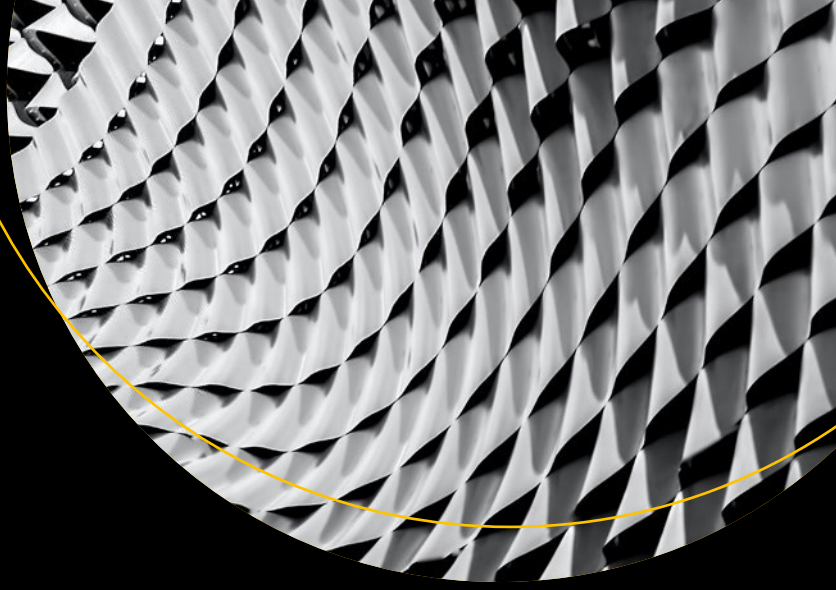

Modern C++ for Absolute Beginners

A Friendly Introduction to C++ Programming Language and C++11 to C++20 Standards

Slobodan Dmitrović

Apress®

Modern C++ for Absolute Beginners

A Friendly Introduction to C++ Programming Language and C++11 to C++20 Standards

Slobodan Dmitrović

Apress®

Modern C++ for Absolute Beginners: A Friendly Introduction to C++ Programming Language and C++11 to C++20 Standards

Slobodan Dmitrović
Belgrade, Serbia

ISBN-13 (pbk): 978-1-4842-6046-3 ISBN-13 (electronic): 978-1-4842-6047-0
https://doi.org/10.1007/978-1-4842-6047-0

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 by Ricardo Gomez Angel 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 editorial@apress.com; for reprint, paperback, or audio rights, please email 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 http://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/9781484260463. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

To M. R., whose work is an inspiration to me.

Table of Contents

About the Author

Slobodan Dmitrović is a software development consultant and an author from Serbia. He specializes in C++ training, technical analysis, and software architecture. He is a highly visible member of the SE European C++ community and a StackOverflow contributor. Slobodan has gained international experience working as a software consultant in Denmark, Poland, Croatia, China, and the Philippines. Slobodan maintains a website at `www.cppandfriends.com`.

About the Technical Reviewer

Chinmaya Patnayak is an embedded software developer at NVIDIA and is skilled in C++, CUDA, deep learning, Linux, and file systems. He has been a speaker and instructor for deep learning at various major technology events across India. Chinmaya holds an M.Sc. degree in physics and B.E. in electrical and electronics engineering from BITS Pilani. He has previously worked with Defence Research and Development Organization (DRDO) on encryption algorithms for video streams. His current interest lies in neural networks for image segmentation and applications in biomedical research and self-driving cars. Find more about him at `chinmayapatnayak.github.io`.

Acknowledgments

I would like to thank my friends and fellow C++ peers who have supported me in writing this book.

I owe my gratitude to outstanding professionals at Apress for their amazing work and support during the entire writing and production process.

I am thankful to the StackOverflow and the entire C++ community for their help and feedback.

My deepest appreciation goes to S. Antonijević, Jovo Arežina, and Saša Popović for their ongoing support.

CHAPTER 1

Introduction

Dear Reader,

Congratulations on choosing to learn the C++ programming language, and thank you for picking up this book. My name is Slobodan Dmitrović, I am a software developer and a technical writer, and I will try to introduce you to a beautiful world of C++ to the best of my abilities.

This book is an effort to introduce the reader to a C++ programming language in a structured, straightforward, and friendly manner. We will use the "just enough theory and plenty of examples" approach whenever possible.

