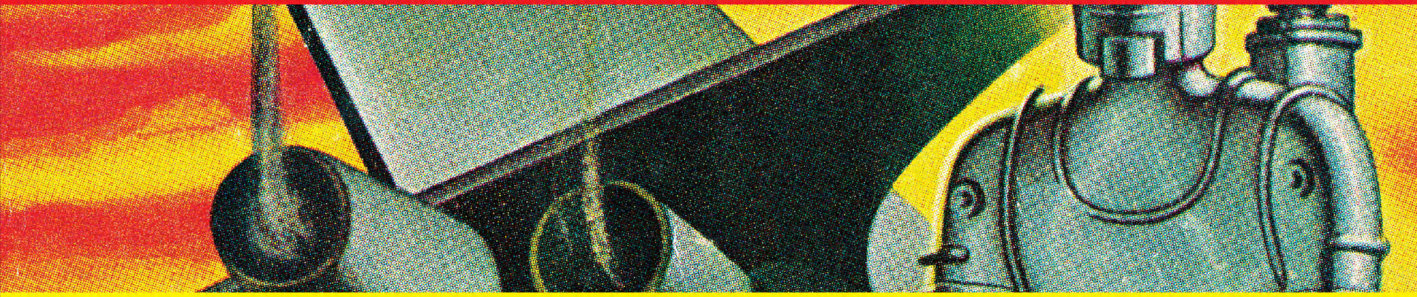
TECH TODAY

PROFESSIONAL C++

6TH EDITION



MARC GREGOIRE

WILEY

PROFESSIONAL C++

Sixth Edition

Marc Gregoire

WILEY

Copyright © 2024 by John Wiley & Sons, Inc. All rights reserved.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada and the United Kingdom.

ISBNs: 9781394193172 (Paperback), 9781394193196 (ePDF), 9781394193189 (ePub)

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4470, or on the web at `www.copyright.com`. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at `www.wiley.com/go/permission`.

Trademarks: WILEY and the Wiley logo are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Further, readers should be aware that websites listed in this work may have changed or disappeared between when this work was written and when it is read. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services or for technical support, please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic formats. For more information about Wiley products, visit our web site at `www.wiley.com`.

Library of Congress Control Number: 2023948608

Cover image: © CSA-Printstock/Getty Images
Cover design: Wiley

*Dedicated to my amazing parents and brother,
whose continuous support and patience help me in
tackling such a big project as writing this book.*

ABOUT THE AUTHOR

MARC GREGOIRE is a software architect from Belgium. He graduated from the University of Leuven, Belgium, with a degree in "Burgerlijk ingenieur in de computer wetenschappen" (equivalent to a master of science in engineering in computer science). The year after, he received an advanced master's degree in artificial intelligence, *cum laude*, at the same university. After his studies, Marc started working for a software consultancy company called Ordina Belgium. As a consultant, he worked for Siemens and Nokia Siemens Networks on critical 2G and 3G software running on Solaris for telecom operators. This required working in international teams stretching from South America and the United States to Europe, the Middle East, Africa, and Asia. Now, Marc is a software project manager and software architect at Nikon Metrology (`industry.nikon.com`), a division of Nikon and a leading provider of precision optical instruments, X-ray machines, and metrology solutions for X-ray, CT, and 3-D geometric inspection.

His main expertise is C++. He has experience with developing C++ programs running 24/7 on Windows and Linux platforms: for example, KNX/EIB home automation software. In addition to C++, Marc also likes C#.

Since April 2007, he has received the annual Microsoft MVP (Most Valuable Professional) award for his Visual C++ expertise.

Marc is the founder of the Belgian C++ Users Group (`becpp.org`), co-author of *C++ Standard Library Quick Reference* 1st and 2nd editions (Apress 2016 and 2019), a technical editor for numerous books for several publishers, and a regular speaker at the CppCon C++ conference (`cppcon.org`). He maintains a blog at `www.nuonsoft.com/blog` and is passionate about traveling and gastronomic restaurants.

ABOUT THE TECHNICAL EDITORS

BRADLEY JONES has programmed in a variety of languages and tools ranging from C to Unity on platforms ranging from Windows to mobile and including the web as well as a little bit of virtual reality and embedded devices just for fun. In addition to programming, he has authored books on C, C++, C#, Windows, the web, and many more technical topics and a few nontechnical topics. Bradley is the owner of Lots of Software, LLC, and has been recognized in the industry as a community influencer as well as has been recognized as a Microsoft MVP, a CODiE Judge, an international technology speaker, a bestselling technical author, and more.

ARTHUR O'DWYER is a professional C++ trainer, software engineer, author, and WG21 committee member. He authored *Mastering the C++17 STL* (Packt Publishing, 2017), founded CppCon's "Back to Basics" track (2019), implemented libc++'s <memory_resource> header (2022), and is responsible for the simplified "implicit move" semantics in C++20 and C++23. He and his wife live in New York.

ACKNOWLEDGMENTS

I THANK THE JOHN WILEY & SONS editorial and production teams for their support. A special thank-you to Jim Minatel, executive editor at Wiley, for giving me a chance to write this sixth edition; Pete Gaughan, senior managing editor; Ashirvad Moses Thyagarajan, managing editor; Kathryn Hogan, PhD, project manager; Archana Pragash, content refinement specialist; and Kim Wimpsett, copyeditor.

A special thank you to technical editors Bradley Jones and Arthur O'Dwyer for checking the technical accuracy of the book. Their feedback and numerous contributions have strengthened this book and are greatly appreciated.

Of course, the support and patience of my parents and my brother were very important in finishing this book. I would also like to express my sincere gratitude to my employer, Nikon Metrology, for supporting me during this project.

Finally, I thank you, the reader, for supporting me over all these years and across numerous editions with this approach to professional C++ software development.

—Marc Gregoire

CONTENTS

INTRODUCTION *xli*

PART I: INTRODUCTION TO PROFESSIONAL C++

CHAPTER 1: A CRASH COURSE IN C++ AND THE STANDARD LIBRARY 3

C++ Crash Course 4
The Obligatory "Hello, World" Program 4
Comments 5
Importing Modules 5
How the Compiler Processes Your Source Code 6
Preprocessor Directives 6
The main() Function 7
Printing Text 7
I/O Streams 7
Returning from a Function 9
Namespaces 9
Nested Namespace 11
Namespace Alias 11
Literals 11
Variables 12
Numerical Limits 15
Zero Initialization 15
Casting 16
Floating-Point Numbers 16
Operators 18
Enumerations 21
Old-Style Enumerations 22
Structs 23
Conditional Statements 24
if/else Statements 24
switch Statements 25
The Conditional Operator 27
Logical Evaluation Operators 27
Three-Way Comparisons 29
Functions 30

Function Return Type Deduction 31
Current Function's Name 32
Function Overloading 32
Attributes 32
[[nodiscard]] 33
[[maybe_unused]] 33
[[noreturn]] 34
[[deprecated]] 34
[[likely]] and [[unlikely]] 35
[[assume]] 35
C-Style Arrays 36
std::array 37
std::vector 38
std::pair 39
std::optional 40
Structured Bindings 41
Loops 41
The while Loop 41
The do/while Loop 42
The for Loop 42
The Range-Based for Loop 42
Initializer Lists 43
Strings in C++ 43
C++ as an Object-Oriented Language 44
Defining Classes 44
Using Classes 47
Scope Resolution 47
Uniform Initialization 48
Designated Initializers 51
Pointers and Dynamic Memory 52
The Stack and the Free Store 52
Working with Pointers 53
Dynamically Allocated Arrays 54
Null Pointer Constant 55
The Use of const 56
const as a Qualifier for a Type 56
const Member Functions 58
References 59
Reference Variables 59
Reference Data Members 62
Reference Parameters 62

