**03/05/2020**

# Chapter 1: Getting Started

## 1. Hello World!

Every C++ program must have exactly one global function named `main()`. The program starts by executing that function. The `int` value returned by `main()`, if any, is the program's return value to "the system." If no value is returned, the system will receive a value indicating successful completion. A non-zero value from `main()` indicates failure.

Not every Operating System (OS) and execution environment make use of that return value: Linux/Unix-based and Mac-based environments often do, but Windows-based environments usually do not. For example, if you run Xcode in Mac and return zero from `main()`, you will see a line at the end indicating it.

```
Program ended with exit code: 0
```

In this course we use Linux (Ubuntu) as the default OS. This is to provide to you an opportunity to explore different environments and tools so that you have a feel of what OS you like.

Typically, a program produces some output. Here is a program that writes `Hello World!`

Hello.cpp

```
#include <iostream>

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

The line `#include<iostream>` instructs the compiler to include the declarations of the *standard stream I/O facilities* defined in `iostream`. Without these declarations, the expression

```
std::cout << "Hello World!\n";
```

will make no sense to the computer. The operator `<<` ("put to") writes its second argument/operand onto its first. In this case, the string literal `"Hello World!\n"` is written onto the standard output stream `std::cout`. A string literal is a sequence of characters surrounded by double quotes. In a string literal, the backslash character `\` followed by another character denotes a single “special character.” In this case, \n is the newline character. The std:: specifies that the name cout belongs to the standard-library (std) *namespace*.

If you need to get input from the user through the standard input stream, you can use the following syntax which is also part of the *standard stream I/O facilities*. The cin is the standard input, the operator >> is an extraction operator, and endl indicates an end-of-line.

```
int i;
std::cout << "Please input an integer: ";
std::cin >> i;
std::cout << "The value of your input: " << i << std::endl;
```

## 2. A Hello World Program with a Class and an Object

In the following example, we define a class named Hello in a .h file (a header file, introduced in more details in the next section), implement the details of the class in a .cpp file, then use the class in the main() in another .cpp file.

hello.h

```
#ifndef HELLO_H
#define HELLO_H
namespace hello{
 class Hello{
     public:
     void helloWorld();
 };
}
#endif
```

**(Class Member Function)** A member function is a function that is defined by a class. Member functions are defined once for the class but are treated as members of each object. In the previous lines of code, the helloWorld() is a member function declared in the .h file of class Hello. The member function’s definition is placed in the corresponding .cpp file:

hello.cpp

```
#include <iostream>
#include "hello.h"

namespace hello{
   void Hello::helloWorld(){
      std::cout<<"Hello World!! C++!!!"<<std::endl;
   }
}
```

A dot operator (.) is used to call a member function:

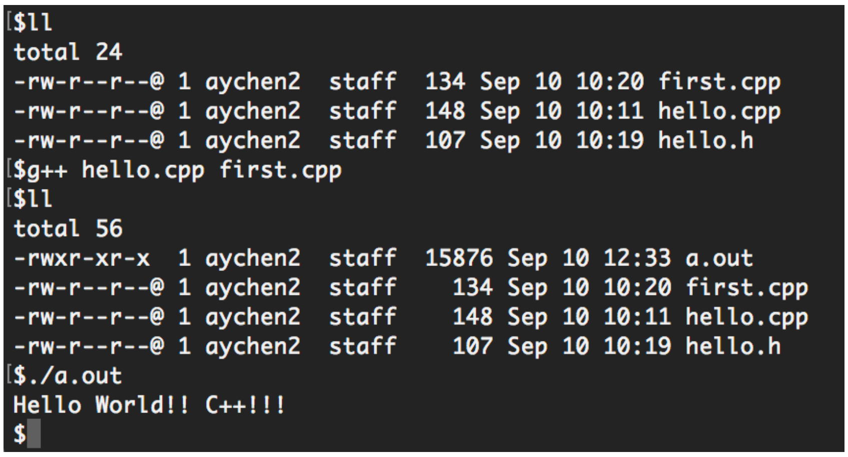
first.cpp

```
#include "hello.h"

int main(){
    hello::Hello h;
    h.helloWorld();

    return 0;
}
```

To run the code:



```
[$ll
total 24
-rw-r--r--@ 1 aychen2  staff  134 Sep 10 10:20 first.cpp
-rw-r--r--@ 1 aychen2  staff  148 Sep 10 10:11 hello.cpp
-rw-r--r--@ 1 aychen2  staff  107 Sep 10 10:19 hello.h
[$g++ hello.cpp first.cpp
[$ll
total 56
-rwxr-xr-x  1 aychen2  staff  15876 Sep 10 12:33 a.out
-rw-r--r--@ 1 aychen2  staff    134 Sep 10 10:20 first.cpp
-rw-r--r--@ 1 aychen2  staff    148 Sep 10 10:11 hello.cpp
-rw-r--r--@ 1 aychen2  staff    107 Sep 10 10:19 hello.h
[$./a.out
Hello World!! C++!!!
$
```

where g++ is the symbolic link to the c++ compiler.

## Header Files (.h)

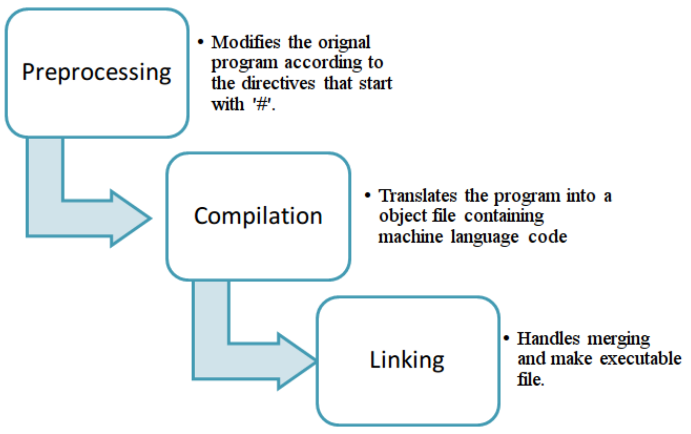
In order to ensure that the class definition is the same in each file, classes are usually defined in header files. Typically, classes are stored in headers whose name derives from the name of the class. For example, the `string` library type is defined in the `string` header. Similarly, as we've already seen, we defined our `Hello` class in a header file named `hello.h`.

The most common technique for making it safe to include a header multiple times relies on the preprocessor. The preprocessor—which C++ inherits from C—is a program that runs before the compiler to modify our source. Our program already rely on one preprocessor facility, `#include`. When the preprocessor sees `#include`, it replaces the `#include` with the contents (lines of code) of the specified header (such as `iostream`).

C++ programs also use the preprocessor to define header guards. Header guards rely on preprocessor variables. **Preprocessor variables have one of two possible states: defined** or **not defined.** The `#define` directive takes a name and defines that name as a preprocessor variable. There are two other directives that **test** whether a given preprocessor variable has or has not been defined: `#ifdef` is true if the variable has been defined, and `#ifndef` is true if the variable has not been defined. If the test is true, then everything following the `#ifdef` or `#ifndef` is processed up to the matching `#endif`. See the definition of the hello.h file as an example.

## 3. Steps to Build a C++ program

After you write a C++ program, the next step is to build the program before running it. The compilation is the process which convert the program written in human readable language (like C++, Java, or Python) into a machine code, directly understood by the Central Processing Unit (CPU). There are many stages involved in creating an executable file from the source file.


Preprocessing
• Modifies the orignal program according to the directives that start with '#'.
Compilation
• Translates the program into a object file containing machine language code
Linking
• Handles merging and make executable file.

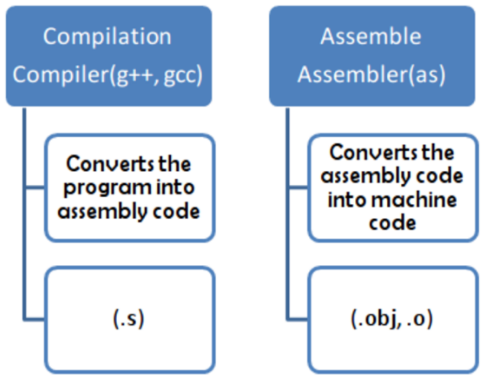
The stages to build an executable file from C++ source code include **Preprocessing**, **Compiling** and **Linking**. This means that even if the program gets compiled, it may result in not running as errors may arise during linking.