To me, C++ is a wonderful product of the human intellect. Over the years, I have certainly come to think of it as a thing of beauty and elegance. C++ is a language like no other, surprising in its complexity, yet wonderfully sleek and elegant in so many ways. It is also a language that cannot be learned by guessing, one that is easy to get wrong and challenging to get right.

In this book, we will get familiar with the language basics first. Then, we will move onto standard-library. Once we got these covered, we will describe the modern C++ standards in more detail.

After each section, there are source code exercises to help us adopt the learned material more efficiently. Let us get started!

S. Dmitrović, *Modern C++ for Absolute Beginners*, https://doi.org/10.1007/978-1-4842-6047-0_1

CHAPTER 2

What is C++?

C++ is a programming language. A standardized, general-purpose, object-oriented, compiled language. C++ is accompanied by a set of functions and containers called the C++ Standard-Library. Bjarne Stroustrup created C++ as an extension to a C programming language. Still, C++ evolved to be a completely different programming language.

Let us emphasize this: C and C++ are two different languages. C++ started as "C with classes," but it is now a completely different language. So, C++ is not C; C++ is not C with classes; it is just C++. And there is no such thing as a C/C++ programming language.

C++ is widely used for the so-called systems programming as well as application programming. C++ is a language that allows us to *get down to the metal* where we can perform low-level routines if needed, or soar high with abstraction mechanisms such as templates and classes.

2.1 C++ Standards

C++ is governed by the ISO C++ standard. There are multiple ISO C++ standards listed here in chronological order: C++03, C++11, C++14, C++17, and the upcoming C++20 standard.

Every C++ standard starting with the C++11 onwards is referred to as "Modern C++." And modern C++ is what we will be teaching in this book.

S. Dmitrović, *Modern C++ for Absolute Beginners*, https://doi.org/10.1007/978-1-4842-6047-0_2

CHAPTER 3

C++ Compilers

C++ programs are usually a collection of C++ code spread across one or multiple source files. The C++ compiler compiles these files and turns them into object files. Object files are linked together by a linker to create an executable file or a library. At the time of the writing, some of the more popular C++ compilers are:

- The g++ frontend (as part of the GCC)
- Visual C++ (as part of the Visual Studio IDE)
- Clang (as part of the LLVM)

3.1 Installing C++ Compilers

The following sections explain how to install C++ compilers on Linux and Windows and how to compile and run our C++ programs.

3.1.1 On Linux

To install a C++ compiler on Linux, type the following inside the terminal:

```
sudo apt-get install build-essential
```

To compile the C++ source file **source.cpp**, we type:

```
g++ source.cpp
```

This command will produce an executable with the default name of **a.out**. To run the executable file, type:

```
./a.out
```


S. Dmitrović, *Modern C++ for Absolute Beginners*, https://doi.org/10.1007/978-1-4842-6047-0_3

To compile for a C++11 standard, we add the -std=c++11 flag:

```
g++ -std=c++11 source.cpp
```

To enable warnings, we add the -*Wall* flag:

```
g++ -std=c++11 -Wall source.cpp
```

To produce a custom executable name, we add the -*o* flag followed by an executable name:

```
g++ -std=c++11 -Wall source.cpp -o myexe
```

The same rules apply to the Clang compiler. Substitute *g++* with *clang++*.

3.1.2 On Windows

On Windows, we can install a free copy of Visual Studio.

Choose *Create a new project,* make sure the *C++* language option is selected, and choose - *Empty Project* – click *Next* and click *Create*. Go to the Solution Explorer panel, right-click on the project name, choose *Add – New Item – C++ File (.cpp)*, type the name of a file (**source.cpp**), and click *Add*. Press F5 to run the program.

We can also do the following: choose *Create a new project,* make sure the *C++* language option is selected, and choose – *Console App* – click *Next* and click *Create*.

If a *Create a new project* button is not visible, choose *File – New – Project* and repeat the remaining steps.

