



Contents

Chapters 23–26 and Appendices F–J are PDF documents posted online at the book’s Companion Website, which is accessible from

<http://www.pearsonhighered.com/deitel>

See the inside front cover for more information.

Preface

xxiii

Before You Begin

xxxix

I	Introduction to Computers and C++	I
1.1	Introduction	2
1.2	Computers and the Internet in Industry and Research	3
1.3	Hardware and Software	5
1.3.1	Moore’s Law	5
1.3.2	Computer Organization	6
1.4	Data Hierarchy	7
1.5	Machine Languages, Assembly Languages and High-Level Languages	10
1.6	C and C++	11
1.7	Programming Languages	12
1.8	Introduction to Object Technology	14
1.9	Typical C++ Development Environment	17
1.10	Test-Driving a C++ Application	20
1.10.1	Compiling and Running an Application in Visual Studio 2015 for Windows	20
1.10.2	Compiling and Running Using GNU C++ on Linux	25
1.10.3	Compiling and Running with Xcode on Mac OS X	27
1.11	Operating Systems	32
1.11.1	Windows—A Proprietary Operating System	32
1.11.2	Linux—An Open-Source Operating System	32
1.11.3	Apple’s OS X; Apple’s iOS for iPhone®, iPad® and iPod Touch® Devices	33
1.11.4	Google’s Android	33
1.12	The Internet and the World Wide Web	34
1.13	Some Key Software Development Terminology	36
1.14	C++11 and C++14: The Latest C++ Versions	38

1.15	Boost C++ Libraries	39
1.16	Keeping Up to Date with Information Technologies	39

2 Introduction to C++ Programming, Input/Output and Operators 44

2.1	Introduction	45
2.2	First Program in C++: Printing a Line of Text	45
2.3	Modifying Our First C++ Program	49
2.4	Another C++ Program: Adding Integers	50
2.5	Memory Concepts	54
2.6	Arithmetic	55
2.7	Decision Making: Equality and Relational Operators	59
2.8	Wrap-Up	63

3 Introduction to Classes, Objects, Member Functions and Strings 73

3.1	Introduction	74
3.2	Test-Driving an Account Object	75
3.2.1	Instantiating an Object	75
3.2.2	Headers and Source-Code Files	76
3.2.3	Calling Class Account's getName Member Function	76
3.2.4	Inputting a string with getline	77
3.2.5	Calling Class Account's setName Member Function	77
3.3	Account Class with a Data Member and Set and Get Member Functions	78
3.3.1	Account Class Definition	78
3.3.2	Keyword class and the Class Body	79
3.3.3	Data Member name of Type string	79
3.3.4	setName Member Function	80
3.3.5	getName Member Function	82
3.3.6	Access Specifiers private and public	82
3.3.7	Account UML Class Diagram	83
3.4	Account Class: Initializing Objects with Constructors	84
3.4.1	Defining an Account Constructor for Custom Object Initialization	85
3.4.2	Initializing Account Objects When They're Created	86
3.4.3	Account UML Class Diagram with a Constructor	88
3.5	Software Engineering with Set and Get Member Functions	88
3.6	Account Class with a Balance; Data Validation	89
3.6.1	Data Member balance	89
3.6.2	Two-Parameter Constructor with Validation	91
3.6.3	deposit Member Function with Validation	91
3.6.4	getBalance Member Function	91
3.6.5	Manipulating Account Objects with Balances	92
3.6.6	Account UML Class Diagram with a Balance and Member Functions deposit and getBalance	94
3.7	Wrap-Up	94

