



Contents

Preface

xxiii

Before You Begin

liii

I	Intro to Computers and C++	I
1.1	Introduction	2
1.2	Hardware	4
1.2.1	Computer Organization	4
1.2.2	Moore's Law, Multi-Core Processors and Concurrent Programming	7
1.3	Data Hierarchies	10
1.4	Machine Languages, Assembly Languages and High-Level Languages	13
1.5	Operating Systems	14
1.6	C and C++	18
1.7	C++ Standard Library and Open-Source Libraries	19
1.8	Other Popular Programming Languages	21
1.9	Introduction to Object Orientation	23
1.10	Simplified View of a C++ Development Environment	25
1.11	Test-Driving a C++20 Application Various Ways	28
1.11.1	Compiling and Running on Windows with Visual Studio Community Edition	29
1.11.2	Compiling and Running with GNU C++ on Linux	32
1.11.3	Compiling and Running with g++ in the GCC Docker Container	34
1.11.4	Compiling and Running with clang++ in a Docker Container	36
1.11.5	Compiling and Running with Xcode on macOS	37
1.12	Internet, World Wide Web, the Cloud and IoT	41
1.12.1	The Internet: A Network of Networks	41
1.12.2	The World Wide Web: Making the Internet User-Friendly	42
1.12.3	The Cloud	42
1.12.4	The Internet of Things (IoT)	42
1.12.5	Edge Computing	43
1.12.6	Mashups	43
1.13	Metaverse	44
1.13.1	Virtual Reality (VR)	44
1.13.2	Augmented Reality (AR)	45
1.13.3	Mixed Reality	46
1.13.4	Blockchain	46
1.13.5	Bitcoin and Cryptocurrency	46

1.13.6	Ethereum	47
1.13.7	Non-Fungible Tokens (NFTs)	47
1.13.8	Web3	47
1.14	Software Development Technologies	48
1.15	How Big Is Big Data?	49
1.15.1	Big-Data Analytics	52
1.15.2	Data Science and Big Data Are Making a Difference: Use Cases	53
1.16	AI—at the Intersection of Computer Science and Data Science	54
1.16.1	Artificial Intelligence (AI)	54
1.16.2	Artificial General Intelligence (AGI)	55
1.16.3	Artificial Intelligence Milestones	55
1.16.4	Machine Learning	56
1.16.5	Deep Learning	56
1.16.6	Reinforcement Learning	57
1.16.7	Generative AI—ChatGPT and Dall-E 2	57
1.17	Wrap-Up	59
2	Intro to C++20 Programming	65
2.1	Introduction	66
2.2	First Program in C++: Displaying a Line of Text	66
2.3	Modifying Our First C++ Program	70
2.4	Another C++ Program: Adding Integers	71
2.5	Memory Concepts	75
2.6	Arithmetic	76
2.7	Decision Making: Equality and Relational Operators	79
2.8	Objects Natural Case Study: Creating and Using Objects of Standard-Library Class <code>string</code>	83
2.9	Wrap-Up	86
3	Algorithm Development and Control Statements: Part I	93
3.1	Introduction	94
3.2	Algorithms	95
3.3	Pseudocode	95
3.4	Control Structures	96
3.4.1	Sequence Structure	97
3.4.2	Selection Statements	98
3.4.3	Iteration Statements	98
3.4.4	Summary of Control Statements	99
3.5	<code>if</code> Single-Selection Statement	100
3.6	<code>if...else</code> Double-Selection Statement	101
3.6.1	Nested <code>if...else</code> Statements	102
3.6.2	Blocks	104
3.6.3	Conditional Operator (<code>?:</code>)	104
3.7	<code>while</code> Iteration Statement	105

3.8	Formulating Algorithms: Counter-Controlled Iteration	107
3.8.1	Pseudocode Algorithm with Counter-Controlled Iteration	107
3.8.2	Implementing Counter-Controlled Iteration	108
3.8.3	Integer Division and Truncation	110
3.8.4	Arithmetic Overflow	110
3.8.5	Input Validation	110
3.9	Formulating Algorithms: Sentinel-Controlled Iteration	111
3.9.1	Top-Down, Stepwise Refinement: The Top and First Refinement	112
3.9.2	Proceeding to the Second Refinement	112
3.9.3	Implementing Sentinel-Controlled Iteration	114
3.9.4	Mixed-Type Expressions and Implicit Type Promotions	116
3.9.5	Formatting Floating-Point Numbers	117
3.10	Formulating Algorithms: Nested Control Statements	118
3.10.1	Problem Statement	118
3.10.2	Top-Down, Stepwise Refinement: Pseudocode Representation of the Top	119
3.10.3	Top-Down, Stepwise Refinement: First Refinement	119
3.10.4	Top-Down, Stepwise Refinement: Second Refinement	120
3.10.5	Complete Second Refinement of the Pseudocode	120
3.10.6	Implementing the Program	121
3.10.7	Preventing Narrowing Conversions with Braced Initialization	123
3.11	Compound Assignment Operators	125
3.12	Increment and Decrement Operators	125
3.13	Fundamental Types Are Not Portable	128
3.14	Objects Natural Case Study: Super-Sized Integers	129
3.15	Wrap-Up	134

