

Contents

Introduction	1
PART I: The Basics	
LESSON 1: Getting Started	5
A Brief History of C++	6
Connection to C	6
Advantages of C++	6
Evolution of the C++ Standard	7
Who Uses Programs Written in C++?	7
Programming a C++ Application	7
Steps to Generating an Executable	7
Analyzing Errors and “Debugging”	8
Integrated Development Environments	8
Programming Your First C++ Application	9
Building and Executing Your First C++ Application	10
Understanding Compiler Errors	12
What’s New in C++?	12
LESSON 2: The Anatomy of a C++ Program	17
Parts of the Hello World Program	18
Preprocessor Directive <code>#include</code>	18
The Body of Your Program <code>main()</code>	19
Returning a Value	20
The Concept of Namespaces	21
Comments in C++ Code	22
Functions in C++	23
Basic Input Using <code>std::cin</code> and Output Using <code>std::cout</code>	26
LESSON 3: Using Variables, Declaring Constants	31
What Is a Variable?	32
Memory and Addressing in Brief	32
Declaring Variables to Access and Use Memory	32

Declaring and Initializing Multiple Variables of a Type	34
Understanding the Scope of a Variable	35
Global Variables	37
Naming Conventions	38
Common Compiler-Supported C++ Variable Types	39
Using Type <code>bool</code> to Store Boolean Values	40
Using Type <code>char</code> to Store Character Values	41
The Concept of Signed and Unsigned Integers	41
Signed Integer Types <code>short</code> , <code>int</code> , <code>long</code> , and <code>long long</code>	42
Unsigned Integer Types <code>unsigned short</code> , <code>unsigned int</code> , <code>unsigned long</code> , and <code>unsigned long long</code>	42
Avoid Overflow Errors by Selecting Correct Data Types	43
Floating-Point Types <code>float</code> and <code>double</code>	45
Determining the Size of a Variable Using <code>sizeof</code>	46
Avoid Narrowing Conversion Errors by Using List Initialization	48
Automatic Type Inference Using <code>auto</code>	48
Using <code>typedef</code> to Substitute a Variable's Type	50
What Is a Constant?	50
Literal Constants	51
Declaring Variables as Constants Using <code>const</code>	52
Constant Expressions Using <code>constexpr</code>	53
Enumerations	55
Defining Constants Using <code>#define</code>	57
Keywords You Cannot Use as Variable or Constant Names	58
LESSON 4: Managing Arrays and Strings	63
What Is an Array?	64
The Need for Arrays	64
Declaring and Initializing Static Arrays	65
How Data Is Stored in an Array	66
Accessing Data Stored in an Array	67
Modifying Data Stored in an Array	69
Multidimensional Arrays	71
Declaring and Initializing Multidimensional Arrays	72
Accessing Elements in a Multidimensional Array	73

Dynamic Arrays	74
C-style Character Strings	76
C++ Strings: Using <code>std::string</code>	79
LESSON 5: Working with Expressions, Statements, and Operators	85
Statements	86
Compound Statements or Blocks	87
Using Operators	87
The Assignment Operator (<code>=</code>)	87
Understanding L-values and R-values	87
Operators to Add (+), Subtract (-), Multiply (*), Divide (/), and Modulo Divide (%)	88
Operators to Increment (++) and Decrement (--)	89
To Postfix or to Prefix?	90
Equality Operators (<code>==</code>) and (<code>!=</code>)	92
Relational Operators	92
Logical Operations NOT, AND, OR, and XOR	95
Using C++ Logical Operators NOT (<code>!</code>), AND (<code>&&</code>), and OR (<code> </code>)	96
Bitwise NOT (<code>~</code>), AND (<code>&</code>), OR (<code> </code>), and XOR (<code>^</code>) Operators	100
Bitwise Right Shift (<code>>></code>) and Left Shift (<code><<</code>) Operators	102
Compound Assignment Operators	104
Using Operator <code>sizeof</code> to Determine the Memory Occupied by a Variable	106
Operator Precedence	108
LESSON 6: Controlling Program Flow	113
Conditional Execution Using <code>if ... else</code>	114
Conditional Programming Using <code>if ... else</code>	115
Executing Multiple Statements Conditionally	117
Nested <code>if</code> Statements	118
Conditional Processing Using <code>switch-case</code>	122
Conditional Execution Using Operator (<code>? :</code>)	126
Getting Code to Execute in Loops	128
A Rudimentary Loop Using <code>goto</code>	128
The <code>while</code> Loop	130
The <code>do...while</code> Loop	132
The <code>for</code> Loop	133
The Range-Based <code>for</code> Loop	137

