C++ Crash Course: Hello, World!

Nick
April 13, 2019

Introduction

The first program many people write is a simple *Hello, world!* one. This series will be no different. However, this series is intended to be a crash course, so we will focus more on understanding the functionality of C++, rather than the syntax.

In this tutorial we will discuss the following:

- The Compiler
- Comments
- Including Files
- Namespaces
- Main Functions
- Printing Using a Stream Object
- Return Statements

Code Overview

Source Code 1: Example hello world program in C++ called hello_world.cpp

```cpp
// Prints "Hello, world!"
// By: Nick from CoffeeBeforeArch
#include <iostream>
using namespace std;
int main(){
    cout << "Hello, world!\n";
    return 0;
}
```

0.1 The Compiler

The CPU does not know how to read C++ code directly. In order to transform our code,shown in Listing 1, into the instructions that a CPU can understand, we use a compiler.

Compilers can be thought of as translators and optimizers. Each CPU has a set of instructions that can be used to perform arithmetic, load/store values to memory, and jump/branch to other parts of code.

Compilers take high level code, in our case C++, and re-write it using those instructions. Compilers will also optimize the code (to the best of its ability) during this translation.

While we typically call the creation of an executable from C++ code compilation, there are more underlying processes going on behind the scenes. These include:

- Pre-processing
- Compilation and Assembly
- Linking

These will be discussed in greater detail in another tutorial. For now, and in most cases, we mean creating an executable from C++ code when we say compilation.

For our example (assuming a GNU+Linux system), we compile our code using **g++**. Pre-processing, compilation and assembly, and linking are all done when we do the command:

g++ -o hello_world hello_world.cpp

Here, we call g++, tell it to create an executable named hello_world with **-o hello_world**, and give it our input file **hello_world.cpp**. Alternatively, we could have specified the output file after the input file, and the compilation would have worked the same way (as long as we move the -o as well).

0.2 Comments

Every good program should have information about what it does, and who wrote it (at the very least). Comments can be done in two ways:

- Using the // characters
- Using the /* and */ characters

Anything on the same line and after a // will be ignored by the compiler. Anything in between /* and */ will be ignored by the compiler as well.

You might use // for a comment on a single line You might use /* and */ to surround comments that will span multiple lines. This avoids needing to type // on each line.

Our original comments could be transformed to listing 2, which is treated by the compiler the same way.

Source Code 2: Example of using block comments in C++

```
/*
Prints "Hello, world!"
By: Nick from CoffeeBeforeArch
*/
```

0.3 Including files

We don't want to re-invent the wheel every time we write a program. A lot of commonly implemented functions/classes/etc. can be found in the C++ Standard Library. We can gain access to the standard library using **include<someFile>**.

Angle brackets are used to tell the compiler that the file we are looking for is located in a standard list of system directories it knows where to find. When we include the file, you can think of it as being copy-and-pasted right where the include statement is. We can also include files that are not in the standard system directories by replacing the angle brackets with quotes. An example of this would be **include "path/to/a/file"**.

For our hello world program, we are including from **iostream** (part of the C++ Standard Library). This gives us access to various input and output stream objects.

0.4 Namespaces

Namespaces are a convenient tool that allow us to group named things, like functions and objects, into a specific scope. In the case of our example, the **using namespace std;** directive tells the program that the following code will be using functions and objects from the **std** namespace (where the C++ Standard Library is defined). If we did not include this line, we would need to tell the compiler which namespace to look for at each use of something from the standard library.

As an example, **cout**, a stream object used in the program, would have to be used as **std::cout**. The **::** is the scope resolution operator. This is used to tell the compiler to use the cout stream object found in the namespace std.

0.5 Main Functions

Every C++ program must have a main function. This serves as the entry point for execution (where program begins when you run an executable). The main function has two standard signatures (ways that it can be defined). You can use **int main()**, as used in our example, or **int main(int argc, char *argv[ ])**.

We use the first when we are not passing in arguments to the program when we launch the executable, and the second when we are. We will discuss more about function signatures and passing command line arguments in other tutorials.

0.6 Printing Using a Stream Object

In order to print out something to the console, we use the cout stream object we included from iostream. To print, we use **cout** <<. Almost anything following the angle brackets will be printed to the console (there are some limitations that we will discuss later). In our example, we print a string, which is defined within quotation marks (note, the \n is a newline character, and will place your console cursor on the next line after printing hello world).

0.7 Return

The signature of main function we are using is defined as **int main()**, where **int**, in this context, means the main function will return an integer (whole number). We typically return 0 for normal program exit from a main function. We can return non-0 codes when we want to exit abnormally, but there is no standard as to how those are interpreted.

1 Conclusion

And that's it! From this example you should understand from a high level the reasoning behind each part of a C++ hello world program. In later tutorials, we will discuss more about the language, and how to implement some more interesting programs.