



# 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  
<https://doi.org/10.1007/978-1-4842-6047-0>

ISBN-13 (electronic): 978-1-4842-6047-0

Copyright © 2020 by Slobodan Dmitrović

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 by Ricardo Gomez Angel on Unsplash ([www.unsplash.com](http://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](mailto:orders-ny@springer-sbm.com), or visit [www.springeronline.com](http://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](mailto:editorial@apress.com); for reprint, paperback, or audio rights, please email [bookpermissions@springernature.com](mailto: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](http://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

<b>About the Author</b> .....	<b>xv</b>
<b>About the Technical Reviewer</b> .....	<b>xvii</b>
<b>Acknowledgments</b> .....	<b>xix</b>
<b>Chapter 1: Introduction</b> .....	<b>1</b>
<b>Chapter 2: What is C++?</b> .....	<b>3</b>
2.1 C++ Standards.....	3
<b>Chapter 3: C++ Compilers</b> .....	<b>5</b>
3.1 Installing C++ Compilers .....	5
3.1.1 On Linux.....	5
3.1.2 On Windows.....	6
<b>Chapter 4: Our First Program</b> .....	<b>7</b>
4.1 Comments.....	8
4.2 Hello World Example .....	8
<b>Chapter 5: Types</b> .....	<b>11</b>
5.1 Fundamental Types .....	11
5.1.1 Boolean.....	11
5.1.2 Character Type.....	12
5.1.3 Integer Types .....	14
5.1.4 Floating-Point Types .....	15
5.1.5 Type void.....	16
5.2 Type Modifiers.....	16
5.3 Variable Declaration, Definition, and Initialization.....	17

TABLE OF CONTENTS

- Chapter 6: Exercises ..... 19**
  - 6.1 Hello World and Comments ..... 19
  - 6.2 Declaration ..... 19
  - 6.3 Definition ..... 20
  - 6.4 Initialization ..... 20
- Chapter 7: Operators ..... 21**
  - 7.1 Assignment Operator ..... 21
  - 7.2 Arithmetic Operators ..... 21
  - 7.3 Compound Assignment Operators ..... 23
  - 7.4 Increment/Decrement Operators ..... 24
- Chapter 8: Standard Input ..... 25**
- Chapter 9: Exercises ..... 27**
  - 9.1 Standard Input ..... 27
  - 9.2 Two Inputs ..... 27
  - 9.3 Multiple Inputs ..... 28
  - 9.4 Inputs and Arithmetic Operations ..... 28
  - 9.5 Post-Increment and Compound Assignment ..... 29
  - 9.6 Integral and Floating-point Division ..... 29
- Chapter 10: Arrays ..... 31**
- Chapter 11: Pointers ..... 33**
- Chapter 12: References ..... 37**
- Chapter 13: Introduction to Strings ..... 39**
  - 13.1 Defining a String ..... 39
  - 13.2 Concatenating Strings ..... 40
  - 13.3 Accessing Characters ..... 41
  - 13.4 Comparing Strings ..... 41
  - 13.5 String Input ..... 42

13.6 A Pointer to a String .....	43
13.7 Substrings.....	43
13.8 Finding a Substring .....	44
<b>Chapter 14: Automatic Type Deduction .....</b>	<b>47</b>
<b>Chapter 15: Exercises .....</b>	<b>49</b>
15.1 Array Definition .....	49
15.2 Pointer to an Object .....	49
15.3 Reference Type.....	50
15.4 Strings.....	50
15.5 Strings from Standard Input.....	51
15.6 Creating a Substring .....	51
15.7 Finding a single Character .....	52
15.8 Finding a Substring .....	52
15.9 Automatic Type Deduction.....	53
<b>Chapter 16: Statements .....</b>	<b>55</b>
16.1 Selection Statements.....	55
16.1.1 if Statement.....	55
16.1.2 Conditional Expression .....	57
16.1.3 The Logical Operators.....	58
16.1.4 switch Statement .....	63
16.2 Iteration Statements .....	64
16.2.1 for Statement.....	64
16.2.2 while Statement .....	65
16.2.3 do Statement .....	66
<b>Chapter 17: Constants .....</b>	<b>67</b>
<b>Chapter 18: Exercises .....</b>	<b>69</b>
18.1 A Simple if-statement .....	69
18.2 Logical Operators.....	69
18.3 The switch-statement .....	70