4	Algorithm Development and Control Statements: Part I	103
4.1	Introduction	104
4.2	Algorithms	105
4.3	Pseudocode	105
4.4	Control Structures	106
4.4.1	Sequence Structure	106
4.4.2	Selection Statements	108
4.4.3	Iteration Statements	108
4.4.4	Summary of Control Statements	109
4.5	if Single-Selection Statement	109
4.6	if...else Double-Selection Statement	110
4.6.1	Nested if...else Statements	111
4.6.2	Dangling-else Problem	113
4.6.3	Blocks	113
4.6.4	Conditional Operator (?:)	114
4.7	Student Class: Nested if...else Statements	115
4.8	while Iteration Statement	117
4.9	Formulating Algorithms: Counter-Controlled Iteration	119
4.9.1	Pseudocode Algorithm with Counter-Controlled Iteration	119
4.9.2	Implementing Counter-Controlled Iteration	120
4.9.3	Notes on Integer Division and Truncation	122
4.9.4	Arithmetic Overflow	122
4.9.5	Input Validation	123
4.10	Formulating Algorithms: Sentinel-Controlled Iteration	123
4.10.1	Top-Down, Stepwise Refinement: The Top and First Refinement	124
4.10.2	Proceeding to the Second Refinement	124
4.10.3	Implementing Sentinel-Controlled Iteration	126
4.10.4	Converting Between Fundamental Types Explicitly and Implicitly	129
4.10.5	Formatting Floating-Point Numbers	130
4.10.6	Unsigned Integers and User Input	130
4.11	Formulating Algorithms: Nested Control Statements	131
4.11.1	Problem Statement	131
4.11.2	Top-Down, Stepwise Refinement: Pseudocode Representation of the Top	132
4.11.3	Top-Down, Stepwise Refinement: First Refinement	132
4.11.4	Top-Down, Stepwise Refinement: Second Refinement	132
4.11.5	Complete Second Refinement of the Pseudocode	133
4.11.6	Program That Implements the Pseudocode Algorithm	134
4.11.7	Preventing Narrowing Conversions with List Initialization	135
4.12	Compound Assignment Operators	136
4.13	Increment and Decrement Operators	137
4.14	Fundamental Types Are Not Portable	140
4.15	Wrap-Up	140

5 Control Statements: Part 2; Logical Operators 159

5.1	Introduction	160
5.2	Essentials of Counter-Controlled Iteration	160
5.3	for Iteration Statement	161
5.4	Examples Using the for Statement	165
5.5	Application: Summing Even Integers	166
5.6	Application: Compound-Interest Calculations	167
5.7	Case Study: Integer-Based Monetary Calculations with Class DollarAmount	171
5.7.1	Demonstrating Class DollarAmount	172
5.7.2	Class DollarAmount	175
5.8	do...while Iteration Statement	179
5.9	switch Multiple-Selection Statement	180
5.10	break and continue Statements	186
5.10.1	break Statement	186
5.10.2	continue Statement	187
5.11	Logical Operators	188
5.11.1	Logical AND (&&) Operator	188
5.11.2	Logical OR () Operator	189
5.11.3	Short-Circuit Evaluation	190
5.11.4	Logical Negation (!) Operator	190
5.11.5	Logical Operators Example	191
5.12	Confusing the Equality (==) and Assignment (=) Operators	192
5.13	Structured-Programming Summary	194
5.14	Wrap-Up	199

6 Functions and an Introduction to Recursion 211

6.1	Introduction	212
6.2	Program Components in C++	213
6.3	Math Library Functions	214
6.4	Function Prototypes	215
6.5	Function-Prototype and Argument-Coercion Notes	218
6.5.1	Function Signatures and Function Prototypes	219
6.5.2	Argument Coercion	219
6.5.3	Argument-Promotion Rules and Implicit Conversions	219
6.6	C++ Standard Library Headers	220
6.7	Case Study: Random-Number Generation	222
6.7.1	Rolling a Six-Sided Die	223
6.7.2	Rolling a Six-Sided Die 60,000,000 Times	224
6.7.3	Randomizing the Random-Number Generator with srand	225
6.7.4	Seeding the Random-Number Generator with the Current Time	227
6.7.5	Scaling and Shifting Random Numbers	227
6.8	Case Study: Game of Chance; Introducing Scoped enums	228
6.9	C++11 Random Numbers	232
6.10	Scope Rules	233