Reference Return Values 65
Deciding Between References and Pointers 65
const_cast() 69
Exceptions 70
Type Aliases 71
typedefs 72
Type Inference 72
The auto Keyword 72
The decltype Keyword 75
The Standard Library 75
Your First Bigger C++ Program 76
An Employee Records System 76
The Employee Class 76
Employee.cppm 76
Employee.cpp 78
EmployeeTest.cpp 79
The Database Class 80
Database.cppm 80
Database.cpp 81
DatabaseTest.cpp 82
The User Interface 82
Evaluating the Program 85
Summary 85
Exercises 85

CHAPTER 2: WORKING WITH STRINGS AND STRING VIEWS 87

Dynamic Strings 88
C-Style Strings 88
String Literals 90
Raw String Literals 90
The C++ std::string Class 92
What Is Wrong with C-Style Strings? 92
Using the std::string Class 92
std::string Literals 95
CTAD with std::vector and Strings 96
Numeric Conversions 96
High-Level Numeric Conversions 96
Low-Level Numeric Conversions 98
The std::string_view Class 100
std::string_view and Temporary Strings 102
std::string_view Literals 102

Nonstandard Strings 103
Formatting and Printing Strings 103
Format Strings 104
Argument Indices 105
Printing to Different Destinations 106
Compile-Time Verification of Format Strings 106
Non-Compile-Time Constant Format Strings 106
Handling Errors in Non-Compile-Time Constant Format Strings 107
Format Specifiers 107
width 108
[fill]align 108
sign 109
# 109
type 109
precision 110
0 111
L 111
Formatting Escaped Characters and Strings 111
Formatting Ranges 112
Support for Custom Types 114
Summary 117
Exercises 117

CHAPTER 3: CODING WITH STYLE 119

The Importance of Looking Good 119
Thinking Ahead 120
Elements of Good Style 120
Documenting Your Code 120
Reasons to Write Comments 120
Commenting to Explain Usage 120
Commenting to Explain Complicated Code 122
Commenting to Convey Meta-information 124
Copyright Comment 125
Commenting Styles 125
Commenting Every Line 125
Prefix Comments 126
Fixed-Format Comments 127
Ad Hoc Comments 129
Self-Documenting Code 129
Decomposition 129
Decomposition through Refactoring 130

Decomposition by Design 131
Decomposition in This Book 131
Naming 132
Choosing a Good Name 132
Naming Conventions 133
Counters 133
Prefixes 133
Hungarian Notation 134
Getters and Setters 134
Capitalization 134
Namespaced Constants 134
Using Language Features with Style 135
Use Constants 135
Use References Instead of Pointers 136
Use Custom Exceptions 136
Formatting 137
The Curly Brace Alignment Debate 137
Coming to Blows over Spaces and Parentheses 138
Spaces, Tabs, and Line Breaks 139
Stylistic Challenges 139
Summary 140
Exercises 140

PART II: PROFESSIONAL C++ SOFTWARE DESIGN

CHAPTER 4: DESIGNING PROFESSIONAL C++ PROGRAMS 145

What is Programming Design? 146
The Importance of Programming Design 147
Designing For C++ 149
Two Rules for Your Own C++ Designs 150
Abstraction 150
Benefiting from Abstraction 150
Incorporating Abstraction in Your Design 151
Reuse 152
Writing Reusable Code 153
Reusing Designs 153
Reusing Existing Code 154
A Note on Terminology 155
Deciding Whether to Reuse Code or Write It Yourself 156
Advantages to Reusing Code 156

Disadvantages to Reusing Code 157
Putting It Together to Make a Decision 158
Guidelines for Choosing a Library to Reuse 158
Understand the Capabilities and Limitations 158
Understand the Learning Cost 159
Understand the Performance 159
Understand Platform Limitations 162
Understand Licensing 162
Understand Support and Know Where to Find Help 162
Prototype 163
Open-Source Libraries 163
The C++ Standard Library 165
Designing a Chess Program 166
Requirements 166
Design Steps 167
Divide the Program into Subsystems 167
Choose Threading Models 169
Specify Class Hierarchies for Each Subsystem 170
Specify Classes, Data Structures, Algorithms, and Patterns for Each Subsystem 170
Specify Error Handling for Each Subsystem 173
Summary 174
Exercises 175

CHAPTER 5: DESIGNING WITH CLASSES 177

Am I Thinking Procedurally? 178
The Object-Oriented Philosophy 178
Classes 178
Components 179
Properties 179
Behaviors 180
Bringing It All Together 180
Living In a World of Classes 181
Over-Classification 182
Overly General Classes 182
Class Relationships 183
The Has-a Relationship 183
The Is-a Relationship (Inheritance) 184
Inheritance Techniques 185
Polymorphism 186
The Fine Line Between Has-a and Is-a 186

The Not-a Relationship 190
Hierarchies 191
Multiple Inheritance 192
Mixin Classes 193
Summary 194
Exercises 194

CHAPTER 6: DESIGNING FOR REUSE 197

The Reuse Philosophy 198
How to Design Reusable Code 198
Use Abstraction 199
Structure Your Code for Optimal Reuse 200
Avoid Combining Unrelated or Logically Separate Concepts 201
Use Templates for Generic Data Structures and Algorithms 203
Provide Appropriate Checks and Safeguards 205
Design for Extensibility 206
Design Usable Interfaces 208
Consider the Audience 208
Consider the Purpose 209
Design Interfaces That Are Easy to Use 210
Design General-Purpose Interfaces 214
Reconciling Generality and Ease of Use 215
Designing a Successful Abstraction 216
The SOLID Principles 216
Summary 217
Exercises 217

PART III: C++ CODING THE PROFESSIONAL WAY

CHAPTER 7: MEMORY MANAGEMENT 221

Working with Dynamic Memory 222
How to Picture Memory 222
Allocation and Deallocation 223
Using new and delete 223
What About My Good Friend malloc? 224
When Memory Allocation Fails 225
Arrays 225
Arrays of Primitive Types 226
Arrays of Objects 228

Deleting Arrays 228
Multidimensional Arrays 229
Working with Pointers 233
A Mental Model for Pointers 233
Casting with Pointers 234
Array-Pointer Duality 234
Arrays Decay to Pointers 234
Not All Pointers Are Arrays! 236
Low-Level Memory Operations 236
Pointer Arithmetic 236
Custom Memory Management 237
Garbage Collection 238
Object Pools 238
Common Memory Pitfalls 239
Underallocating Data Buffers and Out-of-Bounds Memory Access 239
Memory Leaks 240
Finding and Fixing Memory Leaks in Windows with Visual C++ 241
Finding and Fixing Memory Leaks in Linux with Valgrind 243
Double-Deletion and Invalid Pointers 243
Smart Pointers 244
unique_ptr 245
Creating unique_ptrs 245
Using unique_ptrs 247
unique_ptr and C-Style Arrays 248
Custom Deleters 248
shared_ptr 249
Creating and Using shared_ptrs 249
The Need for Reference Counting 250
Casting a shared_ptr 251
Aliasing 252
weak_ptr 252
Passing to Functions 253
Returning from Functions 253
enable_shared_from_this 254
Interoperability of Smart Pointers with C-Style Functions 255
The Old and Removed auto_ptr 255
Summary 256
Exercises 256

CHAPTER 8: GAINING PROFICIENCY WITH CLASSES AND OBJECTS 259

Introducing the Spreadsheet Example 260
Writing Classes 260
Class Definitions 260
Class Members 261
Access Control 261
Order of Declarations 262
In-Class Member Initializers 263
Defining Member Functions 263
Accessing Data Members 264
Calling Other Member Functions 264
Using Objects 265
Objects on the Stack 266
Objects on the Free Store 266
The this Pointer 267
Explicit Object Parameter 268
Understanding Object Life Cycles 269
Object Creation 269
Writing Constructors 270
Using Constructors 270
Providing Multiple Constructors 271
Default Constructors 272
Constructor Initializers aka Ctor-Initializers 276
Copy Constructors 279
Initializer-List Constructors 281
Delegating Constructors 283
Converting Constructors and Explicit Constructors 284
Summary of Compiler-Generated Constructors 285
Object Destruction 286
Assigning to Objects 288
Declaring an Assignment Operator 288
Defining an Assignment Operator 289
Explicitly Defaulted and Deleted Assignment Operator 290
Compiler-Generated Copy Constructor and Copy Assignment Operator 291
Distinguishing Copying from Assignment 291
Objects as Return Values 291
Copy Constructors and Object Members 292
Summary 293
Exercises 293

