

GLOBAL
EDITION



C++ How to Program

TENTH EDITION

Paul Deitel • Harvey Deitel

Introducing the New C++14 Standard



Pearson

C++

HOW TO PROGRAM

C++ How to Program, Global Edition

Table of Contents

Cover

Title Page

Copyright Page

Contents

Preface

Before You Begin

1 Introduction to Computers and C++ 41

1.1 Introduction 42

1.2 Computers and the Internet in Industry and Research 43

1.3 Hardware and Software 45

1.3.2 Computer Organization 46

1.4 Data Hierarchy 47

1.5 Machine Languages, Assembly Languages and High-Level
Languages 50

1.6 C and C++ 51

1.7 Programming Languages 52

1.8 Introduction to Object Technology 54

1.9 Typical C++ Development Environment 57

1.10 Test-Driving a C++ Application 60

1.10.1 Compiling and Running an Application in Visual Studio 2015 for
Windows 60

1.10.2 Compiling and Running Using GNU C++ on Linux 65

1.10.3 Compiling and Running with Xcode on Mac OS X 67

Table of Contents

1.11 Operating Systems 72

1.11.1 WindowsA Proprietary Operating System 72

1.11.2 LinuxAn Open-Source Operating System 72

1.11.3 Apples OS X; Apples iOS for iPhone®, iPad® and iPod Touch®
Devices 73

1.11.4 Googles Android 73

1.12 The Internet and the World Wide Web 74

1.13 Some Key Software Development Terminology 76

1.14 C++11 and C++14: The Latest C++ Versions 78

1.15 Boost C++ Libraries 79

1.16 Keeping Up to Date with Information Technologies 79

2 Introduction to C++ Programming, Input/Output and Operators 84

2.1 Introduction 85

2.2 First Program in C++: Printing a Line of Text 85

2.3 Modifying Our First C++ Program 89

2.4 Another C++ Program: Adding Integers 90

2.5 Memory Concepts 94

2.6 Arithmetic 95

2.7 Decision Making: Equality and Relational Operators 99

2.8 Wrap-Up 103

3 Introduction to Classes, Objects, Member Functions and Strings 113

3.1 Introduction 114

3.2 Test-Driving an Account Object 115

3.2.1 Instantiating an Object 115

3.2.2 Headers and Source-Code Files 116

Table of Contents

3.2.3 Calling Class Accounts getName Member Function 116

3.2.4 Inputting a string with getline 117

3.2.5 Calling Class Accounts setName Member Function 117

3.3 Account Class with a Data Member and Set and Get Member Functions 118

3.3.1 Account Class Definition 118

3.3.2 Keyword class and the Class Body 119

3.3.3 Data Member name of Type string 119

3.3.4 setName Member Function 120

3.3.5 getName Member Function 122

3.3.6 Access Specifiers private and public 122

3.3.7 Account UML Class Diagram 123

3.4 Account Class: Initializing Objects with Constructors 124

3.4.1 Defining an Account Constructor for Custom Object Initialization 125

3.4.2 Initializing Account Objects When Theyre Created 126

3.4.3 Account UML Class Diagram with a Constructor 128

3.5 Software Engineering with Set and Get Member Functions 128

3.6 Account Class with a Balance; Data Validation 129

3.6.1 Data Member balance 129

3.6.2 Two-Parameter Constructor with Validation 131

3.6.3 deposit Member Function with Validation 131

3.6.4 getBalance Member Function 131

3.6.5 Manipulating Account Objects with Balances 132

3.6.6 Account UML Class Diagram with a Balance and Member Functions
deposit and getBalance 134

3.7 Wrap-Up 134

4 Algorithm Development and Control Statements: Part 1 143

4.1 Introduction 144

Table of Contents

4.2 Algorithms	145
4.3 Pseudocode	145
4.4 Control Structures	146
4.4.1 Sequence Structure	146
4.4.2 Selection Statements	148
4.4.3 Iteration Statements	148
4.4.4 Summary of Control Statements	149
4.5 if Single-Selection Statement	149
4.6 ifelse Double-Selection Statement	150
4.6.1 Nested ifelse Statements	151
4.6.2 Dangling-else Problem	153
4.6.3 Blocks	153
