C++ Introduction

What is C++?

C++ is a cross-platform language that can be used to create high-performance applications.

C++ was developed by Bjarne Stroustrup, as an extension to the C language.

C++ gives programmers a high level of control over system resources and memory.

Why Use C++

C++ is one of the world's most popular programming languages.

C++ can be found in today's operating systems, Graphical User Interfaces, and embedded systems.

C++ is an object-oriented programming language which gives a clear structure to programs and allows code to be reused, lowering development costs.

C++ is portable and can be used to develop applications that can be adapted to multiple platforms.

C++ is fun and easy to learn!

As C++ is close to C, C# and Java, it makes it easy for programmers to switch to C++ or vice versa.

Difference between C and C++

C++ was developed as an extension of C, and both languages have almost the same syntax.

The main difference between C and C++ is that C++ support classes and objects, while C does not.

C++ Getting Started

To start using C++, you need two things:

- A text editor, like Notepad, to write C++ code
- A compiler, like GCC, to translate the C++ code into a language that the computer will understand

C++ Quickstart

Let's create our first C++ file.

Open the Compiler and go to **File > New > Empty File**.

Write the following C++ code and save the file as `myfirstprogram.cpp` (**File > Save File as**):

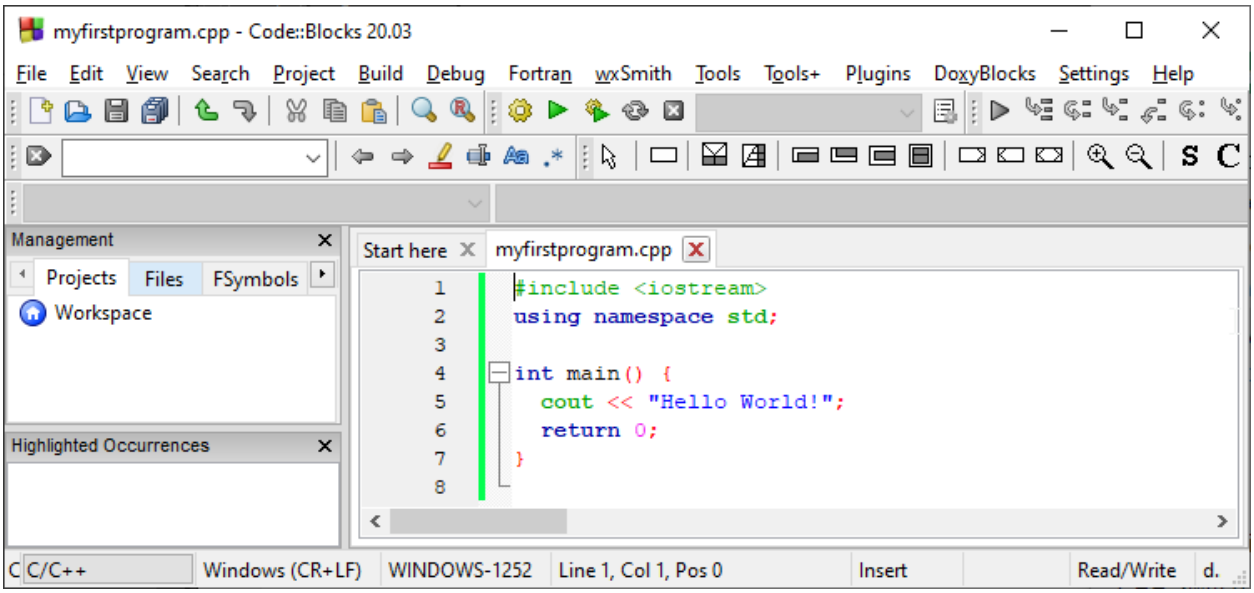
myfirstprogram.cpp

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

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

Don't worry if you don't understand the code above - we will discuss it in detail in later chapters. For now, focus on how to run the code.

In Codeblocks, it should look like this:



Then, go to **Build > Build and Run** to run (execute) the program. The result will look something to this:

```
Hello World!
Process returned 0 (0x0) execution time : 0.011 s
Press any key to continue.
```

Congratulations! You have now written and executed your first C++ program.

C++ Syntax

Let's break up the following code to understand it better:

Example

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


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

Example explained

Line 1: `#include <iostream>` is a **header file library** that lets us work with input and output objects, such as `cout` (used in line 5). Header files add functionality to C++ programs.

Line 2: `using namespace std` means that we can use names for objects and variables from the standard library.

Don't worry if you don't understand how `#include <iostream>` and `using namespace std` works. Just think of it as something that (almost) always appears in your program.

Line 3: A blank line. C++ ignores white space. But we use it to make the code more readable.

Line 4: Another thing that always appear in a C++ program, is `int main()`. This is called a **function**. Any code inside its curly brackets `{}` will be executed.

Line 5: `cout` (pronounced "see-out") is an **object** used together with the *insertion operator* (`<<`) to output/print text. In our example it will output "Hello World!".