CHAPTER 9: MASTERING CLASSES AND OBJECTS 295

Friends 296
Dynamic Memory Allocation in Objects 297
The Spreadsheet Class 297
Freeing Memory with Destructors 300
Handling Copying and Assignment 301
The Spreadsheet Copy Constructor 303
The Spreadsheet Assignment Operator 303
Disallowing Assignment and Pass-by-Value 306
Handling Moving with Move Semantics 307
Rvalue References 307
Decay Copy 310
Implementing Move Semantics 310
Testing the Spreadsheet Move Operations 314
Implementing a Swap Function with Move Semantics 316
Using std::move() in Return Statements 317
Optimal Way to Pass Arguments to Functions 318
Rule of Zero 319
More About Member Functions 320
static Member Functions 320
const Member Functions 321
mutable Data Members 322
Member Function Overloading 323
Overloading Based on const 323
Explicitly Deleting Overloads 325
Ref-Qualified Member Functions 325
Inline Member Functions 327
Default Arguments 329
Constexpr and Consteval 330
The constexpr Keyword 330
The consteval Keyword 331
constexpr and consteval Classes 332
Different Kinds of Data Members 333
static Data Members 333
Inline Variables 334
Accessing static Data Members from within Class Member Functions 334
constexpr static Data Members 335
Accessing static Data Members from Outside Class Member Functions 336
Reference Data Members 336

Nested Classes 338
Enumerations Inside Classes 339
Operator Overloading 339
Example: Implementing Addition for SpreadsheetCells 340
First Attempt: The add Member Function 340
Second Attempt: Overloaded operator+ as a Member Function 341
Third Attempt: Global operator+ 342
Overloading Arithmetic Operators 343
Overloading the Arithmetic Shorthand Operators 344
Overloading Comparison Operators 345
Overloading Comparison Operators Before C++20 345
Overloading Comparison Operators Since C++20 347
Compiler-Generated Comparison Operators 348
Building Stable Interfaces 350
Using Interface and Implementation Classes 350
Summary 354
Exercises 354

CHAPTER 10: DISCOVERING INHERITANCE TECHNIQUES 357

Building Classes with Inheritance 358
Extending Classes 358
A Client's View of Inheritance 359
A Derived Class's View of Inheritance 360
Preventing Inheritance 362
Overriding Member Functions 362
The virtual Keyword 362
Syntax for Overriding a Member Function 363
A Client's View of Overridden Member Functions 363
The override Keyword 365
The Truth about virtual 366
Preventing Overriding 370
Inheritance For Reuse 370
The WeatherPrediction Class 370
Adding Functionality in a Derived Class 371
Replacing Functionality in a Derived Class 373
Respect Your Parents 373
Parent Constructors 373
Parent Destructors 375
virtual Member Function Calls within Constructors and Destructor 376
Referring to Parent Names 377
Casting Up and Down 379

Inheritance for Polymorphism 380
- Return of the Spreadsheet 380
- Designing the Polymorphic Spreadsheet Cell 381
- The SpreadsheetCell Base Class 382
  - A First Attempt 382
  - Pure virtual Member Functions and Abstract Base Classes 382
- The Individual Derived Classes 383
  - StringSpreadsheetCell Class Definition 383
  - StringSpreadsheetCell Implementation 384
  - DoubleSpreadsheetCell Class Definition and Implementation 384
- Leveraging Polymorphism 385
- Future Considerations 386
- Providing Implementations for Pure virtual Member Functions 388

Multiple Inheritance 388
- Inheriting from Multiple Classes 389
- Naming Collisions and Ambiguous Base Classes 390
  - Name Ambiguity 390
  - Ambiguous Base Classes 391
  - Uses for Multiple Inheritance 392

Interesting and Obscure Inheritance Issues 392
- Changing the Overridden Member Function's Return Type 393
- Adding Overloads of virtual Base Class Member Functions to Derived Classes 396
- Inherited Constructors 396
  - Hiding of Inherited Constructors 397
  - Inherited Constructors and Multiple Inheritance 398
  - Initialization of Data Members 399
- Special Cases in Overriding Member Functions 400
  - The Base Class Member Function Is static 400
  - The Base Class Member Function Is Overloaded 401
  - The Base Class Member Function Is private 403
  - The Base Class Member Function Has Default Arguments 404
  - The Base Class Member Function Has a Different Access Specification 405
- Copy Constructors and Assignment Operators in Derived Classes 407
- Run-Time Type Facilities 408
- Non-public Inheritance 410
- Virtual Base Classes 411

Casts 414
- static_cast() 414

reinterpret_cast() 415
dynamic_cast() 416
std::bit_cast() 417
Summary of Casts 418
Summary 418
Exercises 419

CHAPTER 11: MODULES, HEADER FILES, AND MISCELLANEOUS TOPICS 421

Modules 422
Unmodularizing Code 423
Standard Named Modules 423
Module Interface Files 423
Module Implementation Files 425
Splitting Interface from Implementation 426
Visibility vs. Reachability 427
Submodules 428
Module Partitions 429
Implementation Partitions 431
Private Module Fragment 432
Header Units 433
Importable Standard Library Headers 434
Preprocessor Directives 436
Preprocessor Macros 437
Linkage 438
Internal Linkage 439
The extern Keyword 440
Header Files 441
One Definition Rule (ODR) 441
Duplicate Definitions 442
Circular Dependencies 442
Querying Existence of Headers 443
Module Import Declarations 443
Feature-Test Macros for Core Language Features 444
The Static Keyword 445
static Data Members and Member Functions 445
static Variables in Functions 445
Order of Initialization of Nonlocal Variables 446
Order of Destruction of Nonlocal Variables 446

C-Style Variable-Length Argument Lists 447
Accessing the Arguments 448
Why You Shouldn't Use C-Style Variable-Length Argument Lists 448
Summary 449
Exercises 449

CHAPTER 12: WRITING GENERIC CODE WITH TEMPLATES 451

Overview of Templates 452
Class Templates 453
Writing a Class Template 453
Coding Without Templates 453
A Template Grid Class 456
Using the Grid Template 460
How the Compiler Processes Templates 461
Selective/Implicit Instantiation 462
Explicit Instantiation 462
Template Requirements on Types 462
Distributing Template Code Between Files 463
Member Function Definitions in Same File as Class Template Definition 463
Member Function Definitions in Separate File 463
Template Parameters 464
Non-type Template Parameters 464
Default Values for Template Parameters 466
Class Template Argument Deduction 467
Member Function Templates 468
Member Function Templates with Non-type Template Parameters 471
Using Member Function Templates with Explicit Object Parameters to Avoid Code Duplication 473
Class Template Specialization 474
Deriving from Class Templates 477
Inheritance vs. Specialization 478
Alias Templates 479
Function Templates 479
Function Overloads vs. Function Template 481
Function Template Overloading 481
Function Templates as Friends of Class Templates 482
More on Template Type Parameter Deduction 484
Return Type of Function Templates 484
Abbreviated Function Template Syntax 486

Variable Templates 487
Concepts 487
Syntax 488
Constraints Expression 488
Requires Expressions 489
Combining Concept Expressions 491
Predefined Standard Concepts 491
Type-Constrained auto 492
Type Constraints and Function Templates 493
Constraint Subsumption 495
Type Constraints and Class Templates 495
Type Constraints and Class Member Functions 496
Constraint-Based Class Template Specialization and Function Template Overloading 496
Best Practices 497
Summary 498
Exercises 498