4 Control Statements, Part 2 **I45**

4.1	Introduction	146
4.2	Essentials of Counter-Controlled Iteration	146
4.3	for Iteration Statement	148
4.4	Examples Using the for Statement	151
4.5	Application: Summing Even Integers; Introducing C++20 Text Formatting	152
4.6	Application: Compound-Interest Calculations; Introducing Format Specifiers	153
4.7	do...while Iteration Statement	158
4.8	switch Multiple-Selection Statement	159
4.9	Selection Statements with Initializers	165
4.10	break and continue Statements	167
4.11	Logical Operators	169
4.11.1	Logical AND (&&) Operator	169
4.11.2	Logical OR () Operator	170
4.11.3	Short-Circuit Evaluation	170
4.11.4	Logical Negation (!) Operator	171
4.11.5	Example: Producing Logical-Operator Truth Tables	171

4.12	Confusing the Equality (==) and Assignment (=) Operators	173
4.13	Structured-Programming Summary	175
4.14	Objects Natural Case Study: Precise Monetary Calculations with the Boost Multiprecision Library	180
4.15	Wrap-Up	183

5 Functions and an Intro to Function Templates 189

5.1	Introduction	190
5.2	C++ Program Components	191
5.3	Math Library Functions	192
5.4	Function Definitions and Function Prototypes	194
5.5	Order of Evaluation of a Function's Arguments	197
5.6	Function-Prototype and Argument-Coercion Notes	197
5.6.1	Function Signatures and Function Prototypes	198
5.6.2	Argument Coercion	198
5.6.3	Argument-Promotion Rules and Implicit Conversions	198
5.7	C++ Standard Library Headers	200
5.8	Case Study: Random-Number Generation	203
5.8.1	Rolling a Six-Sided Die	204
5.8.2	Rolling a Six-Sided Die 60,000,000 Times	205
5.8.3	Seeding the Random-Number Generator	207
5.8.4	Seeding the Random-Number Generator with <code>random_device</code>	208
5.9	Case Study: Game of Chance; Introducing Scoped <code>enums</code>	209
5.10	Function-Call Stack and Activation Records	214
5.11	Inline Functions	217
5.12	References and Reference Parameters	219
5.13	Default Arguments	222
5.14	Function Overloading	224
5.15	Function Templates	227
5.16	Recursion	229
5.16.1	Factorials	230
5.16.2	Recursive Factorial Example	230
5.16.3	Recursive Fibonacci Series Example	234
5.16.4	Recursion vs. Iteration	237
5.17	Scope Rules	239
5.18	Unary Scope Resolution Operator	244
5.19	<code>Lnfylun Lhqtomh Wjtz Qarcv: Qjwazkrplm xzz Xndmwwqhlz</code>	245
5.20	Wrap-Up	248

6 arrays, vectors, Ranges and Functional-Style Programming 259

6.1	Introduction	260
6.2	arrays	261
6.3	Declaring arrays	262
6.4	Initializing array Elements in a Loop	262

6.5	Initializing an array with an Initializer List	265
6.6	Range-Based <code>for</code> Statement	267
6.7	Calculating array Element Values and an Intro to <code>constexpr</code>	269
6.8	Totaling array Elements	271
6.9	Using a Primitive Bar Chart to Display array Data Graphically	272
6.10	Using array Elements as Counters	274
6.11	Using arrays to Summarize Survey Results	275
6.12	Sorting and Searching arrays	277
6.13	Multidimensional arrays	279
6.14	Intro to Functional-Style Programming	283
6.14.1	What vs. How	283
6.14.2	Passing Functions as Arguments to Other Functions: Introducing Lambda Expressions	284
6.14.3	Filter, Map and Reduce: Intro to C++20's Ranges Library	286
6.15	Objects-Natural Case Study: C++ Standard Library Class Template <code>vector</code>	290
6.16	Wrap-Up	297
7	(Downplaying) Pointers in Modern C++	309
7.1	Introduction	310
7.2	Pointer Variable Declarations and Initialization	312
7.2.1	Declaring Pointers	312
7.2.2	Initializing Pointers	312
7.2.3	Null Pointers	312
7.3	Pointer Operators	313
7.3.1	Address (<code>&</code>) Operator	313
7.3.2	Indirection (<code>*</code>) Operator	314
7.3.3	Using the Address (<code>&</code>) and Indirection (<code>*</code>) Operators	315
7.4	Pass-by-Reference with Pointers	316
7.5	Built-In Arrays	320
7.5.1	Declaring and Accessing a Built-In Array	320
7.5.2	Initializing Built-In Arrays	321
7.5.3	Passing Built-In Arrays to Functions	322
7.5.4	Declaring Built-In Array Parameters	322
7.5.5	Standard Library Functions <code>begin</code> and <code>end</code>	323
7.5.6	Built-In Array Limitations	323
7.6	Using C++20 <code>to_array</code> to Convert a Built-in Array to a <code>std::array</code>	324
7.7	Using <code>const</code> with Pointers and the Data Pointed To	325
7.7.1	Using a Nonconstant Pointer to Nonconstant Data	326
7.7.2	Using a Nonconstant Pointer to Constant Data	326
7.7.3	Using a Constant Pointer to Nonconstant Data	327
7.7.4	Using a Constant Pointer to Constant Data	327
7.8	<code>sizeof</code> Operator	329
7.9	Pointer Expressions and Pointer Arithmetic	331
7.9.1	Adding Integers to and Subtracting Integers from Pointers	332

