defined simple data type (enumeration type), the **`namespace`** mechanism of ANSI/ISO Standard C++, and the `string` type. The earlier versions of C did not include the enumeration type. Enumeration types have very limited use; their main purpose is to make the program readable. This book is organized such that readers can skip the section on enumeration types during the first reading without experiencing any discontinuity, and then later go through this section.

Chapter 9 discusses arrays in detail. Chapter 10 introduces records (**`struct`**s). The introduction of **`struct`**s in this book is similar to C **`struct`**s. This chapter is optional; it is not a prerequisite for any of the remaining chapters.

Chapter 11 begins the study of object-oriented programming (OOP) and introduces classes. The first half of this chapter shows how classes are defined and used in a program. The second half of the chapter introduces abstract data types (ADTs). This chapter shows how classes in C++ are a natural way to implement ADTs. Chapter 12 continues with the fundamentals of object-oriented design (OOD) and OOP, and discusses inheritance and composition. It

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

explains how classes in C++ provide a natural mechanism for OOD and how C++ supports OOP. Chapter 12 also discusses how to find the objects in a given problem.

Chapter 13 studies pointers in detail. After introducing pointers and how to use them in a program, this chapter highlights the peculiarities of classes with pointer data members and how to avoid them. Moreover, this chapter also discusses how to create and work with dynamic two-dimensional arrays. Chapter 13 also discusses abstract classes and a type of polymorphism accomplished via virtual functions.

Chapter 14 continues the study of OOD and OOP. In particular, it studies polymorphism in C++. Chapter 14 specifically discusses two types of polymorphism—overloading and templates.

Chapter 15 discusses exception handling in detail. Chapter 16 introduces and discusses recursion. This is a stand-alone chapter, so it can be studied anytime after Chapter 10.

Chapters 17 and 18 are devoted to the study of data structures. Discussed in detail are linked lists in Chapter 17 and stacks and queues in Chapter 18. The programming code developed in these chapters is generic. These chapters effectively use the fundamentals of OOD.

Chapter 19 discusses various searching and sorting algorithms. In addition to showing how these algorithms work, it also provides relevant analysis and results concerning the performance of the algorithms. The algorithm analysis allows the user to decide which algorithm to use in a particular application. This chapter also includes several sorting algorithms. The instructor can decide which algorithms to cover.

CENGAGEbrain.com

Chapter 20 provides an introduction to binary trees. Various traversal algorithms, as well as the basic properties of binary trees, are discussed and illustrated. Special binary trees, called binary search trees, are introduced. Searching, as well as item insertion and deletion from a binary search tree, are described and illustrated. Chapter 20 also discusses nonrecursive binary tree traversal algorithms. Furthermore, to enhance the flexibility of traversal algorithms, it shows how to construct and pass functions as parameters to other functions. This chapter also discusses AVL (height balanced) trees in detail. Due to text length considerations, discussion on AVL trees is provided as a separate section and is available on the Web site accompanying this book.

Graph algorithms are discussed in Chapter 21. After introducing the basic graph theory terminology, the representation of graphs in computer memory is discussed. This chapter also discusses graph traversal algorithms, the shortest path algorithm, and the minimal spanning tree algorithm. Topological sort is also discussed in this chapter and is available on the Web site accompanying this book.

C++ is equipped with a powerful library—the Standard Template Library (STL)—of data structures and algorithms that can be used effectively in a wide variety of applications. Chapter 22 describes the STL in detail. After introducing the three basic components of the STL, it shows how sequence containers are used in a program. Special containers, such as stack and queue, are also discussed. The latter half of this chapter shows how various STL algorithms can be used in a program. This chapter is fairly long; depending on the availability of time, the

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

instructor can at least cover the sequence containers, iterators, the classes `stack` and `queue`, and certain algorithms.

Appendix A lists the reserved words in C++. Appendix B shows the precedence and associativity of the C++ operators. Appendix C lists the ASCII (American Standard Code for Information Interchange) and EBCDIC (Extended Binary Coded Decimal Interchange Code) character sets. Appendix D lists the C++ operators that can be overloaded.

Appendix E has three objectives. First, we discuss how to convert a number from decimal to binary and binary to decimal. We then discuss binary and random access files in detail. Finally, we describe the naming conventions of the header files in both ANSI/ISO Standard C++ and Standard C++. Appendix F discusses some of the most widely used library routines, and includes the names of the standard C++ header files. The programs in Appendix G show how to print the memory size for the built-in data types on your system as well as how to use a random number generator. Appendix H gives selected references for further study. Appendix I provides the answers to odd-numbered exercises in the book.

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

How to Use the Book

This book can be used in various ways. Figure 1 shows the dependency of the chapters.

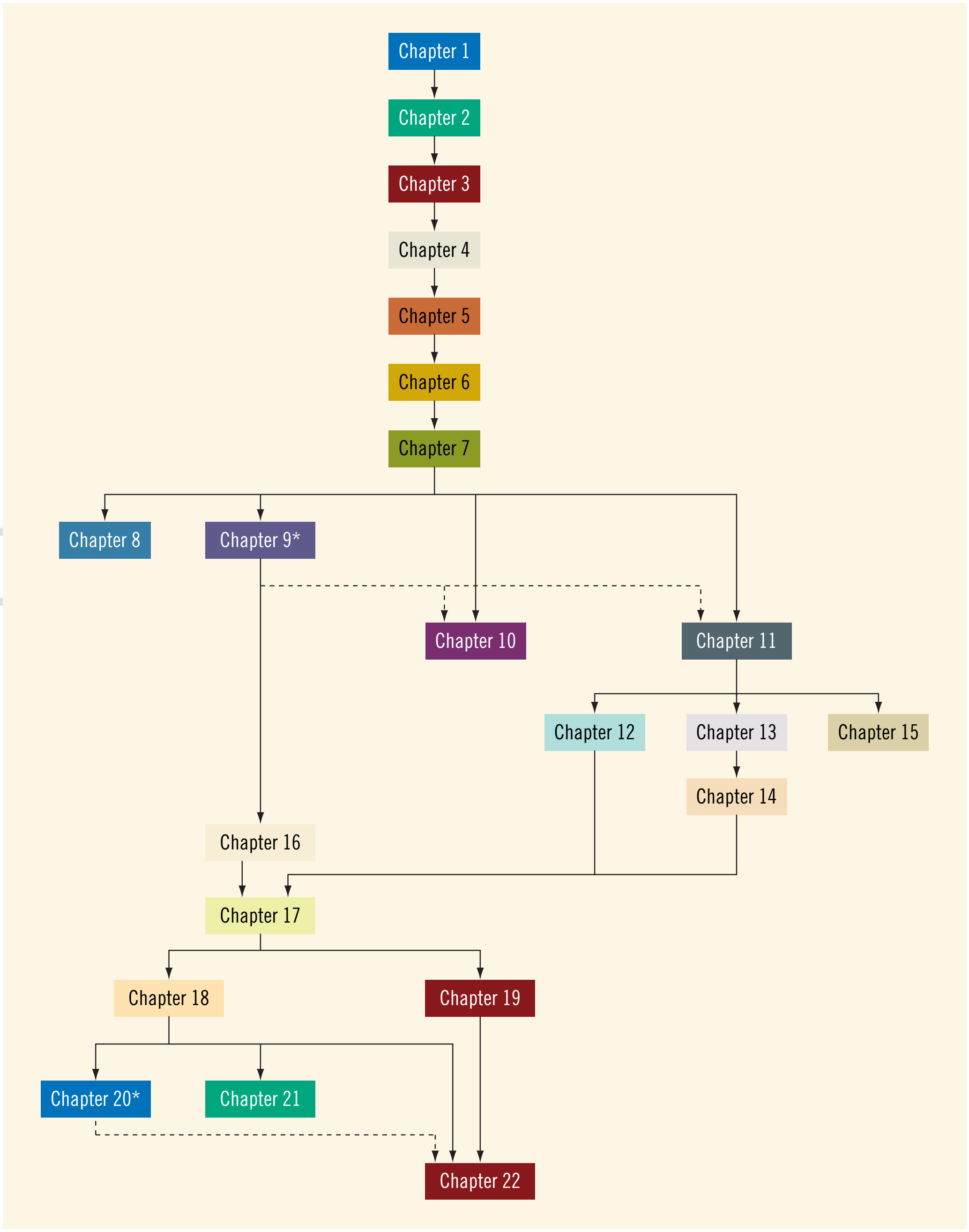


FIGURE 1 Chapter dependency diagram

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

In Figure 1, dotted lines mean that the preceding chapter is used in one of the sections of the chapter and is not necessarily a prerequisite for the next chapter. For example, Chapter 9 covers arrays in detail. In Chapters 10 and 11, we show the relationship between arrays and `struct`s and arrays and classes, respectively. However, if Chapter 11 is studied before Chapter 9, then the section dealing with arrays in Chapter 11 can be skipped without any discontinuation. This particular section can be studied after studying Chapter 9.

It is recommended that the first seven chapters be covered sequentially. After covering the first seven chapters, if the reader is interested in learning OOD and OOP early, then Chapter 11 can be studied right after Chapter 7. Chapter 8 can be studied anytime after Chapter 7. After studying the first seven chapters in sequence, some of the approaches are:

1. Study chapters in the sequence: 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22.
2. Study chapters in the sequence: 9, 11, 13, 14, 12, 16, 17, 18, 15, 19, 20, 21, 22.
3. Study chapters in the sequence: 11, 9, 13, 14, 12, 16, 17, 18, 15, 19, 20, 21, 22.

As the chapter dependency diagram shows, Chapters 18 and 19 can be covered in any sequence. However, typically, Chapters 18 and 19 are studied in sequence. Ideally, one should study Chapters 17, 18, 19, and 20 in sequence. Chapters 21 and 22 can be studied in any sequence.

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Features of the Book

From beginning to end, the concepts are introduced at a pace that is conducive to learning. *The writing style of this book is simple and straightforward, and it parallels the teaching style of a classroom*. Before introducing a key concept, we explain why certain elements are necessary. The concepts introduced are then described using examples and small programs.

Each chapter has two types of programs. The first type are small programs that are part of the numbered Examples (*e.g.*, Example 4-1), and are used to explain key concepts. In these examples, each line of the programming code is numbered. The program, illustrated through a Sample Run, is then explained line-by-line. The rationale behind each line is discussed in detail.

The Programming Examples form the backbone of the book and are designed to be methodical and user-friendly. Each Programming Example starts with a Problem Analysis and is followed by the Algorithm Design. Every step of the algorithm is then coded in C++. In addition to teaching problem-solving techniques, these detailed programs show the user how to implement concepts in an actual C++ program. I strongly recommend that students study the Programming Examples very carefully in order to effectively learn C++.

CENGAGEbrain.com

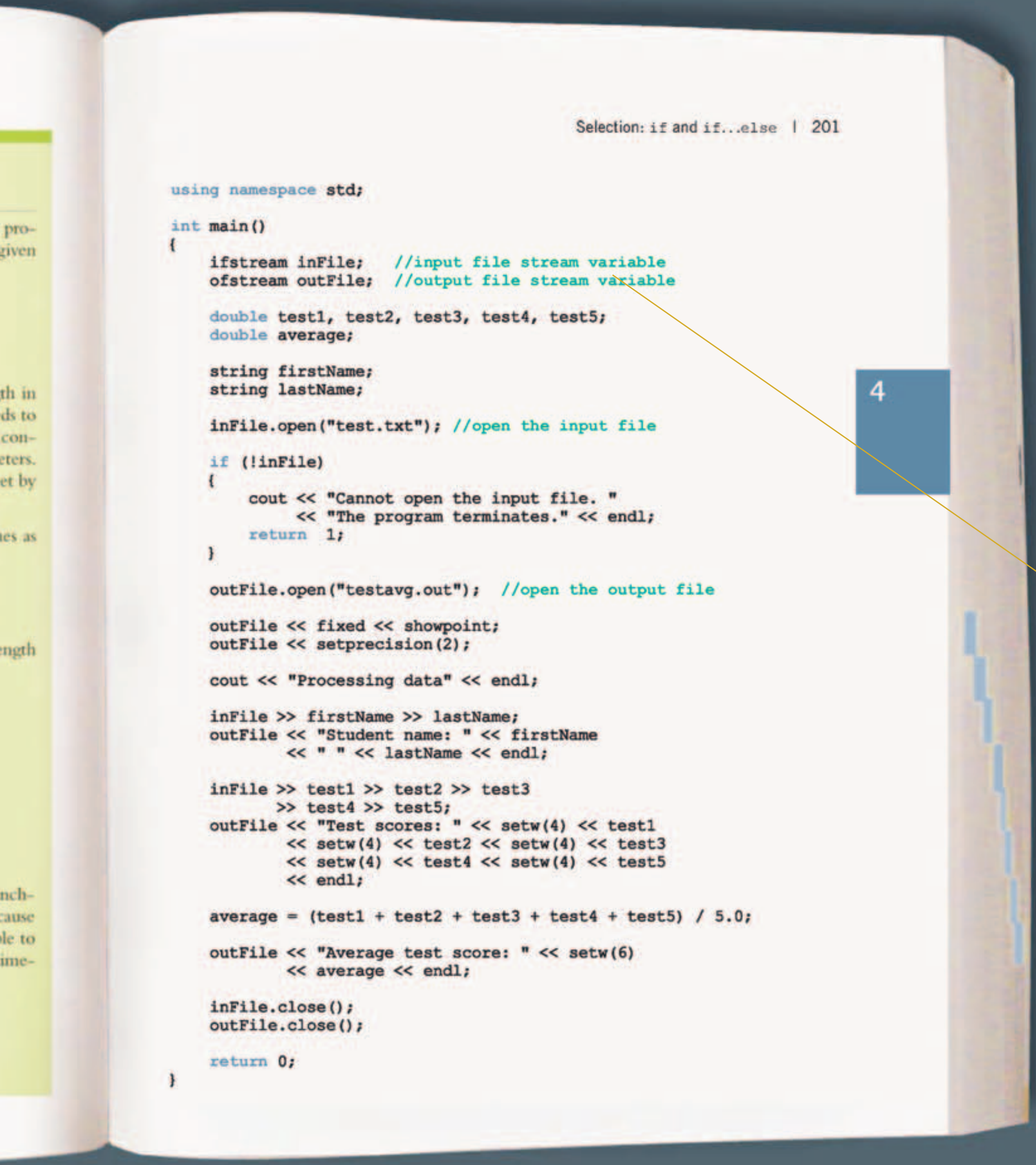
Quick Review sections at the end of each chapter reinforce learning. After reading the chapter, students can quickly walk through the highlights of the chapter and test themselves using the ensuing Exercises. Many readers refer to the Quick Review as an easy way to review the chapter before an exam.

The features of the text are clearly illustrated on the following pages.

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Features of the Book

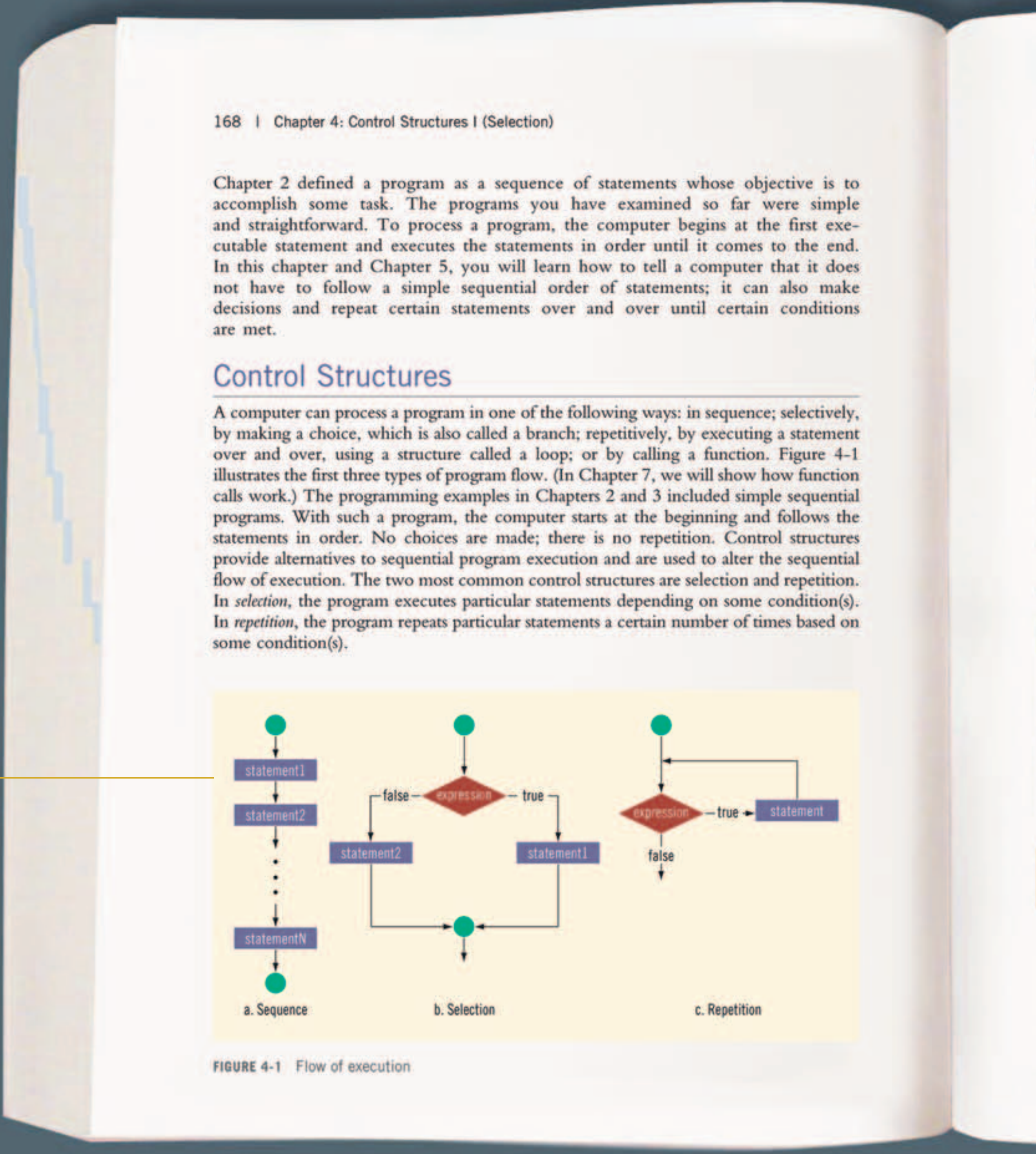

Selection: if and if...else | 201

```
using namespace std;

int main()
{
    ifstream inFile;   //input file stream variable
    ofstream outFile;  //output file stream variable

    double test1, test2, test3, test4, test5;
    double average;

    string firstName;
    string lastName;

    inFile.open("test.txt"); //open the input file

    if (!inFile)
    {
        cout << "Cannot open the input file. "
             << "The program terminates." << endl;
        return  1;
    }

    outFile.open("testavg.out");  //open the output file

    outFile << fixed << showpoint;
    outFile << setprecision(2);

    cout << "Processing data" << endl;

    inFile >> firstName >> lastName;
    outFile << "Student name: " << firstName
            << " " << lastName << endl;

    inFile >> test1 >> test2 >> test3
           >> test4 >> test5;
    outFile << "Test scores: " << setw(4) << test1
            << setw(4) << test2 << setw(4) << test3
            << setw(4) << test4 << setw(4) << test5
            << endl;

    average = (test1 + test2 + test3 + test4 + test5) / 5.0;

    outFile << "Average test score: " << setw(6)
            << average << endl;

    inFile.close();
    outFile.close();

    return 0;
}
```

Four-color interior design shows accurate C++ code and related comments.

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User



Chapter 2 defined a program as a sequence of statements whose objective is to accomplish some task. The programs you have examined so far were simple and straightforward. To process a program, the computer begins at the first executable statement and executes the statements in order until it comes to the end. In this chapter and Chapter 5, you will learn how to tell a computer that it does not have to follow a simple sequential order of statements; it can also make decisions and repeat certain statements over and over until certain conditions are met.

Control Structures

A computer can process a program in one of the following ways: in sequence; selectively, by making a choice, which is also called a branch; repetitively, by executing a statement over and over, using a structure called a loop; or by calling a function. Figure 4-1 illustrates the first three types of program flow. (In Chapter 7, we will show how function calls work.) The programming examples in Chapters 2 and 3 included simple sequential programs. With such a program, the computer starts at the beginning and follows the statements in order. No choices are made; there is no repetition. Control structures provide alternatives to sequential program execution and are used to alter the sequential flow of execution. The two most common control structures are selection and repetition. In *selection*, the program executes particular statements depending on some condition(s). In *repetition*, the program repeats particular statements a certain number of times based on some condition(s).