4.6.4 Conditional Operator (?:)	154
4.7 Student Class: Nested ifelse Statements	155
4.8 while Iteration Statement	157
4.9 Formulating Algorithms: Counter-Controlled Iteration	159
4.9.1 Pseudocode Algorithm with Counter-Controlled Iteration	159
4.9.2 Implementing Counter-Controlled Iteration	160
4.9.3 Notes on Integer Division and Truncation	162
4.9.4 Arithmetic Overflow	162
4.9.5 Input Validation	163
4.10 Formulating Algorithms: Sentinel-Controlled Iteration	163
4.10.1 Top-Down, Stepwise Refinement: The Top and First Refinement	164
4.10.2 Proceeding to the Second Refinement	164
4.10.3 Implementing Sentinel-Controlled Iteration	166
4.10.4 Converting Between Fundamental Types Explicitly and Implicitly	169
4.10.5 Formatting Floating-Point Numbers	170
4.10.6 Unsigned Integers and User Input	170

Table of Contents

4.11 Formulating Algorithms: Nested Control Statements 171

4.11.1 Problem Statement 171

4.11.2 Top-Down, Stepwise Refinement: Pseudocode Representation of the
Top 172

4.11.3 Top-Down, Stepwise Refinement: First Refinement 172

4.11.4 Top-Down, Stepwise Refinement: Second Refinement 172

4.11.5 Complete Second Refinement of the Pseudocode 173

4.11.6 Program That Implements the Pseudocode Algorithm 174

4.11.7 Preventing Narrowing Conversions with List Initialization 175

4.12 Compound Assignment Operators 176

4.13 Increment and Decrement Operators 177

4.14 Fundamental Types Are Not Portable 180

4.15 Wrap-Up 180

5 Control Statements: Part 2; Logical Operators 199

5.1 Introduction 200

5.2 Essentials of Counter-Controlled Iteration 200

5.3 for Iteration Statement 201

5.4 Examples Using the for Statement 205

5.5 Application: Summing Even Integers 206

5.6 Application: Compound-Interest Calculations 207

5.7 Case Study: Integer-Based Monetary Calculations with Class DollarAmount 211

5.7.1 Demonstrating Class DollarAmount 212

5.7.2 Class DollarAmount 215

5.8 dowhile Iteration Statement 219

5.9 switch Multiple-Selection Statement 220

5.10 break and continue Statements 226

5.10.1 break Statement 226

Table of Contents

5.10.2 continue Statement	227
5.11 Logical Operators	228
5.11.1 Logical AND (&&) Operator	228
5.11.2 Logical OR () Operator	229
5.11.3 Short-Circuit Evaluation	230
5.11.4 Logical Negation (!) Operator	230
5.11.5 Logical Operators Example	231
5.12 Confusing the Equality (==) and Assignment (=) Operators	232
5.13 Structured-Programming Summary	234
5.14 Wrap-Up	239
6 Functions and an Introduction to Recursion	251
6.1 Introduction	252
6.2 Program Components in C++	253
6.3 Math Library Functions	254
6.4 Function Prototypes	255
6.5 Function-Prototype and Argument-Coercion Notes	258
6.5.1 Function Signatures and Function Prototypes	259
6.5.2 Argument Coercion	259
6.5.3 Argument-Promotion Rules and Implicit Conversions	259
6.6 C++ Standard Library Headers	260
6.7 Case Study: Random-Number Generation	262
6.7.1 Rolling a Six-Sided Die	263
6.7.2 Rolling a Six-Sided Die 60,000,000 Times	264
6.7.3 Randomizing the Random-Number Generator with srand	265
6.7.4 Seeding the Random-Number Generator with the Current Time	267
6.7.5 Scaling and Shifting Random Numbers	267
6.8 Case Study: Game of Chance; Introducing Scopedenums	268

Table of Contents

- 6.9 C++11 Random Numbers 272
- 6.10 Scope Rules 273
- 6.11 Function-Call Stack and Activation Records 277
- 6.12 Inline Functions 281
- 6.13 References and Reference Parameters 282
- 6.14 Default Arguments 285
- 6.15 Unary Scope Resolution Operator 287
- 6.16 Function Overloading 288
- 6.17 Function Templates 291
- 6.18 Recursion 294
- 6.19 Example Using Recursion: Fibonacci Series 297
- 6.20 Recursion vs. Iteration 300
- 6.21 Wrap-Up 303