6.11	Function-Call Stack and Activation Records	237
6.12	Inline Functions	241
6.13	References and Reference Parameters	242
6.14	Default Arguments	245
6.15	Unary Scope Resolution Operator	247
6.16	Function Overloading	248
6.17	Function Templates	251
6.18	Recursion	254
6.19	Example Using Recursion: Fibonacci Series	257
6.20	Recursion vs. Iteration	260
6.21	Wrap-Up	263

7 Class Templates array and vector; Catching Exceptions **283**

7.1	Introduction	284
7.2	arrays	284
7.3	Declaring arrays	286
7.4	Examples Using arrays	286
7.4.1	Declaring an array and Using a Loop to Initialize the array's Elements	287
7.4.2	Initializing an array in a Declaration with an Initializer List	288
7.4.3	Specifying an array's Size with a Constant Variable and Setting array Elements with Calculations	289
7.4.4	Summing the Elements of an array	290
7.4.5	Using a Bar Chart to Display array Data Graphically	291
7.4.6	Using the Elements of an array as Counters	292
7.4.7	Using arrays to Summarize Survey Results	293
7.4.8	Static Local arrays and Automatic Local arrays	296
7.5	Range-Based for Statement	298
7.6	Case Study: Class GradeBook Using an array to Store Grades	300
7.7	Sorting and Searching arrays	306
7.7.1	Sorting	306
7.7.2	Searching	306
7.7.3	Demonstrating Functions sort and binary_search	306
7.8	Multidimensional arrays	307
7.9	Case Study: Class GradeBook Using a Two-Dimensional array	311
7.10	Introduction to C++ Standard Library Class Template vector	317
7.11	Wrap-Up	323

8 Pointers **339**

8.1	Introduction	340
8.2	Pointer Variable Declarations and Initialization	341
8.2.1	Declaring Pointers	341
8.2.2	Initializing Pointers	342
8.2.3	Null Pointers Prior to C++11	342

8.3	Pointer Operators	342
8.3.1	Address (&) Operator	342
8.3.2	Indirection (*) Operator	343
8.3.3	Using the Address (&) and Indirection (*) Operators	344
8.4	Pass-by-Reference with Pointers	345
8.5	Built-In Arrays	349
8.5.1	Declaring and Accessing a Built-In Array	349
8.5.2	Initializing Built-In Arrays	350
8.5.3	Passing Built-In Arrays to Functions	350
8.5.4	Declaring Built-In Array Parameters	351
8.5.5	C++11: Standard Library Functions <code>begin</code> and <code>end</code>	351
8.5.6	Built-In Array Limitations	351
8.5.7	Built-In Arrays Sometimes Are Required	352
8.6	Using <code>const</code> with Pointers	352
8.6.1	Nonconstant Pointer to Nonconstant Data	353
8.6.2	Nonconstant Pointer to Constant Data	353
8.6.3	Constant Pointer to Nonconstant Data	354
8.6.4	Constant Pointer to Constant Data	355
8.7	<code>sizeof</code> Operator	356
8.8	Pointer Expressions and Pointer Arithmetic	358
8.8.1	Adding Integers to and Subtracting Integers from Pointers	359
8.8.2	Subtracting Pointers	360
8.8.3	Pointer Assignment	361
8.8.4	Cannot Dereference a <code>void*</code>	361
8.8.5	Comparing Pointers	361
8.9	Relationship Between Pointers and Built-In Arrays	361
8.9.1	Pointer/Offset Notation	362
8.9.2	Pointer/Offset Notation with the Built-In Array's Name as the Pointer	362
8.9.3	Pointer/Subscript Notation	362
8.9.4	Demonstrating the Relationship Between Pointers and Built-In Arrays	363
8.10	Pointer-Based Strings (Optional)	364
8.11	Note About Smart Pointers	367
8.12	Wrap-Up	367

