



Beginning C++20

From Novice to Professional

—

Sixth Edition

—

Ivor Horton
Peter Van Weert

Apress®

Beginning C++20

From Novice to Professional

Sixth Edition



Ivor Horton
Peter Van Weert

Apress®

Beginning C++20: From Novice to Professional

Ivor Horton
Stratford-upon-Avon, Warwickshire, UK

Peter Van Weert
Kessel-Lo, Belgium

ISBN-13 (pbk): 978-1-4842-5883-5
<https://doi.org/10.1007/978-1-4842-5884-2>

ISBN-13 (electronic): 978-1-4842-5884-2

Copyright © 2020 by Ivor Horton and Peter Van Weert

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Steve Anglin
Development Editor: Matthew Moodie
Coordinating Editor: Mark Powers

Cover designed by eStudioCalamar

Cover image designed by Casey Horner on Unsplash (www.unsplash.com)

Distributed to the book trade worldwide by Apress Media, LLC, 1 New York Plaza, New York, NY 10004, U.S.A. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484258835. For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper

*This is for Alexander and Henry, who are both going to learn
programming soon.*

*If their amazing expertise with Minecraft is anything to go by,
they will be brilliant at it.*

—Ivor Horton

For my wonderful family. For all your love and support.

—Peter Van Weert

Table of Contents

About the Authors.....	xxiii
About the Technical Reviewer	xxv
Introduction	xxvii
■ Chapter 1: Basic Ideas.....	1
Modern C++.....	1
Standard Libraries.....	2
C++ Program Concepts.....	3
Source Files.....	3
Comments and Whitespace.....	4
Standard Library Modules	5
Functions.....	5
Statements	6
Data Input and Output.....	7
return Statements.....	7
Namespaces	8
Names and Keywords.....	8
Classes and Objects	9
Templates.....	9
Code Appearance and Programming Style.....	10
Creating an Executable	11
Procedural and Object-Oriented Programming	12

Representing Numbers.....	13
Binary Numbers.....	13
Hexadecimal Numbers	15
Negative Binary Numbers.....	16
Octal Values	18
Big-Endian and Little-Endian Systems	18
Floating-Point Numbers.....	19
Representing Characters.....	21
ASCII Codes	21
UCS and Unicode	22
C++ Source Characters.....	22
Escape Sequences	23
Summary	25
■ Chapter 2: Introducing Fundamental Types of Data	27
Variables, Data, and Data Types	27
Defining Integer Variables.....	28
Zero Initialization	32
Defining Variables with Fixed Values	32
Integer Literals	32
Decimal Integer Literals.....	32
Hexadecimal Literals	34
Octal Literals.....	34
Binary Literals.....	34
Calculations with Integers.....	35
Compound Arithmetic Expressions	36
Assignment Operations	37
The op= Assignment Operators	40
The sizeof Operator	41
Incrementing and Decrementing Integers.....	42
Postfix Increment and Decrement Operations.....	42

Defining Floating-Point Variables	44
Floating-Point Literals	45
Floating-Point Calculations	45
Mathematical Constants	46
Mathematical Functions	47
Invalid Floating-Point Results	50
Pitfalls	51
Mixed Expressions and Type Conversion	52
Explicit Type Conversion	53
Old-Style Casts	56
Formatting Strings	56
Formatting Stream Output	57
String Formatting with <code>std::format()</code>	57
Finding the Limits	63
Finding Other Properties of Fundamental Types	64
Working with Character Variables	65
Working with Unicode Characters	66
The <code>auto</code> Keyword	67
Summary	68
■ Chapter 3: Working with Fundamental Data Types	71
Operator Precedence and Associativity	71
Bitwise Operators	73
The Bitwise Shift Operators	74
Logical Operations on Bit Patterns	77
The Lifetime of a Variable	84
Global Variables	85
Enumerated Data Types	89
Aliases for Data Types	93
Summary	94

■ Chapter 4: Making Decisions	97
Comparing Data Values	97
Applying the Comparison Operators	98
Comparing Floating-Point Values	100
The Spaceship Operator	100
The if Statement.....	103
Nested if Statements	107
Character Classification and Conversion	108
The if-else Statement.....	110
Nested if-else Statements	112
Understanding Nested ifs	113
Logical Operators	115
Logical AND	115
Logical OR.....	116
Logical Negation.....	116
Combining Logical Operators	117
Logical Operators on Integer Operands	119
Logical Operators vs. Bitwise Operators	119
The Conditional Operator.....	121
The switch Statement	123
Fallthrough	128
Statement Blocks and Variable Scope.....	130
Initialization Statements.....	131
Summary	132
■ Chapter 5: Arrays and Loops	135
Arrays	135
Using an Array	135
Understanding Loops	137
The for Loop	138
Avoiding Magic Numbers	141