Modifying Loop Behavior Using <code>continue</code> and <code>break</code>	139
Loops That Don't End—That Is, Infinite Loops	140
Controlling Infinite Loops	141
Programming Nested Loops	143
Using Nested Loops to Walk a Multidimensional Array	145
Using Nested Loops to Calculate Fibonacci Numbers	147
LESSON 7: Organizing Code with Functions	151
The Need for Functions	152
What Is a Function Prototype?	153
What Is a Function Definition?	154
What Is a Function Call, and What Are Arguments?	154
Programming a Function with Multiple Parameters	155
Programming Functions with No Parameters or No Return Values	156
Function Parameters with Default Values	157
Recursion—Functions That Invoke Themselves	159
Functions with Multiple Return Statements	161
Using Functions to Work with Different Forms of Data	162
Overloading Functions	163
Passing an Array of Values to a Function	165
Passing Arguments by Reference	166
How Function Calls Are Handled by the Microprocessor	168
Inline Functions	169
Automatic Return Type Deduction	171
Lambda Functions	172
LESSON 8: Pointers and References Explained	177
What Is a Pointer?	178
Declaring a Pointer	178
Determining the Address of a Variable Using the Reference Operator (<code>&</code>)	179
Using Pointers to Store Addresses	180
Access Pointed Data Using the Dereference Operator (<code>*</code>)	183
What Is the <code>sizeof()</code> of a Pointer?	185
Dynamic Memory Allocation	187
Using Operators <code>new</code> and <code>delete</code> to Allocate and Release Memory Dynamically	187
Effect of Incrementing and Decrementing Operators (<code>++</code> and <code>--</code>) on Pointers	191

Using the <code>const</code> Keyword on Pointers	193
Passing Pointers to Functions	194
Similarities between Arrays and Pointers	195
Common Programming Mistakes When Using Pointers	198
Memory Leaks	198
When Pointers Don't Point to Valid Memory Locations	199
Dangling Pointers (Also Called Stray or Wild Pointers)	200
Checking Whether Allocation Request Using <code>new</code> Succeeded	202
Pointer Programming Best-Practices	204
What Is a Reference?	205
What Makes References Useful?	206
Using Keyword <code>const</code> on References	208
Passing Arguments by Reference to Functions	208

PART II: Fundamentals of Object-Oriented C++ Programming

LESSON 9: Classes and Objects	215
The Concept of Classes and Objects	216
Declaring a Class	216
An Object as an Instance of a Class	217
Accessing Members Using the Dot Operator (.)	218
Accessing Members Using the Pointer Operator (<code>-></code>)	219
Keywords <code>public</code> and <code>private</code>	220
Abstraction of Data via Keyword <code>private</code>	222
Constructors	224
Declaring and Implementing a Constructor	224
When and How to Use Constructors	225
Overloading Constructors	227
Class Without a Default Constructor	228
Constructor Parameters with Default Values	230
Constructors with Initialization Lists	231
Destructor	233
Declaring and Implementing a Destructor	234
When and How to Use a Destructor	234
Copy Constructor	237
Shallow Copying and Associated Problems	237
Ensuring Deep Copy Using a Copy Constructor	240
Move Constructors Help Improve Performance	244

Different Uses of Constructors and the Destructor.....	246
Class That Does Not Permit Copying.....	246
Singleton Class That Permits a Single Instance.....	247
Class That Prohibits Instantiation on the Stack.....	249
Using Constructors to Convert Types.....	251
this Pointer.....	254
sizeof() a Class.....	255
How <code>struct</code> Differs from <code>class</code>	257
Declaring a <code>friend</code> of a <code>class</code>	258
union: A Special Data Storage Mechanism.....	260
Declaring a Union.....	260
Where Would You Use a <code>union</code> ?.....	261
Using Aggregate Initialization on Classes and Structs.....	263
<code>constexpr</code> with Classes and Objects.....	266
LESSON 10: Implementing Inheritance	271
Basics of Inheritance.....	272
Inheritance and Derivation.....	272
C++ Syntax of Derivation.....	274
Access Specifier Keyword <code>protected</code>	276
Base Class Initialization—Passing Parameters to the Base Class.....	279
Derived Class Overriding Base Class's Methods.....	281
Invoking Overridden Methods of a Base Class.....	283
Invoking Methods of a Base Class in a Derived Class.....	284
Derived Class Hiding Base Class's Methods.....	286
Order of Construction.....	288
Order of Destruction.....	288
Private Inheritance.....	291
Protected Inheritance.....	293
The Problem of Slicing.....	297
Multiple Inheritance.....	297
Avoiding Inheritance Using <code>final</code>	300
LESSON 11: Polymorphism	305
Basics of Polymorphism.....	306
Need for Polymorphic Behavior.....	306
Polymorphic Behavior Implemented Using Virtual Functions.....	308