TABLE OF CONTENTS

- 18.4 The for-loop..... 71
- 18.5 Array and the for-loop ..... 72
- 18.6 The const Type Qualifier..... 72
- Chapter 19: Functions ..... 75**
- 19.1 Introduction..... 75
- 19.2 Function Declaration ..... 75
- 19.3 Function Definition ..... 76
- 19.4 Return Statement..... 79
- 19.5 Passing Arguments ..... 80
  - 19.5.1 Passing by Value/Copy..... 80
  - 19.5.2 Passing by Reference ..... 81
  - 19.5.3 Passing by Const Reference ..... 82
- 19.6 Function Overloading ..... 83
- Chapter 20: Exercises ..... 85**
- 20.1 Function Definition ..... 85
- 20.2 Separate Declaration and Definition ..... 85
- 20.3 Function Parameters..... 86
- 20.4 Passing Arguments ..... 87
- 20.5 Function Overloads ..... 87
- Chapter 21: Scope and Lifetime..... 89**
- 21.1 Local Scope..... 89
- 21.2 Block Scope ..... 89
- 21.3 Lifetime ..... 90
- 21.4 Automatic Storage Duration ..... 90
- 21.5 Dynamic Storage Duration ..... 90
- 21.6 Static Storage Duration..... 91
- 21.7 Operators new and delete..... 91



<b>Chapter 22: Exercises</b> .....	<b>93</b>
22.1 Automatic Storage Duration .....	93
22.2 Dynamic Storage Duration .....	93
22.3 Automatic and Dynamic Storage Durations .....	94
<b>Chapter 23: Classes - Introduction</b> .....	<b>95</b>
23.1 Data Member Fields .....	96
23.2 Member Functions .....	96
23.3 Access Specifiers .....	99
23.4 Constructors .....	102
23.4.1 Default Constructor .....	102
23.4.2 Member Initialization .....	104
23.4.3 Copy Constructor .....	104
23.4.4 Copy Assignment .....	107
23.4.5 Move Constructor .....	109
23.4.6 Move Assignment .....	111
23.5 Operator Overloading .....	112
23.6 Destructors .....	118
<b>Chapter 24: Exercises</b> .....	<b>121</b>
24.1 Class Instance .....	121
24.2 Class with Data Members .....	121
24.3 Class with Member Function .....	122
24.4 Class with Data and Function Members .....	122
24.5 Class Access Specifiers .....	123
24.6 User-defined Default Constructor and Destructor .....	124
24.7 Constructor Initializer List .....	125
24.8 User-defined Copy Constructor .....	126
24.9 User-defined Move Constructor .....	127
24.10 Overloading Arithmetic Operators .....	128

TABLE OF CONTENTS

- Chapter 25: Classes – Inheritance and Polymorphism ..... 131**
  - 25.1 Inheritance ..... 131
  - 25.2 Polymorphism ..... 135
- Chapter 26: Exercises ..... 141**
  - 26.1 Inheritance ..... 141
- Chapter 27: The static Specifier ..... 145**
- Chapter 28: Templates ..... 149**
- Chapter 29: Enumerations ..... 155**
- Chapter 30: Exercises ..... 159**
  - 30.1 Static variable ..... 159
  - 30.2 Static data member ..... 160
  - 30.3 Static member function ..... 160
  - 30.4 Function Template..... 161
  - 30.5 Class Template ..... 162
  - 30.6 Scoped Enums ..... 163
  - 30.7 Enums in a switch..... 164
- Chapter 31: Organizing code ..... 165**
  - 31.1 Header and Source Files ..... 165
  - 31.2 Header Guards ..... 166
  - 31.3 Namespaces ..... 166
- Chapter 32: Exercises ..... 171**
  - 32.1 Header and Source Files ..... 171
  - 32.2 Multiple Source Files ..... 172
  - 32.3 Namespaces ..... 173
  - 32.4 Nested Namespaces ..... 174