7.9.2	Subtracting One Pointer from Another	333
7.9.3	Pointer Assignment	333
7.9.4	Cannot Dereference a <code>void*</code> Pointer	333
7.9.5	Comparing Pointers	334
7.10	Objects-Natural Case Study: C++20 <code>spans</code> —Views of Contiguous Container Elements	334
7.11	A Brief Intro to Pointer-Based Strings	340
7.11.1	Command-Line Arguments	341
7.11.2	Revisiting C++20's <code>to_array</code> Function	342
7.12	Looking Ahead to Other Pointer Topics	344
7.13	Wrap-Up	344

8 strings, string_views, Text Files, CSV Files and Regex 357

8.1	Introduction	358
8.2	<code>string</code> Assignment and Concatenation	359
8.3	Comparing strings	361
8.4	Substrings	363
8.5	Swapping strings	364
8.6	<code>string</code> Characteristics	365
8.7	Finding Substrings and Characters in a <code>string</code>	367
8.8	Replacing and Erasing Characters in a <code>string</code>	370
8.9	Inserting Characters into a <code>string</code>	372
8.10	Numeric Conversions	373
8.11	<code>string_view</code>	374
8.12	Files and Streams	377
8.13	Creating a Sequential File	378
8.14	Reading Data from a Sequential File	381
8.15	Reading and Writing Quoted Text	384
8.16	Updating Sequential Files	385
8.17	String Stream Processing	386
8.18	Raw String Literals	389
8.19	Objects-Natural Data Science Case Study: Reading and Analyzing a CSV File Containing <i>Titanic</i> Disaster Data	390
8.19.1	Using <code>rapidcsv</code> to Read the Contents of a CSV File	390
8.19.2	Reading and Analyzing the <i>Titanic</i> Disaster Dataset	392
8.20	Objects-Natural Data Science Case Study: Intro to Regular Expressions	399
8.20.1	Matching Complete Strings to Patterns	400
8.20.2	Replacing Substrings	405
8.20.3	Searching for Matches	405
8.21	Wrap-Up	408

9 Custom Classes 431

9.1	Introduction	432
-----	--------------	-----

9.2	Test-Driving an Account Object	433
9.3	Account Class with a Data Member and <i>Set</i> and <i>Get</i> Member Functions	435
9.3.1	Class Definition	435
9.3.2	Access Specifiers <i>private</i> and <i>public</i>	437
9.4	Account Class: Custom Constructors	438
9.5	Software Engineering with <i>Set</i> and <i>Get</i> Member Functions	442
9.6	Account Class with a Balance	444
9.7	Time Class Case Study: Separating Interface from Implementation	447
9.7.1	Interface of a Class	448
9.7.2	Separating the Interface from the Implementation	448
9.7.3	Class Definition	449
9.7.4	Member Functions	450
9.7.5	Including the Class Header in the Source-Code File	450
9.7.6	Scope Resolution Operator (<i>::</i>)	451
9.7.7	Member Function <i>setTime</i> and Throwing Exceptions	451
9.7.8	Member Functions <i>to24HourString</i> and <i>to12HourString</i>	451
9.7.9	Implicitly Inlining Member Functions	452
9.7.10	Member Functions vs. Global Functions	452
9.7.11	Using Class <i>Time</i>	452
9.7.12	Object Size	454
9.8	Compilation and Linking Process	454
9.9	Class Scope and Accessing Class Members	455
9.10	Access Functions and Utility Functions	456
9.11	Time Class Case Study: Constructors with Default Arguments	457
9.11.1	Class <i>Time</i>	457
9.11.2	Overloaded Constructors and Delegating Constructors	462
9.12	Destructors	463
9.13	When Constructors and Destructors Are Called	464
9.14	Time Class Case Study: A Subtle Trap — Returning a Reference or a Pointer to a <i>private</i> Data Member	468
9.15	Default Assignment Operator	470
9.16	<i>const</i> Objects and <i>const</i> Member Functions	472
9.17	Composition: Objects as Members of Classes	474
9.18	<i>friend</i> Functions and <i>friend</i> Classes	480
9.19	The <i>this</i> Pointer	482
9.19.1	Implicitly and Explicitly Using the <i>this</i> Pointer to Access an Object's Data Members	482
9.19.2	Using the <i>this</i> Pointer to Enable Cascaded Function Calls	484
9.20	<i>static</i> Class Members: Classwide Data and Member Functions	487
9.21	Aggregates in C++20	492
9.21.1	Initializing an Aggregate	493
9.21.2	C++20 Designated Initializers	493
9.22	Concluding Our Objects Natural Case Study Track: Studying the Vigenère Secret-Key Cipher Implementation	494
9.23	Wrap-Up	507