CHAPTER 13: DEMYSTIFYING C++ I/O 501

Using Streams 502
What Is a Stream, Anyway? 502
Stream Sources and Destinations 504
Output with Streams 504
Output Basics 504
Member Functions of Output Streams 505
Handling Output Errors 506
Output Manipulators 508
Input with Streams 510
Input Basics 510
Handling Input Errors 511
Input Member Functions 512
Input Manipulators 516
Input and Output with Objects 517
Custom Manipulators 519
String Streams 519
Span-Based Streams 521
File Streams 522
Text Mode vs. Binary Mode 523
Jumping Around with seek() and tell() 523
Linking Streams Together 526
Read an Entire File 526

Bidirectional I/O 527
Filesystem Support Library 528
Path 528
Directory Entry 530
Helper Functions 530
Directory Iteration 530
Summary 531
Exercises 532

CHAPTER 14: HANDLING ERRORS 533

Errors and Exceptions 534
What Are Exceptions, Anyway? 534
Why Exceptions in C++ Are a Good Thing 535
Recommendation 536
Exception Mechanics 536
Throwing and Catching Exceptions 537
Exception Types 540
Catching Exception Objects as Reference-to-const 541
Throwing and Catching Multiple Exceptions 541
Matching and const 543
Matching Any Exception 543
Uncaught Exceptions 544
noexcept Specifier 546
noexcept(expression) Specifier 546
noexcept(expression) Operator 546
Throw Lists 547
Exceptions and Polymorphism 547
The Standard Exception Hierarchy 547
Catching Exceptions in a Class Hierarchy 549
Writing Your Own Exception Classes 550
Nested Exceptions 553
Rethrowing Exceptions 555
Stack Unwinding and Cleanup 556
Use Smart Pointers 558
Catch, Cleanup, and Rethrow 558
Source Location 559
Source Location for Logging 560
Automatically Embed a Source Location in Custom Exceptions 560
Stack Trace 561

The Stack Trace Library 561
Automatically Embed a Stack Trace in Custom Exceptions 563
Common Error-Handling Issues 564
Memory Allocation Errors 565
Non-throwing new 565
Customizing Memory Allocation Failure Behavior 566
Errors in Constructors 567
Function-Try-Blocks for Constructors 569
Errors in Destructors 572
Exception Safety Guarantees 573
Summary 573
Exercises 573

CHAPTER 15: OVERLOADING C++ OPERATORS 577

Overview of Operator Overloading 578
Why Overload Operators? 578
Limitations to Operator Overloading 578
Choices in Operator Overloading 579
Member Function or Global Function 579
Choosing Argument Types 580
Choosing Return Types 581
Choosing Behavior 581
Operators You Shouldn't Overload 581
Summary of Overloadable Operators 582
Rvalue References 586
Precedence and Associativity 587
Relational Operators 588
Alternative Notation 589
Overloading The Arithmetic Operators 589
Overloading Unary Minus and Unary Plus 589
Overloading Increment and Decrement 590
Overloading the Bitwise and Binary Logical Operators 591
Overloading the Insertion and Extraction Operators 591
Overloading the Subscripting Operator 593
Providing Read-Only Access with operator[] 596
Multidimensional Subscripting Operator 598
Non-integral Array Indices 599
static Subscripting Operator 599
Overloading the Function Call Operator 600
static Function Call Operator 601

Overloading the Dereferencing Operators 602
Implementing operator* 603
Implementing operator–> 604
What in the World Are operator.* and operator–>*? 604
Writing Conversion Operators 605
Operator auto 606
Solving Ambiguity Problems with Explicit Conversion Operators 606
Conversions for Boolean Expressions 607
Overloading the Memory Allocation and Deallocation Operators 609
How new and delete Really Work 609
The New-Expression and operator new 609
The Delete-Expression and operator delete 610
Overloading operator new and operator delete 610
Explicitly Deleting or Defaulting operator new and operator delete 613
Overloading operator new and operator delete with Extra Parameters 613
Overloading operator delete with Size of Memory as Parameter 614
Overloading User-Defined Literal Operators 615
Standard Library Literals 615
User-Defined Literals 616
Cooked-Mode Literal Operator 616
Raw-Mode Literal Operator 617
Summary 618
Exercises 618

CHAPTER 16: OVERVIEW OF THE C++ STANDARD LIBRARY 619

Coding Principles 620
Use of Templates 621
Use of Operator Overloading 621
Overview of the C++ Standard Library 621
Strings 621
Regular Expressions 622
I/O Streams 622
Smart Pointers 622
Exceptions 623
Standard Integer Types 623
Numerics Library 623
Integer Comparisons 624
Bit Manipulation 624
Time and Date Utilities 625
Random Numbers 625

- Initializer Lists 626
- Pair and Tuple 626
- Vocabulary Types 626
- Function Objects 627
- Filesystem 627
- Multithreading 627
- Type Traits 627
- Standard Library Feature-Test Macros 627
- <version> 629
- Source Location 629
- Stack Trace 629
- Containers 629
  - Sequential Containers 630
  - Sequential Views 632
  - Container Adapters 632
  - Ordered Associative Containers 634
  - Unordered Associative Containers/Hash Tables 635
  - Flat Associative Container Adapters 635
  - bitset 636
  - Summary of Standard Library Containers 636
- Algorithms 639
  - Non-modifying Sequence Algorithms 640
  - Modifying Sequence Algorithms 642
  - Operational Algorithms 643
  - Swap Algorithms 644
  - Partitioning Algorithms 644
  - Sorting Algorithms 645
  - Binary Search Algorithms 645
  - Set Algorithms on Sorted Sequences 645
  - Other Algorithms on Sorted Sequences 646
  - Heap Algorithms 646
  - Minimum/Maximum Algorithms 646
  - Numerical Processing Algorithms 647
  - Permutation Algorithms 648
  - Choosing an Algorithm 648
- Ranges Library 649
- What's Missing from the Standard Library 650

Summary 650

Exercises 650

CHAPTER 17: UNDERSTANDING ITERATORS AND THE RANGES LIBRARY 653

Iterators 654
Getting Iterators for Containers 656
Iterator Traits 658
Examples 659
Function Dispatching Using Iterator Traits 660
Stream Iterators 661
Output Stream Iterator: ostream_iterator 662
Input Stream Iterator: istream_iterator 663
Input Stream Iterator: istreambuf_iterator 663
Iterator Adapters 663
Insert Iterators 664
Reverse Iterators 665
Move Iterators 666
Ranges 668
Constrained Algorithms 669
Projection 670
Views 671
Modifying Elements Through a View 677
Mapping Elements 677
Range Factories 678
Input Streams as Views 679
Converting a Range into a Container 680
Summary 681
Exercises 681

CHAPTER 18: STANDARD LIBRARY CONTAINERS 683

Containers Overview 684
Requirements on Elements 685
Exceptions and Error Checking 687
Sequential Containers 687
vector 687
vector Overview 687
vector Details 690
Move Semantics 703
vector Example: A Round-Robin Class 704
The vector<bool> Specialization 709
deque 709
list 710

Accessing Elements 710
Iterators 711
Adding and Removing Elements 711
list Size 711
Special list Operations 711
list Example: Determining Enrollment 713
forward_list 714
array 717
Sequential Views 718
span 718
mdspan 720
Container Adapters 722
queue 722
queue Operations 722
queue Example: A Network Packet Buffer 723
priority_queue 725
priority_queue Operations 725
priority_queue Example: An Error Correlator 726
stack 727
stack Operations 728
stack Example: Revised Error Correlator 728
Associative Containers 728
Ordered Associative Containers 728
The pair Utility Class 729
map 729
multimap 738
set 742
multiset 744
Unordered Associative Containers Or Hash Tables 744
Hash Functions 744
unordered_map 746
unordered_multimap 750
unordered_set/unordered_multiset 751
Flat Set and Flat Map Associative Container Adapters 751
Performance of Associative Containers 752
Other Containers 752
Standard C-Style Arrays 752
Strings 753
Streams 754
bitset 754
bitset Basics 755

Bitwise Operators 755
bitset Example: Representing Cable Channels 756
Summary 759
Exercises 759