Defining the Array Size with the Braced Initializer	142
Determining the Size of an Array	143
Controlling a for Loop with Floating-Point Values	145
More Complex for Loop Control Expressions.....	147
The Comma Operator.....	148
The Range-Based for Loop.....	149
The while Loop	150
The do-while Loop.....	153
Nested Loops.....	155
Skipping Loop Iterations.....	158
Breaking Out of a Loop.....	159
Indefinite Loops	159
Controlling a for Loop with Unsigned Integers	163
Arrays of Characters.....	165
Multidimensional Arrays.....	168
Initializing Multidimensional Arrays.....	170
Multidimensional Character Arrays	173
Allocating an Array at Runtime.....	174
Alternatives to Using an Array	177
Using array<T,N> Containers	177
Using std::vector<T> Containers.....	183
Summary	188
■ Chapter 6: Pointers and References	191
What Is a Pointer?	191
The Address-Of Operator.....	194
The Indirection Operator.....	195
Why Use Pointers?	197
Pointers to Type char.....	197
Arrays of Pointers	200

Constant Pointers and Pointers to Constants	202
Pointers and Arrays	205
Pointer Arithmetic	205
Using Pointer Notation with an Array Name	208
Dynamic Memory Allocation	210
The Stack and the Free Store	211
Using the new and delete Operators	212
Dynamic Allocation of Arrays	213
Member Selection Through a Pointer	217
Hazards of Dynamic Memory Allocation	218
Dangling Pointers and Multiple Deallocations	218
Allocation/Deallocation Mismatch	218
Memory Leaks	219
Fragmentation of the Free Store	219
Golden Rule of Dynamic Memory Allocation	220
Raw Pointers and Smart Pointers	221
Using <code>unique_ptr<T></code> Pointers	222
Using <code>shared_ptr<T></code> Pointers	225
Understanding References	229
Defining References	229
Using a Reference Variable in a Range-Based for Loop	231
Summary	232
■ Chapter 7: Working with Strings	235
A Better Class of String	235
Defining string Objects	236
Operations with String Objects	240
Accessing Characters in a String	244
Accessing Substrings	246
Comparing Strings	247

Searching Strings	255
Modifying a String	262
std::string vs. std::vector<char>	267
Converting Strings into Numbers	268
Strings of International Characters	268
Strings of wchar_t Characters.....	269
Objects That Contain Unicode Strings	270
Raw String Literals	271
Summary	272
■ Chapter 8: Defining Functions	275
Segmenting Your Programs.....	275
Functions in Classes.....	276
Characteristics of a Function.....	276
Defining Functions	276
The Function Body	279
Return Values.....	280
Function Declarations.....	281
Passing Arguments to a Function.....	282
Pass-by-Value.....	283
Pass-by-Reference	291
Default Argument Values	298
Multiple Default Parameter Values	299
Arguments to main().....	301
Returning Values from a Function	302
Returning a Pointer.....	302
Returning a Reference.....	307
Returning vs. Output Parameters	307
Return Type Deduction.....	308
Static Variables.....	310

Function Overloading	311
Overloading and Pointer Parameters.....	313
Overloading and Reference Parameters.....	313
Overloading and const Parameters	315
Overloading and Default Argument Values	317
Recursion	318
Basic Examples of Recursion	319
Recursive Algorithms.....	320
Summary.....	327
■ Chapter 9: Vocabulary Types	333
Working with Optional Values.....	333
std::optional	335
String Views: The New Reference-to-const-string	337
Using String View Function Parameters	339
A Proper Motivation	340
Spans: The New Reference-to-vector or -array	340
Spans vs. Views.....	342
Spans of const Elements	342
Fixed-Size Spans.....	343
Summary.....	344
■ Chapter 10: Function Templates	347
Function Templates	347
Creating Instances of a Function Template	348
Template Type Parameters	350
Explicit Template Arguments	351
Function Template Specialization.....	351
Function Templates and Overloading	352
Function Templates with Multiple Parameters	354
Return Type Deduction in Templates	355
decltype(auto).....	356

Default Values for Template Parameters	357
Non-Type Template Parameters	358
Templates for Functions with Fixed-Size Array Arguments	360
Abbreviated Function Templates	362
Limitations to Abbreviated Function Templates	362
Summary	363
■ Chapter 11: Modules and Namespaces	365
Modules	365
Your First Module	367
Export Blocks	369
Separating Interface from Implementation	370
Reachability vs. Visibility	375
Exporting Import Declarations	376
Managing Larger Modules	377
Global Module Fragments	381
Namespaces	382
The Global Namespace	382
Defining a Namespace	383
Nested Namespaces	385
Namespaces and Modules	386
Functions and Namespaces	387
Using Directives and Declarations	389
Namespace Aliases	391
Summary	391
■ Chapter 12: Defining Your Own Data Types	395
Classes and Object-Oriented Programming	395
Encapsulation	396
Inheritance	399
Polymorphism	400
Terminology	402