CHAPTER 4

Our First Program

Let us create a blank text file using the text editor or C++ IDE of our choice and name it *source.cpp*. First, let us create an empty C++ program that does nothing. The content of the *source.cpp* file is:

```
int main(){}
```

The function `main` is the main program entry point, the start of our program. When we run our executable, the code inside the `main` function body gets executed. A function is of type `int` (and returns a result to the system, but let us not worry about that just yet). The reserved name `main` is a function name. It is followed by a list of parameters inside the parentheses `()` followed by a function body marked with braces `{}`. Braces marking the beginning and the end of a function body can also be on separate lines:

```
int main()
{

}
```

This simple program does nothing, it has no parameters listed inside parentheses, and there are no statements inside the function body. It is essential to understand that this is the main program signature.

There is also another `main` function signature accepting two different parameters used for manipulating the command line arguments. For now, we will only use the first form .

S. Dmitrović, *Modern C++ for Absolute Beginners*, https://doi.org/10.1007/978-1-4842-6047-0_4

4.1 Comments

Single line comments in C++ start with double slashes // and the compiler ignores them. We use them to comment or document the code or use them as notes:

```
int main()
{
    // this is a comment
}
```

We can have multiple single-line comments:

```
int main()
{
    // this is a comment
    // this is another comment
}
```

Multi-line comments start with the /* and end with the */. They are also known as C-style comments. Example:

```
int main()
{
    /* This is a
    multi-line comment */
}
```

4.2 Hello World Example

Now we are ready to get the first glimpse at our "Hello World" example. The following program is the simplest "Hello World" example. It prints out Hello World. in the console window:

```
#include <iostream>

int main()
{
    std::cout << "Hello World.";
}
```

Believe it or not, the detailed analysis and explanation of this example is 15 pages long. We can go into it right now, but we will be no wiser at this point as we first need to know what headers, streams, objects, operators, and string literals are. Do not worry. We will get there.

A brief(ish) explanation

The `#include <iostream>` statement includes the `iostream` header into our source file via the `#include` directive. The `iostream` header is part of the standard library. We need its inclusion to use the `std::cout` object, also known as a standard-output stream. The `<<` operator inserts our Hello World string literal into that output stream. String literal is enclosed in double quotes `""`. The `;` marks the end of the statement. Statements are pieces of the C++program that get executed. Statements end with a semicolon `;` in C++. The `std` is the standard-library namespace and `::` is the scope resolution operator. Object `cout` is inside the `std` namespace, and to access it, we need to prepend the call with the `std::`. We will get more familiar with all of these later in the book, especially the `std::` part.

A brief explanation

In a nutshell, the `std::cout <<` is the natural way of outputting data to the standard output/console window in C++.

We can output multiple string literals by separating them with multiple `<<` operators:

```
#include <iostream>

int main()
{
    std::cout << "Some string." << " Another string.";
}
```

To output on a new line, we need to output a new-line character `\n` literal. The characters are enclosed in single quotes `'\n'`.

Example:

```
#include <iostream>

int main()
{
    std::cout << "First line" << '\n' << "Second line.";
}
```

The \ represents an escape sequence, a mechanism to output certain special characters such as new-line character `'\n'`, single quote character `'\''` or a double quote character `'\"'`.

Characters can also be part of the single string literal:

```
#include <iostream>

int main()
{
    std::cout << "First line\nSecond line.";
}
```

Do not use using namespace std;

Many examples on the web introduce the entire std namespace into the current scope via the `using namespace std;` statement only to be able to type `cout` instead of the `std::cout`. While this might save us from typing five additional characters, it is **wrong** for many reasons. We do not want to introduce the entire *std* namespace into the current scope because we want to avoid name clashes and ambiguity. Good to remember: do not introduce the entire `std` namespace into a current scope via the `using namespace std;` statement. So, instead of this wrong approach:

```
#include <iostream>

using namespace std; // do not use this

int main()
{
    cout << "A bad example.";
}
```

use the following:

```
#include <iostream>

int main()
{
    std::cout << "A good example.";
}
```

For calls to objects and functions that reside inside the `std` namespace, add the `std::` prefix where needed.

CHAPTER 5

Types

Every entity has a type. What is a type? A type is a set of possible values and operations. Instances of types are called objects. An object is some region in memory that has a value of particular type (not to be confused with an instance of a class which is also called object).

5.1 Fundamental Types

C++ has some built-in types. We often refer to them as fundamental types. A declaration is a statement that introduces a name into a current scope.