CHAPTER 19: FUNCTION POINTERS, FUNCTION OBJECTS, AND LAMBDA EXPRESSIONS 761

Function Pointers 762
findMatches() Using Function Pointers 762
findMatches() As a Function Template 764
Windows DLLs and Function Pointers 765
Pointers to Member Functions (And Data Members) 765
Function Objects 767
Writing Your First Function Object 767
Function Objects in the Standard Library 767
Arithmetic Function Objects 768
Comparison Function Objects 769
Logical Function Objects 771
Bitwise Function Objects 771
Adapter Function Objects 771
Polymorphic Function Wrappers 775
std::function 775
std::move_only_function 776
Lambda Expressions 777
Syntax 777
Lambda Expressions as Parameters 783
Generic Lambda Expressions 783
Lambda Capture Expressions 784
Templated Lambda Expressions 785
Lambda Expressions as Return Type 785
Lambda Expressions in Unevaluated Contexts 786
Default Construction, Copying, and Assigning 786
Recursive Lambda Expressions 787
Invokers 787
Summary 788
Exercises 788

CHAPTER 20: MASTERING STANDARD LIBRARY ALGORITHMS 791

Overview of Algorithms 792
The find and find_if Algorithms 793
The accumulate Algorithm 795

Move Semantics with Algorithms 796
Algorithm Callbacks 796
Algorithm Details 797
Non-modifying Sequence Algorithms 798
Search Algorithms 798
Specialized Searchers 799
Comparison Algorithms 800
Counting Algorithms 802
Modifying Sequence Algorithms 803
generate 804
transform 804
copy 805
move 806
replace 808
erase 808
remove 809
unique 810
shuffle 811
sample 811
reverse 812
Shifting Elements 812
Operational Algorithms 813
for_each 813
for_each_n 814
Partition Algorithms 814
Sorting Algorithms 815
Binary Search Algorithms 817
Set Algorithms 818
Minimum/Maximum Algorithms 820
Parallel Algorithms 822
Numerical Processing Algorithms 823
iota 824
Reduce Algorithms 824
Scan Algorithms 825
Constrained Algorithms 826
Constrained find 826
Constrained generate 826
Constrained for_each 827
Constrained-Only Algorithms 827
Summary 828
Exercises 828

CHAPTER 21: STRING LOCALIZATION AND REGULAR EXPRESSIONS 831

Localization 832
Wide Characters 832
Non-Western Character Sets 833
Localizing String Literals 835
Locales and Facets 836
Locales 836
Global Locale 837
Using Locales 837
Character Classification 839
Character Conversion 839
Using Facets 839
Conversions 840
Regular Expressions 841
ECMAScript Syntax 842
Anchor 842
Wildcard 843
Alternation 843
Grouping 843
Quantifier 843
Precedence 844
Character Set Matches 844
Word Boundary 846
Back Reference 847
Lookahead 847
Regular Expressions and Raw String Literals 848
Common Regular Expressions 848
The regex Library 848
regex_match() 849
regex_match() Examples 850
regex_search() 852
regex_search() Examples 852
regex_iterator 853
regex_iterator Examples 853
regex_token_iterator 854
regex_token_iterator Examples 855
regex_replace() 857
regex_replace() Examples 857
Summary 859
Exercises 859

CHAPTER 22: DATE AND TIME UTILITIES 861

Compile-Time Rational Numbers 862
Duration 864
Examples and Converting Durations 865
Predefined Durations 867
Standard Literals 868
hh_mm_ss 868
Clock 868
Printing Current Time 870
Execution Timing 870
Time Point 871
Date 873
Creating Dates 873
Printing Dates 875
Arithmetic with Dates 876
Time Zone 877
Summary 878
Exercises 878

CHAPTER 23: RANDOM NUMBER FACILITIES 879

C-Style Random Number Generation 880
Random Number Engines 881
Random Number Engine Adapters 882
Predefined Engines and Engine Adapters 883
Generating Random Numbers 884
Random Number Distributions 885
Summary 889
Exercises 889

CHAPTER 24: ADDITIONAL VOCABULARY TYPES 891

Variant 892
Any 894
Tuple 895
Decompose Tuples 897
Structured Bindings 897
tie 898
Concatenation 898
Comparisons 898
make_from_tuple 899
apply 900

Optional: Monadic Operations 900
Expected 901
Exceptions, Error Return Codes, and expected 904
Summary 904
Exercises 905

PART IV: MASTERING ADVANCED FEATURES OF C++

CHAPTER 25: CUSTOMIZING AND EXTENDING THE STANDARD LIBRARY 909

Allocators 910
Extending the Standard Library 911
Why Extend the Standard Library? 912
Writing a Standard Library Algorithm 912
find_all 912
Modernized find_all 914
Writing a Standard Library Container 915
A Basic Directed Graph 915
Making directed_graph a Standard Library Container 925
Additional Standard Library–Like Functionality 939
Further Improvements 942
Other Container Types 942
Summary 942
Exercises 943

CHAPTER 26: ADVANCED TEMPLATES 945

More About Template Parameters 946
More About Template Type Parameters 946
Introducing Template Template Parameters 949
More About Non-type Template Parameters 951
Class Template Partial Specialization 952
Emulating Function Partial Specialization with Overloading 955
Template Recursion 957
An N-Dimensional Grid: First Attempt 957
A Real *N*-Dimensional Grid 958
Variadic Templates 960
Type-Safe Variable-Length Argument Lists 961
constexpr if 963
Variable Number of Mixin Classes 963
Fold Expressions 964

Metaprogramming 966
Factorial at Compile Time 967
Loop Unrolling 968
Printing Tuples 968
constexpr if 970
Using a Compile-Time Integer Sequence with Folding 971
Type Traits 972
Using Type Categories 973
Using Type Relationships 975
Using the conditional Type Trait 976
Using Type Modification Type Traits 978
Using enable_if 978
Using constexpr if to Simplify enable_if Constructs 981
Logical Operator Traits 982
Static Assertions 982
Metaprogramming Conclusion 983
Summary 983
Exercises 984

CHAPTER 27: MULTITHREADED PROGRAMMING WITH C++ 985

Introduction 986
Race Conditions 988
Tearing 989
Deadlocks 989
False Sharing 991
Threads 991
Thread with Function Pointer 991
Thread with Function Object 993
Thread with Lambda 994
Thread with Member Function Pointer 994
Thread-Local Storage 995
Canceling Threads 996
Automatically Joining Threads 996
Cooperative Cancellation 996
Retrieving Results from Threads 998
Copying and Rethrowing Exceptions 998
Atomic Operations Library 1001
Atomic Operations 1002
Atomic Smart Pointers 1004
Atomic References 1004
Using Atomic Types 1004

Waiting on Atomic Variables 1006
Mutual Exclusion 1007
Mutex Classes 1008
Spinlock 1008
Non-timed Mutex Classes 1009
Timed Mutex Classes 1010
Locks 1011
lock_guard 1011
unique_lock 1011
shared_lock 1012
Acquiring Multiple Locks at Once 1012
scoped_lock 1013
std::call_once 1014
Examples Using Mutexes 1015
Thread-Safe Writing to Streams 1015
Double-Checked Locking 1017
Condition Variables 1019
Spurious Wake-Ups 1020
Using Condition Variables 1020
Latches 1021
Barriers 1023
Semaphores 1024
Futures 1025
std::promise and std::future 1026
std::packaged_task 1027
std::async 1028
Exception Handling 1029
std::shared_future 1029
Example: Multithreaded Logger Class 1030
Thread Pools 1035
Coroutines 1036
Threading Design and Best Practices 1038
Summary 1039
Exercises 1040

