

# Fundamentals of C++ and more

By Isaiah Carrington

## Table Of Contents

Overview .....	3
Resources .....	3
Compiler .....	3
Sources .....	3
Section 1: basic concepts .....	4
1.1 Writing your first program .....	4
1.2 Headers and namespaces .....	6
1.3 New lines and endl .....	7
The endl manipulator .....	7
The new lines character .....	9
1.4 Comments .....	10
Single Line Comments .....	11
Multi-Line Comments .....	11
1.5 Introduction to variables .....	13
Creating a variable .....	13
Rules for variable names .....	13
Assigning a value to a variable .....	13

## Overview

Hey, Isaiah here. First off, I want to thank you for taking your time to read this e-book. A lot of effort has gone into it to get it to the point where it is now, and I am glad it was able to find its way into your hands device.

So as implied by the title of this book, I will be going over the fundamentals of the C++ programming language to equip you with the knowledge you need for any purpose you may need, whether it is just to pick up a new skill, pass a course, refresh your mind etc. This book will be all you will need to get started... or so I hope.

Not only will we be focusing on the fundamentals, but we will also look at semi-advance concepts such as object-oriented programming and recursion.

That brings me to the end of this overview. First section we will be looking at is basic concepts. I hope you find this helpful.

## Resources

### Compiler

In order to run a C++ program, it first has to be compiled, and to compile a C++ program, you need a C++ compiler. Different compilers may have different results, so throughout this book, I will stick with using this website ([https://www.onlinegdb.com/online\\_C++\\_compiler](https://www.onlinegdb.com/online_C++_compiler)). The biggest downside of this, is that this compiler is only available if you have internet. If you wish to download a compiler, then I recommend DevC++, as it is very easy to use or Visual Studio.

### Sources

This book is written using my accumulation of experience in learning C++ as a base, despite it not being my main language (python has that spot).

The biggest source that i will be using for this book is Sololearn's C++ course (<https://www.sololearn.com/learning/1051>), which is a resource I highly recommend.

## Section 1: basic concepts

Whether you are new to C++ or a returning veteran just looking to brush up, we all start at the basics.

In this section, the focus will be on, making your first program, headers and namespaces, new lines, comments, variables, auto, basic arithmetic and assignment and incrementation.

As mentioned in the introduction (resources), I will be making compilations and executions, making use of the website provided [here](#).

### 1.1 Writing your first program.

To start things off, you are going to be making your first program in C++, the very infamous “hello world”. The point of this program is so that you get a view of the most basic structure a C++ program should have, and how to display simple outputs to the screen.

Before we begin, first launch your compiler, whether you’re using the website I recommended or a different one.

Feel free to copy the code below and put it in your compiler (or type it in), then build and run it.

Don’t worry about understanding every line of the code for now, everything will be explained.