7 Class Templates array and vector; Catching Exceptions 323

- 7.1 Introduction 324
- 7.2 arrays 324
- 7.3 Declaring arrays 326
- 7.4 Examples Using arrays 326
 - 7.4.1 Declaring an array and Using a Loop to Initialize the arrays Elements 327
 - 7.4.2 Initializing an array in a Declaration with an Initializer List 328
 - 7.4.3 Specifying an arrays Size with a Constant Variable and Setting array
Elements with Calculations 329
 - 7.4.4 Summing the Elements of an array 330
 - 7.4.5 Using a Bar Chart to Display array Data Graphically 331
 - 7.4.6 Using the Elements of an array as Counters 332
 - 7.4.7 Using arrays to Summarize Survey Results 333

Table of Contents

7.4.8 Static Local arrays and Automatic Local arrays	336
7.5 Range-Based for Statement	338
7.6 Case Study: Class GradeBook Using an array to Store Grades	340
7.7 Sorting and Searching arrays	346
7.7.1 Sorting	346
7.7.2 Searching	346
7.7.3 Demonstrating Functions sort and binary_search	346
7.8 Multidimensional arrays	347
7.9 Case Study: Class GradeBook Using a Two-Dimensional array	351
7.10 Introduction to C++ Standard Library Class Template vector	357
7.11 Wrap-Up	363
8 Pointers	379
8.1 Introduction	380
8.2 Pointer Variable Declarations and Initialization	381
8.2.1 Declaring Pointers	381
8.2.2 Initializing Pointers	382
8.2.3 Null Pointers Prior to C++11	382
8.3 Pointer Operators	382
8.3.1 Address (&) Operator	382
8.3.2 Indirection (*) Operator	383
8.3.3 Using the Address (&) and Indirection (*) Operators	384
8.4 Pass-by-Reference with Pointers	385
8.5 Built-In Arrays	389
8.5.1 Declaring and Accessing a Built-In Array	389
8.5.2 Initializing Built-In Arrays	390
8.5.3 Passing Built-In Arrays to Functions	390

Table of Contents

8.5.4 Declaring Built-In Array Parameters	391
8.5.5 C++11: Standard Library Functions begin and end	391
8.5.6 Built-In Array Limitations	391
8.5.7 Built-In Arrays Sometimes Are Required	392
8.6 Using const with Pointers	392
8.6.1 Nonconstant Pointer to Nonconstant Data	393
8.6.2 Nonconstant Pointer to Constant Data	393
8.6.3 Constant Pointer to Nonconstant Data	394
8.6.4 Constant Pointer to Constant Data	395
8.7 sizeof Operator	396
8.8 Pointer Expressions and Pointer Arithmetic	398
8.8.1 Adding Integers to and Subtracting Integers from Pointers	399
8.8.2 Subtracting Pointers	400
8.8.3 Pointer Assignment	401
8.8.4 Cannot Dereference a void*	401
8.8.5 Comparing Pointers	401
8.9 Relationship Between Pointers and Built-In Arrays	401
8.9.1 Pointer/Offset Notation	402
8.9.2 Pointer/Offset Notation with the Built-In Arrays Name as the Pointer	402
8.9.3 Pointer/Subscript Notation	402
8.9.4 Demonstrating the Relationship Between Pointers and Built-In Arrays	403
8.10 Pointer-Based Strings (Optional)	404
8.11 Note About Smart Pointers	407
8.12 Wrap-Up	407
9 Classes: A Deeper Look	425
9.1 Introduction	426
9.2 Time Class Case Study: Separating Interface from	