Defining a Class	402
Creating Objects of a Class.....	404
Constructors.....	405
Default Constructors.....	405
Defining a Class Constructor	406
Using the default Keyword	408
Defining Functions Outside the Class.....	408
Default Arguments for Constructor Parameters	409
Using a Member Initializer List.....	410
Using the explicit Keyword	411
Delegating Constructors	413
The Copy Constructor	415
Defining Classes in Modules	417
Accessing Private Class Members	418
The this Pointer	420
Returning this from a Function.....	421
const Objects and const Member Functions	422
const Member Functions	423
const Correctness.....	425
Overloading on const.....	426
Casting Away const	427
Using the mutable Keyword.....	428
Friends	429
The Friend Functions of a Class	429
Friend Classes	431
Arrays of Class Objects	432
The Size of a Class Object.....	434
Static Members of a Class	434
Static Member Variables.....	435
Accessing Static Member Variables	438
Static Constants	438

Static Member Variables of the Class Type Itself.....	439
Static Member Functions.....	440
Destructors.....	441
Using Pointers as Class Members.....	444
The Truckload Example.....	444
Nested Classes.....	459
Nested Classes with Public Access	460
Summary.....	464
■ Chapter 13: Operator Overloading.....	467
Implementing Operators for a Class.....	467
Operator Overloading.....	468
Implementing an Overloaded Operator.....	468
Nonmember Operator Functions	471
Implementing Full Support for an Operator	471
Operators That Can Be Overloaded	473
Restrictions and Key Guideline.....	474
Operator Function Idioms	476
Supporting All Comparison Operators	477
Defaulting Comparison Operators	481
Overloading the << Operator for Output Streams.....	483
Overloading the Arithmetic Operators	485
Implementing One Operator in Terms of Another	489
Member vs. Nonmember Functions	491
Operator Functions and Implicit Conversions.....	492
Overloading Unary Operators	493
Overloading the Increment and Decrement Operators.....	494
Overloading the Subscript Operator.....	496
Modifying the Result of an Overloaded Subscript Operator.....	499
Function Objects	501

Overloading Type Conversions	503
Potential Ambiguities with Conversions	504
Overloading the Assignment Operator.....	504
Implementing the Copy Assignment Operator	505
Copy Assignment vs. Copy Construction	508
Deleting the Copy Assignment Operator	509
Assigning Different Types	510
Summary	510
■ Chapter 14: Inheritance	513
Classes and Object-Oriented Programming	513
Hierarchies	514
Inheritance in Classes	515
Inheritance vs. Aggregation and Composition	516
Deriving Classes	516
Protected Members of a Class	519
The Access Level of Inherited Class Members	520
Access Specifiers and Class Hierarchies.....	521
Choosing Access Specifiers in Class Hierarchies	522
Changing the Access Specification of Inherited Members	524
Constructors in a Derived Class	524
The Copy Constructor in a Derived Class.....	528
The Default Constructor in a Derived Class	530
Inheriting Constructors	530
Destructors Under Inheritance	531
The Order in Which Destructors Are Called.....	533
Duplicate Member Variable Names	533
Duplicate Member Function Names	534
Multiple Inheritance	535
Multiple Base Classes.....	535
Inherited Member Ambiguity	537

Repeated Inheritance	541
Virtual Base Classes	542
Converting Between Related Class Types	542
Summary	543
■ Chapter 15: Polymorphism	547
Understanding Polymorphism	547
Using a Base Class Pointer	547
Calling Inherited Functions	549
Virtual Functions	553
Requirements for Virtual Function Operation	556
Using the override Specifier	557
Using final	558
Virtual Functions and Class Hierarchies	559
Access Specifiers and Virtual Functions	560
Default Argument Values in Virtual Functions	562
Using References to Call Virtual Functions	563
Polymorphic Collections	564
Destroying Objects Through a Pointer	565
Virtual Destructors	566
Converting Between Pointers to Class Objects	567
Dynamic Casts	570
Casting Pointers Dynamically	570
Converting References	573
Calling the Base Class Version of a Virtual Function	573
Calling Virtual Functions from Constructors or Destructors	574
The Cost of Polymorphism	577
Determining the Dynamic Type	578

