C++/Introduction

Hello World!

The first program that most aspiring programmers write is the classic "Hello World" program. The purpose of this program is to display the text "Hello World!" to the user. The "Hello World" example is somewhat famous as it is often the first program presented when introducing a programming language[1].

```
#include <iostream>

using namespace std;

int main()
{
   cout << "Hello World!" << endl;
   cin.get();

   return 0;
}
```

NOTE:

The 'return 0;' as shown above, is not a necessary addition to the 'hello world' program. A return value of 0 in main simply signals to the operating system that everything went smoothly. By default, a C++ program will always return 0 if there is no return at the end of main.

Understanding the Code

Before discussing the particulars, it is useful to think of a computer program simultaneously in terms of both its structure and its meaning.

A C++ program is structured in a specific, particular manner. C++ is a language and therefore has a grammar similar to a spoken language like English. The grammar of computer languages is usually much, much simpler than spoken languages but comes with the disadvantage of having stricter rules. Applying this structure or grammar to the language is what allows the computer to understand the program and what it is supposed to do.

The overall program has a structure, but it is also important to understand the purpose of part of that structure. By analogy, a textbook can be split into sections, chapters, paragraphs, sentences, and words (the structure), but it is also necessary to understand the overall meaning of the words, the sentences, and chapters to fully understand the content of the textbook. You can think of this as the semantics of the program.

A line-by-line analysis of the program should give a better idea of both the structure and meaning of the classic "Hello World" program.

A Detailed Explanation of the Code

#include

```
#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 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

```
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."

Please note that you should either remember the fact that the iostream file uses the 'std' name space 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 in every time you wish to write a new C++ program is entirely normal, indeed justified; may we 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()

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: perform the actions specified by the statements in your program.

cin, cout

```
cout << "Hello, World!" << endl;
cin.get();
```

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)`, such as `cin.get()` in the example above (where `cin` is the object, `get` is the function name, and there are no arguments in the argument list). However, symbols such as << can also behave as functions, as illustrated by the use of `cout` above. This capability is called operator overloading which will be discussed later on.

{ }

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 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.

At one point, IBM tried paying its programmers by the number of lines of code they wrote each week. This did not work very well for C programmers who could make one statement span thousands of lines by simply holding down the enter key to insert lots of new lines between the words of their programs.

return

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

Compiling the code

In order for the computer to execute the code you have written, it needs to first be compiled by a C++ compiler. The compiler translates the textual representation of the program into a form that a computer can execute more efficiently.

What the compiler does

In very broad terms, the compiler is a translator that acts as an intermediary between the programmer and the CPU on the computer. A high-level language like C++ is actually a sort of 'compromise' language between the native language of the CPU (generally referred to as machine language) and the native language of the programmer (say English). Computers do not natively understand human languages, yet for someone to write computer code in the native language of the machine would be too difficult and time consuming. Therefore, the purpose of the computer language itself is to define a mid-point that is closer to how humans think and organize procedures but is still unambiguously translatable to the native machine language.

The compiler therefore is reading in the code written by the programmer and translating it into machine language code, that the computer can execute directly.

C++ is a compiled language that is converted to machine language by the compiler. Beginner programmers will likely also come across the notion of interpreted languages and interpreters. Since this text covers C++, interpreted languages are not covered in detail; however, in brief, an interpreter is like a compiler that converts the program into machine language at the time it is run on the computer rather than in advance as is done with a compiled language. An example of a interpreted language is Java.

Running the compiler

The code needs to be compiled with a compiler to finish the process. What if you don't have one? Well, the good news is, there are several good compilers that are available for free. The GNU Compiler Collection (GCC) has versions available for most systems and does a good job of implementing the ISO C++ standard. The clang compiler has complete support for C++11 and FreeBSD fully support clang and C++11. However, many people prefer to use an Integrated Development Environment (IDE) which provides a user friendly environment for developing programs. For MacOS X, there is Xcode which uses gcc for compiling C++. For Windows, there is Dev C++ which also uses gcc for compiling C++, Microsoft Visual C++ (and its free Express version), TCLITE, and ports of the GNU Compiler Collection distributed within Cygwin and MinGW. You might also enjoy using Geany (https://www.geany.org/Download/Releases)

Each compiler is invoked in a specific way. For example, if you wish to use GCC, type the following into a terminal:

```
c++ ex.cpp -o example
```

Replace `ex.cpp` with the name of the source file containing the program you wish to compile. The file name you choose must have an extension of either .cpp or .c++

Replace `example` with the file name you wish to use to invoke the executable program.

If the compiler detects any errors it will write them out at the terminal so that you can take action to fix them by editing your source file. If no errors are detected the compiler will produce an executable program file, in this case called example in the same directory as the source file.

To invoke the compiled program and thus have your computer execute it, enter:

```
./example
```

and observe that your computer performs the actions you specified in your source file.

If you wish to use a different compiler, please consult the documentation describing that compiler for the correct way to invoke it.

Exercises

Exercise 1

Copy the following, then edit it so it compiles correctly and prints "Hello, World!" on the screen

```
#include     <iostream>
using namespace std;
int main()
{
    cout << "Hello World!"<<endl;
    return 0;
}
```

A more complete version that includes line termination and a proper return code would be:

```
#include  <iostream>
using namespace std;
int main()
{
    cout << "Hello World" <<endl;
    return 0;
}
```

Exercise 2

Change the "Hello, World!" example above to display another line. If Spock were doing this exercise, he might add to it so that it would display:

```
Hello, World!
Live long and prosper.
```

```
#include <iostream>
using namespace std;
int main()
{
  cout<<"Hello, World!"<<endl;
  cout<<"Live long and prosper."<<endl;
  cin.get();
  return 0;
}
```

Exercise 3

Supposing you did use a:

```
using namespace std;
```

statement to reduce the amount of typing required, try removing this statement and see if you can still get your program to compile and run without it.

Alternatively if you did not use such a statement, try adding it and seeing how many 'std::' prefixes you can remove as a consequence and still have your program compile and run successfully.

Where To Go Next

Topics in C++		
Beginners	**Data Structures**	**Advanced**
• Lesson 1: Introduction to C++ • Lesson 2: Variables and User Input • Lesson 3: Simple Math • Lesson 4: Conditional Statements • Lesson 5: Loops • Lesson 6: Functions and Recursion • Lesson 7: More Functions	• Lesson 8: Pointers • Lesson 9: Classes and Inheritance • Lesson 10: Templates 1 • Lesson 11: Templates 2	• Lesson 12: The STL • Lesson 13: STL Algorithms
Part of the School of Computer Science		

References

1. Examples of programs in different programming languages (http://www.ntecs.de/old-hp/uu9r/lang/html/lang.en.html)

- C Preprocessor article at Wikipedia.
- Cplusplus Sample Codes (http://cplusplus.happycodings.com/)
- Stroustrup's C++ Style and Technique FAQ (http://www.stroustrup.com/bs_faq2.html%7CBjarne)