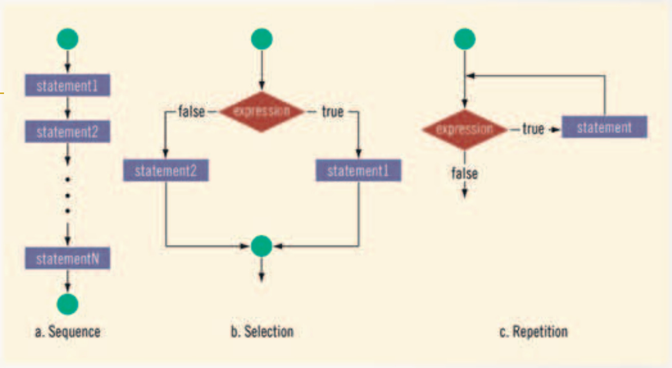


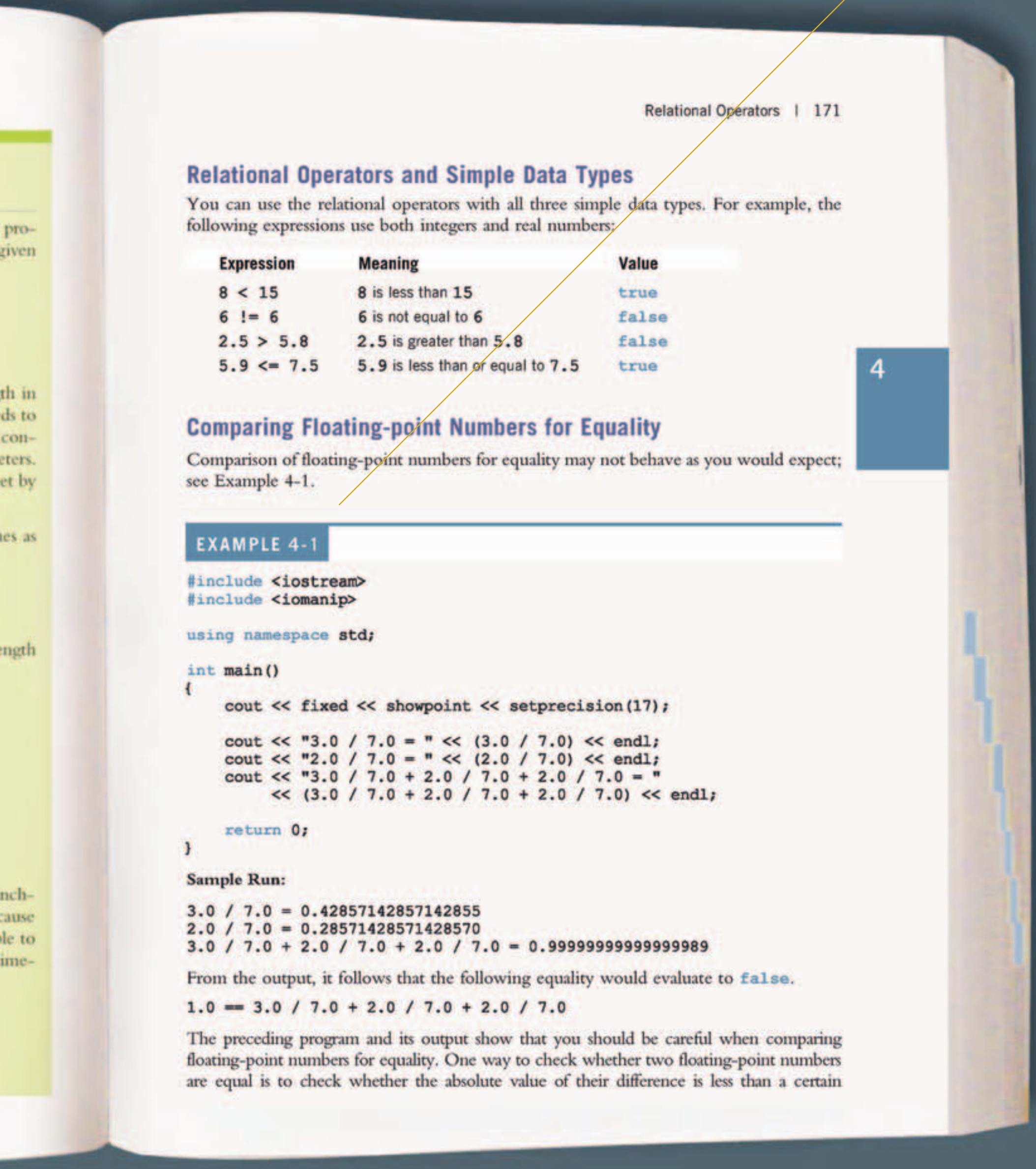
FIGURE 4-1 Flow of execution

More than 420 visual diagrams, both extensive and exhaustive, illustrate difficult concepts.

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Numbered *Examples* illustrate the key concepts with their relevant code. The programming code in these examples is followed by a Sample Run. An explanation then follows that describes what each line in the code does.



Relational Operators and Simple Data Types

You can use the relational operators with all three simple data types. For example, the following expressions use both integers and real numbers:

Expression	Meaning	Value
`8 < 15`	8 is less than 15	`true`
`6 != 6`	6 is not equal to 6	`false`
`2.5 > 5.8`	2.5 is greater than 5.8	`false`
`5.9 <= 7.5`	5.9 is less than or equal to 7.5	`true`

4

Comparing Floating-point Numbers for Equality

Comparison of floating-point numbers for equality may not behave as you would expect; see Example 4-1.

EXAMPLE 4-1

```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
    cout << fixed << showpoint << setprecision(17);

    cout << "3.0 / 7.0 = " << (3.0 / 7.0) << endl;
    cout << "2.0 / 7.0 = " << (2.0 / 7.0) << endl;
    cout << "3.0 / 7.0 + 2.0 / 7.0 + 2.0 / 7.0 = "
         << (3.0 / 7.0 + 2.0 / 7.0 + 2.0 / 7.0) << endl;

    return 0;
}
```

Sample Run:

```
3.0 / 7.0 = 0.42857142857142855
2.0 / 7.0 = 0.28571428571428570
3.0 / 7.0 + 2.0 / 7.0 + 2.0 / 7.0 = 0.99999999999999989
```

From the output, it follows that the following equality would evaluate to `false`.

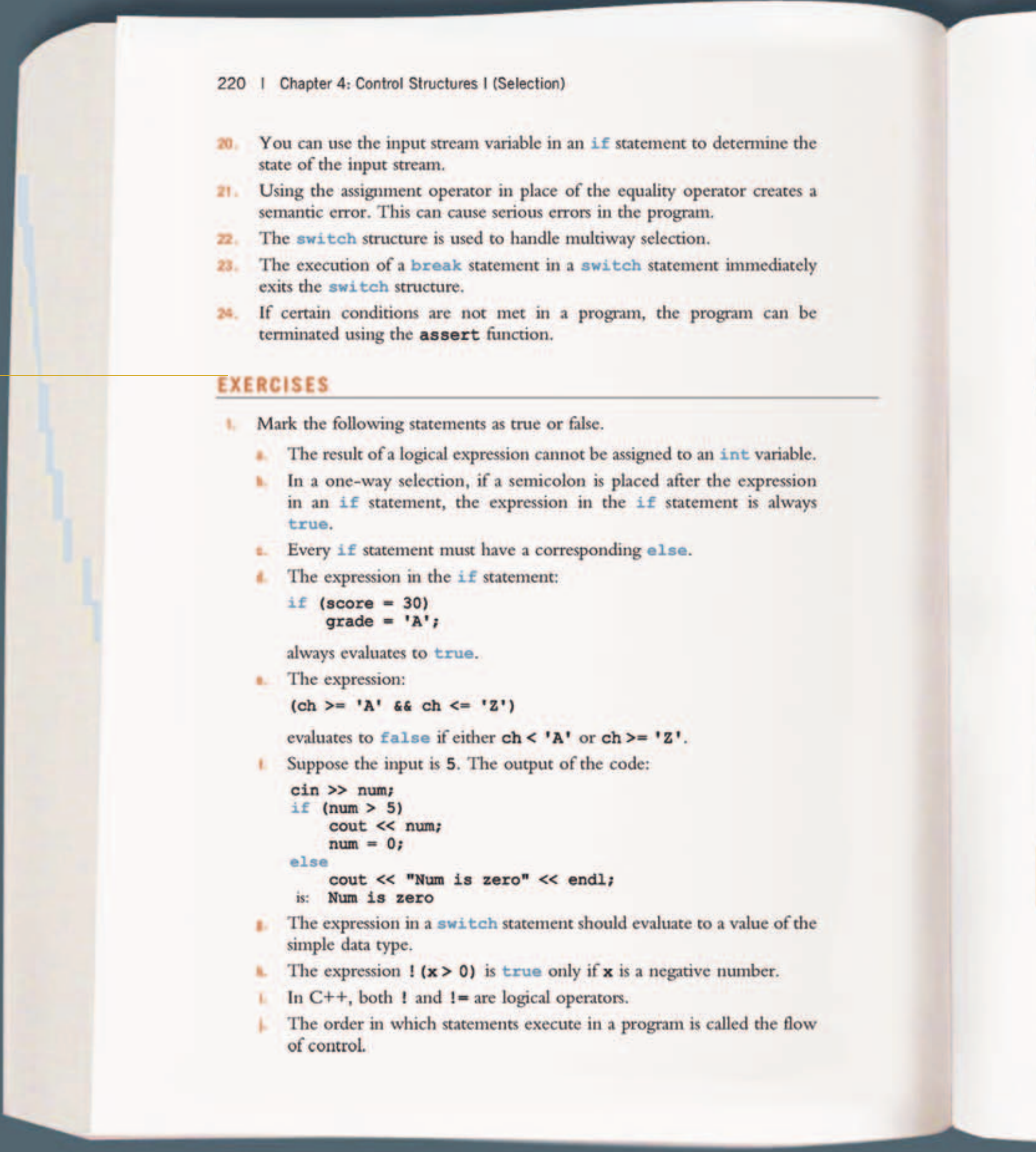
```
1.0 == 3.0 / 7.0 + 2.0 / 7.0 + 2.0 / 7.0
```

The preceding program and its output show that you should be careful when comparing floating-point numbers for equality. One way to check whether two floating-point numbers are equal is to check whether the absolute value of their difference is less than a certain

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Exercises further reinforce learning and ensure that students have, in fact, mastered the material.



20. You can use the input stream variable in an `if` statement to determine the state of the input stream.
21. Using the assignment operator in place of the equality operator creates a semantic error. This can cause serious errors in the program.
22. The `switch` structure is used to handle multiway selection.
23. The execution of a `break` statement in a `switch` statement immediately exits the `switch` structure.
24. If certain conditions are not met in a program, the program can be terminated using the **`assert`** function.

EXERCISES

1. Mark the following statements as true or false.
   a. The result of a logical expression cannot be assigned to an `int` variable.
   b. In a one-way selection, if a semicolon is placed after the expression in an `if` statement, the expression in the `if` statement is always `true`.
   c. Every `if` statement must have a corresponding `else`.
   d. The expression in the `if` statement:

```
if (score = 30)
    grade = 'A';
```

   always evaluates to `true`.
   e. The expression:

```
(ch >= 'A' && ch <= 'Z')
```

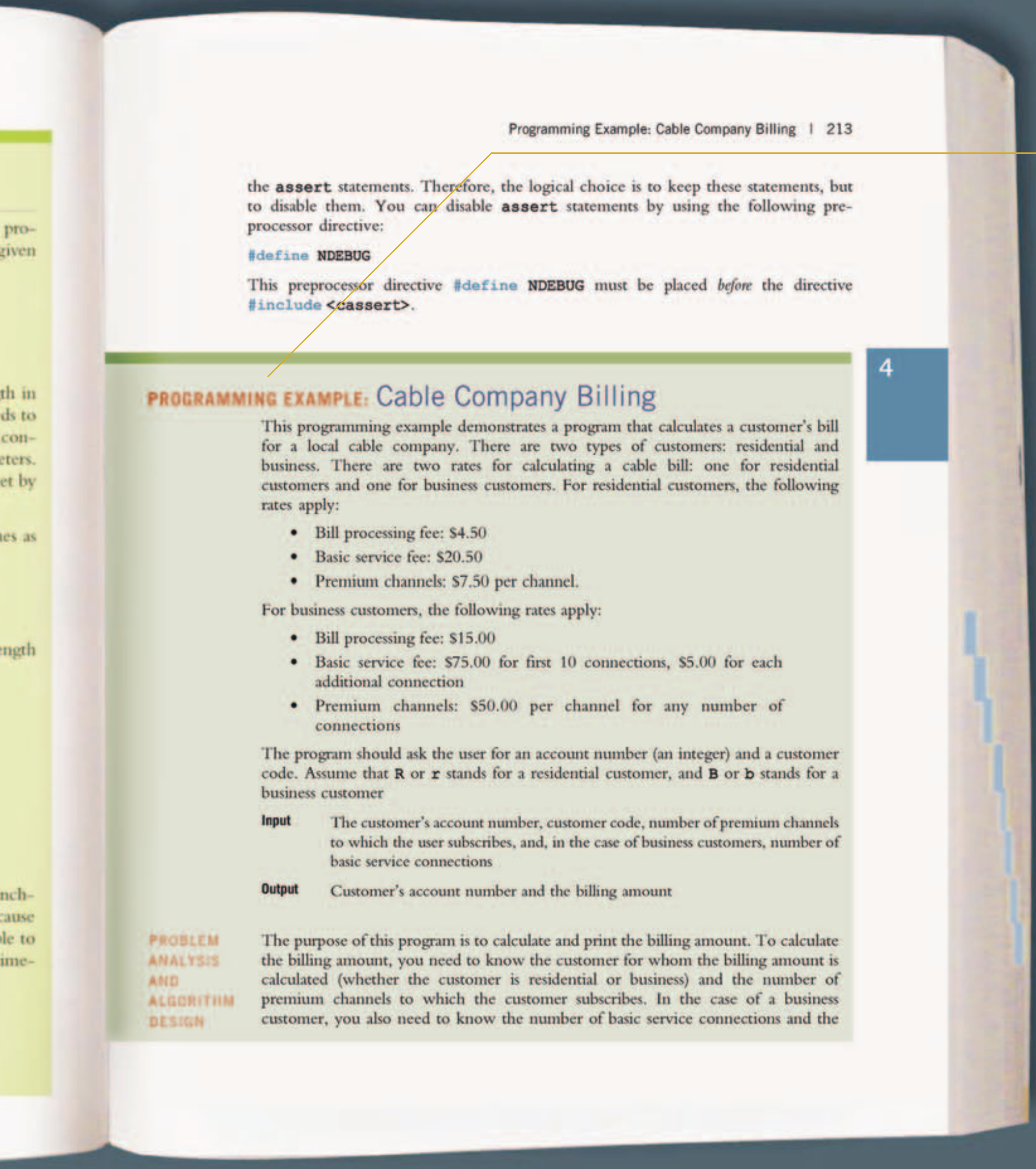
   evaluates to `false` if either `ch < 'A'` or `ch >= 'Z'`.
   f. Suppose the input is `5`. The output of the code:

```
cin >> num;
if (num > 5)
    cout << num;
    num = 0;
else
    cout << "Num is zero" << endl;
```

   is: `Num is zero`
   g. The expression in a `switch` statement should evaluate to a value of the simple data type.
   h. The expression `!(x > 0)` is `true` only if `x` is a negative number.
   i. In C++, both `!` and `!=` are logical operators.
   j. The order in which statements execute in a program is called the flow of control.

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User



the **`assert`** statements. Therefore, the logical choice is to keep these statements, but to disable them. You can disable **`assert`** statements by using the following preprocessor directive:

```
#define NDEBUG
```

This preprocessor directive `#define` **`NDEBUG`** must be placed *before* the directive `#include` **`<cassert>`**.

4

PROGRAMMING EXAMPLE: Cable Company Billing

This programming example demonstrates a program that calculates a customer's bill for a local cable company. There are two types of customers: residential and business. There are two rates for calculating a cable bill: one for residential customers and one for business customers. For residential customers, the following rates apply:

- Bill processing fee: $4.50
- Basic service fee: $20.50
- Premium channels: $7.50 per channel.

For business customers, the following rates apply:

- Bill processing fee: $15.00
- Basic service fee: $75.00 for first 10 connections, $5.00 for each additional connection
- Premium channels: $50.00 per channel for any number of connections

The program should ask the user for an account number (an integer) and a customer code. Assume that **`R`** or **`r`** stands for a residential customer, and **`B`** or **`b`** stands for a business customer

Input The customer's account number, customer code, number of premium channels to which the user subscribes, and, in the case of business customers, number of basic service connections

Output Customer's account number and the billing amount

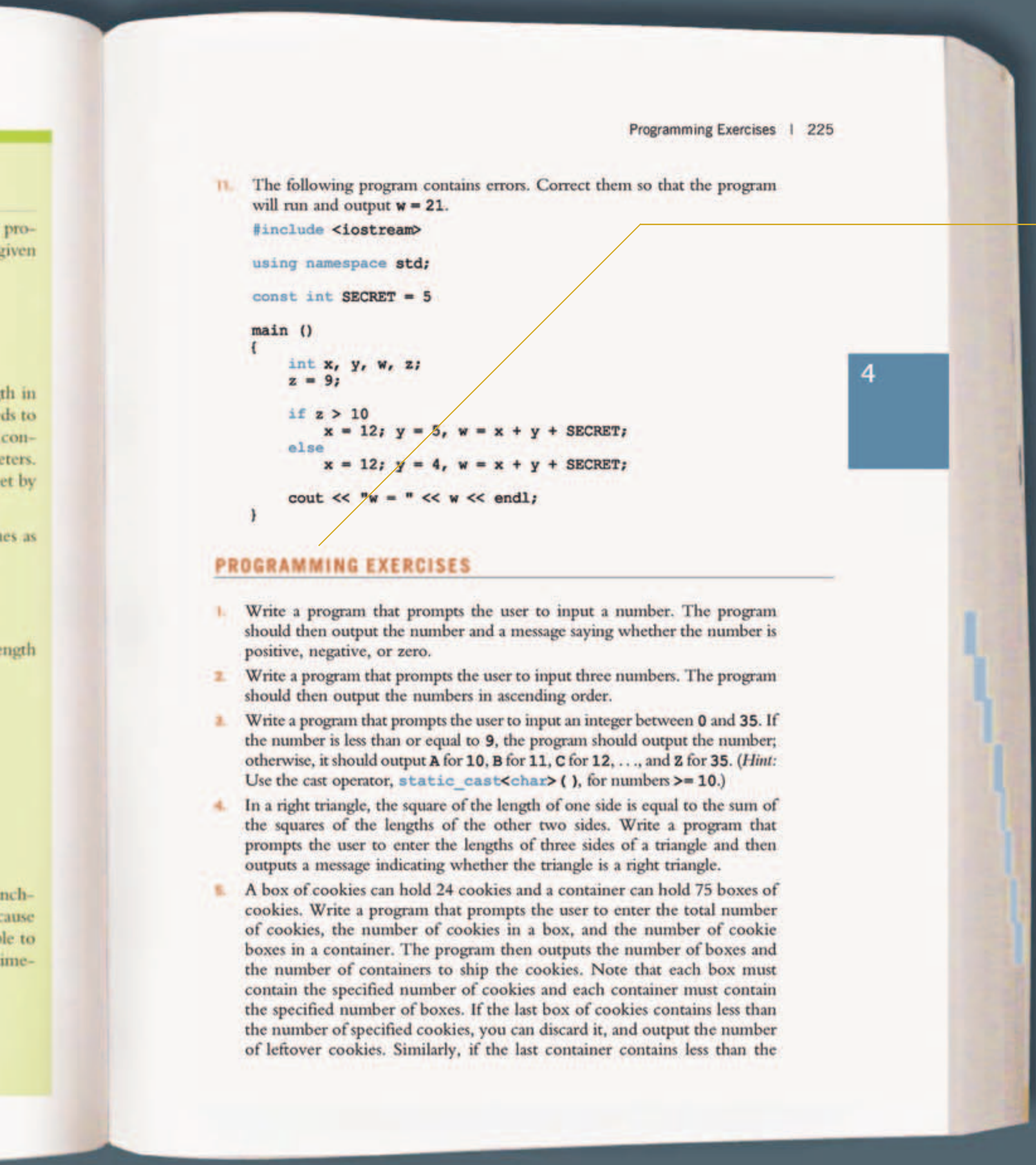
PROBLEM ANALYSIS AND ALGORITHM DESIGN

The purpose of this program is to calculate and print the billing amount. To calculate the billing amount, you need to know the customer for whom the billing amount is calculated (whether the customer is residential or business) and the number of premium channels to which the customer subscribes. In the case of a business customer, you also need to know the number of basic service connections and the

Programming Examples are complete programs featured in each chapter. These examples include the accurate, concrete stages of Input, Output, Problem Analysis and Algorithm Design, and a Complete Program Listing.

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User



11. The following program contains errors. Correct them so that the program will run and output w = 21.

```
#include <iostream>

using namespace std;

const int SECRET = 5

main ()
{
    int x, y, w, z;
    z = 9;

    if z > 10
        x = 12; y = 5, w = x + y + SECRET;
    else
        x = 12; y = 4, w = x + y + SECRET;

    cout << "w = " << w << endl;
}
```

PROGRAMMING EXERCISES

1. Write a program that prompts the user to input a number. The program should then output the number and a message saying whether the number is positive, negative, or zero.
2. Write a program that prompts the user to input three numbers. The program should then output the numbers in ascending order.
3. Write a program that prompts the user to input an integer between 0 and 35. If the number is less than or equal to 9, the program should output the number; otherwise, it should output A for 10, B for 11, C for 12, . . ., and Z for 35. (*Hint:* Use the cast operator, static_cast<char>( ), for numbers >= 10.)
4. In a right triangle, the square of the length of one side is equal to the sum of the squares of the lengths of the other two sides. Write a program that prompts the user to enter the lengths of three sides of a triangle and then outputs a message indicating whether the triangle is a right triangle.
5. A box of cookies can hold 24 cookies and a container can hold 75 boxes of cookies. Write a program that prompts the user to enter the total number of cookies, the number of cookies in a box, and the number of cookie boxes in a container. The program then outputs the number of boxes and the number of containers to ship the cookies. Note that each box must contain the specified number of cookies and each container must contain the specified number of boxes. If the last box of cookies contains less than the number of specified cookies, you can discard it, and output the number of leftover cookies. Similarly, if the last container contains less than the

Programming Exercises challenge students to write C++ programs with a specified outcome.

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Supplemental Resources

The following supplemental materials are available when this book is used in a classroom setting. All instructor materials as outlined below are available on a single CD-ROM.