Pure Virtual Functions	582
Abstract Classes	583
An Abstract Box Class	584
Abstract Classes as Interfaces	586
Summary	589
■ Chapter 16: Runtime Errors and Exceptions.....	593
Handling Errors	593
Understanding Exceptions.....	594
Throwing an Exception	595
The Exception-Handling Process.....	597
Code That Causes an Exception to Be Thrown	599
Nested try Blocks	600
Class Objects as Exceptions.....	601
Matching a Catch Handler to an Exception.....	603
Catching Derived Class Exceptions with a Base Class Handler.....	606
Rethrowing Exceptions.....	608
Unhandled Exceptions.....	611
Catching All Exceptions	612
Functions That Don't Throw Exceptions	615
The noexcept Specifier	615
Exceptions and Destructors.....	616
Exceptions and Resource Leaks.....	617
Resource Acquisition Is Initialization	618
Standard RAII Classes for Dynamic Memory	620
Standard Library Exceptions	622
The Exception Class Definitions	624
Using Standard Exceptions.....	625
Summary	629

■ Chapter 17: Class Templates	633
Understanding Class Templates	633
Defining Class Templates	635
Template Type Parameters	636
A Simple Class Template	637
Defining Member Functions of a Class Template	638
Constructor Templates	639
The Destructor Template	640
Subscript Operator Templates	640
The Assignment Operator Template	643
Class Template Instantiation	648
Explicit Template Instantiation	649
Testing the Array Class Template	649
Non-Type Class Template Parameters	652
Templates for Member Functions with Non-Type Parameters	653
Arguments for Non-Type Parameters	658
Non-Type Template Arguments vs. Constructor Arguments	658
Default Values for Template Parameters	659
Class Template Argument Deduction	660
Class Template Specialization	662
Defining a Class Template Specialization	663
Partial Template Specialization	664
Class Templates with Nested Classes	664
Function Templates for Stack Members	667
Dependent Names Nuisances	672
Dependent Type Names	673
Dependent Base Classes	675
Summary	677

■ Chapter 18: Move Semantics	681
Lvalues and Rvalues.....	681
Rvalue References.....	683
Moving Objects.....	683
Defining Move Members.....	686
Explicitly Moved Objects	690
Move-Only Types	691
Extended Use of Moved Objects	692
A Barrel of Contradictions	693
std::move() Does Not Move.....	693
An Rvalue Reference Is an Lvalue	694
Defining Functions Revisited.....	694
Pass-by-Rvalue-Reference	695
The Return of Pass-by-Value	697
Return-by-Value	700
Defining Move Members Revisited.....	702
Always Add noexcept.....	702
The “Move-and-Swap” Idiom	706
Special Member Functions.....	708
Default Move Members	709
The Rule of Five	709
The Rule of Zero	710
Summary.....	711
■ Chapter 19: First-Class Functions	713
Pointers to Functions	714
Defining Pointers to Functions	714
Callback Functions for Higher-Order Functions.....	717
Type Aliases for Function Pointers.....	720

Function Objects	721
Basic Function Objects	721
Standard Function Objects	723
Parameterized Function Objects.....	724
Lambda Expressions	726
Defining a Lambda Expression	726
Naming a Lambda Closure	728
Passing a Lambda Expression to a Function Template.....	728
Generic Lambdas.....	729
The Capture Clause	730
The <code>std::function<></code> Template	735
Summary	737
■ Chapter 20: Containers and Algorithms	741
Containers	741
Sequence Containers.....	742
Stacks and Queues	745
Associative Containers	747
Iterators.....	755
The Iterator Design Pattern	755
Iterators for Standard Library Containers	757
Iterators for Arrays.....	767
Algorithms	768
A First Example.....	768
Finding Elements	770
Handling Multiple Output Values.....	772
The Remove-Erase Idiom.....	774
Sorting	776
Parallel Algorithms	777

Ranges and Views	778
Range-Based Algorithms	778
Views	780
Summary	785
■ Chapter 21: Constrained Templates and Concepts	791
Unconstrained Templates	792
Constrained Templates	793
Concepts	794
Concept Definitions and Expressions	795
Requires Expressions	796
Asserting That a Type Models a Concept	801
Standard Concepts	801
Requires Clauses	803
Shorthand Notation	804
Constrained Function Templates	805
Constrained Class Templates	807
Constrained Class Members	808
Constraint-Based Specialization	809
Constraining Auto	812
Summary	813
■ Index	815

About the Authors



Ivor Horton graduated as a mathematician and was lured into information technology with promises of great rewards for very little work. In spite of the reality being a great deal of work for relatively modest rewards, he has continued to work with computers to the present day. He has been engaged at various times in programming, systems design, consultancy, and the management and implementation of projects of considerable complexity.