9 Classes: A Deeper Look 385

9.1	Introduction	386
9.2	Time Class Case Study: Separating Interface from Implementation	387
9.2.1	Interface of a Class	388
9.2.2	Separating the Interface from the Implementation	388
9.2.3	Time Class Definition	388
9.2.4	Time Class Member Functions	390
9.2.5	Scope Resolution Operator (<code>::</code>)	391
9.2.6	Including the Class Header in the Source-Code File	391

9.2.7	Time Class Member Function <code>setTime</code> and Throwing Exceptions	392
9.2.8	Time Class Member Function <code>toUniversalString</code> and String Stream Processing	392
9.2.9	Time Class Member Function <code>toStandardString</code>	393
9.2.10	Implicitly Inlining Member Functions	393
9.2.11	Member Functions vs. Global Functions	393
9.2.12	Using Class Time	394
9.2.13	Object Size	396
9.3	Compilation and Linking Process	396
9.4	Class Scope and Accessing Class Members	398
9.5	Access Functions and Utility Functions	399
9.6	Time Class Case Study: Constructors with Default Arguments	399
9.6.1	Constructors with Default Arguments	399
9.6.2	Overloaded Constructors and C++11 Delegating Constructors	404
9.7	Destructors	405
9.8	When Constructors and Destructors Are Called	405
9.8.1	Constructors and Destructors for Objects in Global Scope	406
9.8.2	Constructors and Destructors for Non-static Local Objects	406
9.8.3	Constructors and Destructors for static Local Objects	406
9.8.4	Demonstrating When Constructors and Destructors Are Called	406
9.9	Time Class Case Study: A Subtle Trap—Returning a Reference or a Pointer to a private Data Member	409
9.10	Default Memberwise Assignment	411
9.11	<code>const</code> Objects and <code>const</code> Member Functions	413
9.12	Composition: Objects as Members of Classes	415
9.13	<code>friend</code> Functions and <code>friend</code> Classes	421
9.14	Using the <code>this</code> Pointer	423
9.14.1	Implicitly and Explicitly Using the <code>this</code> Pointer to Access an Object's Data Members	424
9.14.2	Using the <code>this</code> Pointer to Enable Cascaded Function Calls	425
9.15	<code>static</code> Class Members	429
9.15.1	Motivating Classwide Data	429
9.15.2	Scope and Initialization of <code>static</code> Data Members	429
9.15.3	Accessing <code>static</code> Data Members	430
9.15.4	Demonstrating <code>static</code> Data Members	430
9.16	Wrap-Up	433

10 Operator Overloading; Class `string` **447**

10.1	Introduction	448
10.2	Using the Overloaded Operators of Standard Library Class <code>string</code>	449
10.3	Fundamentals of Operator Overloading	453
10.3.1	Operator Overloading Is Not Automatic	453
10.3.2	Operators That You Do Not Have to Overload	453
10.3.3	Operators That Cannot Be Overloaded	454
10.3.4	Rules and Restrictions on Operator Overloading	454

10.4	Overloading Binary Operators	455
10.5	Overloading the Binary Stream Insertion and Stream Extraction Operators	455
10.6	Overloading Unary Operators	459
10.7	Overloading the Increment and Decrement Operators	460
10.8	Case Study: A Date Class	461
10.9	Dynamic Memory Management	466
10.10	Case Study: Array Class	468
	10.10.1 Using the Array Class	469
	10.10.2 Array Class Definition	473
10.11	Operators as Member vs. Non-Member Functions	480
10.12	Converting Between Types	481
10.13	<code>explicit</code> Constructors and Conversion Operators	482
10.14	Overloading the Function Call Operator <code>()</code>	485
10.15	Wrap-Up	485