Table of Contents

Implementation 427

9.2.1 Interface of a Class 428

9.2.2 Separating the Interface from the Implementation 428

9.2.3 Time Class Definition 428

9.2.4 Time Class Member Functions 430

9.2.5 Scope Resolution Operator (::) 431

9.2.6 Including the Class Header in the Source-Code File 431

9.2.7 Time Class Member Function setTime and Throwing Exceptions 432

9.2.8 Time Class Member Function toUniversalString and String Stream
Processing 432

9.2.9 Time Class Member Function toStandardString 433

9.2.10 Implicitly Inlining Member Functions 433

9.2.11 Member Functions vs. Global Functions 433

9.2.12 Using Class Time 434

9.2.13 Object Size 436

9.3 Compilation and Linking Process 436

9.4 Class Scope and Accessing Class Members 438

9.5 Access Functions and Utility Functions 439

9.6 Time Class Case Study: Constructors with Default Arguments 439

9.6.1 Constructors with Default Arguments 439

9.6.2 Overloaded Constructors and C++11 Delegating Constructors 444

9.7 Destructors 445

9.8 When Constructors and Destructors Are Called 445

9.8.1 Constructors and Destructors for Objects in Global Scope 446

9.8.2 Constructors and Destructors for Non-static Local Objects 446

9.8.3 Constructors and Destructors for static Local Objects 446

9.8.4 Demonstrating When Constructors and Destructors Are Called 446

9.9 Time Class Case Study: A Subtle Trap Returning a Reference or a

Table of Contents

Pointer to a private Data Member	449
9.10 Default Memberwise Assignment	451
9.11 const Objects and const Member Functions	453
9.12 Composition: Objects as Members of Classes	455
9.13 friend Functions and friend Classes	461
9.14 Using the this Pointer	463
9.14.1 Implicitly and Explicitly Using the this Pointer to Access an Objects Data Members	464
9.14.2 Using the this Pointer to Enable Cascaded Function Calls	465
9.15 static Class Members	469
9.15.1 Motivating Classwide Data	469
9.15.2 Scope and Initialization of static Data Members	469
9.15.3 Accessing static Data Members	470
9.15.4 Demonstrating static Data Members	470
9.16 Wrap-Up	473
10 Operator Overloading; Class string	487
10.1 Introduction	488
10.2 Using the Overloaded Operators of Standard Library Class string	489
10.3 Fundamentals of Operator Overloading	493
10.3.1 Operator Overloading Is Not Automatic	493
10.3.2 Operators That You Do Not Have to Overload	493
10.3.3 Operators That Cannot Be Overloaded	494
10.3.4 Rules and Restrictions on Operator Overloading	494
10.4 Overloading Binary Operators	495
10.5 Overloading the Binary Stream Insertion and Stream Extraction Operators	495
10.6 Overloading Unary Operators	499
10.7 Overloading the Increment and Decrement Operators	500

Table of Contents

10.8 Case Study: A Date Class	501
10.9 Dynamic Memory Management	506
10.10 Case Study: Array Class	508
10.10.1 Using the Array Class	509
10.10.2 Array Class Definition	513
10.11 Operators as Member vs. Non-Member Functions	520
10.12 Converting Between Types	521
10.13 explicit Constructors and Conversion Operators	522
10.14 Overloading the Function Call Operator ()	525
10.15 Wrap-Up	525
11 Object-Oriented Programming: Inheritance	537
11.1 Introduction	538
11.2 Base Classes and Derived Classes	539
11.2.1 CommunityMember Class Hierarchy	539
11.2.2 Shape Class Hierarchy	540
11.3 Relationship between Base and Derived Classes	541
11.3.1 Creating and Using a CommissionEmployee Class	541
11.3.2 Creating a BasePlusCommissionEmployee Class Without Using Inheritance	546
11.3.3 Creating a CommissionEmployeeBasePlusCommissionEmployee Inheritance Hierarchy	551
11.3.4 CommissionEmployeeBasePlusCommissionEmployee Inheritance Hierarchy Using protected Data	555
11.3.5 CommissionEmployeeBasePlusCommissionEmployee Inheritance Hierarchy Using private Data	559
11.4 Constructors and Destructors in Derived Classes	563
11.5 public, protected and private Inheritance	565
11.6 Wrap-Up	566

Table of Contents

12 Object-Oriented Programming: Polymorphism	571
12.1 Introduction	572
12.2 Introduction to Polymorphism: Polymorphic Video Game	573
12.3 Relationships Among Objects in an Inheritance Hierarchy	574
12.3.1 Invoking Base-Class Functions from Derived-Class Objects	574
12.3.2 Aiming Derived-Class Pointers at Base-Class Objects	577
12.3.3 Derived-Class Member-Function Calls via Base-Class Pointers	578
12.4 Virtual Functions and Virtual Destructors	580
12.4.1 Why virtual Functions Are Useful	580
12.4.2 Declaring virtual Functions	580
12.4.3 Invoking a virtual Function Through a Base-Class Pointer or Reference	581
12.4.4 Invoking a virtual Function Through an Objects Name	581
12.4.5 virtual Functions in the CommissionEmployee Hierarchy	581
12.4.6 virtual Destructors	586
12.4.7 C++11: final Member Functions and Classes	586
12.5 Type Fields and switch Statements	587
12.6 Abstract Classes and Pure virtual Functions	587
12.6.1 Pure virtual Functions	588
12.6.2 Device Drivers: Polymorphism in Operating Systems	589
12.7 Case Study: Payroll System Using Polymorphism	589
12.7.1 Creating Abstract Base Class Employee	590
12.7.2 Creating Concrete Derived Class SalariedEmployee	593
12.7.3 Creating Concrete Derived Class CommissionEmployee	596
12.7.4 Creating Indirect Concrete Derived Class BasePlusCommissionEmployee	598
12.7.5 Demonstrating Polymorphic Processing	600
12.8 (Optional) Polymorphism, Virtual Functions and Dynamic Binding	