PART V: C++ SOFTWARE ENGINEERING

CHAPTER 28: MAXIMIZING SOFTWARE ENGINEERING METHODS 1043

The Need for Process 1044
Software Life Cycle Models 1045

The Waterfall Model 1045
Benefits of the Waterfall Model 1046
Drawbacks of the Waterfall Model 1046
Sashimi Model 1047
Spiral-like Models 1047
Benefits of a Spiral-like Model 1048
Drawbacks of a Spiral-like Model 1049
Agile 1050
Software Engineering Methodologies 1050
Scrum 1050
Roles 1051
The Process 1051
Benefits of Scrum 1053
Drawbacks of Scrum 1053
The Unified Process 1053
The Rational Unified Process 1054
RUP as a Product 1055
RUP as a Process 1055
RUP in Practice 1055
Extreme Programming 1056
XP in Theory 1056
XP in Practice 1060
Software Triage 1060
Building Your Own Process and Methodology 1061
Be Open to New Ideas 1061
Bring New Ideas to the Table 1061
Recognize What Works and What Doesn't Work 1061
Don't Be a Renegade 1061
Version Control 1062
Summary 1064
Exercises 1064

CHAPTER 29: WRITING EFFICIENT C++ 1065

Overview of Performance and Efficiency 1066
Two Approaches to Efficiency 1066
Two Kinds of Programs 1066
Is C++ an Inefficient Language? 1066
Language-Level Efficiency 1067
Handle Objects Efficiently 1068
Pass-by-Value or Pass-by-Reference 1068
Return-by-Value or Return-by-Reference 1070

Catch Exceptions by Reference 1070
Use Move Semantics 1070
Avoid Creating Temporary Objects 1070
Pre-allocate Memory 1071
Use Inline Functions 1071
Mark Unreachable Code 1072
Design-Level Efficiency 1073
Cache Where Necessary 1073
Use Object Pools 1074
An Object Pool Implementation 1074
Using the Object Pool 1078
Profiling 1079
Profiling Example with gprof 1080
First Design Attempt 1080
Profiling the First Design Attempt 1083
Second Design Attempt 1085
Profiling the Second Design Attempt 1087
Profiling Example with Visual C++ 2022 1088
Summary 1090
Exercises 1090

CHAPTER 30: BECOMING ADEPT AT TESTING 1093

Quality Control 1094
Whose Responsibility Is Testing? 1094
The Life Cycle of a Bug 1094
Bug-Tracking Tools 1095
Unit Testing 1097
Approaches to Unit Testing 1097
The Unit Testing Process 1098
Define the Granularity of Your Tests 1098
Brainstorm the Individual Tests 1100
Create Sample Data and Results 1101
Write the Tests 1101
Run the Tests 1102
Unit Testing in Action 1102
Introducing the Microsoft Visual C++ Testing Framework 1103
Writing the First Test 1105
Building and Running Tests 1105
Negative Tests 1106
Adding the Real Tests 1107

Debugging Tests 1109
Basking in the Glorious Light of Unit Test Results 1110
Fuzz Testing 1110
Higher-Level Testing 1110
Integration Tests 1110
Sample Integration Tests 1111
Integration Testing Techniques 1112
System Tests 1112
Regression Tests 1112
Tips For Successful Testing 1113
Summary 1114
Exercises 1114

CHAPTER 31: CONQUERING DEBUGGING 1117

The Fundamental Law of Debugging 1118
Bug Taxonomies 1118
Avoid Bugs 1118
Plan For Bugs 1119
Error Logging 1119
Debug Traces 1121
Debug Mode 1122
Ring Buffers 1126
Assertions 1129
Crash Dumps 1131
Debugging Techniques 1131
Reproducing Bugs 1132
Debugging Reproducible Bugs 1133
Debugging Nonreproducible Bugs 1133
Debugging Regressions 1134
Debugging Memory Problems 1134
Categories of Memory Errors 1135
Tips for Debugging Memory Errors 1137
Debugging Multithreaded Programs 1139
Debugging Example: Article Citations 1140
Buggy Implementation of an ArticleCitations Class 1140
Testing the ArticleCitations Class 1143
Lessons from the ArticleCitations Example 1151
Summary 1152
Exercises 1152

CHAPTER 32: INCORPORATING DESIGN TECHNIQUES AND FRAMEWORKS 1155

"I Can Never Remember How To. . ." 1156
. . .Write a Class 1156
. . .Derive from an Existing Class 1158
. . .Write a Lambda Expression 1158
. . .Use the Copy-and-Swap Idiom 1159
. . .Throw and Catch Exceptions 1160
. . .Write a Class Template 1161
. . .Constrain Template Parameters 1162
. . .Write to a File 1162
. . .Read from a File 1162
There Must Be a Better Way 1163
Resource Acquisition Is Initialization 1163
Double Dispatch 1166
Attempt #1: Brute Force 1167
Attempt #2: Single Polymorphism with Overloading 1168
Attempt #3: Double Dispatch 1169
Mixin Classes 1171
Using Multiple Inheritance 1171
Using Class Templates 1173
Using CRTP 1174
Using CRTP and Deducing this 1175
Object-Oriented Frameworks 1175
Working with Frameworks 1176
The Model-View-Controller Paradigm 1176
Summary 1177
Exercises 1178

CHAPTER 33: APPLYING DESIGN PATTERNS 1179

The Strategy Pattern 1180
Example: A Logging Mechanism 1180
Implementation of a Strategy-Based Logger 1181
Using the Strategy-Based Logger 1182
The Abstract Factory Pattern 1183
Example: A Car Factory Simulation 1183
Implementation of an Abstract Factory 1184
Using an Abstract Factory 1185
The Factory Method Pattern 1186
Example: A Second Car Factory Simulation 1186

Implementation of a Factory Method 1187
Using a Factory Method 1189
Other Uses 1190
Other Factory Patterns 1191
The Adapter Pattern 1192
Example: Adapting a Logger Class 1192
Implementation of an Adapter 1193
Using an Adapter 1194
The Proxy Pattern 1194
Example: Hiding Network Connectivity Issues 1194
Implementation of a Proxy 1195
Using a Proxy 1196
The Iterator Pattern 1196
The Observer Pattern 1197
Example: Exposing Events from Subjects 1197
Implementation of an Observable 1197
Using an Observer 1199
The Decorator Pattern 1200
Example: Defining Styles in Web Pages 1200
Implementation of a Decorator 1201
Using a Decorator 1202
The Chain of Responsibility Pattern 1202
Example: Event Handling 1203
Implementation of a Chain of Responsibility 1203
Using a Chain of Responsibility 1204
The Singleton Pattern 1205
Example: A Logging Mechanism 1206
Implementation of a Singleton 1206
Using a Singleton 1208
Summary 1209
Exercises 1209

CHAPTER 34: DEVELOPING CROSS-PLATFORM AND CROSS-LANGUAGE APPLICATIONS 1211

Cross-Platform Development 1212
Architecture Issues 1212
Size of Integers 1212
Binary Compatibility 1213
Address Sizes 1214
Byte Order 1214
Implementation Issues 1216

Compiler Quirks and Extensions 1216
Library Implementations 1216
Handling Different Implementations 1217
Platform-Specific Features 1217
Cross-Language Development 1219
Mixing C and C++ 1219
Shifting Paradigms 1219
Linking with C Code 1222
Calling C++ Code from C# 1224
Use C# Code from C++ and C++ from C# with C++/CLI 1226
Calling C++ Code from Java with JNI 1227
Calling Scripts from C++ Code 1229
Calling C++ Code from Scripts 1230
A Practical Example: Encrypting Passwords 1230
Calling Assembly Code from C++ 1232
Summary 1233
Exercises 1234

PART VI: APPENDICES

APPENDIX A: C++ INTERVIEWS 1239

APPENDIX B: ANNOTATED BIBLIOGRAPHY 1265

APPENDIX C: STANDARD LIBRARY HEADER FILES 1277

APPENDIX D: INTRODUCTION TO UML 1287

INDEX *1293*

INTRODUCTION

The development of C++ started in 1982 by Bjarne Stroustrup, a Danish computer scientist, as the successor of C with Classes. In 1985, the first edition of *The C++ Programming Language* book was released. The first standardized version of C++ was released in 1998, called C++98. In 2003, C++03 came out and contained a few small updates. After that, it was silent for a while, but traction slowly started building up, resulting in a major update of the language in 2011, called C++11. From then on, the C++ Standard Committee has been on a three-year cycle to release updated versions, giving us C++14, C++17, C++20, and now C++23. All in all, with the release of C++23 in 2023, C++ is almost 40 years old and still going strong. In most rankings of programming languages in 2023, C++ is in the top four. It is being used on an extremely wide range of hardware, going from small devices with embedded microprocessors all the way up to multi-rack supercomputers. Besides wide hardware support, C++ can be used to tackle almost any programming job, be it games on mobile platforms, performance-critical artificial intelligence (AI) and machine learning (ML) software, components for self-driving cars, real-time 3-D graphics engines, low-level hardware drivers, entire operating systems, software stacks for networking equipment, web browsers, and so on. The performance of C++ programs is hard to match with any other programming language, and as such, it is the de facto language for writing fast, powerful, and enterprise-class programs. Big tech companies, such as Microsoft, Facebook, Amazon, Google, and many more, use services written in C++ to run their infrastructure. As popular as C++ has become, the language can be difficult to grasp in full. There are simple, but powerful, techniques that professional C++ programmers use that don't show up in traditional texts, and there are useful parts of C++ that remain a mystery even to experienced C++ programmers.