10 OOP: Inheritance and Runtime Polymorphism 529

10.1	Introduction	530
10.2	Base Classes and Derived Classes	532
10.2.1	CommunityMember Class Hierarchy	533
10.2.2	Shape Class Hierarchy and public Inheritance	534
10.3	Relationship Between Base and Derived Classes	535
10.3.1	Creating and Using a SalariedEmployee Class	535
10.3.2	Creating a SalariedEmployee–SalariedCommissionEmployee Inheritance Hierarchy	538
10.4	Constructors and Destructors in Derived Classes	544
10.5	Intro to Runtime Polymorphism: Polymorphic Video Game	545
10.6	Relationships Among Objects in an Inheritance Hierarchy	546
10.6.1	Invoking Base-Class Functions from Derived-Class Objects	547
10.6.2	Aiming Derived-Class Pointers at Base-Class Objects	549
10.6.3	Derived-Class Member-Function Calls via Base-Class Pointers	550
10.7	Virtual Functions and Virtual Destructors	551
10.7.1	Why virtual Functions Are Useful	551
10.7.2	Declaring virtual Functions	552
10.7.3	Invoking a virtual Function	553
10.7.4	virtual Functions in the SalariedEmployee Hierarchy	553
10.7.5	virtual Destructors	557
10.7.6	final Member Functions and Classes	557
10.8	Abstract Classes and Pure virtual Functions	558
10.8.1	Pure virtual Functions	559
10.8.2	Device Drivers: Polymorphism in Operating Systems	559
10.9	Case Study: Payroll System Using Runtime Polymorphism	560
10.9.1	Creating Abstract Base Class Employee	561
10.9.2	Creating Concrete Derived Class SalariedEmployee	563
10.9.3	Creating Concrete Derived Class CommissionEmployee	565
10.9.4	Demonstrating Runtime Polymorphic Processing	567
10.10	Runtime Polymorphism, Virtual Functions and Dynamic Binding “Under the Hood”	570
10.11	Program to an Interface, Not an Implementation	573
10.11.1	Rethinking the Employee Hierarchy—CompensationModel Interface	575
10.11.2	Class Employee	575
10.11.3	CompensationModel Implementations	577
10.11.4	Testing the New Hierarchy	579
10.11.5	Dependency Injection Design Benefits	580
10.12	Wrap-Up	581

11 Operator Overloading, Copy/Move Semantics and Smart Pointers 587

11.1	Introduction	588
11.2	Using the Overloaded Operators of Standard Library Class string	590

11.3	Operator Overloading Fundamentals	596
11.3.1	Operator Overloading Is Not Automatic	596
11.3.2	Operators That Cannot Be Overloaded	596
11.3.3	Operators That You Do Not Have to Overload	596
11.3.4	Rules and Restrictions on Operator Overloading	597
11.4	(Downplaying) Dynamic Memory Management with <code>new</code> and <code>delete</code>	598
11.5	Modern C++ Dynamic Memory Management: RAII and Smart Pointers	600
11.5.1	Smart Pointers	601
11.5.2	Demonstrating <code>unique_ptr</code>	601
11.5.3	<code>unique_ptr</code> Ownership	603
11.5.4	<code>unique_ptr</code> to a Built-In Array	603
11.6	<code>MyArray</code> Case Study: Crafting a Valuable Class with Operator Overloading	604
11.6.1	Special Member Functions	605
11.6.2	Using Class <code>MyArray</code>	606
11.6.3	<code>MyArray</code> Class Definition	615
11.6.4	Constructor That Specifies a <code>MyArray</code> 's Size	617
11.6.5	Passing a Braced Initializer to a Constructor	618
11.6.6	Copy Constructor and Copy Assignment Operator	619
11.6.7	Move Constructor and Move Assignment Operator	622
11.6.8	Destructor	626
11.6.9	<code>toString</code> and <code>size</code> Functions	626
11.6.10	Overloading the Equality (<code>==</code>) and Inequality (<code>!=</code>) Operators	627
11.6.11	Overloading the Subscript (<code>[]</code>) Operator	629
11.6.12	Overloading the Unary <code>bool</code> Conversion Operator	630
11.6.13	Overloading the Preincrement Operator	631
11.6.14	Overloading the Postincrement Operator	632
11.6.15	Overloading the Addition Assignment Operator (<code>+=</code>)	633
11.6.16	Overloading the Binary Stream Extraction (<code>>></code>) and Stream Insertion (<code><<</code>) Operators	633
11.6.17	<code>friend</code> Function <code>swap</code>	636
11.7	C++20 Three-Way Comparison Operator (<code><=></code>)	636
11.8	Converting Between Types	640
11.9	<code>explicit</code> Constructors and Conversion Operators	641
11.10	Overloading the Function Call Operator (<code>()</code>)	644
11.11	Wrap-Up	644