Table of Contents

Under the Hood 603

12.9 Case Study: Payroll System Using Polymorphism and Runtime
Type Information with Downcasting, `dynamic_cast`, `typeid` and
`type_info` 607

12.10 Wrap-Up 610

13 Stream Input/Output: A Deeper Look 617

13.1 Introduction 618

13.2 Streams 619

13.2.1 Classic Streams vs. Standard Streams 619

13.2.2 `iostream` Library Headers 620

13.2.3 Stream Input/Output Classes and Objects 620

13.3 Stream Output 621

13.3.1 Output of `char*` Variables 621

13.3.2 Character Output Using Member Function `put` 622

13.4 Stream Input 622

13.4.1 `get` and `getline` Member Functions 623

13.4.2 `istream` Member Functions `peek`, `putback` and `ignore` 626

13.4.3 Type-Safe I/O 626

13.5 Unformatted I/O Using `read`, `write` and `gcount` 626

13.6 Stream Manipulators: A Deeper Look 627

13.6.1 Integral Stream Base: `dec`, `oct`, `hex` and `setbase` 628

13.6.2 Floating-Point Precision (`precision`, `setprecision`) 628

13.6.3 Field Width (`width`, `setw`) 630

13.6.4 User-Defined Output Stream Manipulators 631

13.7 Stream Format States and Stream Manipulators 632

13.7.1 Trailing Zeros and Decimal Points (`showpoint`) 633

13.7.2 Justification (`left`, `right` and `internal`) 634

13.7.3 Padding (`fill`, `setfill`) 635

Table of Contents

13.7.4	Integral Stream Base (dec, oct, hex, showbase)	637
13.7.5	Floating-Point Numbers; Scientific and Fixed Notation (scientific, fixed)	637
13.7.6	Uppercase/Lowercase Control (uppercase)	638
13.7.7	Specifying Boolean Format (boolalpha)	639
13.7.8	Setting and Resetting the Format State via Member Function flags	640
13.8	Stream Error States	641
13.9	Tying an Output Stream to an Input Stream	644
13.10	Wrap-Up	645
14	File Processing	655
14.1	Introduction	656
14.2	Files and Streams	656
14.3	Creating a Sequential File	657
14.3.1	Opening a File	658
14.3.2	Opening a File via the open Member Function	659
14.3.3	Testing Whether a File Was Opened Successfully	659
14.3.4	Overloaded bool Operator	660
14.3.5	Processing Data	660
14.3.6	Closing a File	660
14.3.7	Sample Execution	661
14.4	Reading Data from a Sequential File	661
14.4.1	Opening a File for Input	662
14.4.2	Reading from the File	662
14.4.3	File-Position Pointers	662
14.4.4	Case Study: Credit Inquiry Program	663
14.5	C++14: Reading and Writing Quoted Text	666
14.6	Updating Sequential Files	667
14.7	Random-Access Files	668