Ivor has many years of experience in designing and implementing systems for engineering design and manufacturing control. He has developed occasionally useful applications in a wide variety of programming languages and has taught primarily scientists and engineers to do likewise. His currently published works include tutorials on C, C++, and Java. At the present time, when he is not writing programming books or providing advice to others, he spends his time fishing, traveling, and enjoying life in general.



Peter Van Weert is a Belgian software engineer whose main interests and expertise are application software development, programming languages, algorithms, and data structures.

He received his master of science degree in computer science *summa cum laude* with congratulations of the Board of Examiners from the University of Leuven. In 2010, he completed his PhD thesis on the design and efficient compilation of rule-based programming languages at the research group for declarative programming languages and artificial intelligence. During his doctoral studies he was a teaching assistant for object-oriented programming (Java), software analysis and design, and declarative programming.

Peter then joined Nikon Metrology, where he worked on large-scale, industrial application software in the area of 3D laser scanning and point cloud inspection for over six years. Today, Peter is senior C++ engineer

and Scrum team leader at Medicim, the R&D unit for digital dentistry software of Envista Holdings. At Medicim, he codevelops a suite of applications for dental professionals, capable of capturing patient data from a wide range of hardware, with advanced diagnostic functionality and support for implant planning and prosthetic design.

Common themes in his professional career include advanced desktop application development, mastering and refactoring of code bases of millions of lines of C++ code, high-performant, real-time processing of 3D data, concurrency, algorithms and data structures, interfacing with cutting-edge hardware, and leading agile development teams.

In his spare time Peter has co-authored two editions of two books on C++, as well as two award-winning Windows apps. Peter is also a regular expert speaker at, and board member of, the Belgian C++ Users Group.

About the Technical Reviewer

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’s 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 architect at Nikon Metrology (www.nikonmetrology.com), a division of Nikon and a leading provider of precision optical instruments, X-ray machines, and metrology solutions for X-ray, CT, and 3D geometric inspection.

His main expertise is C/C++, specifically Microsoft VC++ and the MFC framework. He has experience in developing C++ programs running 24/7 on Windows and Linux platforms: for example, KNX/EIB home automation software. In addition to C/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 (www.becpp.org), author of “Professional C++” 2nd, 3rd, and 4th Editions (Wiley/Wrox), co-author of “C++ Standard Library Quick Reference” 1st and 2nd Editions (Apress), technical editor for numerous books for several publishers, and regular speaker at the CppCon C++ conference. He maintains a blog at www.nuonsoft.com/blog/.

Introduction

Welcome to *Beginning C++20*. This is a revised and updated version of Ivor Horton’s original book called *Beginning ANSI C++*. The C++ language has been extended and improved considerably since then, so much so that it was no longer possible to squeeze detailed explanations of all of C++ into a single book. This tutorial will teach the essentials of the C++ language and Standard Library features, which will be more than enough for you to write your own C++ applications. With the knowledge from this book, you should have no difficulty in extending the depth and scope of your C++ expertise.

We have assumed no prior programming knowledge. If you are keen to learn and have an aptitude for thinking logically, getting a grip on C++ will be easier than you might imagine. By developing C++ skills, you’ll be learning a language that is already used by millions and that provides the capability for application development in just about any context.

C++ is very powerful. Arguably, it’s more powerful than most programming languages. So, yes, like with any powerful tool you can wield some considerable damage if you use it without proper training. We often compare C++ to a Swiss Army knife: age-old, trusted, incredibly versatile, yet potentially mind-boggling and full of pointy things that could really hurt you. Once someone clearly explains to you what all the different tools are meant for, however, and teaches you some elementary knife safety rules, then you’ll never have to look for another pocketknife again.

C++ does not need to be dangerous or difficult at all either. C++ today is much more accessible than many people assume. The language has come a long way since its conception nearly 40 years ago. We have learned how to wield all its mighty blades and tools in the safest and most effective way possible. And, more importantly perhaps, the C++ language and its Standard Library have evolved accordingly to facilitate this. The past decade has seen the rise of what is now known as “modern C++.” Modern C++ emphasizes the use of newer, more expressive, safer language features, combined with tried and tested best practices and coding guidelines. Once you know and apply a handful of simple rules and techniques, C++ loses much of its complexity. The key is that someone properly and gradually explains not simply what you *can* do with C++ but rather what you *should* do with C++. And that’s where this book comes in!