Electronic Instructor's Manual

The Instructor's Manual that accompanies this textbook includes:

- Additional instructional material to assist in class preparation, including suggestions for lecture topics.
- Solutions to all the end-of-chapter materials, including the Programming Exercises.

ExamView®

CENGAGEbrain.com

This textbook is accompanied by ExamView, a powerful testing software package that allows instructors to create and administer printed, computer (LAN-based), and Internet exams. ExamView includes hundreds of questions that correspond to the topics covered in this text, enabling students to generate detailed study guides that include page references for further review. These computer-based and Internet testing components allow students to take exams at their computers, and save the instructor time because each exam is graded automatically.

PowerPoint Presentations

This book comes with Microsoft PowerPoint slides for each chapter. These are included as a teaching aid for classroom presentations, either to make available to students on the network for chapter review, or to be printed for classroom distribution. Instructors can add their own slides for additional topics that they introduce to the class.

Distance Learning

Course Technology Cengage Learning is proud to present online courses in WebCT and Blackboard to provide the most complete and dynamic learning experience possible. For more information on how to bring distance learning to your course, contact your local Course Technology Cengage Learning sales representative.

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Source Code

The source code, in ANSI/ISO Standard C++, is available at www.course.com, and is also available on the Teaching Tools CD-ROM. The input files needed to run some of the programs are also included with the source code.

Solution Files

The solution files for all programming exercises, in ANSI/ISO C++, are available at www.course.com, and are also available on the Teaching Tools CD-ROM. The input files needed to run some of the programming exercises are also included with the solution files.

Student Online Companion

This robust Web site, accessible at www.course.com/malik/cpp, offers students a plethora of review and self-assessment options. Each chapter includes a Concepts Review, Chapter Summary, Key Terms, Self-Tests, and Assignments. In addition, the Online Companion features related Web links, source code for all chapters, and compiler tutorials.

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

ACKNOWLEDGEMENTS

There are many people that I must thank who, one way or another, contributed to the success of this book. First, I would like to thank all the students who, during the preparation, were spontaneous in telling me if certain portions needed to be reworded for better understanding and clearer reading. Next, I would like to thank those who e-mailed numerous comments to improve upon the third edition. I am thankful to Professors S.C. Cheng, John N. Mordeson, and Vasant Raval for constantly supporting this project. I must thank Lee I. Fenicle, Director, Office of Technology Transfer, Creighton University, for his involvement, support, and for providing encouraging words when I needed them. I am also very grateful to the reviewers who reviewed earlier versions of this book and offered many critical suggestions on how to improve it.

I would like to thank the reviewers of the proposal package: William Barrett, San Jose State University; Frank Ducrest, University of Louisiana at Lafayette; Val Manes, South Dakota School of Mines and Technology; Kurt Schmidt, Drexel University; Cassandra Thomas, Tuskegee University. The reviewers will recognize that their criticisms have not been overlooked and, in fact, made this a better book.

All this would not have been possible without the careful planning of Senior Product Manager Alyssa Pratt. I extend my sincere thanks to Alyssa, as well as to Content Project Manager, Jill Braiewa. I also thank Tintu Thomas of Integra Software Services for assisting us in keeping the project on schedule and Green Pen Quality Assurance for carefully testing the code.

This book is dedicated to my parents, who I thank for their blessings.

Finally, I am thankful for the support of my wife Sadhana and especially my daughter Shelly. They cheered me up whenever I was overwhelmed during the writing of this book. I welcome any comments concerning the text. Comments may be forwarded to the following e-mail address: malik@creighton.edu.

D. S. Malik

CENGAGE brain

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Appendix A
Reserved Words

`and`	`and_eq`	`asm`	`auto`
`bitand`	`bitor`	`bool`	`break`
`case`	`catch`	`char`	`class`
`compl`	`const`	`const_cast`	`continue`
`default`	`delete`	`do`	`double`
`dynamic_cast`	`else`	`enum`	`explicit`
`export`	`extern`	`false`	`float`
`for`	`friend`	`goto`	`if`
`include`	`inline`	`int`	`long`
`mutable`	`namespace`	`new`	`not`
`not_eq`	`operator`	`or`	`or_eq`
`private`	`protected`	`public`	`register`
`reinterpret_cast`	`return`	`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`	

CENGAGE brain .com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Appendix B
Operator Precedence

The following table shows the precedence (highest to lowest) and associativity of the operators in C++.

Operator	Associativity
:: (binary scope resolution)	Left to right
:: (unary scope resolution)	Right to left
()	Left to right
[] -> .	Left to right
++ -- (as postfix operators)	Right to left
typeid dynamic_cast	Right to left
static_cast const_cast	Right to left
reinterpret_cast	Right to left
++ -- (as prefix operators) ! + (unary) - (unary)	Right to left
~ & (address of) * (dereference)	Right to left
new delete sizeof	Right to left
->* -- .*	Left to right
* / %	Left to right
+ -	Left to right
<< >>	Left to right
< <= > >=	Left to right
== !=	Left to right
&	Left to right
^	Left to right
|	Left to right
&&	Left to right

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Operator	Associativity
`||`	Left to right
`?:`	Right to left
`=` `+=` `-=` `*=` `/=` `%=`	Right to left
`<<=` `>>=` `&=` `|=` `^=`	Right to left
`throw`	Right to left
`,` (the sequencing operator)	Left to right

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Appendix C
Character Sets

ASCII (American Standard Code for Information Interchange)

The following table shows the ASCII character set.

ASCII

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	lf	vt	ff	cr	so	si	dle	dc1	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	<u>b</u>	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	del		

The numbers 0–12 in the first column specify the left digit(s), and the numbers 0–9 in the second row specify the right digit of each character in the ASCII data set. For example, the character in the row marked 6 (the number in the first column) and the column marked 5 (the number in the second row) is `A`. Therefore, the character at position 65 (which is the 66th character) is `A`. Moreover, the character <u>b</u> at position 32 represents the space character.

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

The first 32 characters, that is, the characters at positions 00–31 and at position 127 are nonprintable characters. The following table shows the abbreviations and meanings of these characters.

nul	null character	ff	form feed	can	cancel
soh	start of header	cr	carriage return	em	end of medium
stx	start of text	so	shift out	sub	substitute
etx	end of text	si	shift in	esc	escape
eot	end of transmission	dle	data link escape	fs	file separator
enq	enquiry	dc1	device control 1	gs	group separator
ack	acknowledge	dc2	device control 2	rs	record separator
bel	bell	dc3	device control 3	us	unit separator
bs	back space	dc4	device control 4	$\underline{b}$	space
ht	horizontal tab	nak	negative acknowledge	del	delete
lf	line feed	syn	synchronous idle		
vt	vertical tab	etb	end of transmitted block		

CENGAGE brain

EBCDIC (Extended Binary Coded Decimal Interchange Code)

The following table shows some of the characters in the EBCDIC character set.

EBCDIC										
	0	1	2	3	4	5	6	7	8	9
6					$\underline{b}$					
7						.	<	(	+	|
8	&									
9	!	$	*	)	;	¬	-	/		
10								,	%	_
11	>	?								
12		`	:	#	@	'	=	"		a
13	b	c	d	e	f	g	h	i		

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

EBCDIC										
14						j	k	l	m	n
15	o	p	q	r						
16		~	s	t	u	v	w	x	y	z
17										
18	[	]								
19				A	B	C	D	E	F	G
20	H	I								J
21	K	L	M	N	O	P	Q	R		
22							S	T	U	V
23	W	X	Y	Z						
24	0	1	2	3	4	5	6	7	8	9

CENGAGEbrain.com

The numbers 6–24 in the first column specify the left digit(s), and the numbers 0–9 in the second row specify the right digits of the characters in the EBCDIC data set. For example, the character in the row marked 19 (the number in the first column) and the column marked 3 (the number in the second row) is A. Therefore, the character at position 193 (which is the 194th character) is A. Moreover, the character b̲ at position 64 represents the space character. The preceding table does not show all the characters in the EBCDIC character set. In fact, the characters at positions 00–63 and 250–255 are nonprintable control characters.

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Appendix D Operator Overloading

The following table lists the operators that can be overloaded.

Operators that can be overloaded							
+	-	*	/	%	^	&	|
!	&&	||	=	==	<	<=	>
>=	!=	+=	-=	*=	/=	%=	^=
|=	&=	<<	>>	>>=	<<=	++	--
->*	,	->	[]	()	~	**new**	**delete**

CENGAGEbrain.com

The following table lists the operators that cannot be overloaded.

Operators that cannot be overloaded				
.	.*	::	?:	**sizeof**

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Appendix E
Additional C++ Topics

Binary (Base 2) Representation of a Nonnegative Integer

Converting a Base 10 Number to a Binary Number (Base 2)

Chapter 1 remarked that `A` is the 66th character in the ASCII character set, but its position is 65 because the position of the first character is 0. Furthermore, the binary number `1000001` is the binary representation of `65`. The number system that we use daily is called the **decimal number system** or **base 10 system**. The number system that the computer uses is called the **binary number system** or **base 2 system**. In this section, we describe how to find the binary representation of a nonnegative integer and vice versa.

Consider 65. Note that:

$$65 = 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

Similarly:

$$711 = 1 \times 2^9 + 0 \times 2^8 + 1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

In general, if m is a nonnegative integer, then m can be written as:

$$m = a_k \times 2^k + a_{k-1} \times 2^{k-1} + a_{k-2} \times 2^{k-2} + \cdots + a_1 \times 2^1 + a_0 \times 2^0,$$

for some nonnegative integer k, and where $a_i = 0$ or 1, for each $i = 0, 1, 2, \ldots, k$. The binary number $a_k a_{k-1} a_{k-2} \ldots a_1 a_0$ is called the **binary** or **base 2 representation** of m. In this case, we usually write:

$$m_{10} = (a_k a_{k-1} a_{k-2} \cdots a_1 a_0)_2$$

and say that m to the base 10 is $a_k a_{k-1} a_{k-2} \ldots a_1 a_0$ to the base 2.

For example, for the integer 65, $k = 6$, $a_6 = 1$, $a_5 = 0$, $a_4 = 0$, $a_3 = 0$, $a_2 = 0$, $a_1 = 0$, $a_0 = 1$. Thus, $a_6 a_5 a_4 a_3 a_2 a_1 a_0 = 1000001$, so the binary representation of 65 is 1000001, that is:

$$65_{10} = (1000001)_2.$$

If no confusion arises, then we write $(1000001)_2$ as 1000001_2.

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

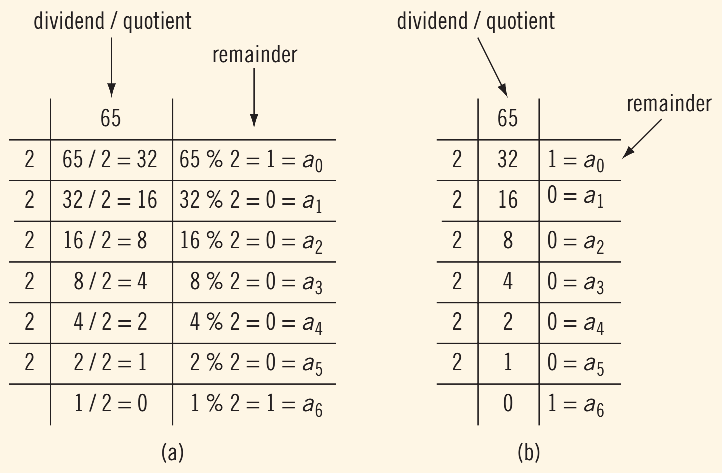
Similarly, for the number 711, $k = 9$, $a_9 = 1$, $a_8 = 0$, $a_7 = 1$, $a_6 = 1$, $a_5 = 0$, $a_4 = 0$, $a_3 = 0$, $a_2 = 1$, $a_1 = 1$, $a_0 = 1$. Thus:

$711_{10} = 1011000111_2$.

It follows that to find the binary representation of a nonnegative, we need to find the coefficients, which are 0 or 1, of various powers of 2. However, there is an easy algorithm, described next, that can be used to find the binary representation of a nonnegative integer. First, note that:

$$0_{10} = 0_2, 1_{10} = 1_2, 2_{10} = 10_2, 3_{10} = 11_2, 4_{10} = 100_2, 5_{10} = 101_2, 6_{10} = 110_2,$$
$$\text{and } 7_{10} = 111_2.$$

Let us consider the integer 65. Note that 65 / 2 = 32 and 65 % 2 = 1, where % is the mod operator. Next, 32 / 2 = 16, and 32 % 2 = 0, and so on. It can be shown that $a_0 = 65 \% 2 = 1$, $a_1 = 32 \% 2 = 0$, and so on. We can show this continuous division and obtaining the remainder with the help of Figure E-1.



	65	
2	65 / 2 = 32	65 % 2 = 1 = a_0
2	32 / 2 = 16	32 % 2 = 0 = a_1
2	16 / 2 = 8	16 % 2 = 0 = a_2
2	8 / 2 = 4	8 % 2 = 0 = a_3
2	4 / 2 = 2	4 % 2 = 0 = a_4
2	2 / 2 = 1	2 % 2 = 0 = a_5
	1 / 2 = 0	1 % 2 = 1 = a_6

	65	
2	32	1 = a_0
2	16	0 = a_1
2	8	0 = a_2
2	4	0 = a_3
2	2	0 = a_4
2	1	0 = a_5
	0	1 = a_6

FIGURE E-1 Determining the binary representation of 65

Notice that in Figure E-1(a), starting at the second row, the second column contains the quotient when the number in the previous row is divided by 2 and the third column contains the remainder of that division. For example, in the second row, 65 / 2 = 32, and 65 % 2 = 1. In the third row, 32 / 2 = 16 and 32 % 2 = 0, and so on. For each row, the number in the second column is divided by 2, the quotient is written in the next row, below the current row, and the remainder is written in the third column. When using a

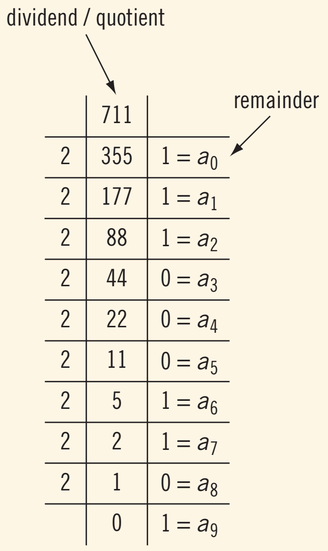
Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

figure, such as E-1, to find the binary representation of a nonnegative integer, typically, we show only the quotients and remainders as in Figure E-1(b). You can write the binary representation of the number starting with the last remainder in the third column, followed by the second last remainder, and so on. Thus:

$65_{10} = 1000001_2$.

Next, consider the number 711. Figure E-2 shows the quotients and the remainders.



	711	
2	355	$1 = a_0$
2	177	$1 = a_1$
2	88	$1 = a_2$
2	44	$0 = a_3$
2	22	$0 = a_4$
2	11	$0 = a_5$
2	5	$1 = a_6$
2	2	$1 = a_7$
2	1	$0 = a_8$
	0	$1 = a_9$

FIGURE E-2 Determining the binary representation of 711

From Figure E-2, it follows that:

$711_{10} = 1011000111_2$.

Converting a Binary Number (Base 2) to Base 10

To convert a number from base 2 to base 10, we first find the weight of each bit in the binary number. The weight of each bit in the binary number is assigned from right to left. The weight of the rightmost bit is 0. The weight of the bit immediately to the left of the rightmost bit is 1, the weight of the bit immediately to the left of it is 2, and so on. Consider the binary number 1001101. The weight of each bit is as follows:

weight	6	5	4	3	2	1	0
	1	0	0	1	1	0	1

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

We use the weight of each bit to find the equivalent decimal number. For each bit, we multiply the bit by 2 to the power of its weight and then we add all of the numbers. For the above binary number, the equivalent decimal number is:

$$1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$= 64 + 0 + 0 + 8 + 4 + 0 + 1$$

$$= 77.$$

Converting a Binary Number (Base 2) to Octal (Base 8) and Hexadecimal (Base 16)

The previous sections described how to convert a binary number to a decimal number (base 2). Even though the language of a computer is binary, if the binary number is too long, then it will be hard to manipulate it manually. To effectively deal with binary numbers, two more number systems, octal (base 8) and hexadecimal (base 16), are of interest to computer scientists.

The digits in the octal number system are 0, 1, 2, 3, 4, 5, 6, and 7. The digits in the hexadecimal number system are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. So A in hexadecimal is 10 in decimal, B in hexadecimal is 11 in decimal, and so on.

CENGAGEbrain.com

The algorithm to convert a binary number into an equivalent number in octal (or hexadecimal) is quite simple. Before we describe the method to do so, let us review some notations. Suppose a_b represents the number *a* to the base *b*. For example, $2A0_{16}$ means 2A0 to the base 16, and 63_8 means 63 to the base 8.

First we describe how to convert a binary number into an equivalent octal number and vice versa. Table E-1 describes the first eight octal numbers.

TABLE E-1 Binary representation of first eight octal numbers

Binary	Octal	Binary	Octal
000	0	100	4
001	1	101	5
010	2	110	6
011	3	111	7

Consider the binary number 1101100010101. To find the equivalent octal number, starting from right to left we consider three digits at a time and write their octal representation. Note that the binary number 1101100010101 has only 13 digits. So when

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

we consider three digits at a time, at the end we will be left with only one digit. In this case, we just add two 0s to the left of the binary number; the equivalent binary number is 001101100010101. Thus,

$$\begin{aligned}
1101100010101_2 &= 001101100010101_2 \\
&= 001\ 101\ 100\ 010\ 101 \\
&= 15425_8 \text{ because } 001_2 = 1_8,\ 101_2 = 5_8,\ 100_2 = 4_8,\ 010_2 = 2_8, \\
&\qquad \text{and } 101_2 = 5_8
\end{aligned}$$

Thus, $1101100010101_2 = 15425_8$.

To convert an octal number into an equivalent binary number, using Table E-1, write the binary representation of each octal digit in the number. For example,

$$\begin{aligned}
3761_8 &= 011\ 111\ 110\ 001_2 \\
&= 011111110001_2 \\
&= 11111110001_2
\end{aligned}$$

Thus, $3761_8 = 11111110001_2$.

CENGAGEbrain.com

Next we discuss how to convert a binary number into an equivalent hexadecimal number and vice versa. The method to do so is similar to converting a number from binary to octal and vice versa, except that here we work with four binary digits. Table E-2 gives the binary representation of the first 16 hexadecimal numbers.

TABLE E-2 Binary representation of first 16 hexadecimal numbers

Binary	Hexadecimal	Binary	Hexadecimal
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Consider the binary number 1111101010001010101_2. Now,

$$\begin{aligned}1111101010001010101_2 &= 111\ 1101\ 0100\ 0101\ 0101_2\\ &= 0111\ 1101\ 0100\ 0101\ 0101_2\text{, add one zero to the left}\\ &= 7D455_{16}\end{aligned}$$

Hence, $1111101010001010101_2 = 7D455_{16}$.

Next, to convert a hexadecimal number into an equivalent binary number, write the four-digit binary representation of each hexadecimal digit into that number. For example,

$$\begin{aligned}A7F32_{16} &= 1010\ 0111\ 1111\ 0011\ 0010_2\\ &= 10100111111100110010_2\end{aligned}$$

Thus, $A7F32_{16} = 10100111111100110010_2$.

More on File Input/Output

In Chapter 3, you learned how to read data from and write data to a file. This section expands on the concepts introduced in that chapter.

CENGAGEbrain.com

Binary Files

In Chapter 3, you learned how to make a program read data from and write data to a file. However, the files that the programs have used until now are called text files. Data in a text file is stored in the character format. For example, consider the number 45. If 45 is stored in a file, then it is stored as a sequence of two characters—the character '4' followed by the character '5'. The eight-bit machine representation of '4' is 00000100 and the eight-bit machine representation of '5' is 00000101. Therefore, in a text file, 45 is stored as 0000010000000101. When this number is read by a C++ program, it must first be converted to its binary format. Suppose that the integers are represented as 16-bit binary numbers. The 16-bit binary representation of 45 is then 0000000000101101. Similarly, when a program stores the number 45 in a text file, it first must be converted to its text format. It thus follows that reading data from and writing data to a text file is not efficient, because the data must be converted from the text to the binary format and vice versa.

On the other hand, when data is stored in a file in the binary format, reading and writing data is faster because no time is lost in converting the data from one format to another format. Such files are called binary files. More formally, **binary files** are files in which data is stored in the binary format. Data in a text file is also called **formatted data**, and in a binary file it is called **raw data**.

C++ allows a programmer to create binary files. This section explains how to create binary files and also how to read data from binary files.

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

To create a binary file, the file must be opened in the binary mode. Suppose outFile is an ofstream variable (object). Consider the following statement:

```
outFile.open("employee.dat", ios::binary);
```

This statement opens the file employee.dat. Data in this file will be written in its binary format. Therefore, the file opening mode ios::binary specifies that the file is opened in the binary mode.

Next, you use the stream function write to write data to the file employee.dat. The syntax to use the function write is:

```
fileVariableName.write(reinterpret_cast<const char *> (buffer),
                       sizeof(buffer));
```

where fileVariableName is the object used to open the output file, and the first argument buffer specifies the starting address of the location in memory where the data is stored. The expression **sizeof**(buffer) specifies the size of the data, in bytes, to be written.

