Compiling C++ Code

This lesson explains how to compile your code in C++, which compilers to use and the most suitable ones for different Operating Systems.

We'll cover the following

For the computer to execute the *Hello World* code you have written, it first needs to 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.

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

- 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

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.

Now that the basics about running the code are clear let's delve into the basic syntax of the C++ itself in the upcoming lesson.