Table of Contents

14.8 Creating a Random-Access File 669

14.8.1 Writing Bytes with ostream Member Function write 669

14.8.2 Converting Between Pointer Types with the reinterpret_cast Operator 669

14.8.3 Credit-Processing Program 670

14.8.4 Opening a File for Output in Binary Mode 673

14.9 Writing Data Randomly to a Random-Access File 673

14.9.1 Opening a File for Input and Output in Binary Mode 675

14.9.2 Positioning the File-Position Pointer 675

14.10 Reading from a Random-Access File Sequentially 675

14.11 Case Study: A Transaction-Processing Program 677

14.12 Object Serialization 683

14.13 Wrap-Up 684

15 Standard Library Containers and Iterators 695

15.1 Introduction 696

15.2 Introduction to Containers 698

15.3 Introduction to Iterators 702

15.4 Introduction to Algorithms 707

15.5 Sequence Containers 707

15.5.1 vector Sequence Container 708

15.5.2 list Sequence Container 715

15.5.3 deque Sequence Container 720

15.6 Associative Containers 721

15.6.1 multiset Associative Container 722

15.6.2 set Associative Container 725

15.6.3 multimap Associative Container 727

15.6.4 map Associative Container 729

15.7 Container Adapters 730

Table of Contents

15.7.1	stack Adapter	731
15.7.2	queue Adapter	733
15.7.3	priority_queue Adapter	734
15.8	Class bitset	735
15.9	Wrap-Up	737
16	Standard Library Algorithms	747
16.1	Introduction	748
16.2	Minimum Iterator Requirements	748
16.3	Lambda Expressions	750
16.3.1	Algorithm for_each	751
16.3.2	Lambda with an Empty Introducer	751
16.3.3	Lambda with a Nonempty IntroducerCapturing Local Variables	752
16.3.4	Lambda Return Types	752
16.4	Algorithms	752
16.4.1	fill, fill_n, generate and generate_n	752
16.4.2	equal, mismatch and lexicographical_compare	755
16.4.3	remove, remove_if, remove_copy and remove_copy_if	758
16.4.4	replace, replace_if, replace_copy and replace_copy_if	761
16.4.5	Mathematical Algorithms	763
16.4.6	Basic Searching and Sorting Algorithms	766
16.4.7	swap, iter_swap and swap_ranges	771
16.4.8	copy_backward, merge, unique and reverse	772
16.4.9	inplace_merge, unique_copy and reverse_copy	775
16.4.10	Set Operations	777
16.4.11	lower_bound, upper_bound and equal_range	780
16.4.12	min, max, minmax and minmax_element	782
16.5	Function Objects	784
16.6	Standard Library Algorithm Summary	787