For example, suppose num is an **int** variable. The following statement writes the value of num in the binary format to the file associated with outFile:

```
outFile.write(reinterpret_cast<const char *> (&num),
              sizeof(num));
```

CENGAGEbrain.com

Similarly, suppose empSalary is an array of, say, 100 components and the component type is **double**. The following statement writes the entire array to the file associated with outFile:

```
outFile.write(reinterpret_cast<const char *> (empSalary),
              sizeof(empSalary));
```

Next, let us discuss how to read data from a binary file. The operation of reading data from a binary file is similar to writing data to a binary file. First, the binary file must be opened. For example, suppose inFile is an ifstream variable, and a program has already created the binary file employee.dat. The following statement opens this file:

```
inFile.open("employee.dat");
```

or:

```
inFile.open("employee.dat", ios::binary);
```

To read data in the binary format, the stream function read is used. The syntax to use the function read is:

```
fileVariableName.read(reinterpret_cast<char *> (buffer),
                      sizeof(buffer));
```

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

The first argument buffer specifies the starting address of the location in memory where the data is to be stored. The expression sizeof(buffer) specifies the size of the data, in bytes, to be read.

The program in the following example further explains how to create binary files and read data from a binary file.

EXAMPLE E-1

```
//Creating and reading binary files

#include <iostream>
#include <fstream>

using namespace std;

struct studentType
{
    char firstName[15];
    char lastName[15];
    int ID;
};

int main()
{
        //create and initialize an array of students' IDs
    int studentIDs[5] = {111111, 222222, 333333,
                         444444, 555555};              //Line 1

         //declare and initialize the struct newStudent
    studentType newStudent = {"John", "Wilson",
                              777777};                 //Line 2

    ofstream outFile;                                  //Line 3

        //open the output file as a binary file
    outFile.open("ids.dat", ios::binary);              //Line 4

         //write the array in the binary format
    outFile.write(reinterpret_cast<const char *> (studentIDs),
                  sizeof(studentIDs));                 //Line 5
        //write the newStudent data in the binary format
    outFile.write(reinterpret_cast<const char *> (&newStudent),
                  sizeof(newStudent));                 //Line 6

    outFile.close();  //close the file                 //Line 7

    ifstream inFile;                                   //Line 8
    int arrayID[5];                                    //Line 9
    studentType student;                               //Line 10
```

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

```
         //open the input file
    inFile.open("ids.dat");                           //Line 11

    if (!inFile)                                      //Line 12
    {
        cout << "The input file does not exist. "
             << "The program terminates!!!!" << endl; //Line 13
        return 1;                                     //Line 14
    }

       //input the data into the array arrayID
    inFile.read(reinterpret_cast<char *> (arrayID),
                sizeof(arrayID));                     //Line 15
       //output the data of the array arrayID
    for (int i = 0; i < 5; i++)                       //Line 16
        cout << arrayID[i] << " ";                    //Line 17
    cout << endl;                                     //Line 18

         //read the student's data
    inFile.read(reinterpret_cast<char *> (&student),
                sizeof(student));                     //Line 19

         //output studentData
    cout << student.ID << " " << student.firstName
         << " " << student.lastName << endl;          //Line 20

    inFile.close();     //close the file              //Line 21

    return 0;                                         //Line 22
}
```

CENGAGEbrain.com

Sample Run:

```
111111 222222 333333 444444 555555
777777 John Wilson
```

The output of the preceding program is self-explanatory. The details are left as an exercise for you.

NOTE

In the program in Example E-1, the statement in Line 2 declares the `struct` variable `newStudent` and also initializes it. Because `newStudent` has three components and we want to initialize all the components, three values are specified in braces separated by commas. In other words, `struct` variables can also be initialized when they are declared.

The program in the following example further explains how to create binary files and then read the data from the binary files.

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

EXAMPLE E-2

```
//Creating and reading a binary file consisting of
//bank customers' data

#include <iostream>
#include <fstream>
#include <iomanip>

using namespace std;

struct customerType
{
    char firstName[15];
    char lastName[15];
    int ID;
    double balance;
};

int main()
{
    customerType cust;                                    //Line 1
    ifstream inFile;                                      //Line 2
    ofstream outFile;                                     //Line 3

    inFile.open("customerData.txt");                      //Line 4

    if (!inFile)                                          //Line 5
    {
        cout << "The input file does not exist. "
             << "The program terminates!!!!" << endl;     //Line 6
        return 1;                                         //Line 7
    }

   outFile.open("customer.dat", ios::binary);             //Line 8

   inFile >> cust.ID >> cust.firstName >> cust.lastName
          >> cust.balance;                                //Line 9

    while (inFile)                                        //Line 10
    {
         outFile.write(reinterpret_cast<const char *> (&cust),
                       sizeof(cust));                     //Line 11
         inFile >> cust.ID >> cust.firstName >> cust.lastName
                >> cust.balance;                          //Line 12
    }

    inFile.close();                                       //Line 13
    inFile.clear();                                       //Line 14
    outFile.close();                                      //Line 15
```

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

```
    inFile.open("customer.dat", ios::binary);          //Line 16

    if (!inFile)                                       //Line 17
    {
        cout << "The input file does not exist. "
             << "The program terminates!!!!" << endl;  //Line 18
        return 1;                                      //Line 19
    }

    cout << left << setw(8) << "ID"
         << setw(16) << "First Name"
         << setw(16) << "Last Name"
         << setw(10) << " Balance" << endl;            //Line 20
    cout << fixed << showpoint << setprecision(2);     //Line 21

        //read and output the data from the binary
        //file customer.dat
    inFile.read(reinterpret_cast<char *> (&cust),
                sizeof(cust));                         //Line 22
    while (inFile)                                     //Line 23
    {
         cout << left << setw(8) << cust.ID
              << setw(16) << cust.firstName
              << setw(16) << cust.lastName
              << right << setw(10) << cust.balance
              << endl;                                 //Line 24
        inFile.read(reinterpret_cast<char *> (&cust),
                    sizeof(cust));                     //Line 25
    }

    inFile.close();     //close the file               //Line 26

    return 0;                                          //Line 27
}
```

Sample Run:

```
ID      First Name      Last Name        Balance
77234   Ashley          White            4563.50
12345   Brad            Smith          128923.45
87123   Lisa            Johnson          2345.93
81234   Sheila          Robinson          674.00
11111   Rita            Gupta           14863.50
23422   Ajay            Kumar           72682.90
22222   Jose            Ramey           25345.35
54234   Sheila          Duffy           65222.00
55555   Tommy           Pitts             892.85
23452   Salma           Quade            2812.90
32657   Jennifer        Ackerman         9823.89
82722   Steve           Sharma          78932.00
```

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Random File Access

In Chapter 3 and the preceding section, you learned how to read data from and write data to a file. More specifically, you used `ifstream` objects to read data from a file and `ofstream` objects to write data to a file. However, the files were read and/or written sequentially. Reading data from a file sequentially does not work very well for a variety of applications. For example, consider a program that processes customers' data in a bank. Typically, there are thousands or even millions of customers in a bank. Suppose we want to access a customer's data from the file that contains such data, say, for an account update. If the data is accessed sequentially, starting from the first position and read until the desired customer's data is found, this process might be extremely time consuming. Similarly, in an airline's reservation system to access a passenger's reservation information sequentially, this might also be very time consuming. In such cases, the data retrieval must be efficient. A convenient way to do this is to be able to read the data randomly from a file, that is, randomly access any record in the file.

In the preceding section, you learned how to use the stream function `read` to read a specific number of bytes, and the function `write` to write a specific number of bytes.

The stream function `seekg` is used to move the read position to any byte in the file. The general syntax to use the function `seekg` is:

CENGAGEbrain.com

```
fileVariableName.seekg(offset, position);
```

The stream function `seekp` is used to move the write position to any byte in the file. The general syntax to use the function `seekp` is:

```
fileVariableName.seekp(offset, position);
```

The `offset` specifies the number of bytes the reading/writing positions are to be moved, and `position` specifies where to begin the offset. The offset can be calculated from the beginning of the file, end of the file, or the current position in the file. Moreover, `offset` is a long integer representation of an offset. Table E-3 shows the values that can be used for `position`.

TABLE E-3 Values of `position`

`position`	Description
`ios::beg`	The offset is calculated from the beginning of the file.
`ios::cur`	The offset is calculated from the current position of the reading marker in the file.
`ios::end`	The offset is calculated from the end of the file.

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

EXAMPLE E-3

Suppose you have the following line of text stored in a file, say, `digitsAndLetters.txt`:

```
0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Also, suppose that `inFile` is an `ifstream` object and the file `digitsAndLetters.txt` has been opened using the object `inFile`. One byte is used to store each character of this line of text. Moreover, the position of the first character is `0`.

Statement	Explanation
`inFile.seekp(10L, ios::beg);`	Sets the reading position of `inFile` to the 11th byte (character), which is at position `10`. That is, it sets the reading position just after the digit `9` or just before the letter `A`.
`inFile.seekp(5L, ios::cur);`	Moves the reading position of `inFile` five bytes to the right of its current position.
`inFile.seekp(-6L, ios::end);`	Sets the reading position of `inFile` to the sixth byte (character) from the end. That is, it sets the reading position just before the letter `U`.

CENGAGEbrain.com

The program in the following example further explains how the functions `seekg` and `seekp` work.

EXAMPLE E-4

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
    char ch;                                              //Line 1
    ifstream inFile;                                      //Line 2

    inFile.open("digitsAndAlphabet.txt");                 //Line 3

    if (!inFile)                                          //Line 4
    {
       cout << "The input file does not exist. "
            << "The program terminates!!!!" << endl;     //Line 5
       return 1;                                          //Line 6
    }
```

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

```
    inFile.get(ch);                                  //Line 7
    cout << "Line 8: The first byte: " << ch << endl; //Line 8

       //position the reading marker six bytes to the
       //right of its current position
    inFile.seekg(6L, ios::cur);                       //Line 9
    inFile.get(ch);   //read the character            //Line 10
    cout << "Line 11: Current byte read: " << ch
         << endl;                                     //Line 11

       //position the reading marker seven bytes
       //from the beginning
    inFile.seekg(7L, ios::beg);                       //Line 12
    inFile.get(ch);   //read the character            //Line 13
    cout << "Line 14: Seventh byte from the beginning: "
         << ch << endl;                               //Line 14

       //position the reading marker 26 bytes
       //from the end
    inFile.seekg(-26L, ios::end);                     //Line 15
    inFile.get(ch);   //read the character            //Line 16
    cout << "Line 17: Byte 26 from the end: " << ch
         << endl;                                     //Line 17

    return 0;                                         //Line 18
}
```

CENGAGE brain .com

Sample Run:

```
Line 8: The first byte: 0
Line 11: Current byte read: 7
Line 14: Seventh byte from the beginning: 7
Line 17: Byte 26 from the end: A
```

The input file contains the following line of text:

```
0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

The following program illustrates how the function `seekg` works with `struct`s.

EXAMPLE E-5

Suppose `customerType` is a `struct` defined as follows:

```
struct customerType
{
    char firstName[15];
    char lastName[15];
    int ID;
    double balance;
};
```

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

The program in Example E-2 created the binary file customer.dat consisting of certain customers' data. You can use the function seekg to move the reading position of this file to any record. Suppose inFile is an ifstream object used to open the binary file customer.dat.

The following statement calculates the size of a customerType **struct** and stores it in the variable custSize:

```
long custSize = sizeof(cust);
```

We can use the value of the variable custSize to move the reading position to a specific record in the file. For example, consider the following statement:

```
inFile.seekg(6 * custSize, ios::beg);
```

This statement moves the reading position just after the sixth customer's record, that is, just before the seventh customer's record.

The following program further illustrates how the function seekg works with **struct**s.

EXAMPLE E-6

CENGAGE brain

```
//Reading a file randomly

#include <iostream>
#include <fstream>
#include <iomanip>

using namespace std;

struct customerType
{
    char firstName[15];
    char lastName[15];
    int ID;
    double balance;
};

void printCustData(const customerType& customer);

int main()
{
    customerType cust;                                   //Line 1
    ifstream inFile;                                     //Line 2

    long custSize = sizeof(cust);                        //Line 3

    inFile.open("customer.dat", ios::binary);            //Line 4
    if (!inFile)                                         //Line 5
```

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

```
    {
        cout << "The input file does not exist. "
             << "The program terminates!!!!" << endl; //Line 6
        return 1;                                     //Line 7
    }

    cout << fixed << showpoint << setprecision(2);    //Line 8

        //randomly read the records and output them
    inFile.seekg(6 * custSize, ios::beg);             //Line 9
    inFile.read(reinterpret_cast<char *> (&cust),
                sizeof(cust));                        //Line 10
    cout << "Seventh customer's data: " << endl;      //Line 11
    printCustData(cust);                              //Line 12

    inFile.seekg(8 * custSize, ios::beg);             //Line 13
    inFile.read(reinterpret_cast<char *> (&cust),
                sizeof(cust));                        //Line 14
    cout << "Ninth customer's data: " << endl;        //Line 15
    printCustData(cust);

    inFile.seekg(-8 * custSize, ios::end);            //Line 16
    inFile.read(reinterpret_cast<char *> (&cust),
                sizeof(cust));                        //Line 17
    cout << "Eighth (from the end) customer's data: "
         << endl;                                     //Line 18
    printCustData(cust);                              //Line 19

    inFile.close();     //close the file              //Line 20

    return 0;                                         //Line 21
}

void printCustData(const customerType& customer)
{
    cout << "  ID: " << customer.ID <<endl
         << "  First Name: " << customer.firstName <<endl
         << "  Last Name: " << customer.lastName <<endl
         << "  Account Balance: $" << customer.balance
         << endl;
}
```

Sample Run:

```
Seventh customer's data:
  ID: 22222
  First Name: Jose
  Last Name: Ramey
  Account Balance: $25345.35
Ninth customer's data:
  ID: 55555
  First Name: Tommy
  Last Name: Pitts
```

CENGAGE brain .com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

```
  Account Balance: $892.85
Eighth (from the end) customer's data:
  ID: 11111
  First Name: Rita
  Last Name: Gupta
  Account Balance: $14863.50
```

The program in Example E-6 illustrates how the function seekg works. Using the function seekg, the reading position in a file can be moved to any location in the file. Similarly, the function seekp can be used to move the write position in a file to any location. Furthermore, these functions can be used to create a binary file in which the data is organized according to the values of either a variable or a particular component of a struct. For example, suppose there are at most, say, 100 students in a class. Each student has a unique ID in the range 1 to 100. Using the students' IDs, we can create a random access binary file in such a way that in the file, a student's data is written at the location specified by its ID. This is like treating the file as an array. The advantage is that, once the file is created, a student's data from the file can be read, directly, using the student's ID. Another advantage is that in the file, the data is sorted according to the IDs.

Here, we are assuming that the student IDs are in the range 1 to 100. However, if you use, say, a three-, four-, or five-digit number as a student ID and there are only a few students in the class, the data in the file could be scattered. In other words, a lot of space could be used just to store only a few students' data. In such cases, more advanced techniques are used to organize the data so that it can be accessed efficiently.

The program in Example E-7 illustrates how to use the students' IDs to organize the data in a binary file. The program also shows how to output the file.

EXAMPLE E-7

```
//Creating and reading a random access file.

#include <iostream>
#include <fstream>
#include <iomanip>

using namespace std;

struct studentType
{
    char firstName[15];
    char lastName[15];
    int ID;
    double GPA;
};

void printStudentData(const studentType& student);
```

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

```
int main()
{
    studentType st;                                     //Line 1
    ifstream inFile;                                    //Line 2
    ofstream outFile;                                   //Line 3

    long studentSize = sizeof(st);                      //Line 4

        //open the input file, which is a text file
    inFile.open("studentData.txt");                     //Line 5

    if (!inFile)                                        //Line 6
    {
        cout << "The input file does not exist. "
             << "The program terminates!!!!" << endl;   //Line 7
        return 1;                                       //Line 8
    }

        //open a binary output file
    outFile.open("student.dat", ios::binary);           //Line 9

    inFile >> st.ID >> st.firstName
           >> st.lastName >> st.GPA;                    //Line 10

    while (inFile)                                      //Line 11
    {
        outFile.seekp((st.ID - 1) * studentSize,
                      ios::beg);                        //Line 12
        outFile.write(reinterpret_cast<const char *> (&st),
                     sizeof(st));                       //Line 13
        inFile >> st.ID >> st.firstName
               >> st.lastName >> st.GPA;                //Line 14
    };

    inFile.close();                                     //Line 15
    inFile.clear();                                     //Line 16
    outFile.close();                                    //Line 17

    cout << left << setw(3) << "ID"
         << setw(16) << "First Name"
         << setw(16) << "Last Name"
         << setw(12) << "Current GPA" << endl;          //Line 18
    cout << fixed << showpoint << setprecision(2);      //Line 19

         //open the input file, which is a binary file
    inFile.open("student.dat", ios::binary);            //Line 20

    if (!inFile)                                        //Line 21
    {
       cout << "The input file does not exist. "
            << "The program terminates!!!!" << endl;    //Line 22
       return 1;                                        //Line 23
    }
```

CENGAGE brain .com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

```
        //read the data at location 0 in the file
    inFile.read(reinterpret_cast<char *> (&st),
                sizeof(st));                               //Line 24
    while (inFile)                                         //Line 25
    {
        if (st.ID != 0)                                    //Line 26
            printStudentData(st);                          //Line 27

            //read the data at the current reading position
        inFile.read(reinterpret_cast<char *> (&st),
                    sizeof(st));                           //Line 28
    };

    return 0;                                              //Line 29
}

void printStudentData(const studentType& student)
{
    cout << left << setw(3) << student.ID
         << setw(16) << student.firstName
         << setw(16) << student.lastName
         << right << setw(10)<< student.GPA
         << endl;
}
```

CENGAGEbrain.com

Sample Run:

```
ID First Name         Last Name          Current GPA
2  Sheila             Duffy                    4.00
10 Ajay               Kumar                    3.60
12 Ashley             White                    3.90
16 Tommy              Pitts                    2.40
23 Rita               Gupta                    3.40
34 Brad               Smith                    3.50
36 Salma              Quade                    3.90
41 Steve              Sharma                   3.50
45 Sheila             Robinson                 2.50
56 Lisa               Johnson                  2.90
67 Jose               Ramey                    3.80
75 Jennifer           Ackerman                 4.00
```

The data in the file studentData.txt is as follows:

```
12 Ashley White 3.9
34 Brad Smith 3.5
56 Lisa Johnson 2.9
45 Sheila Robinson 2.5
23 Rita Gupta 3.4
10 Ajay Kumar 3.6
67 Jose Ramey 3.8
2 Sheila Duffy 4.0
16 Tommy Pitts 2.4
```

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

```
36 Salma Quade 3.9
75 Jennifer Ackerman 4.0
41 Steve Sharma 3.5
```

Naming Conventions of Header Files in ANSI/ISO Standard C++ and Standard C++

The programs in this book are written using ANSI/ISO Standard C++. As indicated in Chapter 1, there are two versions of C++—ANSI/ISO Standard C++ and Standard C++. For the most part, these two standards are the same. The header files in Standard C++ have the extension `.h`, while the header files in ANSI/ISO Standard C++ have no extension. Moreover, the names of certain header files, such as `math.h`, in ANSI/ISO Standard C++ start with the letter `c`. The language C++ evolved from C. Therefore, certain header files—such as `math.h`, `stdlib.h`, and `string.h`—were brought from C into C++. The header files—such as `iostream.h`, `iomanip.h`, and `fstream.h`—were specially designed for C++. Recall that when a header file is included in a program, the global identifiers of the header file also become the global identifiers of the program. In ANSI/ISO Standard C++, to take advantage of the **`namespace`** mechanism, all of the header files were modified so that the identifiers are declared within a **`namespace`**. Recall that the name of this **`namespace`** is `std`.

CENGAGEbrain.com

In ANSI/ISO Standard C++, the extension `.h` of the header files that were specially designed for C++ was dropped. For the header files that were brought from C into C++, the extension `.h` was dropped and the names of these header files start with the letter `c`. Following are the names of the most commonly used header files in Standard C++ and ANSI/ISO Standard C++:

Standard C++ Header File Name	ANSI/ISO Standard C++ Header File Name
`assert.h`	`cassert`
`ctype.h`	`cctype`
`float.h`	`cfloat`
`fstream.h`	`fstream`
`iomanip.h`	`iomanip`
`iostream.h`	`iostream`
`limits.h`	`climits`
`math.h`	`cmath`
`stdlib.h`	`cstdlib`
`string.h`	`cstring`

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

To include a header file, say, `iostream`, the following statement is required:

```
#include <iostream>
```

Furthermore, to use identifiers, such as `cin`, `cout`, `endl`, and so on, the program should use either the statement:

```
using namespace std;
```

or the prefix `std::` before the identifier.

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Appendix F
Header Files

The C++ standard library contains many predefined functions, named constants, and specialized data types. This appendix discusses some of the most widely used library routines (and several named constants). For additional explanation and information on functions, named constants, and so on, check your system documentation. The names of the Standard C++ header files are shown in parentheses.

Header File cassert (assert.h)

The following table describes the function `assert`. Its specification is contained in the header file `cassert` (`assert.h`).

`assert(expression)`	`expression` is any **`int`** `expression`; `expression` is usually a logical expression	• If the value of `expression` is nonzero (**`true`**), the program continues to execute. • If the value of `expression` is 0 (**`false`**), execution of the program terminates immediately. The expression, the name of the file containing the source code, and the line number in the source code are displayed.

NOTE To disable all the `assert` statements, place the preprocessor directive **`#define`** `NDEBUG` before the directive **`#include`** `<cassert>`.

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Header File cctype (ctype.h)

The following table shows various functions from the header file cctype (ctype.h).

Function Name and Parameters	Parameter(s) Types	Function Return Value
isalnum(ch)	ch is a char value	Function returns an int value as follows: • If ch is a letter or a digit character, that is ('A'-'Z', 'a'-'z', '0'-'9'), it returns a nonzero value (true) • 0 (false), otherwise
iscntrl(ch)	ch is a char value	Function returns an int value as follows: • If ch is a control character (in ASCII, a character value 0-31 or 127), it returns a nonzero value (true) • 0 (false), otherwise
isdigit(ch)	ch is a char value	Function returns an int value as follows: • If ch is a digit ('0'-'9'), it returns a nonzero value (true) • 0 (false), otherwise
islower(ch)	ch is a char value	Function returns an int value as follows: • If ch is lowercase ('a'-'z'), it returns a nonzero value (true) • 0 (false), otherwise
isprint(ch)	ch is a char value	Function returns an int value as follows: • If ch is a printable character, including blank (in ASCII, ' ' through '~'), it returns a nonzero value (true) • 0 (false), otherwise
ispunct(ch)	ch is a char value	Function returns an int value as follows: • If ch is a punctuation character, it returns a nonzero value (true) • 0 (false), otherwise
isspace(ch)	ch is a char value	Function returns an int value as follows: • If ch is a white space character (blank, newline, tab, carriage return, form feed), it returns a nonzero value (true) • 0 (false), otherwise

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Function Name and Parameters	Parameter(s) Types	Function Return Value
isupper(ch)	ch is a **char** value	Function returns an **int** value as follows: • If ch is an uppercase letter ('A'-'Z'), it returns a nonzero value (**true**) • 0 (**false**), otherwise
tolower(ch)	ch is a **char** value	Function returns an **int** value as follows: • If ch is an uppercase letter, it returns the ASCII value of the lowercase equivalent of ch • ASCII value of ch, otherwise
toupper(ch)	ch is a **char** value	Function returns an **int** value as follows: • If ch is a lowercase letter, it returns the ASCII value of the uppercase equivalent of ch • ASCII value of ch, otherwise

Header File cfloat (float.h)

In Chapter 2, we listed the largest and smallest values belonging to the floating-point data types. We also remarked that these values are system dependent. These largest and smallest values are stored in named constants. The header file cfloat contains many such named constants. The following table lists some of these constants.

Named Constant	Description
FLT_DIG	Approximate number of significant digits in a **float** value
FLT_MAX	Maximum positive **float** value
FLT_MIN	Minimum positive **float** value
DBL_DIG	Approximate number of significant digits in a **double** value
DBL_MAX	Maximum positive **double** value
DBL_MIN	Minimum positive **double** value
LDBL_DIG	Approximate number of significant digits in a **long double** value
LDBL_MAX	Maximum positive **long double** value
LDBL_MIN	Minimum positive **long double** value

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

A program similar to the following can print the values of these named constants on your system:

```
#include <iostream>
#include <cfloat>

using namespace std;

int main()
{
    cout << "Approximate number of significant digits "
         << "in a float value " << FLT_DIG << endl;
    cout << "Maximum positive float value " << FLT_MAX
         << endl;
    cout << "Minimum positive float value " << FLT_MIN
         << endl;
    cout << "Approximate number of significant digits "
         << "in a double value " << DBL_DIG << endl;
    cout << "Maximum positive double value " << DBL_MAX
         << endl;
    cout << "Minimum positive double value " << DBL_MIN
         << endl;
    cout << "Approximate number of significant digits "
         << "in a long double value " << LDBL_DIG << endl;
    cout << "Maximum positive long double value " << LDBL_MAX
         << endl;
    cout << "Minimum positive long double value " << LDBL_MIN
         << endl;

    return 0;
}
```

CENGAGEbrain.com

Header File climits (limits.h)

In Chapter 2, we listed the largest and smallest values belonging to the integral data types. We also remarked that these values are system dependent. These largest and smallest values are stored in named constants. The header file climits contains many such named constants. The following table lists some of these constants.

Named Constant	Description
CHAR_BIT	Number of bits in a byte
CHAR_MAX	Maximum char value
CHAR_MIN	Minimum char value
SHRT_MAX	Maximum short value
SHRT_MIN	Minimum short value

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Named Constant	Description
INT_MAX	Maximum int value
INT_MIN	Minimum int value
LONG_MAX	Maximum long value
LONG_MIN	Minimum long value
UCHAR_MAX	Maximum unsigned char value
USHRT_MAX	Maximum unsigned short value
UINT_MAX	Maximum unsigned int value
ULONG_MAX	Maximum unsigned long value

A program similar to the following can print the values of these named constants on your system:

```
#include <iostream>
#include <climits>

using namespace std;

int main()
{
    cout << "Number of bits in a byte " << CHAR_BIT << endl;
    cout << "Maximum char value " << CHAR_MAX << endl;
    cout << "Minimum char value " << CHAR_MIN << endl;
    cout << "Maximum short value " << SHRT_MAX << endl;
    cout << "Minimum short value " << SHRT_MIN << endl;
    cout << "Maximum int value " << INT_MAX << endl;
    cout << "Minimum int value " << INT_MIN << endl;
    cout << "Maximum long value " << LONG_MAX << endl;
    cout << "Minimum long value " << LONG_MIN << endl;
    cout << "Maximum unsigned char value " << UCHAR_MAX
         << endl;
    cout << "Maximum unsigned short value " << USHRT_MAX
         << endl;
    cout << "Maximum unsigned int value " << UINT_MAX << endl;
    cout << "Maximum unsigned long value " << ULONG_MAX
         << endl;

    return 0;
}
```

CENGAGE brain .com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Header File `cmath` (`math.h`)

The following table shows various math functions.

Function Name and Parameters	Parameter(s) Type	Function Return Value
`acos(x)`	`x` is a floating-point expression, $-1.0 \le x \le 1.0$	Arc cosine of `x`, a value between 0.0 and π
`asin(x)`	`x` is a floating-point expression, $-1.0 \le x \le 1.0$	Arc sine of `x`, a value between $-\pi/2$ and $\pi/2$
`atan(x)`	`x` is a floating-point expression	Arc tan of `x`, a value between $-\pi/2$ and $\pi/2$
`ceil(x)`	`x` is a floating-point expression	The smallest whole number $\ge$ `x`, ("ceiling" of `x`)
`cos(x)`	`x` is a floating-point expression, `x` is measured in radians	Trigonometric cosine of the angle
`cosh(x)`	`x` is a floating-point expression	Hyperbolic cosine of `x`
`exp(x)`	`x` is a floating-point expression	The value `e` raised to the power of `x`; `(e = 2.718...)`
`fabs(x)`	`x` is a floating-point expression	Absolute value of `x`
`floor(x)`	`x` is a floating-point expression	The largest whole number $\le$ x; ("floor" of x)
`log(x)`	`x` is a floating-point expression, where $x > 0.0$	Natural logarithm (base `e`) of `x`
`log10(x)`	`x` is a floating-point expression, where $x > 0.0$	Common logarithm (base `10`) of `x`
`pow(x,y)`	`x` and `y` are floating-point expressions. If $x = 0.0$, `y` must be positive; if $x \le 0.0$, `y` must be a whole number.	`x` raised to the power of `y`
`sin(x)`	`x` is a floating-point expression; `x` is measured in radians	Trigonometric sine of the angle
`sinh(x)`	`x` is a floating-point expression	Hyperbolic sine of `x`

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Function Name and Parameters	Parameter(s) Type	Function Return Value
`sqrt(x)`	`x` is a floating-point expression, where `x` ≥ `0.0`	Square root of `x`
`tan(x)`	`x` is a floating-point expression; `x` is measured in radians	Trigonometric tangent of the angle
`tanh(x)`	`x` is a floating-point expression	Hyperbolic tangent of `x`

Header File cstddef (stddef.h)

Among others, this header file contains the definition of the following symbolic constant:

`NULL`: The system-dependent null pointer (usually 0)

Header File cstring (string.h)

The following table shows various string functions.

CENGAGEbrain.com

Function Name and Parameters	Parameter(s) Type	Function Return Value
`strcat(destStr, srcStr)`	`destStr` and `srcStr` are null-terminated **`char`** arrays; `destStr` must be large enough to hold the result	The base address of `destStr` is returned; `srcStr`, including the null character, is concatenated to the end of `destStr`
`strcmp(str1, str2)`	`str1` and `str2` are null terminated **`char`** arrays	The returned value is as follows: • An **`int`** value < 0, if `str1 < str2` • An **`int`** value 0, if `str1 = str2` • An **`int`** value > 0, if `str1 > str2`

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Function Name and Parameters	Parameter(s) Type	Function Return Value
`strcpy(destStr, srcStr)`	`destStr` and `srcStr` are null-terminated `char` arrays	The base address of `destStr` is returned; `srcStr` is copied into `destStr`
`strlen(str)`	`str` is a null-terminated `char` array	An integer value ≥ 0 specifying the length of the `str` (excluding the `'\0'`) is returned

HEADER FILE string

This header file—not to be confused with the header file `cstring`—supplies a programmer-defined data type named `string`. Associated with the `string` type are a data type `string::size_type` and a named constant `string::npos`. These are defined as follows:

`string::size_type`	An unsigned integer type
`string::npos`	The maximum value of type `string::size_type`

CENGAGE brain

Several functions are associated with the `string` type. The following table shows some of these functions. Unless stated otherwise, `str`, `str1`, and `str2` are variables (objects) of type `string`. The position of the first character in a `string` variable (such as `str`) is `0`, the second character is `1`, and so on.

Function Name and Parameters	Parameter(s) Type	Function Return Value
`str.c_str()`	None	The base address of a null-terminated C-string corresponding to the characters in `str`.
`getline(istreamVar,str)`	`istreamVar` is an input stream variable (of type `istream` or `ifstream`). `str` is a `string` object (variable).	Characters until the newline character are input from `istreamVar` and stored in `str`. (The newline character is read but not stored into `str`.) The value returned by this function is usually ignored.

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Function Name and Parameters	Parameter(s) Type	Function Return Value
str.empty()	None	Returns **true** if str is empty, that is, the number of characters in str is zero, **false** otherwise.
str.length()	None	A value of type string::size_type giving the number of characters in the string.
str.size()	None	A value of type string::size_type giving the number of characters in the string.
str.find(strExp)	str is a string object and strExp is a string expression evaluating to a string. The string expression, strExp, can also be a character.	The find function searches str to find the first occurrence of the string or the character specified by strExp. If the search is successful, the function find returns the position in str where the match begins. If the search is unsuccessful, the function returns the special value string::npos.
str.substr(pos, len)	Two unsigned integers, pos and len. pos, represent the starting position (of the substring in str), and len represents the length (of the substring). The value of pos must be less than str.length().	A temporary string object that holds a substring of str starting at pos. The length of the substring is, at most, len characters. If len is too large, it means "to the end" of the string in str.
str1.swap(str2);	One parameter of type string. str1 and str2 are string variables.	The contents of str1 and str2 are swapped.

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Function Name and Parameters	Parameter(s) Type	Function Return Value
`str.clear();`	None	Removes all the characters from `str`.
`str.erase();`	None	Removes all the characters from `str`.
`str.erase(m);`	One parameter of type `string::size_type`.	Removes all the characters from `str` starting at index `m`.
`str.erase(m, n);`	Two parameters of type **`int`**.	Starting at index `m`, removes the next `n` characters from `str`. If `n` > length of `str`, removes all the characters starting at the `m`th.
`str.insert(m, n, c);`	Parameters `m` and `n` are of type `string::size_type`; `c` is a character.	Inserts `n` occurrences of the character `c` at index `m` into `str`.
`str1.insert(m, str2);`	Parameter `m` is of type `string::size_type`.	Inserts all the characters of `str2` at index `m` into `str1`.
`str1.replace(m,n,str2);`	Parameters `m` and `n` are of type `string::size_type`.	Starting at index `m`, replaces the next `n` characters of `str1` with all the characters of `str2`. If `n` > length of `str1`, then all the characters until the end of `str1` are replaced.

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Appendix G
Memory Size on a System and Random Number Generator

A program similar to the following prints the memory size of the built-in data types on your system. (The output of the program shows the size of the built-in data type on which this program was run.)

```
#include <iostream>

using namespace std;

int main()
{
   cout << "Size of char = " << sizeof(char) << endl;
   cout << "Size of int = " << sizeof(int) << endl;
   cout << "Size of short = " << sizeof(short) << endl;
   cout << "Size of unsigned int = " << sizeof(unsigned int)
        << endl;
   cout << "Size of long = " << sizeof(long) << endl;
   cout << "Size of bool = " << sizeof(bool) << endl;
   cout << "Size of float = " << sizeof(float) << endl;
   cout << "Size of double = " << sizeof(double) << endl;
   cout << "Size of long double = " << sizeof(long double)
        << endl;
   cout << "Size of unsigned short = "
        << sizeof(unsigned short) << endl;
   cout << "Size of unsigned long = "
        << sizeof(unsigned long) << endl;

   return 0;
}
```

Sample Run:

```
Size of char = 1
Size of int = 4
Size of short = 2
Size of unsigned int = 4
Size of long = 4
Size of bool = 1
Size of float = 4
Size of double = 8
Size of long double = 8
Size of unsigned short = 2
Size of unsigned long = 4
```

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Random Number Generator

To generate a random number, you can use the C++ function rand. To use the function rand, the program must include the header file cstdlib. The header file cstdlib also contains the constant RAND_MAX. Typically, the value of RAND_MAX is 32767. To find the exact value of RAND_MAX, check your system's documentation. The function rand generates an integer between 0 and RAND_MAX. The following program illustrates how to use the function rand. It also prints the value of RAND_MAX:

```
#include <iostream>
#include <cstdlib>
#include <iomanip>

using namespace std;

int main()
{
    cout << fixed << showpoint << setprecision(5);
      cout << "The value of RAND_MAX: " << RAND_MAX << endl;
    cout << "A random number: " << rand() << endl;
    cout << "A random number between 0 and 9: "
         << rand() % 10 << endl;
    cout << "A random number between 0 and 1: "
         << static_cast<double> (rand())
              / static_cast<double>(RAND_MAX)
         << endl;

    return 0;
}
```

Sample Run:

```
The value of RAND_MAX: 32767
A random number: 41
A random number between 0 and 9: 7
A random number between 0 and 1: 0.19330
```

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Appendix H
References

1. G. Booch, *Object-Oriented Analysis and Design*, Second Edition, Addison-Wesley, 1995.
2. E. Horowitz, S. Sahni, and S. Rajasekaran, *Computer Algorithms C++*, Computer Science Press, 1997.
3. N.M. Josuttis, *The C++ Standard Library: A Tutorial and Reference*, Addison-Wesley, Reading, MA, 1999.
4. D.E. Knuth, *The Art of Computer Programming*, Vols. 1-3, Addison-Wesley, 1973, 1969, 1973.
5. S.B. Lippman and J. Lajoie, *C++ Primer*, Third Edition, Addison-Wesley, Reading, MA, 1998.
6. D.S. Malik and M.K. Sen, *Discrete Mathematical Structures: Theory and Applications*, Course Technology, Boston, MA, 2004.
7. E.M. Reingold and W.J. Hensen, *Data Structures in Pascal*, Little Brown and Company, Boston, MA, 1986.
8. R. Sedgewick, *Algorithms in C*, Third Edition, Addison-Wesley, Reading, MA, Parts 1-4, 1998; Part 5, 2002.
9. B. Stroustrup, *The Design and Evolution of C++*, Addison-Wesley, Reading, MA, 1994.

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

INDEX

& (ampersand), 350, 356, 482
* (asterisk), 32, 728, 1094, 1095
\ (backslash), 74, 492
, (comma), 32–33, 37, 84
{} (curly braces), 80, 82, 84
. (decimal point), 130–132, 140
" (double quotes), 51, 75, 443
= (equal sign), 32, 1094
! (exclamation point), 32
/ (forward slash), 32, 1094, 1095
> (greater-than sign), 32
- (hyphen), 32, 1094, 1095
< (less-than sign), 32
() (parentheses), 44–45, 181, 185, 187
. (period), 32, 548, 589
+ (plus sign), 32, 1094, 1095
# (pound sign), 12, 77, 1095
? (question mark), 32
; (semicolon), 32–33, 74, 84, 547, 585
' (single quote), 74
~ (tilde), 615
_ (underscore), 33, 34, 437

A

abs function, 309, 312, 313
abstract classes
  described, **773**
  linked lists and, 981
  overview, 772–780
abstract data types (ADTs), 692, 1264–1272
  defining lists as, 617–618
  described, **616–618**
  graphs and, 1319–1322, 1342
  linked lists and, 964–965, 974–987, 996–997, 1011–1012
  queues and, 1109–1111, 1130–1134, 1137–1138, 1141, 1077–1078
  stacks and, 1056–1057
  templates and, 861
abstract functions, 981
accessor functions, 598–602
accumulate function, 1453–1458
action statements, **185–186**
Ada, 692
addFirst function, 369
addQueue function, 1109–1111, 1114, 1116, 1120, 1124, 1127, 1141–1142
address of operator, **729–730**, 780–783
addressOfX function, 782
addVotes function, 1237–1238
adjacency.
  See also adjacent
  lists, **1317–1318**
  matrix, **1316–1317**
adjacent.
  See also adjacency
  use of the term, **1316**
  vertices, **1315**
adjacent_difference function, 1453–1458
adjacent_find function, 1404, 1431–1435
ADTs (abstract data types), 692, 1264–1272
  defining lists as, 617–618
  described, **616–618**
  graphs and, 1319–1322, 1342
  linked lists and, 964–965, 974–987, 996–997, 1011–1012
  queues and, 1109–1111, 1130–1134, 1137–1138, 1141, 1077–1078
  stacks and, 1056–1057
  templates and, 861
aggregate operations, **481–482**
Aiken, Howard, 3
algebra, 308
algorithm(s)
  arrays and, 477–478, 501
  asymptotic (Big-O) notation and, 1173–1182
  control structures and, 181–182, 213–218, 251–255, 260–264, 277
  converting length and, 90–93
  described, 14–16
  functions and, 324–333, 383, 389
  generic, **1398**
  GPAs and, 695–696
  heap, 1400
  I/O and, 150–158
  linked lists and, 975, 998
  making change and, 83–84
  modifying, 1399
  nonmodifying, 1398
  nonrecursive, 1102–1109
  numeric, 1399–13400
  problem solving and, 14–22
  records and, 555–556, 572–573
  shortest path, **1327–1335**
  stacks and, 1102–1109, 1073–1074, 1095
  STL and, 1397–1408
  text processing and, 526
  traversal, 1288–1292
algorithm header file, 1397, 1408, 1427
ALU (arithmetic logic unit), **4–5**
American Standard Code for Information Interchange. *See* ASCII (American Standard Code for Information Interchange)
ampersand (&), 350, 356, 482
analog signals, **6**

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

ANSI/ISO Standard C++.
  See also Standard C++
  arrays and, 514
  described, **24**, 50–51
  namespaces and, 78, 437, 438
  naming conventions, 1496–1497
  STL and, 1352
  string data type and, 442–443
Apple Computer, 3
application programs, **6**.
  See also programs
arguments, 126.
  See also parameters
arithmetic
  expressions, **41–43**
  operators, **40–45**, 744–745
arithmetic logic unit. *See* ALU (arithmetic logic unit)
array(s)
  base addresses, **485–488**, 511
  -based lists, 1189–1210
  character (c-strings), **492–499**, 514–516, 852–858
  of class objects, 613–615
  components, accessing, 472–475, 502
  of constructors, 613–615
  declaration, 480–481, 491–492, 503, 516–517
  described, **471**
  dynamic, **475**
  implementation of queues as, 1111–1122
  in bounds, **479**
  index out of bounds, **479–480**
  indices, 479–480, 491, 503–507
  initialization, 476, **480–481**, 503, 507
  largest elements in, 477–478
  multidimensional, **500–518**
  n-dimensional, **517**
  one-dimensional, **471**, 475–479, 482
  operator overloading and, 824–825
  overview, 471–492
  parallel, **499–500**
  passing, 511–514
  printing, 476
  processing, 475–479, 481–482, 505–509
  queues and, 1137
  reading data in, 476
  records and, 553–558
  recursion and, 928–940
  stacks and, 1058–1090
  STL and, 1355
  sum and average of, finding, 476–477
  templates and, 864
  two-dimensional, **500–519**, 749–751
array index operator, 850
array subscript operator, **444**, 472
arrayClass class, 824–825
arraySize variable, 375
arrivalTime variable, 1130, 1132–1133
arrivalTimeEmp array, 613–614
ASCII (American Standard Code for Information Interchange)
  arrays and, 492, 494
  control structures and, 172–174
  data types and, 38
  described, **7–8**, 1471–1472
  extraction operator and, 118
  functions and, 311
  recursion and, 941
  type conversion and, 50
assemblers, **9**
assembly language, examples, **9**
assert function, 211–218, 853, 886, 889, 984
assert statement, 909
assert.h. *See* cassert (assert.h) header file
assign function, 1370, 1375
assignment operators, 202–203, 756–758, 821–829, 836, 590–591, 986.
  See also assignment statements
  binary trees and, 1272
  described, **55**, 58
  stacks and, 1066–1068, 1071, 1086–1090
assignment statements, 54–58, 64–66.
  See also assignment operators
  compound, **88–89**
  pointers and, 732
  records and, 550–551
  simple, **88–89**
associativity, **44**, 58, 178
asterisk (*), 32, 728, 1094, 1095
asymptotic (Big-O) notation, 1173–1182
at function, 1370
Augusta, Ada, 2
AVL trees, 1295

B

Babbage, Charles, 2
back, of queues, 1109
back function, 984, 1370, 1375, 1395, 1110, 1120–1121
backslash (/), 74, 492
bad_alloc exceptions, 899–901
base case, **924**, 929
base classes.
  See also inheritance
  constructors and, 670–677
  described, **661**
  overriding member functions of, 663–670
baseClass class, 663, 675, 766–768, 770–771
baseClass.h header file, 769
bCh variable, 683, 684, 685
bClass class, 682–683, 684
bDay object, 689–690
begin function, 984, 1359–1363
Bell Laboratories, 24
billingAmount function, 704
binary code, **7**
binary digits, **7**
binary files, 1482–1487
binary numbers
  conversion and, 941–949, 1477–1482
  described, **7–8**, 1477
binary operators
  described, **41**
  operator overloading and, 810–816

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

binary tree(s)
  binary search trees, 1273–1288
  copying, 1259–1260
  deleting items from, 1279–1287
  described, **1201–1202**, 1252
  height, **1258**
  implementing, 1264–1272
  overview, 1252–1309
  traversal, 1260–1264, 1288–1305
binary_search function, 1427–1430
binaryTreeType class, 1268, 1270–1271, 1293, 1294
binSearch function, 1173, 1188
binToDec function, 945
bits, **7**
blank spaces, 84, 86, 125, 352
blocks, of statements, **191–192**
bool data type, 36, 37
  logical expressions and, 183–184
  while loops and, 243–244, 245
Boolean (logical) expressions
  bool data type and, 183–184
  control structures and, 173–184
  described, **170**, 173
  int data type and, 182–183
  typedef statement and, 425–426
Boolean (logical) operators
  control structures and, 175–184
  described, **175**
bottom-up design, **22**
boxType class, 666–679
branches
  described, **1202**
  directed, **1252**
breadth first traversal, **1325–1327**, 1264
break keyword, 204, 205–207
break statements, **278–281**
breathFirstTraversal function, 1326–1327
bSearchTreeType class, 1294, 1299
bubbleSort function, 1188
Build command, **14**, 624
buildListBackward function, 974, 975
buildListForward function, 973, 975

C

C (high-level language), 9
C# (high-level language), 9
cable company bills, 213–218, 327–333
calculate function, 390
calculateAverage function, 390–392
calculatePay function, 774
callPrint function, 767–768
candidateType class, 1225–1232
candy machine program, 633–647
candyMachine.h header file, 647
case keyword, 204, 206, 208–210
cassert (assert.h) header file, 212, 853, 1510
cAssignmentOprOverload class, 826–827
cast operators, **48**, 66
casting. *See* type conversion
cc command, 624
cctype (ctype.h) header file, 1500–1501
ceil function, 309
Census Bureau (United States), 3
central processing units. *See* CPUs (central processing units)
cfloat (float.h) header file, 1334, 1501–1502
change, making, 93–94
char data type, 36, 37–38, 45
  arrays and, 493, 499
  enumeration types and, 420
  functions and, 351
  templates and, 859
  type conversion and, 50
  variables and, 64, 70
character(s).
  See also character sets; symbols (listed by name)
  comparing, 172–173
  counting, 528
character sets.
  See also ASCII (American Standard Code for Information Interchange); characters
  described, **1471–1473**
  EBCDIC (Extended Binary-Coded Decimal Interchange Code), 8, 38, 492, **1472–1473**
characterCount function, 528
charitable donations program, 150–155
checking account balances, 250–259
child
  left, **1252**
  right, **1252**
cin keyword, 33
cin variable
  described, **78**
  extraction operator and, 117–123
  get function and, 125–126, 131
  header files and, 76
  ignore function and, 126–128
  preprocessor directives and, 116–117
circle class, 661, 662, 772
circular linked lists, 1022–1023. *See also* linked lists
class(es).
  See also class diagrams; classes (listed by name); class members
  built-in operations on, 590
  defining, syntax for, 584
  described, **584**
  identifying, 692–693
  overview, 583–658
  peculiarities related to, 754–765
  pointer variables and, 736–739

CENGAGE brain .com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

class(es). (*continued*)
scope, 592
static members of, **627–632**
structS and, 618–619
virtual destructors and, 772
class diagrams.
See also UML (Unified Modeling Language)
binary trees and, 1268
composition and, 686–687, 689
courseType class and, 696
data abstraction and, 618
described, **587–588**
graphs and, 1320, 1343
inheritance and, 664–667, 676–677
linked lists and, 977, 980–981, 988, 997, 1013
operator overloading and, 838–839, 846
personType class and, 626
queues and, 1111, 1131–1132, 1135, 1138
stacks and, 1057, 1059–1060, 1078
class keyword, 585
class instances. *See* objects
class member(s).
See also private class members; protected class members; public class members
accessing, 589–590
described, **584–585**
functions, implementation of, 593–598
pointers and, 737–738
class objects. *See* objects
class templates.
See also templates
described, **859**
overview, 861–869
classes (listed by name)
arrayClass class, 824–825
baseClass class, 663, 675, 766–768, 770–771
bClass class, 682–683, 684
binaryTreeType class, 1268, 1270–1271, 1293, 1294
boxType class, 666–679
bSearchTreeType class, 1294, 1299
candidateType class, 1225–1232
cAssignmentOprOverload class, 826–827
circle class, 661, 662, 772
classIllusFriend class, 804, 806
classTest class, 850–851
clockType class, 586–623, 660–661, 796, 836–844
courseType class, 696, 704–706
cout class, 680
customerType class, 1130–1131, 1036
cylinderType class, 692–693
dClass class, 683, 684, 685
deque class, 1369–1373
derivedClass class, 663, 675, 766–767, 769, 770–771
divByZero class, 902–903
divisionByZero class, 903–905, 908–910
doDivision class, 910
doublyLinkedList class, 1011–1022
dummyClass class, 612–613
dummyExceptionClass class, 901
elemType class, 1265–1270, 1293
ellipse class, 772
employeeType class, 774–775
exception class, 898–989, 901
extClockType class, 660
fullTimeEmployee class, 774
graphType class, 1319–1320
ifstream class, 680, 681
illustrate class, 627–632
invalid_argument class, 898
ios class, 680
istream class, 131, 680, 681
LinkedListClass class, 982
linkedListIterator class, 976–978
linkedListType class, 976, 981–982, 986, 975, 978, 1004–1005, 1103
linkedQueueType class, 1126–1127
linkedStackType class, 1078–1088
list class, 1373–1380
listType class, 618, 861–868, 1352
logic_error class, 898
msTreeType class, 1343
myClass class, 590, 628
myException class, 915
newString class, 850, 853–858, 864
nodeType class, 976
opOverClass class, 807
orderedArrayListType class, 1173–1182
orderedLinkedList class, 975, 981, 996–1010
ostream class, 131, 680
ostream_iterator class, 1386
out_of_range class, 898–899
overflow_error class, 898
partTimeEmployee class, 660–661, 675–677, 774, 779–780
personType class, 625–626, 660, 675, 677–679, 698–699, 774, 801–804, 1036, 1222–1223
pointerDataClass class, 756, 758–765
printResult class, 1099–1102
queueADT class, 1110–1111, 1117–1118, 1122–1127
queueType class, 1141
rectangle class, 661, 772
rectangleType class, 664–666, 668–679, 808–811, 817–835
runtime_error class, 898
serverType class, 1133–1135
shape class, 772–780
stackADT class, 1056–1058, 1077–1078
stackType class, 1058–1061

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

studentType class, 699–701, 706–707, 736–737
test class, 799
testAddress class, 782–783
testClass class, 608–609, 611
thisPointerClass class, 800
underflow_error class, 898
unorderedArrayListType class, 1162, 1188
unorderedLinkedList class, 975, 981, 987–996, 1088–1089, 1162, 1127–1128, 1217, 1319
unorderedListType class, 995–996
vector class, 1353–1358
videoBinaryTree class, 1300–1302
videoListType class, 1029–1036
videoType class, 1027, 1032, 1296–1300
waitingCustomerQueueType class, 1141–1142
weightedGraphType class, 1328–1329

classifyNumber function, 384
classIllusFriend class, 804, 806
classTest class, 850–851
clear function, 133–135, 144, 1363, 1364, 1508
clearGraph function, 1322
clients, **598**
climits (limits.h) header file, 1502–1503
clockType class, 586–623, 660–661, 796, 836–844
clockType.h header file, 620–622
cmath (math.h) header file, 325, 1504–1505
COBOL, 3, 9
code detection, 518–525
collating sequence, **38**, 172–174
column processing, **505–509**
comma (,), 32–33, 37, 84
comments, **32**, 86
common
  input, **116–117**
  output, **116–117**
compareCode function, 520–521
compareThree function, 318–323
comparion (relational) operators
  described, **551–552**
  records and, 551–552, 553
comparison tree, **1201–1202**
compiler(s)
  cc command and, 624
  comments and, 32
  described, **10**
  errors, 83
  functions and, 325
  objects and, 79–80, 588
  preprocessor directives and, 12–13
  string data type and, 442, 516
  syntax errors and, 12–13
  variables and, 376
compile-time binding, **768**
complex numbers
  described, **845**
  manipulating, 845–850
components, use of the term, **1316**
composition
  constructors and, 689–690
  described, 660, 685–690
  overview, 659–726
compound statements, **191–192**
computer(s)
  language of, 6–8
  history of, 2–6
  systems, elements of, 3–8
computer programs. *See* programs
conditional
  expressions, **203–204**
  operator, **203–204**
connected
  use of the term, **1316**
  vertices, **1315**
const keyword, 356, 483, 587, 592, 599, 817
constructor(s). *See also* default constructors
  arrays of, 613–615
  composition and, 689–690
  described, **604–606**
  exception handling and, 915
  function overloading and, 858
  graphs and, 1134
  inheritance and, 670–679
  invoking, 606–610
  linked lists and, 982, 986, 1014
  parameters and, 607–612
  precautions regarding, 612–613
  properties of, 605
  queues and, 1121–1122
  stacks and, 1066–1067, 1086
  STL and, 1362–1363
container(s)
  adapters, **1393–1396**
  associative, **1386–1393**, 1397
  declaring iterators to, 1358–1359
  described, **1352**
  functions and, 1359–1364
  header files and, 1386–1397
  iterators and, 1386–1397
  types, 1353
continue statements, **278–281**
control structures. *See also* repetition
  checking account balances and, 250–259
  described, **158–159**
  divisibility tests and, 276–278
  Fibonacci numbers and, 259–264
  nested, 241–243, 270, 281–288
  number-guessing games and, 244–247, 249–250
  overview, 167–229
  recursion and, 940–941
  relational operators and, 169–174
  short-circuit evaluation and, 181–182
conversion constructor, 856
CONVERSION memory space, **52–53**
convertEnum function, 428, 430–431
copy algorithm, 1365–1368

CENGAGE brain .com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

copy constructor, 836, 1362
  described, **758–765**
  linked lists and, 981, 986
  stacks and, 1066, 1067, 1071
copy function, 1365–1368
copyList function, 982, 985–986, 1014
copyStack function, 1066, 1085–1086
copyText function, 527–528
copyTree function, 1270–1272
cos function, 309
count function, 1438–1442
count variable, 629
counter-controlled while loops, **236–238**
count_if function, 1438–1442
courseScore variable, 358–359
courseType class, 696, 704–706
cout class, 680
cout keyword, 33, 69
cout statement, 130, 926
  described, **78**
  header files and, 76
  preprocessor directives and, 116–117
cout variable, 497
cpp file extension, 12, 79
CPU (central processing units)
  described, **4–5**
  history of, 3
createGraph function, 1321
createSpanningGraph function, 1343
createVideoLsit function, 1037
createWeightedGraph function, 1328–1329
cstddef (stddef.h) header file, 1505
cstdlib header file, 1510
c_str function, 498–499
cstring (string.h) header file, 852, 1505–1508
C-strings (array characters)
  described, **492–499**, 514–516
  header files and, 852, 1505–1508
  operator overloading and, 852–858
ctype.h. *See* cctype (ctype.h) header file
CU (control unit), **4–5**
curly braces ({}), 80, 82, 84
customerNumber variable, 1130
customer, use of the term, 1129
customerType class, 1130–1131, 1036
cycle, use of the term, **1316**
cylinders, dimensions of, 692–693
cylinderSurfaceArea operation, 693
cylinderType class, 692–693
cylinderVolume operation, 693

D

data.
  See also data abstraction; data types
  comparison, 388–398
  formatted, **1472**
  raw, **1472**
data abstraction
  described, **616–618**
  overview, 583–658
data types.
  See also specific data types
  anonymous, **424–425**
  arithmetic operators and, 40–45
  control structures and, 171
  conversion, 48–50
  described, **34–50**
  extraction operator and, 118–119
  simple, 35–38, 171, 415–468
  user-defined, 415–468
DBL_MAX constant, 1134
dClass class, 683, 684, 685
debugging.
  See also errors
  control structures and, 196–199
  variables and, 66
decimal numbers
  conversion and, 941–949, 1477–1482
  described, **8**, 1477
decimal point (.), 130–132, 140
decision makers, **185**, 233
declaration statements, **80**
decrement operators, **67–69**, 829–830
decToBin function, 949
deep copy, **752–754**
default constructors.
  See also constructors
  binary trees and, 1268
  described, **605**
  invoking, 606–607, 611
  linked lists and, 982, 1014
  stacks and, 1079
default keyword, 204
default parameters, 380–382, 610–612, 674.
  See also parameters
delete function, 1271, 1279–1287
delete keyword, 739
delete operator, 739, 741–743, 755, 982
deleteFromTree function, 1284–1286
deleteNode function, 981, 987, 990–995, 1004–1005, 1021–1022, 1286–1287
deleteQueue function, 1109–1115, 1120–1121, 1125, 1141
departureTimeEmp array, 613–615
depth first traversal, **1323–1325**
depthFirstTraversal function, 1325
deque class, 1369–1373
deque sequence container, **1369–1373**, 1397
dereferencing operator, **730–736**, 1385
derived classes.
  See also inheritance
  class members and, 662–663
  constructors and, 670–679
  described, 660–661
  header files and, 677–679
  protected members in, accessing, 683–685
derivedClass class, 663, 675, 766–767, 769, 770–771

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

derivedClass.h header file, 769–771
destination, use of the term, **1313**
destroy function, 1271
destroyList function, 759, 982, 986
destructors
  described, **615–616**
  graphs and, 1334
  linked lists and, 986
  operator overloading and, 836
  pointers and, 755–756
  queues and, 1121–1122
  stacks and, 1066–1067, 1086
  virtual, **772**
diagonals, reversing, 509–511
different formal parameter lists, **378–379**
digital signals, **6–7**
directed edge, **1252**
directed graphs (digraphs), **1313**, 1315
displayResults function, 428, 431–432
displayRules function, 427–428
divByZero class, 902–903
divideList function, 1213–1214, 1217
divisibility tests, 276–278
divisionByZero class, 903–905, 908–910
divisors, 276–278
do keyword, 274
documentation, 86
doDivision class, 910
doDivision function, 906–907
domains, **617**
dot notation, **130–131**
double data type, 39, 40
  memory and, 54
  functions and, 309, 316, 351
  templates and, 859
double quotes ("), 51, 75, 443
doubleFirst function, 369
doublyLinkedList class, 1011–1022
do. . .while loops.
  See also loops
  break statements and, 278–281
  overview, 274–278
  recursion and, 940–941
  sorting algorithms and, 1197–1199
dummyClass class, 612–613
dummyExceptionClass class, 901
dynamic arrays.
  See also arrays
  described, **475**, 745
  overview, 745–751
dynamic binding. *See* run-time binding
dynamic variables
  described, **739**
  overview, 739–743

E

EBCDIC (Extended Binary-Coded Decimal Interchange Code), 8, 38, 492, 1472–1473
edge(s)
  directed, **1252**
  origin of, **1313**
  parallel, **1315**
  set of, **1313**
  weight of, **1327**, 1339–1342
Eiffel, 692
election results program, 1220–1242
elemType class, 1265–1270, 1293
ellipse class, 772
else statements, pairing, with if statements, **193–194**
employeeType class, 774–775
empty function, 1362, 1383–1384, 1395, 1507
encapsulation, **691**
end function, 1359–1362, 1363
#endif preprocessor directive, 679
enumeration types
  arrays and, 503–507
  defining, 416–417
  described, **35–38**, 416
  functions and, 422–424
  input/output of, 420–422
  operations on, 419
  overview, 416–437
enumerators, **417**
eof function, **248–249**
EOF-controlled while loops, **247–248**, 284–287
equal sign (=), 32, 1094
equality operator, 202–203
equalTime function, 586–599
erase function, 1363, 1364, 1390, 1508
errors.
  See also debugging; exception handling; syntax errors
  control structures and, 211, 212
  information hiding and, 619
  variables and, 63
escape (\) character, 70, 75–76, 148
evaluateExpression function, 1095–1096
evaluateOpr function, 1097–1099
exabytes (EB), **7**
exception(s).
  See also exception handling
  classes, 897–911
  rethrowing, 907–911
  throwing, 891–892, 901–911
exception class, 898–989, 901
exception handling.