In this latest revision of the book, we have gone to great lengths to bring it back in line with the new, modern era of C++ programming we’re living in. As before, we of course do so in the form of a gradual, informal tutorial. We’ll introduce to you all the shiny blades and pointy things C++ has to offer—both old and new—using many hands-on coding samples and exercises. But that’s not all: more than ever before we’ve made sure to always explain which tool is best to use for which purpose, why that is the case, and how to avoid getting cut. We’ve made sure that you will begin C++, from day one, using the safe, productive, modern programming style that employers will expect from you tomorrow.

The C++ language in this book corresponds to the latest International Organization for Standardization (ISO) standard, commonly referred to as C++20. Not everything in C++20 is covered, since some of the extensions compared to previous versions of the language are targeted toward more advanced use.

Using the Book

To learn C++ with this book, you'll need a compiler that conforms to the C++20 standard and a text editor suitable for working with program code. Several compilers are available currently that support, to some extent, C++20 features, many of which are free.

■ **Note** At the time of writing, no compiler fully supports C++20. If the past is any guide, we are confident they will catch up soon, though. https://en.cppreference.com/w/cpp/compiler_support provides an excellent overview of what C++20 features are supported by all major compilers. If your compiler does not support a certain feature yet, you may have to skip some examples or rework them in terms of alternatives.

GCC and Clang are free, open source compilers, with increasing support for C++20. Installing these compilers and putting them together with a suitable editor can be a little tricky if you are new to this kind of thing. An easy way to install a compiler along with a suitable editor is to download a free integrated development environment (IDEs) such as Code::Blocks or Qt Creator. Such IDEs support a complete program development for several compilers, including GCC and Clang.

Another possibility is to use the commercial Microsoft Visual C++ IDE that runs under Microsoft Windows. The Community edition is free for individual use or even small professional teams, and its support for C++20 is on par with GCC and Clang. With Visual Studio you get a comprehensive, easy-to-use professional editor and debugger, as well as support for other languages such as C# and Javascript.

There are other compilers that support C++20 as well, which you can find with a quick online search.

We've organized the material in this book to be read sequentially, so you should start at the beginning and keep going until you reach the end. However, no one ever learned programming by just reading a book. You'll only learn how to program in C++ by writing code, so make sure you key in all the examples—don't just copy them from the download files—and compile and execute the code that you've keyed in. This might seem tedious at times, but it's surprising how much just typing in C++ statements will help your understanding, especially when you may feel you're struggling with some of the ideas. If an example doesn't work, resist the temptation to go straight back to the book to see why. Try to figure out from your code what is wrong. This is good practice for what you'll have to do when you are developing C++ applications for real.

Making mistakes is a fundamental part of the learning process, and the exercises should provide you with ample opportunity for that. It's a good idea to dream up a few exercises of your own. If you are not sure about how to do something, just have a go before looking it up. The more mistakes you make, the greater the insight you'll have into what can, and does, go wrong. Make sure you attempt all the exercises, and remember, don't look at the solutions until you're sure that you can't work them out yourself. Most of these exercises just involve a direct application of what's covered in a chapter—they're just practice, in other words—but some also require a bit of thought or maybe even inspiration.

We wish you every success with C++. Above all, enjoy it!

Ivor Horton
Peter Van Weert

CHAPTER 1



Basic Ideas

In this book we sometimes will use certain code in the examples before having explained it in detail. This chapter is intended to help you when this occurs by presenting an overview of the major elements of C++ and how they hang together. We'll also explain a few concepts relating to the representation of numbers and characters in your computer.

In this chapter, you'll learn

- What is meant by *modern C++*
- What the terms *C++11*, *C++14*, *C++17*, and *C++20* mean
- What the C++ Standard Library is
- What are the elements of a C++ program
- How to document your program code
- How your C++ code becomes an executable program
- How object-oriented programming differs from procedural programming
- What binary, hexadecimal, and octal number systems are
- What floating-point numbers are
- How a computer represents numbers and letters using nothing but bits and bytes
- What Unicode is

Modern C++

Created in the early 1980s by Danish computer scientist Bjarne Stroustrup, C++ is one of the oldest programming languages still in active use. Despite its age, however, C++ is still standing strong, steadily maintaining its top-five position in most popularity rankings for programming languages. Just about any kind of program can be written in C++, from device drivers to operating systems and from payroll and administrative programs to games. Major operating systems, browsers, office suites, email clients, multimedia players, database systems—name one and chances are it's written at least partly in C++.

Above all else, C++ is best suited for applications where performance matters, such as applications that have to process large amounts of data, computer games with high-end graphics, or apps for embedded or mobile devices. Programs written in C++ remain many times faster than those written in other popular languages. C++ is also very effective for developing applications across an enormous range of computing devices and environments, including for personal computers, workstations, mainframe computers, tablets, and mobile phones.