11 Object-Oriented Programming: Inheritance 497

11.1	Introduction	498
11.2	Base Classes and Derived Classes	499
	11.2.1 <code>CommunityMember</code> Class Hierarchy	499
	11.2.2 <code>Shape</code> Class Hierarchy	500
11.3	Relationship between Base and Derived Classes	501
	11.3.1 Creating and Using a <code>CommissionEmployee</code> Class	501
	11.3.2 Creating a <code>BasePlusCommissionEmployee</code> Class Without Using Inheritance	506
	11.3.3 Creating a <code>CommissionEmployee–BasePlusCommissionEmployee</code> Inheritance Hierarchy	511
	11.3.4 <code>CommissionEmployee–BasePlusCommissionEmployee</code> Inheritance Hierarchy Using <code>protected</code> Data	515
	11.3.5 <code>CommissionEmployee–BasePlusCommissionEmployee</code> Inheritance Hierarchy Using <code>private</code> Data	519
11.4	Constructors and Destructors in Derived Classes	523
11.5	<code>public</code> , <code>protected</code> and <code>private</code> Inheritance	525
11.6	Wrap-Up	526

12 Object-Oriented Programming: Polymorphism 531

12.1	Introduction	532
12.2	Introduction to Polymorphism: Polymorphic Video Game	533
12.3	Relationships Among Objects in an Inheritance Hierarchy	534
	12.3.1 Invoking Base-Class Functions from Derived-Class Objects	534
	12.3.2 Aiming Derived-Class Pointers at Base-Class Objects	537
	12.3.3 Derived-Class Member-Function Calls via Base-Class Pointers	538
12.4	Virtual Functions and Virtual Destructors	540
	12.4.1 Why <code>virtual</code> Functions Are Useful	540
	12.4.2 Declaring <code>virtual</code> Functions	540

12.4.3	Invoking a virtual Function Through a Base-Class Pointer or Reference	541
12.4.4	Invoking a virtual Function Through an Object's Name	541
12.4.5	virtual Functions in the CommissionEmployee Hierarchy	541
12.4.6	virtual Destructors	546
12.4.7	C++11: final Member Functions and Classes	546
12.5	Type Fields and switch Statements	547
12.6	Abstract Classes and Pure virtual Functions	547
12.6.1	Pure virtual Functions	548
12.6.2	Device Drivers: Polymorphism in Operating Systems	549
12.7	Case Study: Payroll System Using Polymorphism	549
12.7.1	Creating Abstract Base Class Employee	550
12.7.2	Creating Concrete Derived Class SalariedEmployee	553
12.7.3	Creating Concrete Derived Class CommissionEmployee	556
12.7.4	Creating Indirect Concrete Derived Class BasePlusCommissionEmployee	558
12.7.5	Demonstrating Polymorphic Processing	560
12.8	(Optional) Polymorphism, Virtual Functions and Dynamic Binding "Under the Hood"	563
12.9	Case Study: Payroll System Using Polymorphism and Runtime Type Information with Downcasting, dynamic_cast, typeid and type_info	567
12.10	Wrap-Up	570

13 Stream Input/Output: A Deeper Look **577**

13.1	Introduction	578
13.2	Streams	579
13.2.1	Classic Streams vs. Standard Streams	579
13.2.2	iostream Library Headers	580
13.2.3	Stream Input/Output Classes and Objects	580
13.3	Stream Output	581
13.3.1	Output of char* Variables	581
13.3.2	Character Output Using Member Function put	582
13.4	Stream Input	582
13.4.1	get and getline Member Functions	583
13.4.2	istream Member Functions peek, putback and ignore	586
13.4.3	Type-Safe I/O	586
13.5	Unformatted I/O Using read, write and gcount	586
13.6	Stream Manipulators: A Deeper Look	587
13.6.1	Integral Stream Base: dec, oct, hex and setbase	588
13.6.2	Floating-Point Precision (precision, setprecision)	588
13.6.3	Field Width (width, setw)	590
13.6.4	User-Defined Output Stream Manipulators	591
13.7	Stream Format States and Stream Manipulators	592
13.7.1	Trailing Zeros and Decimal Points (showpoint)	593
13.7.2	Justification (left, right and internal)	594