12 Exceptions and a Look Forward to Contracts 653

12.1	Introduction	654
12.2	Exception-Handling Flow of Control	658
12.2.1	Defining a Custom Exception Class	658
12.2.2	Demonstrating Exception Handling	659
12.2.3	Enclosing Code in a <code>try</code> Block	660
12.2.4	Defining a <code>catch</code> Handler for <code>DivideByZeroExceptions</code>	661
12.2.5	Termination Model of Exception Handling	662
12.2.6	Flow of Control When the User Enters a Nonzero Denominator	663
12.2.7	Flow of Control When the User Enters a Zero Denominator	663

12.3	Exception Safety Guarantees and <code>noexcept</code>	664
12.4	Rethrowing an Exception	665
12.5	Stack Unwinding and Uncaught Exceptions	667
12.6	When to Use Exception Handling	669
12.6.1	<code>assert</code> Macro	671
12.6.2	Failing Fast	671
12.7	Constructors, Destructors and Exception Handling	672
12.7.1	Throwing Exceptions from Constructors	672
12.7.2	Catching Exceptions in Constructors via Function <code>try</code> Blocks	673
12.7.3	Exceptions and Destructors: Revisiting <code>noexcept(false)</code>	675
12.8	Processing <code>new</code> Failures	676
12.8.1	<code>new</code> Throwing <code>bad_alloc</code> on Failure	677
12.8.2	<code>new</code> Returning <code>nullptr</code> on Failure	678
12.8.3	Handling <code>new</code> Failures Using Function <code>set_new_handler</code>	679
12.9	Standard Library Exception Hierarchy	680
12.10	C++'s Alternative to the <code>finally</code> Block: Resource Acquisition Is Initialization (RAII)	683
12.11	Some Libraries Support Both Exceptions and Error Codes	683
12.12	Logging	685
12.13	Looking Ahead to Contracts	685
12.14	Wrap-Up	694

13 Data Structures: Standard Library Containers and Iterators

697

13.1	Introduction	698
13.2	A Brief Intro to Big <i>O</i>	700
13.3	A Brief Intro to Hash Tables	703
13.4	Introduction to Containers	704
13.4.1	Common Nested Types in Sequence and Associative Containers	706
13.4.2	Common Container Member and Non-Member Functions	707
13.4.3	Requirements for Container Elements	710
13.5	Working with Iterators	710
13.5.1	Using <code>istream_iterator</code> for Input and <code>ostream_iterator</code> for Output	710
13.5.2	Iterator Categories	712
13.5.3	Container Support for Iterators	712
13.5.4	Predefined Iterator Type Names	713
13.5.5	Iterator Operators	713
13.6	A Brief Introduction to Algorithms	714
13.7	Sequence Containers	715
13.8	<code>vector</code> Sequence Container	715
13.8.1	Using <code>vectors</code> and Iterators	716
13.8.2	<code>vector</code> Element-Manipulation Functions	719
13.9	<code>list</code> Sequence Container	723
13.10	<code>deque</code> Sequence Container	728

13.11	Associative Containers	730
13.11.1	<code>multiset</code> Associative Container	730
13.11.2	<code>set</code> Associative Container	734
13.11.3	<code>multimap</code> Associative Container	736
13.11.4	<code>map</code> Associative Container	738
13.12	Container Adaptors	739
13.12.1	<code>stack</code> Adaptor	740
13.12.2	<code>queue</code> Adaptor	742
13.12.3	<code>priority_queue</code> Adaptor	743
13.13	<code>bitset</code> Near Container	744
13.14	Wrap-Up	746