Too often, programming books focus on the syntax of the language instead of its real-world use. The typical C++ text introduces a major part of the language in each chapter, explaining the syntax and providing an example. *Professional* C++ does not follow this pattern. Instead of giving you just the nuts and bolts of the language with little practical context, this book will teach you how to use C++ in the real world. It will show you the little-known features that will make your life easier, as well as the programming techniques that separate novices from professional programmers.

WHO THIS BOOK IS FOR

Even if you have used the language for years, you might still be unfamiliar with the more advanced features of C++, or you might not be using the full capabilities of the language. Maybe you don't yet know all the new features introduced with the latest release, C++23. Perhaps you write competent C++ code but would like to learn more about design and good programming style in C++. Or maybe you're relatively new to C++ but want to learn the "right" way to program from the start. This book will meet those needs and bring your C++ skills to the professional level.

Because this book focuses on advancing from basic or intermediate knowledge of C++ to becoming a professional C++ programmer, it assumes that you have some knowledge about programming.

Chapter 1, "A Crash Course in C++ and the Standard Library," covers the basics of C++ as a refresher, but it is not a substitute for actual training in programming. If you are just starting with C++ but you have experience in another programming language such as C, Java, or C#, you should be able to pick up most of what you need from Chapter 1.

In any case, you should have a solid foundation in programming fundamentals. You should know about loops, functions, and variables. You should know how to structure a program, and you should be familiar with fundamental techniques such as recursion. You should have some knowledge of common data structures such as queues, and useful algorithms such as sorting and searching. You don't need to know about object-oriented programming just yet—that is covered in Chapter 5, "Designing with Classes."

You will also need to be familiar with the compiler you will be using to compile your code. Two compilers, Microsoft Visual C++ and GCC, are introduced later in this introduction. For other compilers, refer to the documentation that came with your compiler.

WHAT THIS BOOK COVERS

Professional C++ uses an approach to C++ programming that will both increase the quality of your code and improve your programming efficiency. You will find discussions on new C++23 features throughout this sixth edition. These features are not just isolated to a few chapters or sections; instead, examples have been updated to use new features when appropriate.

Professional C++ teaches you more than just the syntax and language features of C++. It also emphasizes programming methodologies, reusable design patterns, and good programming style. The *Professional C++* methodology incorporates the entire software development process, from designing and writing code to debugging and working in groups. This approach will enable you to master the C++ language and its idiosyncrasies, as well as take advantage of its powerful capabilities for large-scale software development.

Imagine users who have learned all of the syntax of C++ without seeing a single example of its use. They know just enough to be dangerous! Without examples, they might assume that all code should go in the `main()` function of the program or that all variables should be global—practices that are generally not considered hallmarks of good programming.

Professional C++ programmers understand the correct way to use the language, in addition to the syntax. They recognize the importance of good design, the theories of object-oriented programming, and the best ways to use existing libraries. They have also developed an arsenal of useful code and reusable ideas.

By reading and understanding this book, you will become a professional C++ programmer. You will expand your knowledge of C++ to cover lesser known and often misunderstood language features. You will gain an appreciation for object-oriented design and acquire top-notch debugging skills. Perhaps most important, you will finish this book armed with a wealth of reusable ideas that you can actually apply to your daily work.

There are many good reasons to make the effort to be a professional C++ programmer as opposed to a programmer who knows C++. Understanding the true workings of the language will improve the quality of your code. Learning about different programming methodologies and processes will help you to work better with your team. Discovering reusable libraries and common design patterns will improve your daily efficiency and help you stop reinventing the wheel. All of these lessons will make you a better programmer and a more valuable employee. While this book can't guarantee you a promotion, it certainly won't hurt.

HOW THIS BOOK IS STRUCTURED

This book is made up of five parts.

Part I, "Introduction to Professional C++," begins with a crash course in C++ basics to ensure a foundation of C++ knowledge. Following the crash course, Part I goes deeper into working with strings, because strings are used extensively in most examples throughout the book. The last chapter of Part I explores how to write *readable* C++ code.

Part II, "Professional C++ Software Design," discusses C++ design methodologies. You will read about the importance of design, the object-oriented methodology, and the importance of code reuse.

Part III, "C++ Coding the Professional Way," provides a technical tour of C++ from the professional point of view. You will read about the best ways to manage memory in C++, how to create reusable classes, and how to leverage important language features such as inheritance. You will also learn techniques for input and output, error handling, string localization, how to work with regular expressions, and how to structure your code in reusable components called modules. You will read about how to implement operator overloading, how to write templates, how to put restrictions on template parameters using concepts, and how to unlock the power of lambda expressions and function objects. This part also explains the C++ Standard Library, including containers, iterators, ranges, and algorithms. You will also read about some additional libraries that are available in the standard, such as the libraries to work with time, dates, time zones, random numbers, and the filesystem.

Part IV, "Mastering Advanced Features of C++," demonstrates how you can get the most out of C++. This part of the book exposes the mysteries of C++ and describes how to use some of its more advanced features. You will read about how to customize and extend the C++ Standard Library to your needs, advanced details on template programming, including template metaprogramming, and how to use multithreading to take advantage of multiprocessor and multicore systems.

Part V, "C++ Software Engineering," focuses on writing enterprise-quality software. You'll read about the engineering practices being used by programming organizations today; how to write efficient C++ code; software testing concepts, such as unit testing and regression testing; techniques used to debug C++ programs; how to incorporate design techniques, frameworks, and conceptual object-oriented design patterns into your own code; and solutions for cross-language and cross-platform code.

The book concludes with a useful chapter-by-chapter guide to succeeding in a C++ technical interview, an annotated bibliography, a summary of the C++ header files available in the standard, and a brief introduction to the Unified Modeling Language (UML).

This book is not a reference of every single class, member function, and function available in C++. The book *C++17 Standard Library Quick Reference* by Peter Van Weert and Marc Gregoire (Apress, 2019. ISBN: 978-1-4842-4923-9) is a condensed reference to all essential data structures, algorithms, and functions provided by the C++ Standard Library up until the C++17 standard.[1] Appendix B, "Annotated Bibliography," lists a couple more references. Two excellent online references are:

- `cppreference.com`: You can use this reference online or download an offline version for use when you are not connected to the Internet.
- `cplusplus.com/reference`

When I refer to a "Standard Library Reference" in this book, I am referring to one of these detailed C++ references.

The following are additional excellent online resources:

- `github.com/isocpp/CppCoreGuidelines`: The *C++ Core Guidelines* are a collaborative effort led by Bjarne Stroustrup, inventor of the C++ language itself. They are the result of many person-years of discussion and design across a number of organizations. The aim of the guidelines is to help people to use modern C++ effectively. The guidelines are focused on relatively higher-level issues, such as interfaces, resource management, memory management, and concurrency.
- `github.com/Microsoft/GSL`: This is an implementation by Microsoft of the *Guidelines Support Library* (GSL) containing functions and types that are suggested for use by the C++ Core Guidelines. It's a header-only library.
- `isocpp.org/faq`: This is a large collection of frequently asked C++ questions.
- `stackoverflow.com`: Search for answers to common programming questions—or ask your own questions.