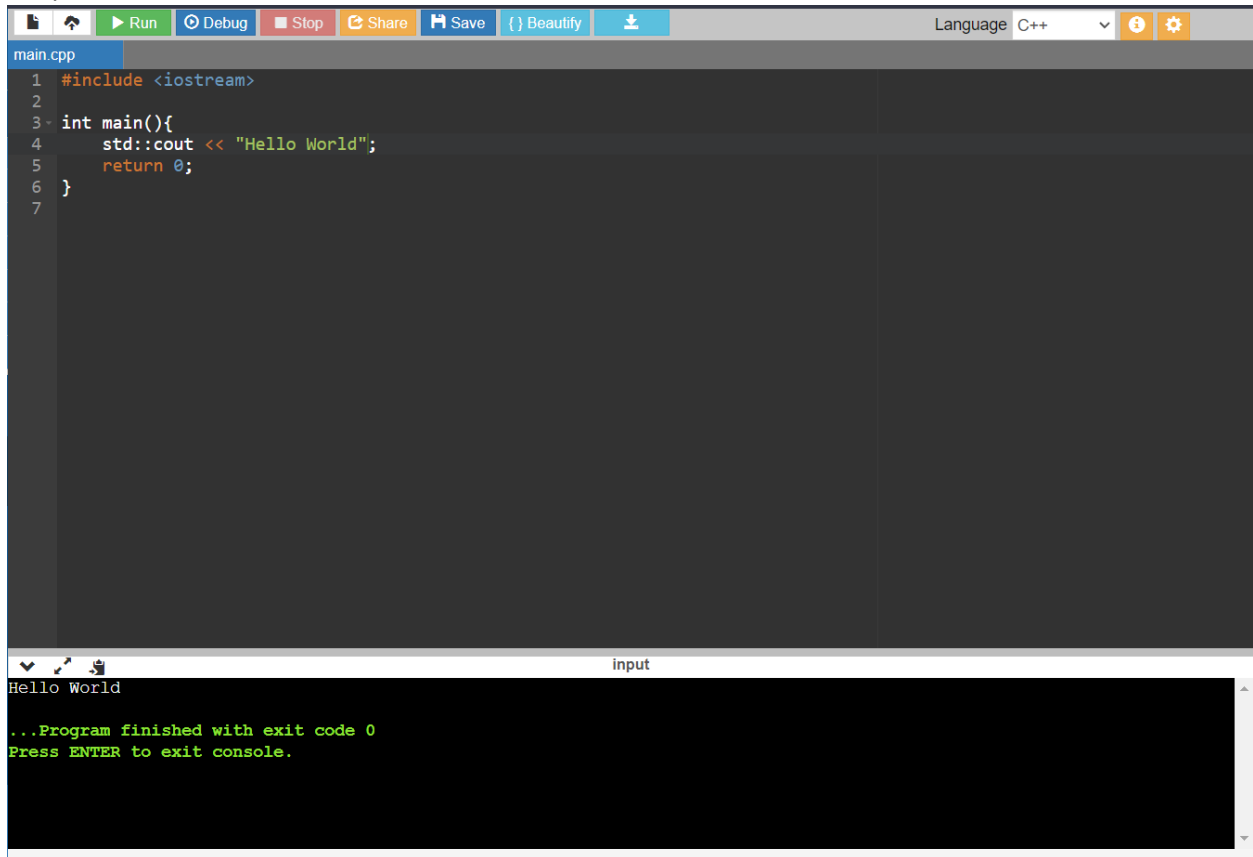
```
#include <iostream>

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

Note: if you receive an error when you copy and paste this code, make sure to replace the double quotes surrounding “hello world”, with double quotes that you type in.

After typing your code, simple hit “run” to run it. If you are using your own compiler, then there should be an option that says “build”. Use that instead

Output:



The image shows a screenshot of a C++ IDE. The top toolbar includes buttons for Run, Debug, Stop, Share, Save, and Beautify. The language is set to C++. The editor window shows the following code in main.cpp:

```
1 #include <iostream>
2
3 int main(){
4     std::cout << "Hello World";
5     return 0;
6 }
7
```

Below the editor is a console window titled 'input'. It displays the output of the program:

```
Hello World
...Program finished with exit code 0
Press ENTER to exit console.
```

Figure 1: your first program

Simple. Just like that, you have written and executed your first program.

Next, we'll look at what this code does.

## 1.2 Headers and namespaces

Headers can be thought of as files, which contain their own code that helps your programs work from behind the scenes.

We have already been introduced to the first header, that being the `<iostream>` header, which we would have used in our first “hello world” program.

`<iostream>`, stands for input output stream, and allows us to get input and output data.

The `#include` keyword is used for adding a built in (standard) or user-defined header files to our program.

A namespace is a declarative region that provides a scope to identifiers (names of elements) inside it.

An example of a namespace is `std`, which you may recognize from our “hello world” program: `std::cout << “hello world”`. What this line means, is that we want our compiler to use the `cout` name from the `std` (standard) namespace. Try removing the `std::` and see what happens when you try to run your code.

There is a different way of accessing the `cout` name without having to prefix it with `std::` everything, but I have intentionally avoided it for now, as it may lead to complications with larger program. I will introduce it a bit later however, as for smaller simpler programs, like the ones we will be doing, it will help speed things up.

### 1.3 New lines and endl

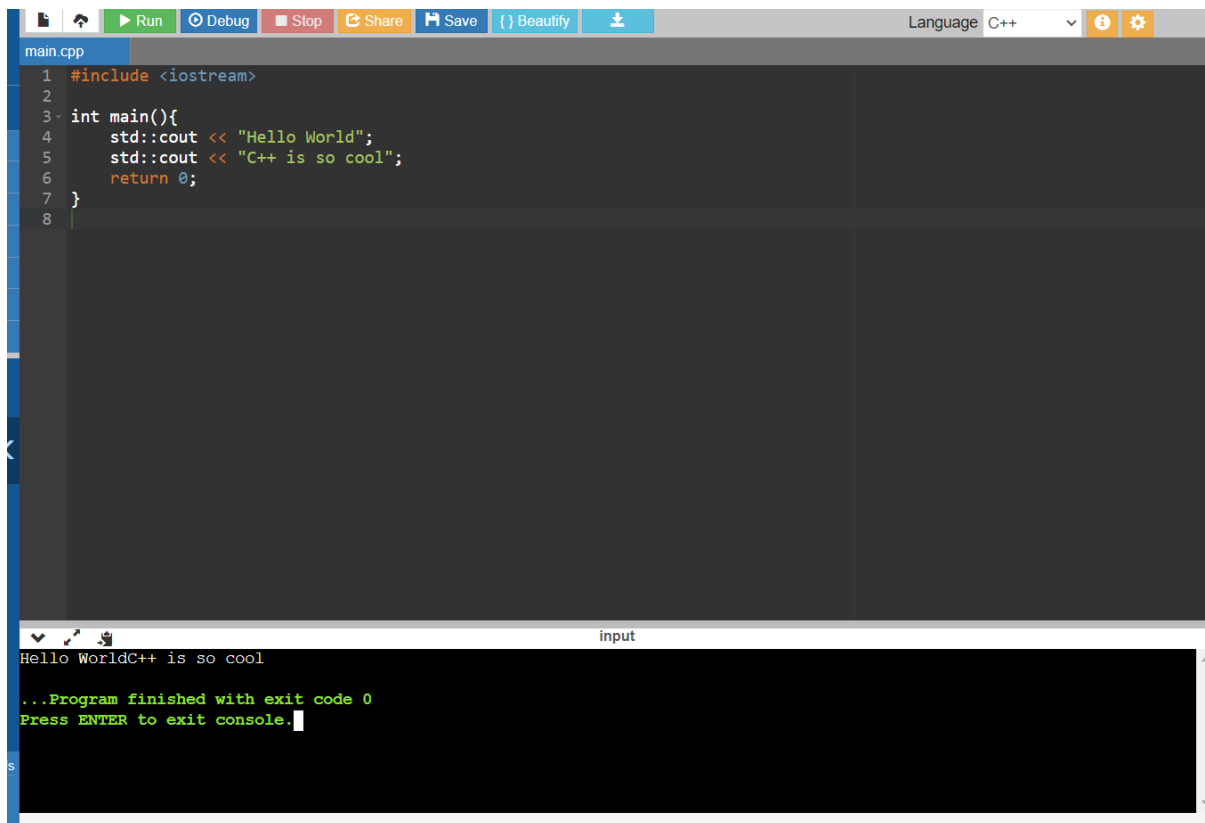
Try compiling and running the below code in your compiler.

```
#include <iostream>

int main(){
    std::cout << "Hello world";
    std::cout << "C++ is so cool";
    return 0;
}
```

*Don't forget your semicolons*

When running this code, you will realize you get the following output:

A screenshot of a C++ IDE. The top part shows a code editor with the following code:

```
1 #include <iostream>
2
3 int main(){
4     std::cout << "Hello World";
5     std::cout << "C++ is so cool";
6     return 0;
7 }
8
```

The bottom part shows a terminal window with the output:

```
input
Hello WorldC++ is so cool
...Program finished with exit code 0
Press ENTER to exit console.
```

Figure 2: Using multiple couts

Not quite the output you expected is it. Despite us having 2 **std::cout** statements on 2 different lines, the output still came out on one line!

This happens because **std::cout** does not automatically generate a new line at the end. So, if we want a new line, we'll have to manually add it ourselves. We will look at 2 ways of doing this.

#### The endl manipulator

The endl manipulator is what is known as a stream manipulator. That is, it is capable of manipulating (making changes to) our output stream.

By placing the endl at the end of our **cout** statement, we can add a new line! Let's try it!!

