Code Explanation

This lesson explains in detail the Hello World code written in the last lesson. It explains the basic syntax used in almost every C++ code

We'll cover the following

Detailed Explanation of the Code

Let's look at the **Hello World!** program we made in the last lesson in detail.

We'll start from the very first line and go step by step!

#include

It is written as:

```
#include <iostream>
```

The *hash sign* **(#)** signifies the start of a preprocessor command. The `#include` command is a specific preprocessor command that effectively copies and pastes the entire text of the file specified between the angle brackets into the source code. In this case, the file is **"iostream"** which is a standard file that should come with the C++ compiler. This file name is short for **"input-output streams"**; in short, it contains code for displaying and getting the text from the user.

The *include* statement allows a programmer to **"include"** this functionality in the program without having to literally cut and paste it into the source code every time. The *iostream* file is part of the C++ standard library, which provides a set of useful and commonly used functionality provided with the compiler. The **“include”** mechanism, however, can be used both for standard code provided by the compiler and for reusable files created by the programmer.

using namespace std

It is written as:

```
using namespace std;
```

C++ supports the concept of namespaces. A *namespace* is essentially a prefix that is applied to all the names in a certain set. One way to think about namespaces is that they are like toolboxes with different useful tools. The using command tells the compiler to allow all the names in the **“std”** namespace to be usable without their prefix. The iostream file defines *three* names used in this program - **cout**, **cin**, and **endl** - which are all defined in the *std namespace*. **“std”** is short for **“standard”** since these names are defined in the standard C++ library that comes with the compiler.

Without using the *std namespace*, the names would have to include the prefix and be written as `std::cout`, `std::cin`, and `std::endl`. If we continue with the toolbox example, this code would be saying, **"Use the cout, cin and endl tools from the std toolbox."**

> **Note:** You should either remember the fact that the iostream file uses the `std` namespace or look it up in the documentation for the **iostream** file because C++ does not make this connection for you explicitly.

A slight feeling of annoyance that you are forced to type this connection every time you wish to write a new C++ program is entirely normal, indeed justified; may I urge you to consider it a small price to pay for avoiding all the tedious work of constantly retyping **std::free** in front of things in your program?

int main()

`int main()` is called as:

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

The *starting* point of all C++ programs is the `main` function. This function is called by the operating system when your program is executed by the computer.

By *execution* we mean:

> Performing the actions specified by the statements in your program.

cin & cout

```
#include <iostream>
using namespace std;

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

The name `cout` is short for **“character output”** and `cin`, correspondingly, is an abbreviation for **“character input”**.

In a typical C++ program, most function calls are of the form `object.function_name(argument1, argument2)`.

Symbols such as `<<` can also behave like functions, as illustrated by the use of `cout` above. This capability is called operator overloading.

{ }

A block of code is defined with the `{ }` tokens. `{` signifies the start of a block of code, and `}` signifies the end.

> **NOTE:** The `{ }` tokens have other uses as well.

semicolons

Statements in C++ must be terminated with a **semicolon**,

- Just as sentences in English must be terminated with a period.
- Just as sentences in English can span several lines, so can the statements in C++.

In fact, you can use as many spaces and new lines between the words of a C++ program as you wish to beautify your code just as spaces are used to justify the text printed on the pages of a book.

return

The `return` statement is written as:

```
return 0; // any type can be returned depending on function type
```

The `return` keyword tells the program to *return* a value to the function `int main` that called this function and then to continue execution in the `int main` function from the point at which this function was called.

> **Note:** The **type** of the value returned by a function must match the **type** specified in the declaration of the function.

Executing the `return` keyword in the `main` function of a program *returns* a value and the execution control to the operating system component that launched this program, in effect, terminating the execution of this program.

Now that we have learned the basics of our code. Let's learn how to compile our code in C++ in the next lesson!