13.7.3	Padding (<code>fill</code> , <code>setfill</code>)	595
13.7.4	Integral Stream Base (<code>dec</code> , <code>oct</code> , <code>hex</code> , <code>showbase</code>)	597
13.7.5	Floating-Point Numbers; Scientific and Fixed Notation (<code>scientific</code> , <code>fixed</code>)	597
13.7.6	Uppercase/Lowercase Control (<code>uppercase</code>)	598
13.7.7	Specifying Boolean Format (<code>boolalpha</code>)	599
13.7.8	Setting and Resetting the Format State via Member Function <code>flags</code>	600
13.8	Stream Error States	601
13.9	Tying an Output Stream to an Input Stream	604
13.10	Wrap-Up	605

14 File Processing

615

14.1	Introduction	616
14.2	Files and Streams	616
14.3	Creating a Sequential File	617
14.3.1	Opening a File	618
14.3.2	Opening a File via the <code>open</code> Member Function	619
14.3.3	Testing Whether a File Was Opened Successfully	619
14.3.4	Overloaded <code>bool</code> Operator	620
14.3.5	Processing Data	620
14.3.6	Closing a File	620
14.3.7	Sample Execution	621
14.4	Reading Data from a Sequential File	621
14.4.1	Opening a File for Input	622
14.4.2	Reading from the File	622
14.4.3	File-Position Pointers	622
14.4.4	Case Study: Credit Inquiry Program	623
14.5	C++14: Reading and Writing Quoted Text	626
14.6	Updating Sequential Files	627
14.7	Random-Access Files	628
14.8	Creating a Random-Access File	629
14.8.1	Writing Bytes with <code>ostream</code> Member Function <code>write</code>	629
14.8.2	Converting Between Pointer Types with the <code>reinterpret_cast</code> Operator	629
14.8.3	Credit-Processing Program	630
14.8.4	Opening a File for Output in Binary Mode	633
14.9	Writing Data Randomly to a Random-Access File	633
14.9.1	Opening a File for Input and Output in Binary Mode	635
14.9.2	Positioning the File-Position Pointer	635
14.10	Reading from a Random-Access File Sequentially	635
14.11	Case Study: A Transaction-Processing Program	637
14.12	Object Serialization	643
14.13	Wrap-Up	644

15 Standard Library Containers and Iterators 655

15.1	Introduction	656
15.2	Introduction to Containers	658
15.3	Introduction to Iterators	662
15.4	Introduction to Algorithms	667
15.5	Sequence Containers	667
15.5.1	vector Sequence Container	668
15.5.2	list Sequence Container	675
15.5.3	deque Sequence Container	680
15.6	Associative Containers	681
15.6.1	multiset Associative Container	682
15.6.2	set Associative Container	685
15.6.3	multimap Associative Container	687
15.6.4	map Associative Container	689
15.7	Container Adapters	690
15.7.1	stack Adapter	691
15.7.2	queue Adapter	693
15.7.3	priority_queue Adapter	694
15.8	Class bitset	695
15.9	Wrap-Up	697

16 Standard Library Algorithms 707

16.1	Introduction	708
16.2	Minimum Iterator Requirements	708
16.3	Lambda Expressions	710
16.3.1	Algorithm for_each	711
16.3.2	Lambda with an Empty Introducer	711
16.3.3	Lambda with a Nonempty Introducer—Capturing Local Variables	712
16.3.4	Lambda Return Types	712
16.4	Algorithms	712
16.4.1	fill, fill_n, generate and generate_n	712
16.4.2	equal, mismatch and lexicographical_compare	715
16.4.3	remove, remove_if, remove_copy and remove_copy_if	718
16.4.4	replace, replace_if, replace_copy and replace_copy_if	721
16.4.5	Mathematical Algorithms	723
16.4.6	Basic Searching and Sorting Algorithms	726
16.4.7	swap, iter_swap and swap_ranges	731
16.4.8	copy_backward, merge, unique and reverse	732
16.4.9	inplace_merge, unique_copy and reverse_copy	735
16.4.10	Set Operations	737
16.4.11	lower_bound, upper_bound and equal_range	740
16.4.12	min, max, minmax and minmax_element	742
16.5	Function Objects	744
16.6	Standard Library Algorithm Summary	747
16.7	Wrap-Up	749