Table of Contents

16.7 Wrap-Up 789

17 Exception Handling: A Deeper Look 797

17.1 Introduction 798

17.2 Exception-Handling Flow of Control; Defining an Exception Class 799

17.2.1 Defining an Exception Class to Represent the Type of Problem That
Might Occur 799

17.2.2 Demonstrating Exception Handling 800

17.2.3 Enclosing Code in a try Block 801

17.2.4 Defining a catch Handler to Process a DivideByZeroException 802

17.2.5 Termination Model of Exception Handling 802

17.2.6 Flow of Program Control When the User Enters a Nonzero
Denominator 803

17.2.7 Flow of Program Control When the User Enters a Denominator of
Zero 803

17.3 Rethrowing an Exception 804

17.4 Stack Unwinding 806

17.5 When to Use Exception Handling 807

17.6 noexcept: Declaring Functions That Do Not Throw Exceptions
808

17.7 Constructors, Destructors and Exception Handling 808

17.7.1 Destructors Called Due to Exceptions 808

17.7.2 Initializing Local Objects to Acquire Resources 809

17.8 Processing new Failures 809

17.8.1 new Throwing bad_alloc on Failure 809

17.8.2 new Returning nullptr on Failure 810

17.8.3 Handling new Failures Using Function set_new_handler 811

17.9 Class unique_ptr and Dynamic Memory Allocation 812

17.9.1 unique_ptr Ownership 814

17.9.2 unique_ptr to a Built-In Array 815

Table of Contents

17.10 Standard Library Exception Hierarchy 815

17.11 Wrap-Up 817

18 Introduction to Custom Templates 823

18.1 Introduction 824

18.2 Class Templates 825

18.2.1 Creating Class Template `Stack<T>` 826

18.2.2 Class Template `Stack<T>`'s Data Representation 827

18.2.3 Class Template `Stack<T>`'s Member Functions 827

18.2.4 Declaring a Class Templates Member Functions Outside the Class
Template Definition 828

18.2.5 Testing Class Template `Stack<T>` 828

18.3 Function Template to Manipulate a Class-Template
Specialization Object 830

18.4 Nontype Parameters 832

18.5 Default Arguments for Template Type Parameters 832

18.6 Overloading Function Templates 833

18.7 Wrap-Up 833

19 Custom Templated Data Structures 837

19.1 Introduction 838

19.1.1 Always Prefer the Standard Library's Containers, Iterators and Algorithms,
if Possible 839

19.1.2 Special Section: Building Your Own Compiler 839

19.2 Self-Referential Classes 839

19.3 Linked Lists 840

19.3.1 Testing Our Linked List Implementation 842

19.3.2 Class Template `ListNode` 845

19.3.3 Class Template `List` 846

19.3.4 Member Function `insertAtFront` 849

Table of Contents

19.3.5 Member Function insertAtBack	850
19.3.6 Member Function removeFromFront	850
19.3.7 Member Function removeFromBack	851
19.3.8 Member Function print	852
19.3.9 Circular Linked Lists and Double Linked Lists	853
19.4 Stacks	854
19.4.1 Taking Advantage of the Relationship Between Stack and List	855
19.4.2 Implementing a Class Template Stack Class Based By Inheriting from List	855
19.4.3 Dependent Names in Class Templates	856
19.4.4 Testing the Stack Class Template	857
19.4.5 Implementing a Class Template Stack Class With Composition of a List Object	858
19.5 Queues	859
19.5.1 Applications of Queues	859
19.5.2 Implementing a Class Template Queue Class Based By Inheriting from List	860
19.5.3 Testing the Queue Class Template	861
19.6 Trees	863
19.6.1 Basic Terminology	863
19.6.2 Binary Search Trees	864
19.6.3 Testing the Tree Class Template	864
19.6.4 Class Template TreeNode	866
19.6.5 Class Template Tree	867
19.6.6 Tree Member Function insertNodeHelper	869
19.6.7 Tree Traversal Functions	869
19.6.8 Duplicate Elimination	870
19.6.9 Overview of the Binary Tree Exercises	870
19.7 Wrap-Up	871

Table of Contents

20 Searching and Sorting 881

20.1 Introduction 882

20.2 Searching Algorithms 883

20.2.1 Linear Search 883

20.2.2 Binary Search 886

20.3 Sorting Algorithms 890

20.3.1 Insertion Sort 891

20.3.2 Selection Sort 893

20.3.3 Merge Sort (A Recursive Implementation) 895

20.4 Wrap-Up 902

21 Class string and String Stream Processing: A Deeper Look 909

21.1 Introduction 910

21.2 string Assignment and Concatenation 911

21.3 Comparing strings 913

21.4 Substrings 916

21.5 Swapping strings 916

21.6 string Characteristics 917

21.7 Finding Substrings and Characters in a string 920

21.8 Replacing Characters in a string 921

21.9 Inserting Characters into a string 923

21.10 Conversion to Pointer-Based char* Strings 924

21.11 Iterators 926

21.12 String Stream Processing 927

21.13 C++11 Numeric Conversion Functions 930

21.14 Wrap-Up 932

22 Bits, Characters, C Strings and structs 939

Table of Contents

22.1	Introduction	940
22.2	Structure Definitions	940
22.3	typedef and using	942
22.4	Example: Card Shuffling and Dealing Simulation	942
22.5	Bitwise Operators	945
22.6	Bit Fields	954
22.7	Character-Handling Library	958
22.8	C String-Manipulation Functions	963
22.9	C String-Conversion Functions	970
22.10	Search Functions of the C String-Handling Library	975
22.11	Memory Functions of the C String-Handling Library	979
22.12	Wrap-Up	983

Chapters on the Web 999

A Operator Precedence and Associativity 1001

B ASCII Character Set 1003

C Fundamental Types 1005

D Number Systems 1007

D.1	Introduction	1008
D.2	Abbreviating Binary Numbers as Octal and Hexadecimal Numbers	1011
D.3	Converting Octal and Hexadecimal Numbers to Binary Numbers	1012
D.4	Converting from Binary, Octal or Hexadecimal to Decimal	1012
D.5	Converting from Decimal to Binary, Octal or Hexadecimal	1013
D.6	Negative Binary Numbers: Twos Complement Notation	1015

Table of Contents

E Preprocessor 1021

E.1 Introduction 1022

E.2 #include Preprocessing Directive 1022

E.3 #define Preprocessing Directive: Symbolic Constants 1023

E.4 #define Preprocessing Directive: Macros 1023

E.5 Conditional Compilation 1025

E.6 #error and #pragma Preprocessing Directives 1027

E.7 Operators # and ## 1027

E.8 Predefined Symbolic Constants 1027

E.9 Assertions 1028

E.10 Wrap-Up 1028

Appendices on the Web 1033

Index 1035

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

Table of Contents

M
N
O
P
Q
R
S
T
V
W
X
Y
Z