Introduction to C++

Ryan Baker

January 3, 2025

Contents

1 Course Introduction

1.1 Overview of Lecture Series

2 Features of C++

2.1 Evolution of C++

C++ was developed in 1979 by Bjarne Stroustrup as a simple extension of C. Since then, it has evolved into a modern multi-paradigm language with major updates every three years (C++26 is on its way). Each update introduces features for better performance, safety, flexibility, and developer experience.

2.2 The C++ Philosophy

C++ is a sharp tool. It prioritizes manual control over all else, allowing for direct memory manipulation and fine-grained resource management. The philosophy is to give the developer the tools for flexibility and performance, but with the responsibility to manage complexity.

2.3 C++ vs. Other Languages

Compiled vs. Interpreted C++ is a *compiled* language, meaning the source code is translated into machine code before it is executed. This allows for a faster run time and more control over hardware aspects. This contrasts with *interpreted* languages, like Python, translated and executed line by line. Interpreted languages tend to be quicker to develop and easier to use.

Strongly Typed Strongly typed languages, such as C++, require the explicit specification of datatypes and enforce type assignments at compile and run time. This makes it harder to make type mistakes and promotes code stability.

Multi-Paridigm A programming paradigm is a high-level way to structure and conceptualize your program. Some languages enforce a certain paradigm, such as procedural programming or object-oriented programming, but not C++. C++ supports multiple programming paradigms, awarding more freedom to the programmer.

3 Environment Setup

3.1 Tools Required

To develop C++, you need two basic tools: a **text editor** and a **compiler**.

3.1.1 Text Editor

- What is a text editor? A tool at edits text. `// duh`
- **Text Editor vs. IDE:** A text editor is a basic tool for writing plain text, while an IDE (Integrated Development Environment) is a more comprehensive tool that includes a code editor, debugging tools, code completion, an build automation.
- Some popular text editors include:
  - **Visual Studio Code:** A free, open-source IDE with support for C++ through extensions.
  - **CLion:** An IDE specifically built for C++ with advanced features.
  - **Vim/Neovim:** My personal choice. Has a steep learning curve, but absolutely worth it.
    - If you decide to go with Vim or Neovim, I recommend spending some time configuring your setup. Feel free to ask me for help.

3.1.2 Compiler

- **Definition:** The compiler converts your C++ source code into machine-readable instructions.
- Common C++ compilers include:
  - **gcc:** The GNU Compiler Collection, a popular open-source compiler. Best for Windows OS and Linux.
  - **clang:** My personal favorite, known for its performance and diagnostics. Best for Mac OS.
  - **MSVC:** An increasingly irrelevant piece of garbage. Second best for Windows.
- Throughout the lecture series, I will be using `clang`. If a certain `clang` flag or directive does not work for your compiler, simply look up its equivalent.

3.2 "Hello, World!" Example

With your text editor of choice, write the following C++ program:

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

Compile and run the program with your compiler of choice. You should see "Hello, World!" printed to the console.

4 Basic Syntax and Structure

4.1 Basic Structure of a C++ Program

4.1.1 `int main()`

- **Entry Point:** The `main()` function is where every C++ program starts executing. It serves as the "entry point" for a program.
- **Return Value:** `main()` returns an integer to indicate the program's exit status. By convention, returning `0` means successful execution.
  - In modern C++, `return 0` is implicit

```
int main() {} // the shortest complete C++ program
```

4.2 Foundational Concepts

4.2.1 Semicolons, `/* comments */`, and Whitespace

- **Semicolons:** Every statement in C++ ends with a semicolon:

  ```
  std::cout << "Hello, World!" << std::endl;
  ```

  - This lets the compiler know that the line is finished
- **Comments:** Comments are a way of "taking notes" within your code. They are ignored by the compiler entirely, but help you and other developers understand the code.
- There are two ways of writing comments:
  - **Single Line:** Single line comments are prefixed with `//`:

    ```
    int main() // main function serves as the entry point
    ```

  - **Block:** Block comments are written with `/* */` notation:

    ```
    int /* why is there a comment here? */ main()
    ```

  - *Ryan's Advice:* Lightly prefer `//` to `/* */` because the parsing of `/* */` is more complicated and can lead to errors if you aren't careful.
- **Whitespace:** C++ could not care less about whitespace. Whitespace includes spaces, tabs, and newlines.
  - This means that is its possible, although not generally recommended, to write a C++ program in one line of code:

```
int main() { std::cout << "Hello, World!" << std::endl; }
```

4.2.2 Line-by-Line Execution

In C++, the program executes statement sequentially, starting from the top of `main()` and moving downward.

```
int main()
{
    std::cout << "First"  << std::endl;  // guaranteed
    std::cout << "Second" << std::endl;  // to print
    std::cout << "Third"  << std::endl;  // in order
}
```

4.3 Input and Output

5 Datatypes and Variables

When we're programming, all we're really

5.1 Primitive Types

5.1.1 `int`, `char`, `bool`, `float`, `void`

`int`: Used to store integer values (e.g., 5, -10, 42)

- Integers can be *signed* (represent ±) or *unsigned* (only positive).
-
- Because a computer's memory is finite, so too is the range of an `int`.
  - The maximum value of an unsigned `int` can be calculated as 2^w where w is the width of the `int` in bits. The minimum is 0.
    - The range of a `uint32_t` is $[0, 2^{32} - 1] = [0, 4294967295]$
  - For signed integers, the maximum value is $2^{w-1} - 1$ and the minimum value is -2^{w-1} where w is the width in bits.
    - The range of an `int32_t` is $[-2147483648, 2147483647]$

`char`: Used to store single characters (e.g., 'a', '2')

- Characters are just integers in disguise. Every character has a corresponding integer value according to the **ASCII Table**.

```
std::cout << int('a') << std::endl; // 97
```

```
std::cout << char(97) << std::endl; // 'a'
```

`bool`:

`float`:

`void`:

5.1.2 `sizeof` Operator

5.2 Declaration and Definition

5.2.1 Assignment Operator `=`

5.2.2 Brace Initialization `{}`

5.3 Arithmetic Operators