5.1.1 Boolean

Let us declare a variable `b` of type `bool`. This type holds values of `true` and `false`.

```
int main()
{
    bool b;
}
```

This example declares a variable `b` of type `bool`. And that is it. The variable is not initialized, no value has been assigned to it at the time of construction. To initialize a variable, we use an assignment operator = followed by an initializer:

```
int main()
{
    bool b = true;
}
```


S. Dmitrović, *Modern C++ for Absolute Beginners*, https://doi.org/10.1007/978-1-4842-6047-0_5

We can also use braces {} for initialization:

```
int main()
{
    bool b{ true };
}
```

These examples declare a (local) variable `b` of type `bool` and initialize it to a value of `true`. Our variable now holds a value of `true`. All local variables should be initialized. Accessing uninitialized variables results in Undefined Behavior, abbreviated as UB. More on this in the following chapters.

5.1.2 Character Type

Type `char`, referred to as *character type*, is used to represent a single character. The type can store characters such as `'a'`, `'Z'` etc. The size of a character type is exactly one byte. Character literals are enclosed in single quotes `''` in C++. To declare and initialize a variable of type `char`, we write:

```
int main()
{
    char c = 'a';
}
```

Now we can print out the value of our char variable:

```
#include <iostream>

int main()
{
    char c = 'a';
    std::cout << "The value of variable c is: " << c;
}
```

Once declared and initialized, we can access our variable and change its value:

```
#include <iostream>

int main()
{
    char c = 'a';
    std::cout << "The value of variable c is: " << c;
    c = 'Z';
    std::cout << " The new value of variable c is: " << c;
}
```

The size of the `char` type in memory is usually one byte. We obtain the size of the type through a `sizeof` operator:

```
#include <iostream>

int main()
{
    std::cout << "The size of type char is: " << sizeof(char)
    << " byte(s)";
}
```

There are other character types such as `wchar_t` for holding characters of Unicode character set, `char16_t` for holding UTF-16 character sets, but for now, let us stick to the type `char`.

A character literal is a character enclosed in single quotes. Example: `'a'`, `'A'`, `'z'`, `'X'`, `'0'` etc.

Every character is represented by an integer number in the character set. That is why we can assign both numeric literals (up to a certain number) and character literals to our `char` variable:

```
int main()
{
    char c = 'a';
    // is the same as if we had
    // char c = 97;
}
```

We can write: `char c = 'a';` or we can write `char c = 97;` which is (probably) the same, as the `'a'` character in ASCII table is represented with the number of `97`. For the most part, we will be using character literals to represent the value of a char object.

5.1.3 Integer Types

Another fundamental type is `int` called integer type. We use it to store integral values (whole numbers), both negative and positive:

```
#include <iostream>

int main()
{
    int x = 123;
    int y = -256;
    std::cout << "The value of x is: " << x << ", the value of y is: "
    << y;
}
```

Here we declared and initialized two variables of type `int`. The size of `int` is usually 4 bytes. We can also initialize the variable with another variable. It will receive a copy of its value. We still have two separate objects in memory:

```
#include <iostream>

int main()
{
    int x = 123;
    int y = x;
    std::cout << "The value of x is: " << x << " ,the value of y is: " << y;
    // x is 123
    // y is 123
    x = 456;
    std::cout << "The value of x is: " << x << " ,the value of y is: " << y;
    // x is now 456
    // y is still 123
}
```

Once we declare a variable, we access and manipulate the variable name by its name only, without the type name.

Integer literals can be decimal, octal, and hexadecimal. Octal literals start with a prefix of `0`, and hexadecimal literals begin with a prefix of `0x`.

```
int main()
{
    int x = 10;     // decimal literal
    int y = 012;    // octal literal
    int z = 0xA;    // hexadecimal literal
}
```

All these variables have been initialized to a value of `10` represented by different integer literals. For the most part, we will be using decimal literals.

There are also other integer types such as `int64_t` and others, but we will stick to `int` for now.

5.1.4 Floating-Point Types

There are three floating-point types in C++: `float`, `double`, `long double`, but we will stick to type `double` (double-precision). We use it for storing floating-point values / real numbers:

```
#include <iostream>

int main()
{
    double d = 3.14;
    std::cout << "The value of d is: " << d;
}
```

Some of the floating-point literals can be:

```
int main()
{
    double x = 213.456;
    double y = 1.;
    double z = 0.15;
```