14 Standard Library Algorithms and C++20 Ranges & Views 773

14.1	Introduction	774
14.2	Algorithm Requirements: C++20 Concepts	776
14.3	Lambdas and Algorithms	778
14.4	Algorithms	781
14.4.1	<code>fill</code> , <code>fill_n</code> , <code>generate</code> and <code>generate_n</code>	781
14.4.2	<code>equal</code> , <code>mismatch</code> and <code>lexicographical_compare</code>	783
14.4.3	<code>remove</code> , <code>remove_if</code> , <code>remove_copy</code> and <code>remove_copy_if</code>	786
14.4.4	<code>replace</code> , <code>replace_if</code> , <code>replace_copy</code> and <code>replace_copy_if</code>	790
14.4.5	Shuffling, Counting, and Minimum and Maximum Element Algorithms	792
14.4.6	Searching and Sorting Algorithms	796
14.4.7	<code>swap</code> , <code>iter_swap</code> and <code>swap_ranges</code>	800
14.4.8	<code>copy_backward</code> , <code>merge</code> , <code>unique</code> , <code>reverse</code> , <code>copy_if</code> and <code>copy_n</code>	802
14.4.9	<code>inplace_merge</code> , <code>unique_copy</code> and <code>reverse_copy</code>	805
14.4.10	Set Operations	807
14.4.11	<code>lower_bound</code> , <code>upper_bound</code> and <code>equal_range</code>	810
14.4.12	<code>min</code> , <code>max</code> and <code>minmax</code>	812
14.4.13	Algorithms <code>gcd</code> , <code>lcm</code> , <code>iota</code> , <code>reduce</code> and <code>partial_sum</code> from Header <code><numeric></code>	813
14.4.14	Heapsort and Priority Queues	816
14.5	Function Objects (Functors)	821
14.6	Projections	825
14.7	C++20 Views and Functional-Style Programming	828
14.7.1	Range Adaptors	828
14.7.2	Working with Range Adaptors and Views	830
14.8	Intro to Parallel Algorithms	834
14.9	Standard Library Algorithm Summary	836
14.10	A Look Ahead to C++23 Ranges	839
14.11	Wrap-Up	840

15	Templates, C++20 Concepts and Metaprogramming	845
15.1	Introduction	846
15.2	Custom Class Templates and Compile-Time Polymorphism	849
15.3	C++20 Function Template Enhancements	854
15.3.1	C++20 Abbreviated Function Templates	854
15.3.2	C++20 Templated Lambdas	856
15.4	C++20 Concepts: A First Look	856
15.4.1	Unconstrained Function Template <code>multiply</code>	857
15.4.2	Constrained Function Template with a C++20 Concepts <code>requires</code> Clause	860
15.4.3	C++20 Predefined Concepts	862
15.5	Type Traits	864
15.6	C++20 Concepts: A Deeper Look	868
15.6.1	Creating a Custom Concept	868
15.6.2	Using a Concept	869
15.6.3	Using Concepts in Abbreviated Function Templates	870
15.6.4	Concept-Based Overloading	871
15.6.5	<code>requires</code> Expressions	874
15.6.6	C++20 Exposition-Only Concepts	877
15.6.7	Techniques Before C++20 Concepts: SFINAE and Tag Dispatch	878
15.7	Testing C++20 Concepts with <code>static_assert</code>	879
15.8	Creating a Custom Algorithm	881
15.9	Creating a Custom Container and Iterators	883
15.9.1	Class Template <code>ConstIterator</code>	885
15.9.2	Class Template <code>Iterator</code>	888
15.9.3	Class Template <code>MyArray</code>	890
15.9.4	<code>MyArray</code> Deduction Guide for Braced Initialization	893
15.9.5	Using <code>MyArray</code> with <code>std::ranges</code> Algorithms	894
15.10	Default Arguments for Template Type Parameters	898
15.11	Variable Templates	898
15.12	Variadic Templates and Fold Expressions	899
15.12.1	<code>tuple</code> Variadic Class Template	899
15.12.2	Variadic Function Templates and an Intro to Fold Expressions	902
15.12.3	Types of Fold Expressions	906
15.12.4	How Unary Fold Expressions Apply Their Operators	906
15.12.5	How Binary-Fold Expressions Apply Their Operators	909
15.12.6	Using the Comma Operator to Repeatedly Perform an Operation	910
15.12.7	Constraining Parameter Pack Elements to the Same Type	911
15.13	Template Metaprogramming	913
15.13.1	C++ Templates Are Turing Complete	914
15.13.2	Computing Values at Compile-Time	914
15.13.3	Conditional Compilation with Template Metaprogramming and <code>constexpr if</code>	919
15.13.4	Type Metafunctions	921
15.14	Wrap-Up	925