```
main.cpp
1 #include <iostream>
2
3 int main(){
4     std::cout << "Hello World" << std::endl;
5     std::cout << "C++ is so cool" << std::endl;
6     return 0;
7 }
8

input
Hello World
C++ is so cool

...Program finished with exit code 0
Press ENTER to exit console.
```

Figure 3: Hello World, but with endl new lines

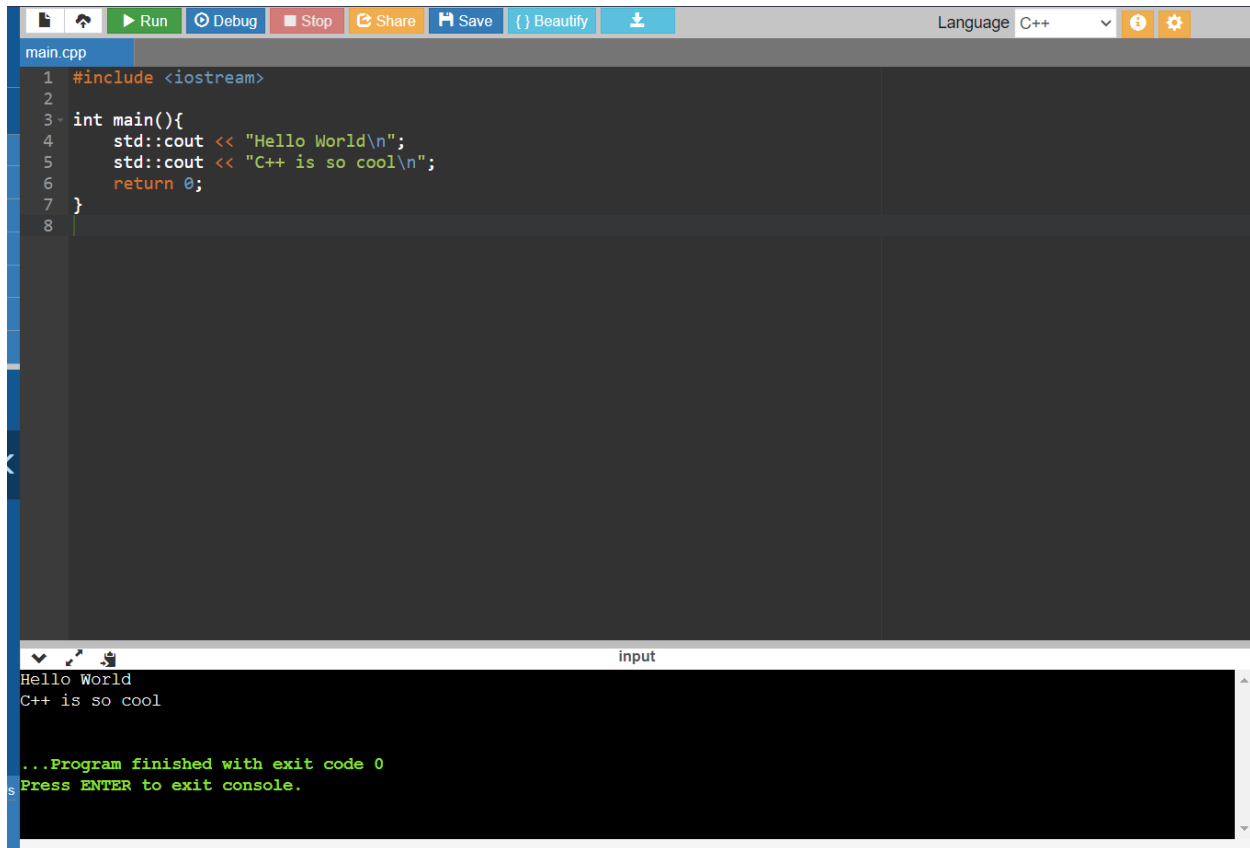
Note: **endl** is also from the standard (**std**) namespace, hence why we have it prefixed with **std::**

There we go, we made our first new line. But there is one other way we will look at now.



## The new lines character

There is a character that we can use to create new lines, and that is the `\n`. By placing this character inside of our string, we can create a new line. Example:



```
main.cpp
1 #include <iostream>
2
3 int main(){
4     std::cout << "Hello World\n";
5     std::cout << "C++ is so cool\n";
6     return 0;
7 }
8

input
Hello World
C++ is so cool

...Program finished with exit code 0
Press ENTER to exit console.
```

Figure 4: Hello world, making use of `\n`

`\n` is an escape character, that is, a character that has its own meaning. We will be looking at these more later. For now, we have learnt how to add new lines to our output!

## 1.4 Comments

In our C++ programs, there may come a time where you would like to explain what a section of your code does. However, you will realize that if you just tried to leave a message in your code, it may cause an error. For example:



```
main.cpp
1  #include <iostream>
2
3  this code prints out hello world
4
5  int main(){
6      std::cout << "Hello World\n";
7      std::cout << "C++ is so cool\n";
8      return 0;
9  }
10
```

input

Compilation failed due to following error(s).

```
3 | this code prints out hello world
  | ^~~~~
```

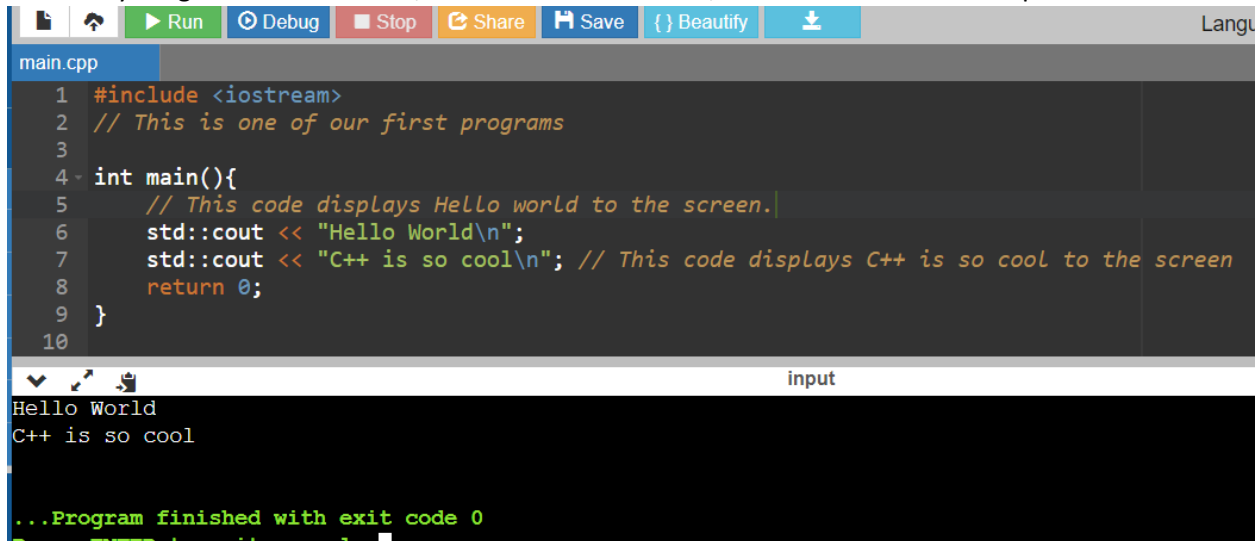
Figure 5: Trying to leave a description of our code

Luckily, this is where comments come in. Comments allow us to leave messages in our code, that are only for whoever may be reading our code, our future selves included. This means that the compiler, will ignore comments, so there's no need to worry about any errors or unexpected code being executed. This makes comments useful for not just leaving explanations for future readers of our code, but also for stopping execution of a certain section of code.

There are 2 types of comments we will be looking at, namely, single-line and multi-line.

## Single Line Comments

These are comments that span a single line. To make this comment, we make use of two slashes (`//`) like this. Everything after the 2 slashes, until the end of the line, will be a comment. Example:



```
main.cpp
1 #include <iostream>
2 // This is one of our first programs
3
4 int main(){
5     // This code displays Hello world to the screen.
6     std::cout << "Hello World\n";
7     std::cout << "C++ is so cool\n"; // This code displays C++ is so cool to the screen
8     return 0;
9 }
10
```

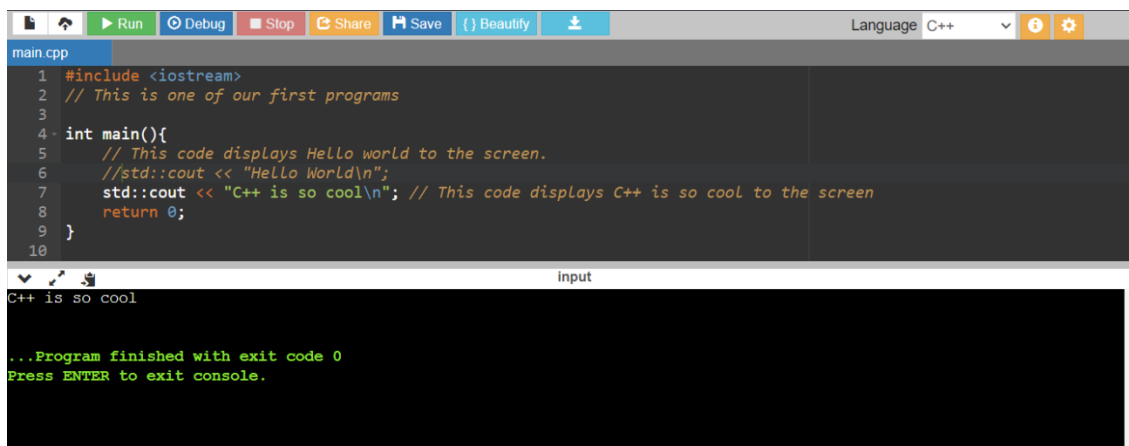
input

```
Hello World
C++ is so cool

...Program finished with exit code 0
Press ENTER to exit console.
```

Figure 6: Using single line comments in our program

Here we will use another example, of us commenting out a line of code.



```
main.cpp
1 #include <iostream>
2 // This is one of our first programs
3
4 int main(){
5     // This code displays Hello world to the screen.
6     //std::cout << "Hello World\n";
7     std::cout << "C++ is so cool\n"; // This code displays C++ is so cool to the screen
8     return 0;
9 }
10
```

input

```
C++ is so cool

...Program finished with exit code 0
Press ENTER to exit console.
```

Figure 7: Commenting out a line of code to stop it from executing

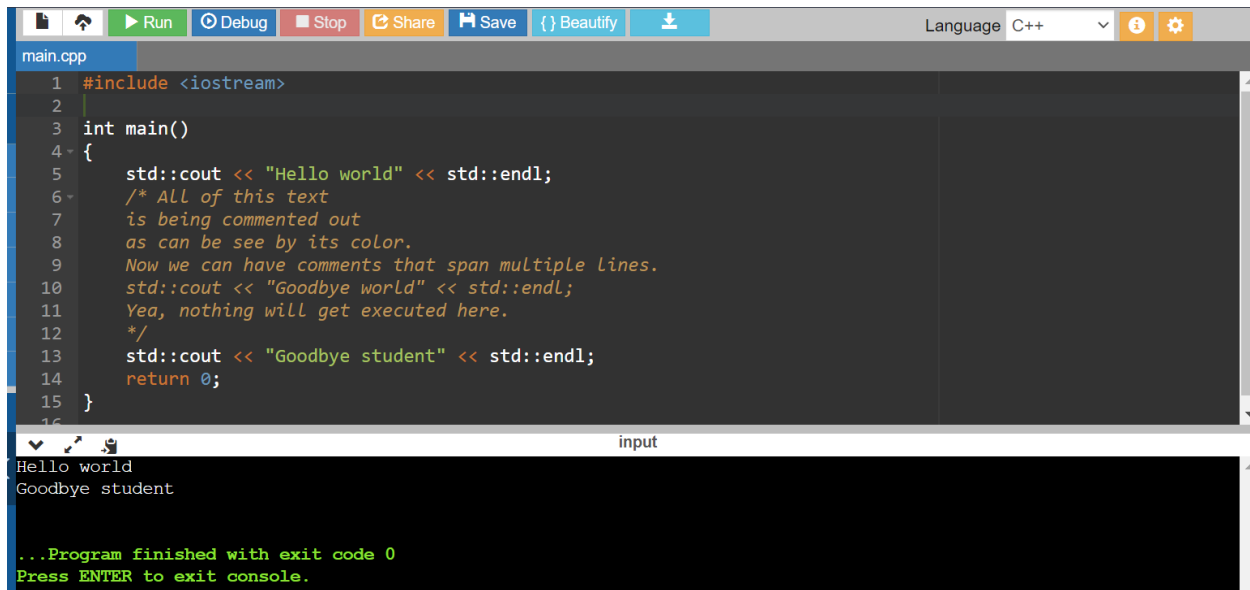
Here we make use of a comment, to comment out a line of code. Notice how the *Hello World* is no longer being displayed.

## Multi-Line Comments

Single line comments are useful, but their limited in the fact that they only span a single line. What if we wanted to comment out several lines at once? Will we have to put the `//` before every line?

Well, that is one way of doing it. However, C++ provides us with a different method, known as the multi-line comment, which makes use of the `/*` and `*/` symbols.

Everything that comes between these 2 symbols, will be commented out. Example:



```
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << "Hello world" << std::endl;
6     /* All of this text
7     is being commented out
8     as can be see by its color.
9     Now we can have comments that span multiple lines.
10    std::cout << "Goodbye world" << std::endl;
11    Yea, nothing will get executed here.
12    */
13    std::cout << "Goodbye student" << std::endl;
14    return 0;
15 }
```

input

```
Hello world
Goodbye student

...Program finished with exit code 0
Press ENTER to exit console.
```

Figure 8: Using Multi-line Comments

## 1.5 Introduction to variables

Variables are sections of a computer's memory that our program reserves for storing data, that we can then get access to by name. If you have also read my book "Student's Guide to Programming" (I highly recommend it if you haven't), then you already know what I am going to say. However, for those of you who haven't, I will continue.

I like to think of variables as containers that we can store any item we want inside of, put a label on it, and then retrieve and make changes to it however we want. At its most basic, this is essentially what variables are. These containers however, only us to store one value at a time. So, if we want to store a different value in that same variable, then we will have to remove the value that is inside of it first. Luckily C++ takes care of the removal part for us.

### Creating a variable

C++ is a static typed language. That is, in order for us to use a variable, we first have to explicitly declare it by giving it a name, and a data type. Also, once we give a data type to a variable, the variable may only hold data of that data type. This is so that C++ knows how much memory to reserve for the variable.

To declare a variable, we use the following statement:

***datatype variable\_name;***

Where datatype is the type of data the variable is supposed contain, and variable\_name is the name of the variable.

### Rules for variable names

When naming a variable, there are naming rules that we must follow. These rules are:

1. Variable names may not contain spaces.
2. Variable names may not begin with a number, but can contain one elsewhere in its name
3. Variable names must not contain symbols (@!\$&{}\*+, etc.)
4. Variable names are normally written in camelCase. Example: myRealNumber
5. Variable names are case sensitive. That is, myVar, MyVar and myvar, refer to 3 different variables.
6. Variable names SHOULD be clear and easy to tell what its purpose is. This is mainly for readability purposes

### Assigning a value to a variable

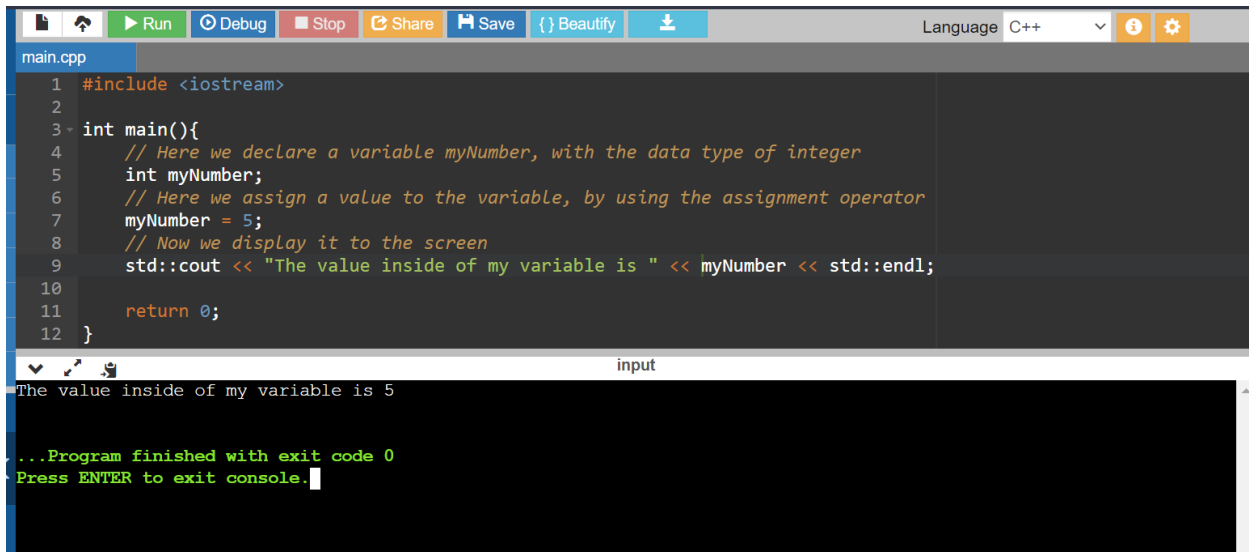
Giving a variable a value, is referred to as assignment. When we give our variable a value for the first time, this is known as initialization. C++ provides us several methods to assign values to variables, some of which I have listed below.

#### *The assignment operator*

The most basic method is the assignment operator (=). This is the simplest and most basic form to use.

***variableName = value;***

Let's look at a brief code example of this method:



```
1 #include <iostream>
2
3 int main(){
4     // Here we declare a variable myNumber, with the data type of integer
5     int myNumber;
6     // Here we assign a value to the variable, by using the assignment operator
7     myNumber = 5;
8     // Now we display it to the screen
9     std::cout << "The value inside of my variable is " << myNumber << std::endl;
10
11     return 0;
12 }
```

input

```
The value inside of my variable is 5

...Program finished with exit code 0
Press ENTER to exit console.
```

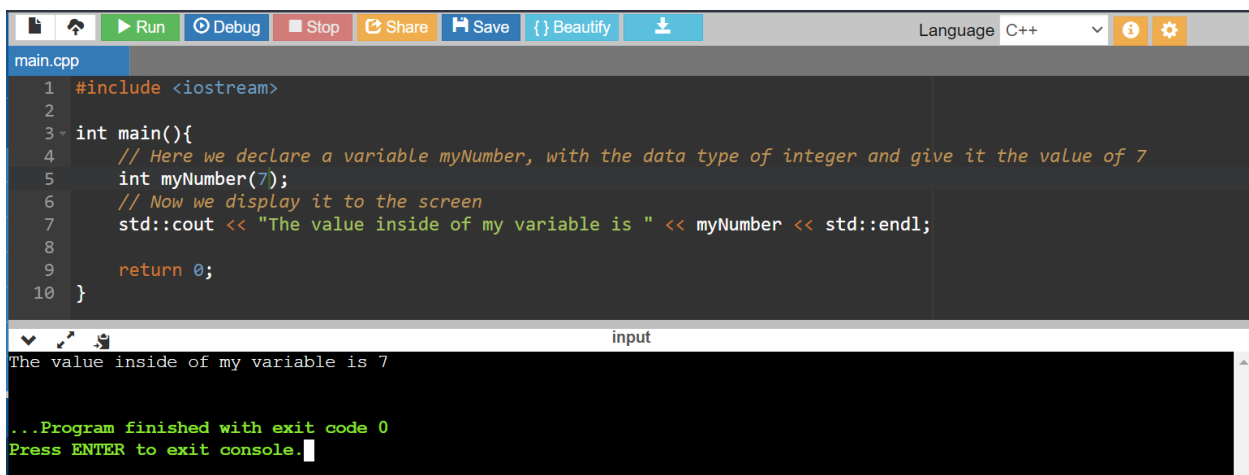
Figure 9: Assigning a value to a variable using the assignment operator

### Using Direct Initialization.

We can use brackets to *initialize* our variable using the following format:

***datatype variableName (value);***

Now for an example of this in code:



```
1 #include <iostream>
2
3 int main(){
4     // Here we declare a variable myNumber, with the data type of integer and give it the value of 7
5     int myNumber(7);
6     // Now we display it to the screen
7     std::cout << "The value inside of my variable is " << myNumber << std::endl;
8
9     return 0;
10 }
```

input

```
The value inside of my variable is 7

...Program finished with exit code 0
Press ENTER to exit console.
```

Figure 10: Initializing a variable using Direct Initialization.