Need for Virtual Destructors.....	310
How Do <code>virtual</code> Functions Work? Understanding the Virtual Function Table.....	314
Abstract Base Classes and Pure Virtual Functions.....	318
Using <code>virtual</code> Inheritance to Solve the Diamond Problem.....	321
Specifier <code>Override</code> to Indicate Intention to <code>Override</code>	326
Use <code>final</code> to Prevent Function Overriding.....	327
Virtual Copy Constructors?.....	328
LESSON 12: Operator Types and Operator Overloading	335
What Are Operators in C++?.....	336
Unary Operators.....	337
Types of Unary Operators.....	337
Programming a Unary Increment/Decrement Operator.....	338
Programming Conversion Operators.....	341
Programming Dereference Operator (*) and Member Selection Operator (->).....	344
Binary Operators.....	346
Types of Binary Operators.....	346
Programming Binary Addition (a+b) and Subtraction (a-b) Operators.....	347
Implementing Addition Assignment (+=) and Subtraction Assignment (-=) Operators.....	350
Overloading Equality (==) and Inequality (!=) Operators.....	352
Overloading <, >, <=, and >= Operators.....	354
Overloading Copy Assignment Operator (=).....	357
Subscript Operator ([]).....	360
Function Operator ().....	364
Move Constructor and Move Assignment Operator for High Performance Programming.....	365
The Problem of Unwanted Copy Steps.....	365
Declaring a Move Constructor and Move Assignment Operator.....	366
User Defined Literals.....	371
Operators That Cannot Be Overloaded.....	373
LESSON 13: Casting Operators	377
The Need for Casting.....	378
Why C-Style Casts Are Not Popular with Some C++ Programmers.....	379

The C++ Casting Operators	379
Using <code>static_cast</code>	380
Using <code>dynamic_cast</code> and Runtime Type Identification	381
Using <code>reinterpret_cast</code>	384
Using <code>const_cast</code>	385
Problems with the C++ Casting Operators	386
LESSON 14: An Introduction to Macros and Templates	391
The Preprocessor and the Compiler	392
Using Macro <code>#define</code> to Define Constants	392
Using Macros for Protection against Multiple Inclusion	395
Using <code>#define</code> to Write Macro Functions	396
Why All the Parentheses?	398
Using Macro <code>assert</code> to Validate Expressions	399
Advantages and Disadvantages of Using Macro Functions	400
An Introduction to Templates	402
Template Declaration Syntax	402
The Different Types of Template Declarations	403
Template Functions	403
Templates and Type Safety	405
Template Classes	406
Declaring Templates with Multiple Parameters	407
Declaring Templates with Default Parameters	408
Sample Template <code>class<> HoldsPair</code>	408
Template Instantiation and Specialization	410
Template Classes and <code>static</code> Members	412
Variable Templates, Also Called Variadic Templates	413
Using <code>static_assert</code> to Perform Compile-Time Checks	417
Using Templates in Practical C++ Programming	418
PART III: Learning the Standard Template Library (STL)	
LESSON 15: An Introduction to the Standard Template Library	421
STL Containers	422
Sequential Containers	422
Associative Containers	423
Container Adapters	425