16 C++20 Modules: Large-Scale Development 933

16.1	Introduction	934
16.2	Compilation and Linking Before C++20	936
16.3	Advantages and Goals of Modules	937
16.4	Example: Transitioning to Modules—Header Units	938
16.5	Modules Can Reduce Translation Unit Sizes and Compilation Times	941
16.6	Example: Creating and Using a Module	942
16.6.1	module Declaration for a Module Interface Unit	943
16.6.2	Exporting a Declaration	945
16.6.3	Exporting a Group of Declarations	945
16.6.4	Exporting a namespace	945
16.6.5	Exporting a namespace Member	946
16.6.6	Importing a Module to Use Its Exported Declarations	946
16.6.7	Example: Attempting to Access Non-Exported Module Contents	948
16.7	Global Module Fragment	951
16.8	Separating Interface from Implementation	951
16.8.1	Example: Module Implementation Units	951
16.8.2	Example: Modularizing a Class	954
16.8.3	:private Module Fragment	958
16.9	Partitions	958
16.9.1	Example: Module Interface Partition Units	959
16.9.2	Module Implementation Partition Units	962
16.9.3	Example: “Submodules” vs. Partitions	962
16.10	Additional Modules Examples	967
16.10.1	Example: Importing the C++ Standard Library as Modules	967
16.10.2	Example: Cyclic Dependencies Are Not Allowed	969
16.10.3	Example: imports Are Not Transitive	970
16.10.4	Example: Visibility vs. Reachability	971
16.11	Migrating Code to Modules	972
16.12	Future of Modules and Modules Tooling	973
16.13	Wrap-Up	975

17 Parallel Algorithms and Concurrency: A High-Level View 987

17.1	Introduction	988
17.2	Standard Library Parallel Algorithms	991
17.2.1	Example: Profiling Sequential and Parallel Sorting Algorithms	991
17.2.2	When to Use Parallel Algorithms	994
17.2.3	Execution Policies	995
17.2.4	Example: Profiling Parallel and Vectorized Operations	996
17.2.5	Additional Parallel Algorithm Notes	998
17.3	Multithreaded Programming	999
17.3.1	Thread States and the Thread Life Cycle	999
17.3.2	Deadlock and Indefinite Postponement	1001

17.4	Launching Tasks with <code>std::jthread</code>	1003
17.4.1	Defining a Task to Perform in a Thread	1003
17.4.2	Executing a Task in a <code>jthread</code>	1005
17.4.3	How <code>jthread</code> Fixes <code>thread</code>	1007
17.5	Producer–Consumer Relationship: A First Attempt	1008
17.6	Producer–Consumer: Synchronizing Access to Shared Mutable Data	1015
17.6.1	Class <code>SynchronizedBuffer</code> : Mutexes, Locks and Condition Variables	1017
17.6.2	Testing <code>SynchronizedBuffer</code>	1023
17.7	Producer–Consumer: Minimizing Waits with a Circular Buffer	1027
17.8	Readers and Writers	1036
17.9	Cooperatively Canceling <code>jthreads</code>	1037
17.10	Launching Tasks with <code>std::async</code>	1040
17.11	Thread-Safe, One-Time Initialization	1047
17.12	A Brief Introduction to Atomics	1048
17.13	Coordinating Threads with C++20 Latches and Barriers	1052
17.13.1	C++20 <code>std::latch</code>	1052
17.13.2	C++20 <code>std::barrier</code>	1055
17.14	C++20 Semaphores	1058
17.15	C++23: A Look to the Future of C++ Concurrency	1062
17.15.1	Parallel Ranges Algorithms	1062
17.15.2	Concurrent Containers	1062
17.15.3	Other Concurrency-Related Proposals	1063
17.16	Wrap-Up	1063

18 C++20 Coroutines **1073**

18.1	Introduction	1074
18.2	Coroutine Support Libraries	1075
18.3	Installing the <code>conurrencpp</code> and <code>generator</code> Libraries	1077
18.4	Creating a Generator Coroutine with <code>co_yield</code> and the <code>generator</code> Library	1077
18.5	Launching Tasks with <code>conurrencpp</code>	1081
18.6	Creating a Coroutine with <code>co_await</code> and <code>co_return</code>	1085
18.7	Low-Level Coroutines Concepts	1093
18.8	Future Coroutines Enhancements	1096
18.9	Wrap-Up	1096

19 Stream I/O & C++20 Text Formatting **1101**

19.1	Introduction	1102
19.2	Streams	1102
19.2.1	Classic Streams vs. Standard Streams	1103
19.2.2	<code>iostream</code> Library Headers	1103
19.2.3	Stream Input/Output Classes and Objects	1103