<b>Chapter 33: Conversions</b> .....	<b>175</b>
33.1 Implicit Conversions.....	175
33.2 Explicit Conversions.....	178
<b>Chapter 34: Exceptions</b> .....	<b>183</b>
<b>Chapter 35: Smart Pointers</b> .....	<b>189</b>
35.1 Unique Pointer .....	189
35.2 Shared Pointer .....	191
<b>Chapter 36: Exercises</b> .....	<b>193</b>
36.1 static_cast Conversion.....	193
36.2 A Simple Unique Pointer:.....	194
36.3 Unique Pointer to an Object of a Class.....	194
36.4 Shared Pointers Exercise .....	195
36.5 Simple Polymorphism .....	195
36.6 Polymorphism II .....	196
36.7 Exception Handling .....	198
36.8 Multiple Exceptions.....	198
<b>Chapter 37: Input/Output Streams</b> .....	<b>201</b>
37.1 File Streams.....	201
37.2 String Streams .....	204
<b>Chapter 38: C++ Standard Library and Friends</b> .....	<b>209</b>
38.1 Containers.....	209
38.1.1 std::vector.....	210
38.1.2 std::array .....	212
38.1.3 std::set.....	212
38.1.4 std::map .....	213
38.1.5 std::pair .....	216
38.1.6 Other Containers.....	217
38.2 The Range-Based for Loop.....	217
38.3 Iterators.....	219

TABLE OF CONTENTS

- 38.4 Algorithms and Utilities ..... 220
  - 38.4.1 `std::sort` ..... 221
  - 38.4.2 `std::find` ..... 222
  - 38.4.3 `std::copy` ..... 224
  - 38.4.4 Min and Max Elements ..... 225
- 38.5 Lambda Expressions ..... 226
- Chapter 39: Exercises ..... 233**
  - 39.1 Basic Vector ..... 233
  - 39.2 Deleting a Single Value ..... 233
  - 39.3 Deleting a Range of Elements ..... 234
  - 39.4 Finding Elements in a Vector ..... 235
  - 39.5 Basic Set ..... 236
  - 39.6 Set Data Manipulation ..... 236
  - 39.7 Set Member Functions ..... 237
  - 39.8 Search for Data in a Set ..... 237
  - 39.9 Basic Map ..... 238
  - 39.10 Inserting Into Map ..... 239
  - 39.11 Searching and Deleting From a Map ..... 240
  - 39.12 Lambda Expressions ..... 241
- Chapter 40: C++ Standards ..... 243**
  - 40.1 C++11 ..... 243
    - 40.1.1 Automatic Type Deduction ..... 244
    - 40.1.2 Range-based Loops ..... 244
    - 40.1.3 Initializer Lists ..... 245
    - 40.1.4 Move Semantics ..... 245
    - 40.1.5 Lambda Expressions ..... 246
    - 40.1.6 The `constexpr` Specifier ..... 247
    - 40.1.7 Scoped Enumerators ..... 247
    - 40.1.8 Smart Pointers ..... 248
    - 40.1.9 `std::unordered_set` ..... 249

40.1.10	<code>std::unordered_map</code> .....	250
40.1.11	<code>std::tuple</code> .....	252
40.1.12	<code>static_assert</code> .....	253
40.1.13	Introduction to Concurrency .....	254
40.1.14	Deleted and Defaulted Functions .....	259
40.1.15	Type Aliases .....	262
40.2	C++14 .....	262
40.2.1	Binary Literals .....	263
40.2.2	Digits Separators .....	264
40.2.3	Auto for Functions .....	264
40.2.4	Generic Lambdas .....	265
40.2.5	<code>std::make_unique</code> .....	265
40.3	C++17 .....	266
40.3.1	Nested Namespaces .....	266
40.3.2	Constexpr Lambdas .....	267
40.3.3	Structured Bindings .....	267
40.3.4	<code>std::filesystem</code> .....	269
40.3.5	<code>std::string_view</code> .....	272
40.3.6	<code>std::any</code> .....	273
40.3.7	<code>std::variant</code> .....	275
40.4	C++20 .....	278
40.4.1	Modules .....	278
40.4.2	Concepts .....	281
40.4.3	Lambda Templates .....	285
40.4.4	[likely] and [unlikely] Attributes .....	286
40.4.5	Ranges .....	287
40.4.6	Coroutines .....	291
40.4.7	<code>std::span</code> .....	291
40.4.8	Mathematical Constants .....	292

TABLE OF CONTENTS

**Summary and Advice ..... 295**

- The go-to Reference ..... 295
- StackOverflow ..... 295
- Other Online Resources ..... 296
- Other C++ Books ..... 296
- Advice ..... 296

**Index..... 297**

# 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](http://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](https://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!

## 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.

## 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
```

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 .

## 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;
}
```

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;
```