17 Exception Handling: A Deeper Look 757

17.1	Introduction	758
17.2	Exception-Handling Flow of Control; Defining an Exception Class	759
17.2.1	Defining an Exception Class to Represent the Type of Problem That Might Occur	759
17.2.2	Demonstrating Exception Handling	760
17.2.3	Enclosing Code in a try Block	761
17.2.4	Defining a catch Handler to Process a DivideByZeroException	762
17.2.5	Termination Model of Exception Handling	762
17.2.6	Flow of Program Control When the User Enters a Nonzero Denominator	763
17.2.7	Flow of Program Control When the User Enters a Denominator of Zero	763
17.3	Rethrowing an Exception	764
17.4	Stack Unwinding	766
17.5	When to Use Exception Handling	767
17.6	noexcept: Declaring Functions That Do Not Throw Exceptions	768
17.7	Constructors, Destructors and Exception Handling	768
17.7.1	Destructors Called Due to Exceptions	768
17.7.2	Initializing Local Objects to Acquire Resources	769
17.8	Processing new Failures	769
17.8.1	new Throwing bad_alloc on Failure	769
17.8.2	new Returning nullptr on Failure	770
17.8.3	Handling new Failures Using Function set_new_handler	771
17.9	Class unique_ptr and Dynamic Memory Allocation	772
17.9.1	unique_ptr Ownership	774
17.9.2	unique_ptr to a Built-In Array	775
17.10	Standard Library Exception Hierarchy	775
17.11	Wrap-Up	777

18 Introduction to Custom Templates 783

18.1	Introduction	784
18.2	Class Templates	785
18.2.1	Creating Class Template Stack<T>	786
18.2.2	Class Template Stack<T>'s Data Representation	787
18.2.3	Class Template Stack<T>'s Member Functions	787
18.2.4	Declaring a Class Template's Member Functions Outside the Class Template Definition	788
18.2.5	Testing Class Template Stack<T>	788
18.3	Function Template to Manipulate a Class-Template Specialization Object	790
18.4	Nontype Parameters	792
18.5	Default Arguments for Template Type Parameters	792
18.6	Overloading Function Templates	793
18.7	Wrap-Up	793

19 Custom Templatized Data Structures 797

19.1	Introduction	798
19.1.1	Always Prefer the Standard Library's Containers, Iterators and Algorithms, if Possible	799
19.1.2	Special Section: Building Your Own Compiler	799
19.2	Self-Referential Classes	799
19.3	Linked Lists	800
19.3.1	Testing Our Linked List Implementation	802
19.3.2	Class Template <code>ListNode</code>	805
19.3.3	Class Template <code>List</code>	806
19.3.4	Member Function <code>insertAtFront</code>	809
19.3.5	Member Function <code>insertAtBack</code>	810
19.3.6	Member Function <code>removeFromFront</code>	810
19.3.7	Member Function <code>removeFromBack</code>	811
19.3.8	Member Function <code>print</code>	812
19.3.9	Circular Linked Lists and Double Linked Lists	813
19.4	Stacks	814
19.4.1	Taking Advantage of the Relationship Between <code>Stack</code> and <code>List</code>	815
19.4.2	Implementing a Class Template <code>Stack</code> Class Based By Inheriting from <code>List</code>	815
19.4.3	Dependent Names in Class Templates	816
19.4.4	Testing the <code>Stack</code> Class Template	817
19.4.5	Implementing a Class Template <code>Stack</code> Class With Composition of a <code>List</code> Object	818
19.5	Queues	819
19.5.1	Applications of Queues	819
19.5.2	Implementing a Class Template <code>Queue</code> Class Based By Inheriting from <code>List</code>	820
19.5.3	Testing the <code>Queue</code> Class Template	821
19.6	Trees	823
19.6.1	Basic Terminology	823
19.6.2	Binary Search Trees	824
19.6.3	Testing the <code>Tree</code> Class Template	824
19.6.4	Class Template <code>TreeNode</code>	826
19.6.5	Class Template <code>Tree</code>	827
19.6.6	<code>Tree</code> Member Function <code>insertNodeHelper</code>	829
19.6.7	<code>Tree</code> Traversal Functions	829
19.6.8	Duplicate Elimination	830
19.6.9	Overview of the Binary Tree Exercises	830
19.7	Wrap-Up	831

