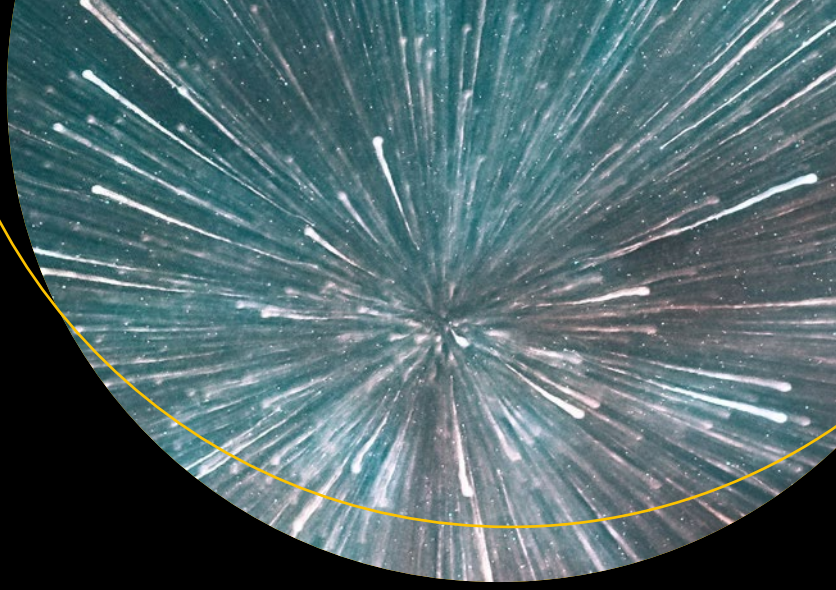

Beginning C++20

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
ISBN-13 (electronic): 978-1-4842-5884-2
https://doi.org/10.1007/978-1-4842-5884-2

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

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.

I. Horton and P. Van Weert, *Beginning C++20*, https://doi.org/10.1007/978-1-4842-5884-2_1

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++.

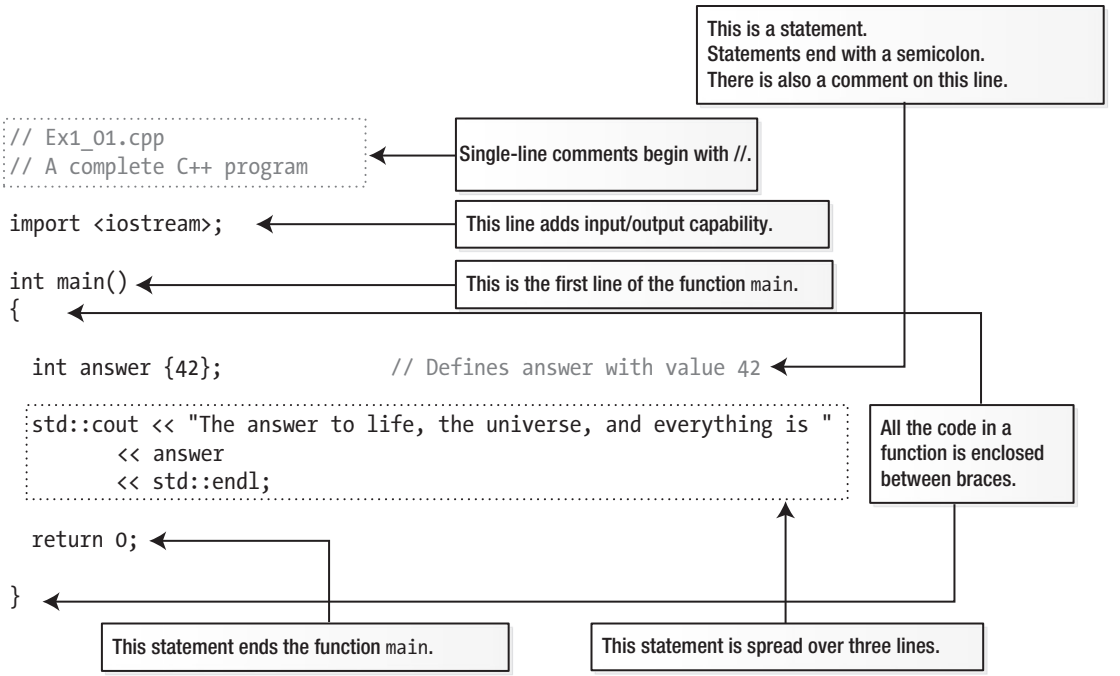

Figure 1-1. *A complete C++ program*

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 exercices.

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).