CONVENTIONS

To help you get the most from the text and keep track of what's happening, a number of conventions are used throughout this book.

> **WARNING** *Boxes like this one hold important, not-to-be-forgotten information that is directly relevant to the surrounding text.*

> **NOTE** *Tips, hints, tricks, and asides to the current discussion are placed in boxes like this one.*

[1] At the time of this writing, an updated edition called *C++23 Standard Library Quick Reference* is being worked on, which is a similar condensed reference but includes all C++20 and C++23 features.

As for styles in the text:

> Important words are *italic* when they are introduced.
>
> Keyboard strokes are shown like this: Ctrl+A.
>
> Filenames and code within the text are shown like so: `monkey.cpp`.
>
> URLs are shown like this: `wiley.com`

Code is presented in three different ways:

```
// Comments in code are shown like this.
In code examples, new and important code is highlighted like this.
Code that's less important in the present context or that has been shown before is
formatted like this.
```

C++23

Paragraphs or sections that are specific to the C++23 standard have a little C++23 icon on the left, just as this paragraph does. C++11, C++14, C++17, and C++20 features are not marked with any icon.

WHAT YOU NEED TO USE THIS BOOK

All you need to use this book is a computer with a C++ compiler. This book focuses only on parts of C++ that have been standardized, and not on vendor-specific compiler extensions.

Any C++ Compiler

You can use whichever C++ compiler you like. If you don't have a C++ compiler yet, you can download one for free. There are a lot of choices. For example, for Windows, you can download Microsoft Visual Studio Community Edition, which is free and includes Visual C++. For Linux, you can use GCC or Clang, which are also free.

The following two sections briefly explain how to use Visual C++ and GCC. Refer to the documentation that came with your compiler for more details.

COMPILERS AND C++23 FEATURE SUPPORT

This book discusses new features introduced with the C++23 standard. At the time of this writing, no compilers were fully C++23-compliant yet. Some new features were only supported by some compilers and not others, while other features were not yet supported by any compiler. Compiler vendors are hard at work to catch up with all new features, and I'm sure it won't take long before there will be full C++23-compliant compilers available. You can keep track of which compiler supports which features at `en.cppreference.com/w/cpp/compiler_support`.

COMPILERS AND C++ MODULE SUPPORT

At the time of this writing, not all compilers fully support modules yet; though all major compilers do, at least partially. This book uses modules everywhere. If your compiler does not yet support modules, you can convert modularized code to non-modularized code, as explained briefly in Chapter 11, "Modules, Header Files, and Miscellaneous Topics."

Example: Microsoft Visual C++ 2022

First, you need to create a project. Start Visual C++ 2022, and on the welcome screen, click the Create A New Project button. If the welcome screen is not shown, select File ➪ New ➪ Project. In the Create A New Project dialog, search for the Console App project template with tags C++, Windows, and Console, and click Next. Specify a name for the project and a location where to save it and click Create.

Once your new project is loaded, you can see a list of project files in the Solution Explorer. If this docking window is not visible, select View ➪ Solution Explorer. A newly created project will contain a file called `<projectname>.cpp` under the Source Files section in the Solution Explorer. You can start writing your C++ code in that `.cpp` file, or if you want to compile source code files from the downloadable source archive for this book, select the `<projectname>.cpp` file in the Solution Explorer and delete it. You can add new files or existing files to a project by right-clicking the project name in the Solution Explorer and then selecting Add ➪ New Item or Add ➪ Existing Item.

At the time of this writing, Visual C++ 2022 does not yet automatically enable C++23 features. To enable C++23 features, in the Solution Explorer window, right-click your project and click Properties. In the Properties window, go to Configuration Properties ➪ General, set the C++ Language Standard option to ISO C++23 Standard or Preview - Features from the Latest C++ Working Draft, whichever is available in your version of Visual C++, and click OK.

Finally, select Build ➪ Build Solution to compile your code. When it compiles without errors, you can run it with Debug ➪ Start Debugging.

NOTE *Microsoft Visual C++ has full support for modules, including the C++23 standard named module* `std`.

Example: GCC

You can create your source code files with any text editor you prefer and save them to a directory. To compile your code, open a terminal and run the following command, specifying all your `.cpp` files that you want to compile:

```
g++ -std=c++2b -o <executable_name> <source1.cpp> [source2.cpp ...]
```

The `-std=c++2b` option is required to tell GCC to enable C++23 features. This option will change to `-std=C++23` once GCC is fully C++23-compliant.

Module Support

Support for modules in GCC is enabled with the `-fmodules-ts` option.

At the time of this writing, GCC does not yet support the C++23 standard named module `std`, introduced in Chapter 1. To make such code compile, you have to replace `import std;` declarations with `import` declarations of individual Standard Library headers. Once that is done, `import` declarations of Standard Library headers, such as the following, require you to precompile them:

```
import <iostream>;
```

Here is an example of precompiling `<iostream>`:

```
g++ -std=c++2b -fmodules-ts -xc++-system-header iostream
```

As an example, the `AirlineTicket` code from Chapter 1 uses modules. To compile it with GCC, first replace the use of `std::println()` with `std::cout` as GCC does not yet support `<print>` functionality at the time of this writing. After that, replace the `import std;` declarations with the appropriate `import` declarations, `<string>` and `<iostream>` for this example. You can find the adapted code in the `Examples\Ch00\AirlineTicket` directory in the downloadable source code archive.

Then, compile the two standard headers `<iostream>` and `<string>`:

```
g++ -std=c++2b -fmodules-ts -xc++-system-header iostream
g++ -std=c++2b -fmodules-ts -xc++-system-header string
```

Compile the module interface file:

```
g++ -std=c++2b -fmodules-ts -c -x c++ AirlineTicket.cppm
```

Finally, compile the application itself:

```
g++ -std=c++2b -fmodules-ts -o AirlineTicket AirlineTicket.cpp
AirlineTicketTest.cpp AirlineTicket.o
```

When it compiles without errors, you can run it as follows:

```
./AirlineTicket
```

NOTE *The process of compiling C++ code using C++ modules with GCC might change in the future. Also, support for the C++23 standard named module* `std` *will be added. In that case, please consult the GCC documentation for an updated procedure on how to compile such code.*

C++23's Support for Printing Ranges

Chapter 2, "Working with Strings and String Views," explains that you can easily print the entire contents of Standard Library containers, such as `std::vector`, to the screen. This is a new feature since C++23 and not all compilers support this yet at the time of this writing.

As an example, Chapter 2 explains that you can write the contents of an `std::vector` as follows. Don't worry if you don't understand all the syntax yet, you will at the end of Chapter 2.

```
std::vector values { 11, 22, 33 };
std::print("{:n}", values);
```

This outputs:

```
11, 22, 33
```

If your compiler does not yet support this C++23 feature to print the contents of a container using `std::print()`, then you can convert the second line of code to the following:

```
for (const auto& value : values) { std::cout << value << ", "; }
```

This outputs:

```
11, 22, 33,
```

Again, don't worry if you don't understand the syntax yet. All will be clear at the end of Chapter 2.

READER SUPPORT FOR THIS BOOK

The following sections describe different options to get support for this book.

Companion Download Files

As you work through the examples in this book, you may choose either to type in all the code manually or to use the source code files that accompany the book. However, I suggest you type in all the code manually because it greatly benefits the learning process and your memory. All of the source code used in this book is available for download at `www.wiley.com/go/proc++6e` or from GitHub at `github.com/Professional-CPP/edition-6`.

NOTE *Because many books have similar titles, you may find it easiest to search by ISBN; for this book, the ISBN is 978-1-394-19317-2.*

Once you've downloaded the code, just decompress it with your favorite decompression tool.

How to Contact the Publisher

If you believe you've found a mistake in this book, please bring it to our attention. At John Wiley & Sons, we understand how important it is to provide our customers with accurate content, but even with our best efforts an error may occur.