19.3	Stream Output	1104
19.3.1	Output of <code>char*</code> Variables	1105
19.3.2	Character Output Using Member Function <code>put</code>	1106
19.4	Stream Input	1106
19.4.1	<code>get</code> and <code>getline</code> Member Functions	1106
19.4.2	<code>istream</code> Member Functions <code>peek</code> , <code>putback</code> and <code>ignore</code>	1109
19.5	Unformatted I/O Using <code>read</code> , <code>write</code> and <code>gcount</code>	1110
19.6	Stream Manipulators	1111
19.6.1	Integral Stream Base: <code>dec</code> , <code>oct</code> , <code>hex</code> and <code>setbase</code>	1112
19.6.2	Floating-Point Precision (<code>setprecision</code> , <code>precision</code>)	1112
19.6.3	Field Width (<code>width</code> , <code>setw</code>)	1114
19.6.4	User-Defined Output Stream Manipulators	1115
19.6.5	Trailing Zeros and Decimal Points (<code>showpoint</code>)	1116
19.6.6	Alignment (<code>left</code> , <code>right</code> and <code>internal</code>)	1117
19.6.7	Padding (<code>fill</code> , <code>setfill</code>)	1118
19.6.8	Integral Stream Base (<code>dec</code> , <code>oct</code> , <code>hex</code> , <code>showbase</code>)	1119
19.6.9	Floating-Point Numbers; Scientific and Fixed Notation (<code>scientific</code> , <code>fixed</code>)	1120
19.6.10	Uppercase/Lowercase Control (<code>uppercase</code>)	1121
19.6.11	Specifying Boolean Format (<code>boolalpha</code>)	1122
19.6.12	Setting and Resetting the Format State via Member Function flags	1123
19.7	Stream Error States	1124
19.8	Tying an Output Stream to an Input Stream	1127
19.9	C++20 Text Formatting	1127
19.9.1	C++20 <code>std::format</code> Presentation Types	1128
19.9.2	C++20 <code>std::format</code> Field Widths and Alignment	1130
19.9.3	C++20 <code>std::format</code> Numeric Formatting	1131
19.9.4	C++20 <code>std::format</code> Field Width and Precision Placeholders	1132
19.10	Wrap-Up	1133

20 Other Topics and a Look Toward the Future of C++

1141

20.1	Introduction	1142
20.2	<code>shared_ptr</code> and <code>weak_ptr</code> Smart Pointers	1143
20.2.1	Reference Counted <code>shared_ptr</code>	1143
20.2.2	<code>weak_ptr</code> : <code>shared_ptr</code> Observer	1147
20.3	Runtime Polymorphism with <code>std::variant</code> and <code>std::visit</code>	1154
20.4	protected Class Members: A Deeper Look	1160
20.5	Non-Virtual Interface (NVI) Idiom	1161
20.6	Inheriting Base-Class Constructors	1168
20.7	Multiple Inheritance	1169
20.7.1	Diamond Inheritance	1174
20.7.2	Eliminating Duplicate Subobjects with <code>virtual</code> Base-Class Inheritance	1176

20.8	<code>public</code> , <code>protected</code> and <code>private</code> Inheritance	1177
20.9	namespaces: A Deeper Look	1179
20.9.1	Defining namespaces	1180
20.9.2	Accessing namespace Members with Qualified Names	1181
20.9.3	<code>using</code> Directives Should Not Be Placed in Headers	1181
20.9.4	Nested Namespaces	1181
20.9.5	Aliases for namespace Names	1181
20.10	Storage Classes and Storage Duration	1182
20.10.1	Storage Duration	1182
20.10.2	Local Variables and Automatic Storage Duration	1182
20.10.3	Static Storage Duration	1183
20.10.4	<code>mutable</code> Class Members	1184
20.11	Operator Keywords	1185
20.12	<code>decltype</code> Operator	1186
20.13	Trailing Return Types for Functions	1187
20.14	<code>[[nodiscard]]</code> Attribute	1187
20.15	Some Key C++23 Features	1189
20.16	Wrap-Up	1194

21 Computer Science Thinking: Searching, Sorting and Big O 1197

21.1	Introduction	1198
21.2	Efficiency of Algorithms: Big O	1199
21.2.1	$O(1)$ Algorithms	1199
21.2.2	$O(n)$ Algorithms	1199
21.2.3	$O(n^2)$ Algorithms	1200
21.3	Linear Search	1201
21.3.1	Implementation	1201
21.3.2	Efficiency of Linear Search	1202
21.4	Binary Search	1203
21.4.1	Implementation	1203
21.4.2	Efficiency of Binary Search	1207
21.5	Insertion Sort	1208
21.5.1	Implementation	1209
21.5.2	Efficiency of Insertion Sort	1210
21.6	Selection Sort	1210
21.6.1	Implementation	1211
21.6.2	Efficiency of Selection Sort	1213
21.7	Merge Sort (A Recursive Implementation)	1213
21.7.1	Implementation	1214
21.7.2	Efficiency of Merge Sort	1219
21.7.3	Summarizing Various Algorithms' Big O Notations	1219
21.8	Wrap-Up	1221

Index 1225