  See also errors; exceptions
  fixing errors and continuing, 912–913
  logging errors and continuing, 914
  mechanisms of, 890–912
  overview, 885–922
  within programs, 886–901
  stack unwinding and, 914–917
  techniques, 912–914
exclamation point (!), 32
executable statements, **80–81**
execution
  flow of, 325
  time, specifying input/output files at, 498
exp function, 309
expr variable, 315
expressions.
  See also specific types
  integral, **45–48**
  mixed, **46–47**
  saving and using the value of, 58–59

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

extClockType class, 660
extern keyword, 374
external variables, **374**
extraction operator, **117–123**, 680, 816–821

F

fabs function, 309
fact function, 926
fail state, **132**
Fibonacci numbers, **259–264**, 932–936
file(s)
  access, random, 1488–1492
  closing, 148
  I/O, 146–149, 1482–1496
  -name extensions, **12**
  opening, **147**, 149
  specifying, at execution time, 498
  stream variables, 148
fill function, 144, 751, 1408–1410
fill_n function, 1408–1410
fillNames function, 1232–1233
find function, 449–451, 1412–1417, 1507
find_end function, 1412–1417
find_first_of function, 1412–1417
find_if function, 1412–1417
first variable, 369, 981
five object, 611–612
fixed manipulator, 136, 137–138
flag variables, **244**
flag-controlled while loops, **243–244**
float data type, 39, 40
float.h header file. *See* cfloat (float.h) header file
floating point data type
  arithmetic operators and, 43
  control structures and, 171–172
  described, **35–40**
  type conversion and, 48–50
floating-point expressions, **45–48**
floor function, 309, 310
for keyword, 265
for loop(s).
  See also loops
  arrays and, 475–476, 506–508, 749–750
  break statements and, 278–281
  enumeration types and, 320
  functions and, 348–349, 352, 371
  nested loops and, 281–288
  operator overloading and, 827
  overview, 264–270
  pointers and, 765
  recursion and, 940–941
  search algorithms and, 1164, 1172, 1181
  sorting algorithms and, 1191, 1200
  STL and, 1358
for loop control variable, **265**
for_each function, 1422–1444
formal parameters, **313–314**.
  See also parameters
formatted data, **1472**
FORTRAN, 3, 9
forward slash (/), 32, 1094, 1095
four object, 611–612
fParamFunc1 function, 1293
fParamFunc2 function, 1293
friend functions
  definition of, 805–807
  described, **804**
  operator overloading and, 835
friend keyword, 805
front, of queues, 1109
front function, 984, 1110, 1119–1120, 1370, 1375, 1395
fstream header file, 146, 681
fullTimeEmployee class, 774
funcExp function, 380–381
function(s).
  See also functions (listed by name)
  abstract, 981
  accessor, 598–602
  arrays and, 482–491, 511–514
  calls, 123
  data types/return types of, 314
  definitions, 313
  described, **30**
  dot notation and, 130–131
  enumeration types and, 422–424
  flow of execution and, 325
  headings (headers), **312–313**
  implementation of, 593–598
  names, 346
  nesting, 370
  objects, **1400–1406**
  overloading, **378–380**, 858
  overriding, 663–670
  overview, 307–414
  as parameters, 1288–1305
  pointers and, 748
  predefined/standard, **30**, 123–131, 308–311
  prototypes, **318–323**, 325, 380
  records and, 552–553
  return values, 748
  signatures, **378**
  templates, **859–861**
  value-returning, **312–313**, 314–315, 370
  void, **312**, 346–354
functionA function, 916
functionB function, 916
functionC function, 916, 917
functions (listed by name).
  See also functions; main function
  abs function, 309, 312, 313
  accumulate function, 1453–1458
  addFirst function, 369
  addQueue function, 1109–1111, 1114, 1116, 1120, 1124, 1127, 1141–1142
  addressOfX function, 782
  addVotes function, 1237–1238
  adjacent_difference function, 1453–1458
  adjacent_find function, 1404, 1431–1435
  assert function, 211–218, 853, 886, 889, 984
  assign function, 1370, 1375
  at function, 1370

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

back function, 984, 1370, 1375, 1395, 1110, 1120–1121
begin function, 984, 1359–1363
billingAmount function, 704
binary_search function, 1427–1430
binSearch function, 1173, 1188
binToDec function, 945
breathFirstTraversal function, 1326–1327
bubbleSort function, 1188
buildListBackward function, 974, 975
buildListForward function, 973, 975
calculate function, 390
calculateAverage function, 390–392
calculatePay function, 774
callPrint function, 767–768
ceil function, 309
characterCount function, 528
classifyNumber function, 384
clear function, 133–135, 144, 1363, 1364, 1508
clearGraph function, 1322
compareCode function, 520–521
compareThree function, 318–323
convertEnum function, 428, 430–431
copy function, 1365–1368
copyList function, 982, 985–986, 1014
copyStack function, 1066, 1085–1086
copyText function, 527–528
copyTree function, 1270–1272
cos function, 309
count function, 1438–1442
count_if function, 1438–1442
createGraph function, 1321
createSpanningGraph function, 1343
createVideoLsit function, 1037
createWeightedGraph function, 1328–1329
c_str function, 498–499
decToBin function, 949
delete function, 1271, 1279–1287
deleteFromTree function, 1284–1286
deleteNode function, 981, 987, 990–995, 1004–1005, 1021–1022, 1286–1287
deleteQueue function, 1109–1115, 1120–1121, 1125, 1141
depthFirstTraversal function, 1325
destroy function, 1271
destroyList function, 759, 982, 986
displayResults function, 428, 431–432
displayRules function, 427–428
divideList function, 1213–1214, 1217
doDivision function, 906–907
doubleFirst function, 369
empty function, 1362, 1383–1384, 1395, 1507
end function, 1359–1362, 1363
eof function, 248–249
equalTime function, 586–599
erase function, 1363, 1364, 1390, 1508
evaluateExpression function, 1095–1096
evaluateOpr function, 1097–1099
exp function, 309
fabs function, 309
fact function, 926
fill function, 144, 751, 1408–1410
fill_n function, 1408–1410
fillNames function, 1232–1233
find function, 449–451, 1412–1417, 1507
find_end function, 1412–1417
find_first_of function, 1412–1417
find_if function, 1412–1417
floor function, 309, 310
for_each function, 1422–1444
fParamFunc1 function, 1293
fParamFunc2 function, 1293
front function, 984, 1110, 1119–1120, 1370, 1375, 1395
funcExp function, 380–381
functionA function, 916
functionB function, 916
functionC function, 916, 917
functionSeven function, 378
functionSix function, 378
functionXYZ function, 379
funcValueParam function, 355
funExample function, 374
funOne function, 362–364, 375
funTwo function, 364
gameResult function, 427–430
generate function, 1410–1412
generate_n function, 1410–1412
get function, 125–126, 128, 130–131, 144
getCredits function, 704
getData function, 567–569
getFirstName function, 802–803
getFreeServerID function, 1139
getGpa function, 701
getHoursEnrolled function, 704
getLastName function, 802–803
getLength function, 669
getline function, 145, 287
getNumber function, 383–384, 387
getNumberOfBusyServers function, 1139
getScore function, 359
getStudentData function, 707–708

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

functions (listed by name). (*continued*)
getTime function, 586–587, 595–596, 841
getWidth function, 669
ignore function, 126–128, 144
incrementHours function, 586–587, 598
incrementMinutes function, 586–587, 596, 598
incrementSeconds function, 586–587, 596, 598
initialize function, 383, 385, 527, 566
initializeList function, 983, 1015, 1089
initializeQueue function, 1109–1110, 1116, 1119, 1124–1127
initializeStack function, 1056, 1061–1062, 1080, 1088
inner_product function, 1453–1458
inorderTitle function, 1301
inorderTraversal function, 1295
insert function, 1359–1360, 1363–1364, 1389, 1508
insertFirst function, 975, 981, 987–990, 1003–1004
insertItem function, 1277
insertLast function, 975, 981, 987, 990–995, 1003–1004
insertNode function, 1003–1004
isEmpty function, 1268, 1320–1321
isEmptyList function, 982, 1089
isEmptyQueue function, 1109–1110
isEmptyStack function, 1056, 1089
isFullQueue function, 1109–1110, 1124
isFullStack function, 1056, 1062, 1078, 1080, 1089
islower function, 312
isNumPalindrome function, 323–325
iter_swap function, 1424–1427
larger function, 316–323, 326–327, 379–380
largerChar function, 379
largerInt function, 379
largestInRows function, 512, 513–514
leavesCount function, 1270
length function, 446–448, 1507
max function, 326–327, 1438–1442
max_element function, 1438–1442
maxSaleBy Quarter function, 571–572
maxSaleByPerson function, 571
max_size function, 1363
merge function, 1376–1377, 1431–1435
mergeList function, 1215–1217, 1219, 1220
mergeSort function, 1217
min function, 1438–1442
min_element function, 1438–1442
minimalSpanning function, 1343–1344
minLocation function, 1192, 1193
move function, 831
nodeCount function, 1270
partial_sum function, 1453–1458
partition function, 1207, 1208, 1209
peek function, 128–130
pigLatinString function, 456–457
pop function, 1055–1056, 1064–1066, 1080, 1083–1085, 1383–1384, 1395
pop_back function, 1373
pop_front function, 1370, 1375
pow function, 309, 310, 312, 325
print function, 668, 671, 675, 696, 701, 703–704, 751, 766, 768–769, 771, 774, 802–804, 820, 983, 1015
printArray function, 490
printGrade function, 359
printGradeReports function, 708
printHeading function, 398
printListReverse function, 1010
printMatrix function, 511–514
printReport function, 570–571
printResults function, 384–385, 391–393, 397, 438, 1239–1240
printShortestDistance function, 1334
printStars function, 348–349, 351–352
printTime function, 586–587, 590, 599, 842
printTreeAndWeight function, 1344–1345
push function, 1055–1056, 1062–1064, 1080–1083, 1383–1384
push_back function, 1358, 1364
push_front function, 1370, 1375
putback function, 128–130
rand function, 1510
random_shuffle function, 1438–1442
rbegin function, 1363, 1366
readCode function, 519–520
readIn function, 553
recMergeSort function, 1217, 1218
recQuickSort function, 1209
remove function, 1375, 1417–1420
remove_copy function, 1417–1420
remove_copy_if function, 1417–1420

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

remove_if function, 1417–1420
rend function, 1363
replace function, 1420–1424, 1508
replace_copy function, 1420–1424
replace_copy_if function, 1420–1424
replace_if function, 1420–1424
resize function, 1364
retrieveAt function, 1237
retrievePlay function, 427–429
reverse function, 1377, 1435–1438
reverse_copy function, 1435–1438
rFibNum function, 932, 934–936
rotate function, 456, 1435–1438
rotate_copy function, 1435–1438
runSimulation function, 1144–1145
saleByQuarter function, 569
search function, 981, 987, 989, 1276–1277, 1427–1430
search_n function, 1427–1430
seekg function, 1489–1490
seekp function, 1489–1493
selectionSort function, 1192
setCourseInfo function, 696–697
setCustomerInfo function, 1132–1133
setData function, 683, 684, 687, 688
setDimension function, 669
setInfo function, 701
setName function, 802–803
setProduct function, 639–640
setServerBusy function, 1139
setSimulationParameters function, 1143
setTime function, 586–587, 594–595, 604–605, 841
setX function, 738
shortestPath function, 1333–1334
showSelection function, 638
size function, 448–449, 1362, 1383–1384, 1395, 1507
sort function, 1188, 1233–1234, 1376, 1427–1430
sortCourses function, 701, 705
splice function, 1376
square root function, 309
squareFirst function, 369
strcmp function, 493–495
strcpy function, 493–495
strlen function, 493–495
substr function, 452–454, 957, 1507
sumRows function, 512, 513–514
swap function, 454–460, 1192, 1193, 1363, 1424–1427, 1507
swap_ranges function, 1424–1427
tail recursive function, 927
testCopyConst function, 765
tolower function, 310
top function, 1055–1056, 1064, 1083, 1080, 1383–1384
totalSaleByPerson function, 569–570
toupper function, 310, 312
transform function, 1422–1444
unique function, 1375
updateServers function, 1140
updateXYZ function, 800–801
validSelection function, 427–428
what function, 898–989
winningObject function, 428, 431
write function, 1488
WriteTotal function, 528
functionSeven function, 378
functionSix function, 378
functionXYZ function, 379
funcValueParam function, 355
funExample function, 374
funOne function, 362–364, 375
funTwo function, 364

G

gameResult function, 427–430
general case, **924**, 929
generate function, 1410–1412
generate_n function, 1410–1412
get function, **125–126**, 128, 130–131, 144
getCredits function, 704
getData function, 567–569
getFirstName function, 802–803
getFreeServerID function, 1139
getGpa function, 701
getHoursEnrolled function, 704
getLastName function, 802–803
getLength function, 669
getline function, 145, 287
getNumber function, 383–384, 387
getNumberOfBusyServers function, 1139
getScore function, 359
getStudentData function, 707–708
getTime function, 586–587, 595–596, 841
getWidth function, 669
gigabytes (GB), **7**
global variables, **374–376**
GPA (grade point averages), 194–195, 546–547, 552–553, 693–714, 736–739, 1072–1076
GPS (global positioning satellites), 3
grade report programs. *See* GPAs (grade point averages)
graph(s).
See also directed graphs (digraphs)
as ADTs, 1319–1322
definitions, 1313–1316
introduction to, 1312–1313
notations, 1313–1316
operations on, 1318–1319
overview, 1311–1350
representation, 1316–1318

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

graph(s). (*continued*)
traversals, 1323–1327
undirected, **1313**, 1314
weighted, **1327**
graphType class, 1319–1320
greater-than sign (>), 32
greedy algorithms. *See* shortest path algorithms

H

Hamblin, Charles L., 1090
has-a relationships. *See* composition
head, 960–961
header files.
See also specific files
control structures and, 212
derived classes and, 677–679
described, **77**, 620
functions and, 123–124
I/O and, 123–124, 139, 141, 144, 146
linked lists and, 995–996, 1005–1006
manipulators and, 139, 141, 144
multiple inclusions of, 678–679
namespaces and, 78, 437, 440–441
naming conventions and, 1496–1497
overview, 1499–1510
stacks and, 1068–1072
STL and, 1386–1397
string data type and, 516
templates and, 863
height variable, 670, 673
hexadecimal numbers, 1480–1482
high-level languages, **9**
Hollerith, Herman, 2
hr variable, 586, 588, 594, 597, 610–611, 796
H_test identifier, 679
hyphen (-), 32, 1094, 1095

I

IBM (International Business Machines), 2, 3, 38
identifiers
described, **33–34**
functions and, 370–374
global, 370–374
legal, 34
local, **370–374**
naming, 84–85
namespaces and, 442
nested, 371
scope of, 370–374
self-documenting, **85**
if statements, 824, 1009, 1192
described, **184–204**
input failure and, 199–201
nested, **192–197**
recursion and, 948
if. . .else statements, **184–204**, 241–243, 359
#ifndef preprocessor directive, 679
ifstream class, 680, 681
ifstream header file, 680
ignore function, 126–128, 144
illustrate class, 627–632
immediate successors, **1319**
implementation
executable code and, 623–628
described, **620–621**, 863
implicit type conversion, **48**
incident, use of the term, **1315**
#include preprocessor directive, 77–78, 116–117, 147, 679, 677
increment operators, **67–69**, 796–797, 829–830
incrementHours function, 586–587, 598
incrementMinutes function, 586–587, 596, 598
incrementSeconds function, 586–587, 596, 598
indirection operator. *See* dereferencing operator
infile variable, 248, 499
infinite loops, **234**
infix notation, **1090**
information hiding, **619–623**

inheritance
constructors and, 670–679
described, **660**, 691
hierarchy, 674, 676
overview, 659–726
multiple, **661**
pointers and, 765–772
private, 661, 662, 668, 681–682
protected, 661, 681–685
public, 662, 663, 667–668, 681–682
single, **661**
stream classes and, 680
initialize function, 383, 385, 527, 566
initialize statement, 371
initializeList function, 983, 1015, 1089
initializeQueue function, 1109–1110, 1116, 1119, 1124–1127
initializeStack function, 1056, 1061–1062, 1080, 1088
inner_product function, 1453–1458
inorder sequence, **1261–1263**
inorder traversal, 1288–1290, 1295
inorderTitle function, 1301
inorderTraversal function, 1295
input.
See also input/output (I/O)
devices, **6**
failure, **122**, 131–133, 199–201
overview, 51–66
statements, 60–66
stream variables, 130–131, 199
input/output (I/O).
See also input
devices, standard, **116–123**
file, 146–149, 1482–1496
formatting output and, 135–140
overview, 110–166
predefined functions and, 123–131
records and, 552
stream classes and, 680–681

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

streams, **116–123**
string data type and, 145, 498–499
insert function, 1359–1360, 1363–1364, 1389, 1508
insert iterators, **1406–1408**
insertFirst function, 975, 981, 987–990, 1003–1004
insertion operator, 680
insertItem function, 1277
insertLast function, 975, 981, 987, 990–995, 1003–1004
insertNode function, 1003–1004
instance variables, **598**
instances. *See* objects
int data type, 36–37, 585
arrays and, 485–487, 512, 749, 751
control structures and, 182–183
dynamic arrays and, 745–648
exception handling and, 891, 894, 908
functions and, 322–323, 351
inheritance and, 683
linked lists and, 970
memory and, 54
operator overloading and, 799
pointers and, 728, 729, 752
queues and, 1130
records and, 559–560
templates and, 859, 862
variables and, 66, 68
integral data type, **35–38**, 491
integral expressions, **45–48**
intersection, of sets, **1313**
intList variable, 554–555
invalid_argument class, 898
I/O. *See* input/output (I/O)
iomanip header file, 139, 141
ios class, 680
iostream header file, 77, 78, 144, 680
IR (instruction register), **5**
is-a relationships. *See* inheritance
isEmpty function, 1268, 1320–1321
isEmptyList function, 982, 1089
isEmptyQueue function, 1109–1110
isEmptyStack function, 1056, 1089
isFullQueue function, 1109–1110, 1124
isFullStack function, 1056, 1062, 1078, 1080, 1089
islower function, 312
isNegative variable, 279
isNumPalindrome function, 323–325
ISO (International Standards Organization), 24.
See also ANSI/ISO Standard C++
istream class, 131, 680, 681
istream data type, 248
istream header file, 146
istream member functions, **123**
iter_swap function, 1424–1427
iteration, 940–941.
See also iterators
iterators.
See also iteration
bidirectional, 1382–1383
described, **976**
forward, 1382
input, 1380–1381
linked lists and, 976–983
output, 1381
random access, 1383–1384
STL and, 1358–1359, 1366–1368, 1380–1397, 1406–1408
stream, 1386

J

Jacquard, Joseph, 2
Java, 9, 692
Jobs, Steven, 3

K

keys, 1163
keywords.
See also keywords (listed by name)
described, **33**
list of, 1467
keywords (listed by name).
See also keywords
break keyword, 204, 205–207
case keyword, 204, 206, 208–210
cin keyword, 33
class keyword, 585
const keyword, 356, 483, 587, 592, 599, 817
cout keyword, 33, 69
default keyword, 204
delete keyword, 739
do keyword, 274
extern keyword, 374
for keyword, 265
friend keyword, 805
namespace keyword, 438
new keyword, 739
private keyword, 585
protected keyword, 585
public keyword, 585
static keyword, 376
struct keyword, 547
switch keyword, 204
this keyword, 799
using keyword, 439
virtual keyword, 768
void keyword, 346
while keyword, 233
kilobytes (KB), **7**
Kruskal's algorithm, 1337

L

larger function, 316–323, 326–327, 379–380
largerChar function, 379
largerInt function, 379
largestInRows function, 512, 513–514
last variable, 981
leaf, 1201–1202, 1257
leavesCount function, 1270
left
manipulator, 143–144
subtree, **1252**
Leibniz, Gottfried von, 2
length, converting, 89–93
length function, 446–448, 1507
length variable, 668, 670, 673
less-than sign (<), 32
letter counts, 525–532
level, of nodes, **1257**

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

level-by-level (breadth-first) traversal, **1325–1327**, 1264
libraries.
  See also STL (Standard Template Library)
  described, **13**
  header files and, 77
LIFO (Last In First Out) data structure, **1055**, 1058, 1352
line counts, 525–532
linked list(s).
  See also lists
  ADTs and, 964–965, 974–987, 996–997, 1011–1012
  -based lists, 1210–1242
  basic operations on, list of, 975
  building, 970–975
  circular, **1022–1023**
  copying, 985–986
  deleting items in, 964–965, 968–969
  described, **960–987**
  destroying, 982, 1014
  doubly, 1011–1022
  first/last elements of, 1016–1022
  initializing, 983, 1015
  inserting items in, 964–968
  iterators, **976–983**
  length of, 983, 1015
  ordered, 996–1010
  overview, 959–1051
  printing, 983, 1008–1010, 1015–1016, 1102–1109
  properties, 961–987
  searching, 988–989, 998, 1016
  traversing, 964
  unordered, 987–996
LinkedListClass class, 982
linkedlist.h header file, 995–996, 1004–1005
linkedListIterator class, 976–978
linkedListType class, 976, 981–982, 986, 975, 978, 1004–1005, 1103
linkedQueueType class, 1126–1127
linkedStack.h header file, 1087–1088
linkedStackType class, 1078–1088
linkers, **13**
links, **960–961**
list(s).
  See also linked lists
  array-based, 1189–1210
  dividing, 1212–1214
  linked list-based, 1210–1242
  merging, 1214–1217
  recursion and, 928–940
list class, 1373–1380
list sequence container, **1373–1380**, 1397
listType class, 618, 861–868, 1352
loaders, **13**
local declarations, **316**
local variables.
  See also variables
  data comparison and, 390
  described, **360**
  functions and, 360–370
logical (Boolean) expressions
  bool data type and, 183–184
  control structures and, 173–184
  described, **170**, 173
  int data type and, 182–183
  typedef statement and, 425–426
logical (Boolean) operators
  control structures and, 175–184
  described, **175**
logic_error class, 898
long double data type, 39
loop(s).
  See also repetition; *specific types*
  arrays and, 475–476
  choosing the right, 278
  control variables (LVCs), **234–236**
  enumeration types and, 420
  EOF-controlled, **247–248**, 284–287
  exiting, 278–281
  flag-controlled, **243–244**
  graphs and, 1315
  linked lists and, 972
  recursion and, 940–941
  search algorithms and, 1171–1172, 1181
  sorting algorithms and, 1187, 1197–1200
  stacks and, 1102–1109
Lukasiewicz, Jan, 1090–1091
LVCs (loop control variables)
  described, **234–236**
  designing loops and, 235–236

M

machine language
  described, **7**
  examples, 9
main function, 82, 319–323, 519–520
  described, **11–12**
  exception handling and, 910, 916
  friend functions and, 806–807
  information hiding and, 620–621
  linked lists and, 1037
  operator overloading and, 800–801
  parameters and, 369
  stacks and, 1095
  variables and, 366, 375, 427
main memory, **5**
make command, **14**, 624
max function, 326–327, 1438–1442
max variable, 316
max_element function, 1438–1442
maxQueueSize variable, 1111, 1118–1119, 1121–1122
maxSaleBy Quarter function, 571–572
maxSaleByPerson function, 571
max_size function, 1363
maxSize variable, 827
megabytes (MB), **7**
member access operator, 589–590, 592
  arrow, **737**
  described, **131**, 549

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

memory
  addresses, **5**, 485–488
  allocating, 52–54, 360–370
  arrays in, 485–488
  cells, **5**
  data types and, 36–40
  dynamic variables and, 739–743
  extraction operator and, 119–120
  functions and, 360–370
  input and, 51–66
  leaks, **742**
  linked lists and, 982
  main, **5**
  peek function and, 128
  pointers and, 730–736
  records and, 547–548
  recursion and, 941
  size, printing, 1509–1510
  static class members and, 628
  variables and, 588