STL Iterators.....	425
STL Algorithms.....	426
The Interaction between Containers and Algorithms Using Iterators.....	427
Using Keyword <code>auto</code> to Let Compiler Define Type	429
Choosing the Right Container.....	429
STL String Classes.....	432
LESSON 16: The STL String Class	435
The Need for String Manipulation Classes.....	436
Working with the STL String Class.....	437
Instantiating the STL String and Making Copies.....	437
Accessing Character Contents of a <code>std::string</code>	440
Concatenating One String to Another.....	442
Finding a Character or Substring in a String.....	444
Truncating an STL <code>string</code>	445
String Reversal.....	448
String Case Conversion.....	449
Template-Based Implementation of an STL String.....	450
C++14 operator <code>""</code> s in <code>std::string</code>	451
LESSON 17: STL Dynamic Array Classes	455
The Characteristics of <code>std::vector</code>	456
Typical Vector Operations.....	456
Instantiating a Vector.....	456
Inserting Elements at the End Using <code>push_back()</code>	458
List Initialization.....	459
Inserting Elements at a Given Position Using <code>insert()</code>	459
Accessing Elements in a Vector Using Array Semantics	462
Accessing Elements in a Vector Using Pointer Semantics	464
Removing Elements from a Vector	465
Understanding the Concepts of Size and Capacity.....	467
The STL <code>deque</code> Class.....	469
LESSON 18: STL <code>list</code> and <code>forward_list</code>	475
The Characteristics of a <code>std::list</code>	476
Basic <code>list</code> Operations.....	476
Instantiating a <code>std::list</code> Object.....	476
Inserting Elements at the Front or Back of the List.....	478

Inserting at the Middle of the List	479
Erasing Elements from the List	482
Reversing and Sorting Elements in a List	483
Reversing Elements Using <code>list::reverse()</code>	484
Sorting Elements	485
Sorting and Removing Elements from a <code>list</code> That Contains Instances of a <code>class</code>	487
<code>std::forward_list</code> Introduced in C++11	490
LESSON 19: STL Set Classes	495
An Introduction to STL Set Classes	496
Basic <code>STL set</code> and <code>multiset</code> Operations	496
Instantiating a <code>std::set</code> Object	497
Inserting Elements in a <code>set</code> or <code>multiset</code>	499
Finding Elements in an STL <code>set</code> or <code>multiset</code>	500
Erasing Elements in an STL <code>set</code> or <code>multiset</code>	502
Pros and Cons of Using STL <code>set</code> and <code>multiset</code>	507
STL Hash Set Implementation <code>std::unordered_set</code> and <code>std::unordered_multiset</code>	507
LESSON 20: STL Map Classes	513
An Introduction to STL Map Classes	514
Basic <code>std::map</code> and <code>std::multimap</code> Operations	515
Instantiating a <code>std::map</code> or <code>std::multimap</code>	515
Inserting Elements in an STL <code>map</code> or <code>multimap</code>	517
Finding Elements in an STL <code>map</code>	519
Finding Elements in an STL <code>multimap</code>	522
Erasing Elements from an STL <code>map</code> or <code>multimap</code>	522
Supplying a Custom Sort Predicate	525
STL's Hash Table-Based Key-Value Container	528
How Hash Tables Work	529
Using <code>unordered_map</code> and <code>unordered_multimap</code>	529
PART IV: More STL	
LESSON 21: Understanding Function Objects	537
The Concept of Function Objects and Predicates	538
Typical Applications of Function Objects	538

Unary Functions	538
Unary Predicate	543
Binary Functions	545
Binary Predicate	547
LESSON 22: Lambda Expressions	553
What Is a Lambda Expression?	554
How to Define a Lambda Expression	555
Lambda Expression for a Unary Function	555
Lambda Expression for a Unary Predicate	557
Lambda Expression with State via Capture Lists [...]	559
The Generic Syntax of Lambda Expressions	560
Lambda Expression for a Binary Function	562
Lambda Expression for a Binary Predicate	564
LESSON 23: STL Algorithms	569
What Are STL Algorithms?	570
Classification of STL Algorithms	570
Non-Mutating Algorithms	570
Mutating Algorithms	571
Usage of STL Algorithms	573
Finding Elements Given a Value or a Condition	573
Counting Elements Given a Value or a Condition	576
Searching for an Element or a Range in a Collection	577
Initializing Elements in a Container to a Specific Value	580
Using <code>std::generate()</code> to Initialize Elements to a Value	582
Generated at Runtime	582
Processing Elements in a Range Using <code>for_each()</code>	583
Performing Transformations on a Range Using <code>std::transform()</code>	585
Copy and Remove Operations	588
Replacing Values and Replacing Element Given a Condition	590
Sorting and Searching in a Sorted Collection and Erasing Duplicates	592
Partitioning a Range	595
Inserting Elements in a Sorted Collection	597