The C++ programming language may be old, but it's still very much alive and kicking. Or, better yet: it's *again* very much alive and kicking. After its initial development and standardization in the 1980s, C++ evolved very slowly, and remained essentially unchanged for decades. Until 2011, when the International Organization for Standardization (ISO) released a new version of the formal C++ standard. This edition of the standard, commonly referred to as *C++11*, revived C++ and catapulted the somewhat dated language right back into the 21st Century. It modernized the language and the way we use it so profoundly that you could almost call C++11 a completely new language.

Programming using the features of C++11 and beyond is referred to as *modern C++*. In this book, we'll show you that modern C++ is about more than simply embracing the language's new features—lambda expressions (Chapter 19), auto type deduction (Chapter 2), and range-based for loops (Chapter 5), to name a few. More than anything else, modern C++ is about modern ways of programming, the consensus of what constitutes good programming style. It's about applying an implicit set of guidelines and best practices, all designed to make C++ programming easier, less error-prone, and more productive. A modern, safe C++ programming style replaces traditional low-level language constructs with the use of containers (Chapters 5 and 20), smart pointers (Chapter 6), or other RAII techniques (Chapter 16). It emphasizes exceptions to report errors (Chapter 16), passing objects by value through move semantics (Chapter 18), writing algorithms instead of loops (Chapter 20), and so on. Of course, all this probably means little to nothing to you yet. But not to worry: in this book, we'll gradually introduce everything you need to know to program in C++ today!

The C++11 standard also appears to have revived the C++ community, which has been actively working hard on extending and further improving the language ever since. Every three years, a new version of the standard is published. After C++11 came C++14, C++17, and, most recently, C++20.

After the smaller, incremental updates of C++14 and C++17, C++20 is again a major milestone. Like C++11 did a decade ago, C++20 will again change the way we program in C++ forever. Modules (Chapter 11) obsolete C++'s formerly antiquated ways of composing larger programs (such as headers and `#include` guards: see online Appendix A), concepts (Chapter 21) facilitate type-safe templates and flexible template specialization (Chapters 9 and 17), and ranges (Chapter 20) revolutionize the way we manipulate data. The best thing about all these C++20 features—and in particular for someone beginning C++ today—is that they make the language easier, more elegant, and more accessible than ever before.

This book relates to C++ as defined by C++20. All code should work on any compiler that complies with the C++20 edition of the standard. The good news is that most major compilers work hard to keep up with all latest developments, so if your compiler does not support a particular feature yet, it soon will.

Standard Libraries

If you had to create everything from scratch every time you wrote a program, it would be tedious indeed. The same functionality is required in many programs—reading data from the keyboard, calculating a square root, sorting data records into a particular sequence, and so on. C++ comes with a large amount of prewritten code that provides facilities such as these so you don't have to write the code yourself. All this standard code is defined in the *Standard Library*.

The Standard Library is a huge collection of routines and definitions that provide functionality that is required by many programs. Examples are numerical calculations, string processing, sorting and searching, organizing and managing data, and input and output. We'll introduce major Standard Library functionalities in virtually every chapter and will later zoom in a bit more specifically on some key data structures and algorithms in Chapter 20. Nevertheless, the Standard Library is so vast that in this book we will only scratch the surface of what is available. You really need several books to fully elaborate all the capabilities it provides. *Beginning STL* (Apress, 2015) is a companion book that is a tutorial on using the Standard Template Library, which is the subset of the C++ Standard Library for managing and processing data in various ways. For a compact yet complete overview of everything the C++17 Standard Library has to offer, we also recommend the book *C++17 Standard Library Quick Reference* (Apress, 2019).

Given the scope of the language and the extent of the library, it's not unusual for a beginner to find C++ somewhat daunting. It is too extensive to learn in its entirety from a single book. However, you don't need to learn all of C++ to be able to write substantial programs. You can approach the language step-by-step, in which case it really isn't difficult. An analogy might be learning to drive a car. You can certainly become a competent and safe driver without necessarily having the expertise, knowledge, and experience to drive in the Indianapolis 500. With this book you can learn everything you need to program effectively in C++. By the time you reach the end, you'll be confidently writing your own applications. You'll also be well equipped to explore the full extent of C++ and its Standard Library.

C++ Program Concepts

There will be much more detail on everything we discuss in this section later in the book. We'll jump straight in with the complete, fully working C++ program shown in Figure 1-1, which also explains what the various bits are. We'll use the example as a base for discussing some more general aspects of C++.