merge function, 1376–1377, 1431–1435
mergeList function, 1215–1217, 1219, 1220
mergeSort function, 1217
min function, 1438–1442
min variable, 586, 588, 594, 597, 610–611, 796
min_element function, 1438–1442
minimal spanning tree, **1335–1345**
minimalSpanning function, 1343–1344
minLocation function, 1192, 1193
mixed expressions, **46–47**
mnemonics, **9**
Modula-2, 692
modular programming, **22**.
  See also functions
modules, 308.
  See also functions
move function, 831
movie ticket sales program, 150–155
msTreeType class, 1343
multidimensional arrays, **500–518**.
  See also arrays
multiset associative container, 1387–1393, 1397
mutator functions, 598–602
myClass class, 590, 628
myClock object, 588–589, 594–598
myException class, 915
myException.h header file, 915–916
myStack.h header file, 1068

N

named constants
  converting length and, 90–91
  described, 52–53
  functions and, 374–376
  making change and, 84–85
namespace keyword, 438
namespaces
  described, **78**
  overview, 437–442
new keyword, 739
new operator, 739, 740–741
newClock.h header file, 837
newline (\n) character, 38, 70, 72–73, 75–76, 125
  string data type and, 145, 396
  text processing and, 526
newString class, 850, 853–858, 864
newStudent variable, 547–548
node(s)
  deleting, 990–991, 1004–1005, 1019–1022
  described, **960–961**, 1201–1202
  inserting, 989–990, 998–1003, 1017–1019
  retrieving data of, 984
  sorting algorithms and, 1201–1202
  structure of, 976
  traversing lists and, 964
nodeCount function, 1270
nodeType class, 976
nonrecursive traversal, 1290–1292
nth Fibonacci number, **260**
null character, 492, 496
null pointers, **739**
null strings, **51**
number(s).
  See also binary numbers; decimal numbers
  classifying, 270–273, 383–387
  converting, 1477–1482
  creating grids of, 282
  determining the largest, 326–327
  divisibility tests, 276–278
  Fibonacci, **259–264**, 932–936
  -guessing games, 20, 244–250
  hexadecimal, 1480–1482
  palindrome, 323–325

O

obj file extension, 79–80
object(s).
  See also OOD (object-oriented design); OOP (object-oriented programming)
  accessing class members and, 589–590
  arrays of, 613–615
  code, saving, 79–80
  declaring, 588–589
  described, **588**
  identifying, 692–693
  programs, **13**
objectOne object, 755–758, 759, 760
objectThree object, 758, 760
objectTwo object, 756–758
ofstream header file, 146
ofstream variable, 360
OOD (object-oriented design).
  See also objects
  described, **22–24**, 584
  inheritance and, 772
  overview, 690–714
  principles of, 691
OOP (object-oriented programming).
  See also objects
  described, **23**
  overview, 690–714
  STL and, 1352

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

open statement, 498
operands, **41**
operating systems, **6**
operations
  described, **617**
  identifying, 692–693
operator(s).
  See also operator overloading
  arithmetic, **40–45**, 744–745
  binary, **41**, 810–816
  cast, **48**, 66
  decrement, **67–69**, 829–830
  increment, **67–69**, 796–797, 829–830
  logical (Boolean) described, **175–184**
  precedence, **40–45**, 177–181, 1469–1470
  relational, **169–174**, 178–179, 419–420
  ternary, **203**
  unary, **41**, 829–835
operator overloading
  binary operators and, 810–816
  described, **797–798**
  member versus nonmember, 835–836
  need for, 796–797
  overview, 795–883
  restrictions, 798
  stacks and, 1067–1068, 1086–1090
  table showing, 1475
  this pointer and, 779–804
opOverClass class, 807
orderedArrayListType class, 1173–1182
orderedLinkedList class, 975, 981, 996–1010
origin, of edges, **1313**
ostream class, 131, 680
ostream_iterator class, 1386
otherClock parameter, 593
out_of_range class, 898–899
output.
  See also I/O (input/output)
  devices, **6**, 69
  formatting, 135–140
  overview of, 69–76
  statements, 10–11, 56
overflow condition, **1064**
overflow_error class, 898

P

p variable, 729
palindrome numbers, 323–325
parallel
  arrays, **499–500**
  edges, **1315**
parameter(s)
  actual, 367–368
  arrays and, 482–485, 488–491, 511–514
  binary trees and, 1271–1272, 1288–1305
  classes and, 592–593
  constructors and, 607–612
  default, 380–382, 610–612, 674
  exception handling and, 905, 916
  formal, **313–314**
  friend functions and, 806
  functions and, 313–317, 346–354, 378–381
  graphs and, 1324
  linked lists and, 986, 1009–1010
  lists, 556, 667
  operator overloading and, 817, 822
  pointers and, 748, 765–767
  queues and, 1143
  records and, 556
  reference, **350–351**, 367–370, 592–593
  sorting algorithms and, 1188, 1208
  stacks and, 1096
  templates and, 859–863
  value, **350–351**, 354–355, 360–370
  virtual functions and, 770
parameterized stream manipulators, **144**
parameterized types, **861**.
  See also class templates
parent, **1252**, 1257
parentheses, 44–45, 181, 185, 187
partial_sum function, 1453–1458
partition function, 1207, 1208, 1209
partTimeEmployee class, 660–661, 675–677, 774, 779–780
partTimeEmployee.h header file, 677–678
Pascal, 9, 692
Pascal, Blaise, 2
Pascaline, 2
path
  binary trees and, 1257
  described, **1202**
  graphs and, 1315–1316
  length of, **1257**
  simple, **1315–1316**
  sorting algorithms and, 1202
  weight of the, 1327
PC (program counter), **5**
peek function, 128–130
period (.), 32, 548, 589
personType class, 625–626, 660, 675, 677–679, 698–699, 774, 801–804, 1036, 1222–1223
personType.h header file, 677–679
petabytes (PB), **7**
pig latin strings, 454–460
pigLatinString function, 456–457
plus sign (+), 32, 1094, 1095
pointer variables
  classes and, 736–739
  constructors and, 760
  declaring, 728–729
  described, **728–745**
  initializing, 739
  operations on, 743–745
  operator overloading and, 824, 836
  structS and, 736–739
pointerDataClass class, 756, 758–765
pointers.
  See also pointer variables
  binary trees and, 1276–1279
  dangling, **743**

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

described, **728–745**
functions and, 748
inheritance and, 765–772
linked lists and, 962, 970, 975, 986, 998–1002, 1013–1014, 1022–1023
peculiarities related to, 754–765
shallow versus deep copy and, 752–754
this, 779–804
polymorphism
described, **691**
parametric, 691
pop function, 1055–1056, 1064–1066, 1080, 1083–1085, 1383–1384, 1395
pop_back function, 1373
pop_front function, 1370, 1375
postfix expressions calculator, 1090–1102
post-increment operator, 831–835
postorder sequence, **1261**
postorder traversal, 1291–1292
posttest loops, **275**
pound sign (#), 12, 77, 1095
pow function, 309, 310, 312, 325
precision
described, **40**
double, **40**
single, **40**
predicates, 1406
prefix (Polish) notation, **1090**
pre-increment operator, 829–830
preorder sequence, **1261**
preorder traversal, 1290–1291
preprocessor directives, 12, 213
described, **77**
#endif, 679
#ifndef, 679
#include, 77–78, 116–117, 147, 679, 677
I/O and, 116–117
pretest loops, **275**
Prim's algorithm, 1337–1338, 1343–1344
primitive data types, 354–355
print function, 668, 671, 675, 696, 701, 703–704, 751, 766, 768–769, 771, 774, 802–804, 820, 983, 1015
printArray function, 490
printGrade function, 359
printGradeReports function, 708
printHeading function, 398
printing
linked lists, 983, 1008–1010, 1015–1016, 1102–1109
memory size, 1509–1510
printListReverse function, 1010
printMatrix function, 511–514
printReport function, 570–571
printResult class, 1099–1102
printResults function, 384–385, 391–393, 397, 438, 1239–1240
printShortestDistance function, 1334
printStars function, 348–349, 351–352
printTime function, 586–587, 590, 599, 842
printTreeAndWeight function, 1344–1345
private class members. *See also* class members
binary trees and, 1268
constructors and, 604–605
described, **585**
friend functions and, 804, 805–806
information hiding and, 585–587, 603–606, 619
inheritance and, 661, 662, 668, 681–682
linked lists and, 982
order of, 603–604
pointers and, 736
private keyword, 585
problem
analysis-coding-execution cycle, 14–22
solving, 14–22
program(s). *See also* programming languages
basics of, 30–34
counter (PC), **5**
creating, 79–82
described, **6**, 30
form, 83–88
object, **13**
processing, 12–14
source, **12**
style, 83–88
terminating, 211–218, 912
programming languages. *See also specific programming languages*
described, **31**
evolution of, 8–10
overview, 1–28
prompt lines, 85–86
protected class members. *See also* class members
binary trees and, 1268
described, **585**, 681
friend functions and, 804
inheritance and, 661, 681–685
protected keyword, 585
protectMembClass.h header file, 684
protectMembInDerivedCl.h header file, 684
pseudocode
control structures and, 196–199
described, **196**
ptrDataClass.h header file, 763
public class members. *See also* class members
constructors and, 604–605
described, **585**
friend functions and, 804
information hiding and, 586–588, 603–605, 627, 628, 629
inheritance and, 661–663, 667–668, 681–682
order of, 603–604
static, 627, 628, 629
STL and, 1358
public keyword, 585
push function, 1055–1056, 1062–1064, 1080–1083, 1383–1384
push_back function, 1358, 1364

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

push_front function, 1370, 1375
putback function, 128–130

Q

question mark (?), 32
queue(s)
application of, 1128–1148
described, **1109–1128**
empty, 1118–1119, 1123–1124
full, 1118–1119, 1123–1124
implementation of, as arrays, 1111–1122
linked implementation of, 1122–1127
operations, 1109–1111
overview, 1054–1159
simulation and, 1128–1148
STL and, 1395–1396
queueADT class, 1110–1111, 1117–1118, 1122–1127
queueFront variable, 1112–1117, 1119–1127
queueRear variable, 1111, 1121, 1124
queueType class, 1141
queuing systems
designing, **1129–1130**
described, 1129

R

rand function, 1510
RAND_MAX constant, 1510
random
file access, 1488–1492
number generation, 20, 1509–1510
random_shuffle function, 1438–1442
raw data, **1472**
rbegin function, 1363, 1366
readCode function, 519–520
readIn function, 553
real data type, 36
Rebuild command, **14**, 624, 682
recMergeSort function, 1217, 1218
records.
See also structS
described, **546–548**
overview, 544–581
sales data analysis program, 563–576
recQuickSort function, 1209
rectangle class, 661, 772
rectangles, 17, 661–679, 772, 808–811, 817–835
rectangleType class, 664–666, 668–679, 808–811, 817–835
recursion.
See also recursive algorithms; recursive functions
converting numbers and, 941–949
described, **924**
direct, **927**
Fibonacci numbers and, 932–936
indirect, **927**
infinite, 927–928
iteration and, 940–941
linked lists and, 1008–1010
overview, 923–957
printing linked lists and, 1102–1109
problem solving and, 928–940
recursive definitions and, 924–928
stacks and, 1102–1109
Tower of Hanoi game and, 936–939
recursive algorithms
described, **925**
problem solving using, 928–940
recursive functions
described, **925**
Fibonacci numbers and, 932–936
stacks and, 1054
ref variable, 369, 370
reference variables, as parameters, 356–359
relational operators
described, **169–174**
enumeration types and, 419–420
operator precedence and, 178–179
string data type and, 173–174
remove function, 1375, 1417–1420
remove_copy function, 1417–1420
remove_copy_if function, 1417–1420
remove_if function, 1417–1420
rend function, 1363
repetition.
See also control structures; loops
arrays and, 475–476
described, **168**
need for, 232–233
overview, 231–305
replace function, 1420–1424, 1508
replace_copy function, 1420–1424
replace_copy_if function, 1420–1424
replace_if function, 1420–1424
reserved words. *See* keywords
resize function, 1364
retrieveAt function, 1237
retrievePlay function, 427–429
return statement, 200, 315–323
reverse function, 1377, 1435–1438
Reverse Polish notation, **1090**
reverse_copy function, 1435–1438
rFibNum function, 932, 934–936
right manipulator, 143–144
right subtree, 1252
rightmost bit, **946**, 947
rock, paper, and scissors game, 426–437
root
described, **1256**
nodes, 1202, 1252, 1254–1255, 1274
rooted tree, **1336**
rotate function, 456, 1435–1438
rotate_copy function, 1435–1438

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

row
  order form, **511**
  processing, **505–509**
runSimulation function, 1144–1145
run-time binding, **768**
runtime_error class, 898
run-together words, **85**

S

salaries, calculating, 18–20
saleByQuarter function, 569
sales
  data analysis program, 563–576
  tax, calculating, 17–18
scientific notation, 38–39
scope resolution operator
  described, **374**, 594
  namespaces and, 442
SDKs (software development kit), 34, 79
  described, **13–14**
  exception handling and, 888, 901
  file extensions and, 80
  header files and, 77
  I/O and, 147
  setprecision manipulator and, 135
  variables and, 63
search algorithms
  asymptotic (Big-O) notation and, 1173–1182
  binary, **1165–1173**
  binary trees and, 1273–1288
  comparison-based, **1182**
  overview, 1161–1248
  sequential, 1163–1175
search function, 981, 987, 989, 1276–1277, 1427–1430
search_n function, 1427–1430
searchSortAlgorithms.h header file, 1162
sec variable, 586, 588, 594, 597, 610–611, 796
second variable, 369
secondary storage
  described, **6**
  I/O and, 146
seekg function, 1489–1490
seekp function, 1489–1493
selection(s)
  multiple, 192–197, 210
  one-way, **185–187**
  two-way, **187–191**
selection structures. *See* control structures
selectionSort function, 1192
selectors, **204**
semanatics, 86
semantics, **84**
semicolon (;), 32–33, 74, 84, 547, 585
sentinel(s)
  -controlled while loops, **239–243**
  described, **239**
sequence containers
  deque, **1369–1373**, 1397
  described, **1352**
  list, **1373–1380**, 1397
  vector, 1353–1362, 1397
server(s)
  described, **1129**
  lists, 1136–1137
  queues and, 1129, 1133–1140
serverType class, 1133–1135
set associative container, 1387–1393, 1397
setCourseInfo function, 696–697
setCustomerInfo function, 1132–1133
setData function, 683, 684, 687, 688
setDimension function, 669
setfill manipulator, 141–142, 151–152
setInfo function, 701
setName function, 802–803
setprecision manipulator, **135–136**, 137–138, 139
setProduct function, 639–640
setServerBusy function, 1139
setSimulationParameters function, 1143
setTime function, 586–587, 594–595, 604–605, 841
setw manipulator, 138–140
setX function, 738
shallow copy, **752–754**
shape class, 772–780
short-circuit evaluation, **181–182**
shortest path algorithm, **1327–1335**
shortestPath function, 1333–1334
showpoint manipulator, 136–138
showSelection function, 638
side effects, 374–376
simple data type, **470**
simple graphs, **1315**
simulation
  described, **1128**
  queues and, 1128–1148
  time-driven, **1129**
single quote ('), 74
size function, 448–449, 1362, 1383–1384, 1395, 1507
Smalltalk, 692
software, **6**
sort function, 1188, 1233–1234, 1376, 1427–1430
sortCourses function, 701, 705
sorting algorithms
  bubble sort, **1183–1188**
  comparison-based, 1201–1202
  described, **1182–1242**
  insertion sort, **1193–1201**
  merge sort, 1210–1242
  overview, 1161–1248
  quick sort, 1202–1210
  selection sort, 1189–1193
source, **1327**
source code
  described, 12, 79
  files (source files), **79**
  saving, 79
source programs, **12**
spaghetti code, 280
spanning tree, **1337**
splice function, 1376
square root function, 309
squareFirst function, 369
stack(s)
  application of, 1090–1102
  container adapters and, 1393–1394
  described, **1054–1057**

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

stack(s) (*continued*)
empty, 1062, 1080
full, 1062, 1080
header files and, 1068–1072
implementation of, as arrays, 1058–1090
initializing, 1061–1062, 1080
linked implementation of, 1076–1088
overview, 1054–1159
postfix expressions calculator and, 1090–1102
unwinding, 914–917
stackADT class, 1056–1058, 1077–1078
stackTop variable, 1058, 1062–1072, 1077–1088
stackType class, 1058–1061
Standard C++, 24, 371.
See also ANSI/ISO Standard C++
library, 51
namespaces and, 437
naming conventions, 1496–1497
string data type and, 516
statement(s).
See also specific types
nested, **192–197**
terminators, **84**
static binding. *See* compile-time binding
static keyword, 376
stepwise refinement, **22**
STL (Standard Template Library).
See also containers
components of, 1352–1380
copy algorithm and, 1365–1368
function objects and, 1400–1406
iterators and, 1358–1359, 1366–1368, 1380–1397, 1406–1408
overview, 1351–1465
strcmp function, 493–495
strcpy function, 493–495
stream(s)
classes, **680–681**
functions, 123
iterators, 1386
variables, 130–131, 360
stream extraction operator, **60**
stream insertion operator, 12, 816–821
string(s).
See also string data type
comparisons, 494–495
arrays of, 514–516
input, 496–497
output, 497
pig latin, 454–460
reading, 496
writing, 496
string data type.
See also strings
control structures and, 173–174
described, **50–51**, 442–446
I/O and, 145
pig latin strings and, 454–460
templates and, 859
using, 78
strlen function, 493–495
strongly connected, use of the term, **1316**
Stroustrup, Bjarne, 24
struct keyword, 547
structS.
See also records
arrays and, 553–558
classes and, 584, 618–619
components of, 546–550
described, **456**
functions and, 552–553
I/O and, 552
linked lists and, 961, 976
pointer variables and, 736–739
seekg function and, 1490–1493
structS within, 559–562
variables and, 547–548, 552–553
structured
design, **22–23**
programming, **22**
structured data type, **470**
strVar variable, 449
student grade program, 155–158
student variable, 547–548
studentType class, 699–701, 706–707, 736–737
subgraphs, **1313**
subprograms, **30**.
See also functions
substr function, 452–454, 957, 1507
sumRows function, 512, 513–514
swap function, 454–460, 1192, 1193, 1363, 1424–1427, 1507
swap_ranges function, 1424–1427
switch keyword, 204
switch structure, 184, 278–281
described, **204–210**
enumeration types and, 421
while loops and, 241–243
symbols, special, **32–33**
symbols (listed by name)
& (ampersand), 350, 356, 482
* (asterisk), 32, 728, 1094, 1095
\ (backslash), 74, 492
, (comma), 32–33, 37, 84
{} (curly braces), 80, 82, 84
. (decimal point), 130–132, 140
" (double quotes), 51, 75, 443
= (equal sign), 32, 1094
! (exclamation point), 32
/ (forward slash), 32, 1094, 1095
> (greater-than sign), 32
- (hyphen), 32, 1094, 1095
< (less-than sign), 32
() (parentheses), 44–45, 181, 185, 187
. (period), 32, 548, 589
+ (plus sign), 32, 1094, 1095
# (pound sign), 12, 77, 1095
? (question mark), 32
; (semicolon), 32–33, 74, 84, 547, 585
' (single quote), 74
~ (tilde), 615
_ (underscore), 33, 34, 437

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

Licensed to: iChapters User

syntax.
 See also syntax errors
 described, **83**
 rules, **31**
syntax errors, 12–13, 86.
 See also debugging; errors
 arrays and, 481, 482
 control structures and, 188
 described, **83**
 dot notation and, 131
 functions and, 317, 379
 namespaces and, 437
system programs, **6**

T

t variable, 375
tables, 500–518
Tabulating Machine Company, 2
tail recursive function, **927**
targets, **1165**
telephone digits, 241–243
templates.
 See also STL (Standard Template Library)
 described, **859**
 linked lists and, 974–975
 overview, 795–883
terabytes (TB), **7**
ternary operators, **203**
test class, 799
test scores, 21–22, 155–158, 193, 200, 209.
 See also GPA (grade point averages)
testAddress class, 782–783
testA.h header file, 679
testClass class, 608–609, 611
testCopyConst function, 765
test.h header file, 678–679
text processing, 525–532
this keyword, 799
this pointer
 described, **779–804**
 operator overloading and, 829–830
thisPointerClass class, 800
throw statements, 908–911
tilde (~), 615
time-driven simulation, **1129**.
 See also simulation
tokens, 32–33
tolower function, 310
top
 -down design, **22**
 of stacks, **1054**, 1055–1056, 1064, 1083, 1080, 1383–1384
top function, 1055–1056, 1064, 1083, 1080, 1383–1384
totalSaleByPerson function, 569–570
toupper function, 310, 312
Tower of Hanoi game, 936–939
transaction time
 described, **1129**
 reducing, 1137, 1140
transactionTime variable, 1130, 1132–1133
transform function, 1422–1444
try/catch blocks, 890–897, 907–911, 914–917
TurboPascal, 692
two-dimensional arrays.
 See also arrays
 described, **500–519**
 dynamic, 749–751
type
 conversion, **48–50**
 names, **617**
typedef iterator, 1358–1359, 1383–1385

U

UML (Unified Modeling Language). class diagrams
 binary trees and, 1268
 composition and, 686–687, 689
 courseType class and, 696
 described, **587–588**
 graphs and, 1320, 1329, 1343
 inheritance and, 664–667, 676–677
 linked lists and, 977, 980–981, 988, 997, 1013
 operator overloading and, 838–839, 846
 personType class and, 626
 queues and, 1111, 1131–1132, 1135, 1138
 stacks and, 1057, 1059–1060, 1078
 studentType class and, 700
unary operators, **41**, 829–835
underflow condition, **1066**
underflow_error class, 898
underscore (_), 33, 34, 437
undirected graphs, **1313**, 1314
Unicode, 8
union, of sets, **1313**
unique function, 1375
unorderedArrayListType class, 1162, 1188
unorderedLinkedList class, 975, 981, 987–996, 1088–1089, 1162, 1127–1128, 1217, 1319
unorderedListType class, 995–996
updateServers function, 1140
updateXYZ function, 800–801
using keyword, 439
using statement, 439, 441

V

validSelection function, 427–428
variable(s)
 automatic, **376–378**
 converting length and, 90–91
 described, 53–54, 59–60, 80, 418, 424, 547–548, 588–589
 dynamic, **739–743**
 enumeration types and, 418–419
 external, **374**
 flag, **244**
 global, **374–376**
 initializing, 55, 59–60, 64–66
 input stream, **117**
 local, 360–370, 390
 making change and, 84
 output stream, **117**
 passing, 592–593
 putting state in, 54–58
 records and, 547–548, 552–553
 reference, 356–359
 static, **376–378**, 627
 stream, **117**

CENGAGEbrain.com

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

vcontainers, size of, 1356
vector class, 1353–1358
vector sequence container, 1353–1362, 1397
video store program, 1023–1042, 1296–1305
videoBinaryTree class, 1300–1302
videoBinaryTree.h header file, 1302
videoListType class, 1029–1036
videoType class, 1027, 1032, 1296–1300
virtual destructors, **772**
virtual functions
  described, **768**
  overview, 765–772
virtual keyword, 768
Visual C++ (Microsoft), 6, 80
Visual Studio .NET (Microsoft), 80, 147
void keyword, 346

W

wages, determining, 189–190
waitingCustomerQueueType class, 1141–1142
waitingTime variable, 1130, 1132–1133
walk-through, **57**, 66
warning messages, 63
weight of the edge, **1327**, 1339–1342
weighted
  graphs, **1327**
  tree, **1336**
weightedGraphType class, 1328–1329
what function, 898–989
while keyword, 233
while loop(s).
  See also looks
  break statements and, 278–281
  counter-controlled, **236–238**
  described, **233**
  designing, 235–236
  EOF-controlled, **247–248**, 284–287
  expressions in, 249–250
  Fibonacci numbers and, 259–264
  flag-controlled, **243–244**
  nested loops and, 283–284
  overview, 233–274
  recursion and, 940–941
  search algorithms and, 1171–1172
  sentinel-controlled, **239–243**
  stacks and, 1108
whitespace, **34**, 145, 526
width variable, 668, 670, 673
winningObject function, 428, 431
Wozniak, Stephen, 3
write function, 1488
WriteTotal function, 528

X

Xerox, 692

Y

y variable, 629
yourClock object, 588–589, 596–597, 604, 607

Z

zettabytes (ZB), 7

Copyright 2009 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.