Note: Every C++ statement ends with a semicolon `;`.

Note: The body of `int main()` could also been written as:
`int main () { cout << "Hello World! "; return 0; }`

Remember: The compiler ignores white spaces. However, multiple lines makes the code more readable.

Line 6: `return 0` ends the main function.

Line 7: Do not forget to add the closing curly bracket `}` to actually end the main function.

Omitting Namespace

You might see some C++ programs that runs without the standard namespace library. The `using namespace std` line can be omitted and replaced with the `std` keyword, followed by the `::` operator for some objects:

Example

```
#include <iostream>

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

C++ Output (Print Text)

The `cout` object, together with the `<<` operator, is used to output values/print text:

Example

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

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

You can add as many `cout` objects as you want. However, note that it does not insert a new line at the end of the output:

Example

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

int main() {
  cout << "Hello World!";
  cout << "I am learning C++";
  return 0;
}
```

New Lines

To insert a new line, you can use the `\n` character:

Example

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

int main() {
  cout << "Hello World! \n";
  cout << "I am learning C++";
  return 0;
}
```

The output will be as the following:

```
Hello World!
I am learning C++
```

Another way to insert a new line, is with the `endl` manipulator:

Example

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

int main() {
  cout << "Hello World!" << endl;
  cout << "I am learning C++";
  return 0;
}
```

The output will be as the following:

```
Hello World!
I am learning C++
```

C++ Comments

Comments can be used to explain C++ code, and to make it more readable. It can also be used to prevent execution when testing alternative code. Comments can be singled-lined or multi-lined.

Single-line Comments

Single-line comments start with two forward slashes (`//`).

Any text between `//` and the end of the line is ignored by the compiler (will not be executed).

This example uses a single-line comment before a line of code:

Example

```
// This is a comment
cout << "Hello World!";
```

This example uses a single-line comment at the end of a line of code:

Example

```
cout << "Hello World!"; // This is a comment
```

C++ Multi-line Comments

Multi-line comments start with `/*` and ends with `*/`.

Any text between `/*` and `*/` will be ignored by the compiler:

Example

```
/* The code below will print the words Hello World!
to the screen, and it is amazing */
cout << "Hello World!";
```

Single or multi-line comments?

It is up to you which you want to use. Normally, we use `//` for short comments, and `/* */` for longer.

C++ Variables

Variables are containers for storing data values.

In C++, there are different **types** of variables (defined with different keywords), for example:

- `int` - stores integers (whole numbers), without decimals, such as 123 or -123
- `double` - stores floating point numbers, with decimals, such as 19.99 or -19.99
- `char` - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- `string` - stores text, such as "Hello World". String values are surrounded by double quotes
- `bool` - stores values with two states: true or false

Declaring (Creating) Variables

To create a variable, specify the type and assign it a value:

Syntax

```
type variableName = value;
```

Where *type* is one of C++ types (such as `int`), and *variableName* is the name of the variable (such as **x** or **myName**). The **equal sign** is used to assign values to the variable.

To create a variable that should store a number, look at the following example:

Example

Create a variable called **myNum** of type `int` and assign it the value **15**:

```
int myNum = 15;
cout << myNum;
```

You can also declare a variable without assigning the value, and assign the value later:

Example

```
int myNum;
myNum = 15;
cout << myNum;
```

Note that if you assign a new value to an existing variable, it will overwrite the previous value:

Example

```
int myNum = 15;  // myNum is 15
myNum = 10;  // Now myNum is 10
cout << myNum;  // Outputs 10
```

The general rules for naming variables are:

- Names can contain letters, digits and underscores
- Names must begin with a letter or an underscore (_)
- Names are case sensitive (`myVar` and `myvar` are different variables)
- Names cannot contain whitespaces or special characters like !, #, %, etc.
- Reserved words (like C++ keywords, such as `int`) cannot be used as names

Other Variable Types

A demonstration of other data types:

Example

```
int myNum = 5;               // Integer (whole number without decimals)
double myFloatNum = 5.99;    // Floating point number (with decimals)
char myLetter = 'D';         // Character
string myText = "Hello";     // String (text)
bool myBoolean = true;       // Boolean (true or false)
```

Display Variables

The `cout` object is used together with the `<<` operator to display variables.

To combine both text and a variable, separate them with the `<<` operator:

Example

```
int myAge = 35;
cout << "I am " << myAge << " years old.";
```

Add Variables Together

To add a variable to another variable, you can use the `+` operator:

Example

```
int x = 5;
int y = 6;
int sum = x + y;
cout << sum;
```

Exercise:

Create a variable named `myNum` and assign the value `50` to it ?

```
    =
```

Declare Many Variables

To declare more than one variable of the **same type**, use a comma-separated list:

Example

```
int x = 5, y = 6, z = 50;
cout << x + y + z;
```

One Value to Multiple Variables

You can also assign the **same value** to multiple variables in one line:

Example

```
int x, y, z;
x = y = z = 50;
cout << x + y + z;
```