LESSON 24: Adaptive Containers: Stack and Queue	603
The Behavioral Characteristics of Stacks and Queues	604
Stacks	604
Queues	604
Using the STL <code>stack</code> Class	605
Instantiating the Stack	605
Stack Member Functions	606
Insertion and Removal at Top Using <code>push()</code> and <code>pop()</code>	607
Using the STL <code>queue</code> Class	609
Instantiating the Queue	609
Member Functions of a <code>queue</code>	610
Insertion at End and Removal at the Beginning of <code>queue</code>	610
via <code>push()</code> and <code>pop()</code>	611
Using the STL Priority Queue	613
Instantiating the <code>priority_queue</code> Class	613
Member Functions of <code>priority_queue</code>	615
Insertion at the End and Removal at the Beginning of <code>priority_queue</code>	615
via <code>push()</code> and <code>pop()</code>	616
LESSON 25: Working with Bit Flags Using STL	621
The <code>bitset</code> Class	622
Instantiating the <code>std::bitset</code>	622
Using <code>std::bitset</code> and Its Members	623
Useful Operators Featured in <code>std::bitset</code>	624
<code>std::bitset</code> Member Methods	625
The <code>vector<bool></code>	627
Instantiating <code>vector<bool></code>	627
<code>vector<bool></code> Functions and Operators	628
PART V: Advanced C++ Concepts	
LESSON 26: Understanding Smart Pointers	633
What Are Smart Pointers?	634
The Problem with Using Conventional (Raw) Pointers	634
How Do Smart Pointers Help?	634
How Are Smart Pointers Implemented?	635
Types of Smart Pointers	636
Deep Copy	637
Copy on Write Mechanism	639

Reference-Counted Smart Pointers	639
Reference-Linked Smart Pointers	640
Destructive Copy	640
Using the <code>std::unique_ptr</code>	643
Popular Smart Pointer Libraries	645
LESSON 27: Using Streams for Input and Output	649
Concept of Streams	650
Important C++ Stream Classes and Objects	651
Using <code>std::cout</code> for Writing Formatted Data to Console	652
Changing Display Number Formats Using <code>std::cout</code>	653
Aligning Text and Setting Field Width Using <code>std::cout</code>	655
Using <code>std::cin</code> for Input	656
Using <code>std::cin</code> for Input into a Plain Old Data Type	656
Using <code>std::cin::get</code> for Input into <code>char*</code> Buffer	657
Using <code>std::cin</code> for Input into a <code>std::string</code>	658
Using <code>std::fstream</code> for File Handling	660
Opening and Closing a File Using <code>open()</code> and <code>close()</code>	660
Creating and Writing a Text File Using <code>open()</code> and <code>operator<<</code>	662
Reading a Text File Using <code>open()</code> and <code>operator>></code>	663
Writing to and Reading from a Binary File	664
Using <code>std::stringstream</code> for String Conversions	666
LESSON 28: Exception Handling	671
What Is an Exception?	672
What Causes Exceptions?	672
Implementing Exception Safety via <code>try</code> and <code>catch</code>	673
Using <code>catch(...)</code> to Handle All Exceptions	673
Catching Exception of a Type	674
Throwing Exception of a Type Using <code>throw</code>	676
How Exception Handling Works	677
Class <code>std::exception</code>	680
Your Custom Exception Class Derived from <code>std::exception</code>	680
LESSON 29: Going Forward	687
What's Different in Today's Processors?	688
How to Better Use Multiple Cores	689
What Is a Thread?	689
Why Program Multithreaded Applications?	690

How Can Threads Transact Data?	691
Using Mutexes and Semaphores to Synchronize Threads	692
Problems Caused by Multithreading	692
Writing Great C++ Code	693
C++17: Expected Features	694
if and switch Support Initializers	695
Copy Elision Guarantee	696
std::string_view Avoids Allocation Overheads	696
std::variant As a Typesafe Alternative to a union	697
Conditional Code Compilation Using if constexpr	697
Improved Lambda Expressions	698
Automatic Type Deduction for Constructors	698
template<auto>	699
Learning C++ Doesn't Stop Here!	699
Online Documentation	699
Communities for Guidance and Help	699

PART VI: Appendixes

APPENDIX A: Working with Numbers: Binary and Hexadecimal	701
APPENDIX B: C++ Keywords	707
APPENDIX C: Operator Precedence	709
APPENDIX D: ASCII Codes	711
APPENDIX E: Answers	717
Index	763