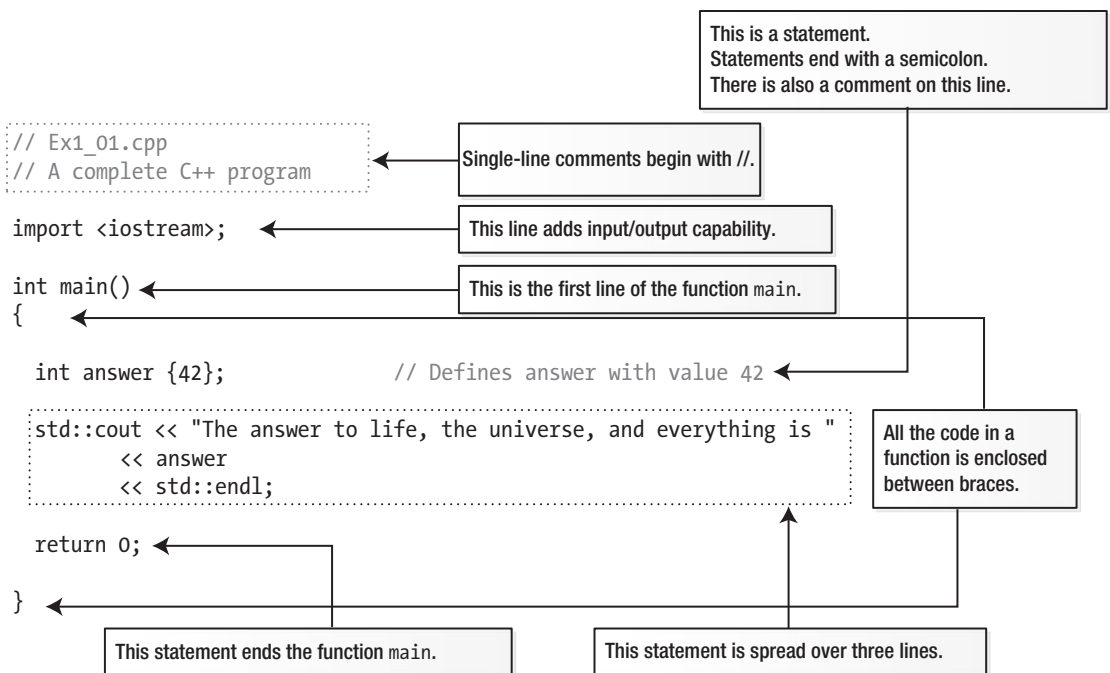


Figure 1-1. A complete C++ program

Source Files

The file depicted in Figure 1-1, `Ex1_01.cpp`, is in the code download for the book. The file extension, `.cpp`, indicates that this is a C++ *source file* or *implementation file*. Source files contain function bodies and thus most of the executable code in a program. The names of source files usually have the extension `.cpp`, although other extensions such as `.cc`, `.cxx`, or `.c++` are sometimes used to identify a C++ source file.

For the first ten chapters all your programs will be small enough to fit in a single C++ source file. Real-life programs, however, are often composed of thousands of files. Not all source files, mind you. In larger program you use other kinds of files as well, mostly to separate out the interfaces of the program's various components (function prototypes, class definitions, module interfaces, and so on) from their implementation in a corresponding source file. In Chapter 11 you will learn about the various kinds of files used to define C++20 modules, and in Appendix A we review the old-fashioned *header files* that are used by legacy code to compose larger programs.

■ **Note** Appendix A is available online together with the source code of all examples and exercises.

Comments and Whitespace

The first two lines in Figure 1-1 are *comments*. You add comments that document your program code to make it easier to understand how it works. The compiler ignores everything that follows two successive forward slashes on a line, so this kind of comment can follow code on a line. In our example, the first line is a comment that indicates the name of the file containing this code. We'll identify the file for each working example in the same way.

■ **Note** The comment with the filename in each file is only there for your convenience. In normal coding there is no need to add such comments; they only introduce unnecessary maintenance overhead when renaming files.

There's another form of comment that you can use when you need to spread a comment over several lines. Here's an example:

```
/* This comment is
   over two lines. */
```

Everything between `/*` and `*/` will be ignored by the compiler. You can embellish this sort of comment to make it stand out. For instance:

```
/******\
 * This comment is *
 * over two lines. *
 \*****/
```

■ **Note** Because you are still learning, we often use code comments in this book to explain even the most basic lines of C++ code. Of course you should not do this in real life, at least not nearly as extensively as we do here. In real life, for instance, you would never clarify the meaning of `int answer {42};` with a comment like we did in Figure 1-1. Once you know and understand the syntax, well-written code should explain itself. Simply echoing code in a comment is generally considered bad practice. Ordinarily, you should only add comments to clarify or document aspects of your code that are not immediately obvious to its intended readers (typically you and/or your co-workers).