20 Searching and Sorting 841

20.1	Introduction	842
20.2	Searching Algorithms	843
20.2.1	Linear Search	843

20.2.2	Binary Search	846
20.3	Sorting Algorithms	850
20.3.1	Insertion Sort	851
20.3.2	Selection Sort	853
20.3.3	Merge Sort (A Recursive Implementation)	855
20.4	Wrap-Up	862

21 Class string and String Stream Processing: A Deeper Look **869**

21.1	Introduction	870
21.2	string Assignment and Concatenation	871
21.3	Comparing strings	873
21.4	Substrings	876
21.5	Swapping strings	876
21.6	string Characteristics	877
21.7	Finding Substrings and Characters in a string	880
21.8	Replacing Characters in a string	881
21.9	Inserting Characters into a string	883
21.10	Conversion to Pointer-Based char* Strings	884
21.11	Iterators	886
21.12	String Stream Processing	887
21.13	C++11 Numeric Conversion Functions	890
21.14	Wrap-Up	892

22 Bits, Characters, C Strings and structs **899**

22.1	Introduction	900
22.2	Structure Definitions	900
22.3	typedef and using	902
22.4	Example: Card Shuffling and Dealing Simulation	902
22.5	Bitwise Operators	905
22.6	Bit Fields	914
22.7	Character-Handling Library	918
22.8	C String-Manipulation Functions	923
22.9	C String-Conversion Functions	930
22.10	Search Functions of the C String-Handling Library	935
22.11	Memory Functions of the C String-Handling Library	939
22.12	Wrap-Up	943

Chapters on the Web **959**

A Operator Precedence and Associativity **961**

B ASCII Character Set **963**

C	Fundamental Types	965
D	Number Systems	967
D.1	Introduction	968
D.2	Abbreviating Binary Numbers as Octal and Hexadecimal Numbers	971
D.3	Converting Octal and Hexadecimal Numbers to Binary Numbers	972
D.4	Converting from Binary, Octal or Hexadecimal to Decimal	972
D.5	Converting from Decimal to Binary, Octal or Hexadecimal	973
D.6	Negative Binary Numbers: Two's Complement Notation	975
E	Preprocessor	981
E.1	Introduction	982
E.2	<code>#include</code> Preprocessing Directive	982
E.3	<code>#define</code> Preprocessing Directive: Symbolic Constants	983
E.4	<code>#define</code> Preprocessing Directive: Macros	983
E.5	Conditional Compilation	985
E.6	<code>#error</code> and <code>#pragma</code> Preprocessing Directives	987
E.7	Operators <code>#</code> and <code>##</code>	987
E.8	Predefined Symbolic Constants	987
E.9	Assertions	988
E.10	Wrap-Up	988
	Appendices on the Web	993
	Index	995

Chapters 23–26 and Appendices F–J are PDF documents posted online at the book's Companion Website, which is accessible from

<http://www.pearsonhighered.com/deitel>

See the inside front cover for more information.

23 Other Topics

24 C++11 and C++14: Additional Features

25 ATM Case Study, Part 1: Object-Oriented Design with the UM

26 ATM Case Study, Part 2: Implementing an Object-Oriented Design

- F** **C Legacy Code Topics**
- G** **UML: Additional Diagram Types**
- H** **Using the Visual Studio Debugger**
- I** **Using the GNU C++ Debugger**
- J** **Using the Xcode Debugger**