### 3.1. Preprocessing

In this phase, the preprocessor modifies the code according to the directives mentioned (that starts with # sign). The C++ preprocessor takes the program and deals with the `#include` directives and the resulting program is a pure C++ program. For example, in C++, `#include<iostream>` tells the preprocessor to read all the contents of the `iostream` header file and include the contents into the program and generate the separate C++ program file. C++ supports many preprocessor directives like `#include`, `#define`, `#if`, `#else` etc.

## 3.2. Compiling

This phase includes the <u>compilation</u> and the <u>assembly</u>.


Compilation
Compiler(g++, gcc)
Converts the program into assembly code
(.s)
Assemble
Assembler(as)
Converts the assembly code into machine code
(.obj, .o)

The <u>compilation</u> translates the program (preprocessed file without any directives) into a low-level assembly level code, specific to a processor architecture.

In the <u>assembly</u> phase, the assembly files are converted into machine level instructions and the file created is a relocatable object file. Hence, the compilation phase generates the relocatable object file which can be used in different places on the same computer without having to re-compile again.

Now, the object file created is in the binary form. In the object file created, each line describes one low level machine level instruction. But you still can't run these object files until they are converted into an executable file.

## 3.3. Linking

Linking as the name suggests, refers to creation of a single executable file from multiple object files. The file created after linking is ready to be loaded into memory and executed by the operating system. There is difference in linking and compilation when it comes to understanding errors.

Compiler shows syntax errors, for example, semi-colon (;) not in place, or data type not defined.

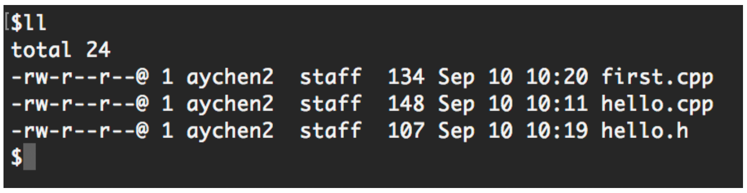
But if there is an error stating that a particular function has been defined multiple times, then this error is from the linker because it's indicating that two or more source code files have the definition for the same function (and these definitions might contradict in the content).

## References on building a program for C++:

https://www.calleerlandsson.com/posts/the-four-stages-of-compiling-a-c-program/
https://www.freelancer.com/community/articles/how-c-works-understanding-compilation
http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/compile.html
http://www.cplusplus.com/articles/2v07M4Gy/

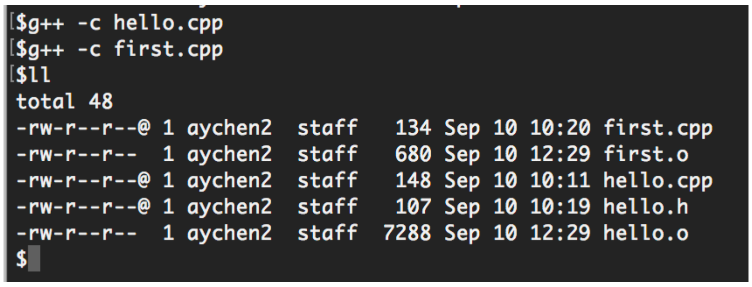
## Going back to our example

We can use the previous program to demonstrate this process for building the executive file. Note that we do not need to always separate these steps, this is just a demonstration. In our directory we have first.cpp, hello.h, and hello.cpp:



```
$ll
total 24
-rw-r--r--@ 1 aychen2  staff  134 Sep 10 10:20 first.cpp
-rw-r--r--@ 1 aychen2  staff  148 Sep 10 10:11 hello.cpp
-rw-r--r--@ 1 aychen2  staff  107 Sep 10 10:19 hello.h
$
```

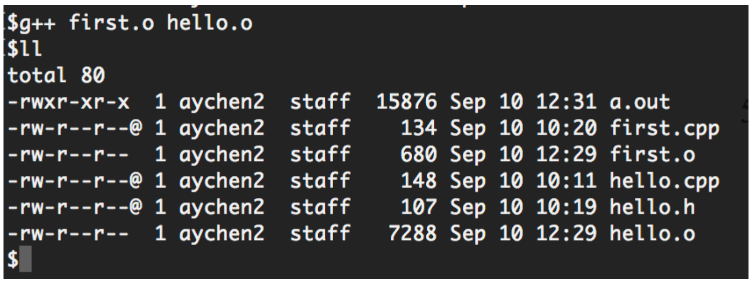
We can first preprocess and compile the two .cpp files separately, and by using the -c flag, their object files in .o format are created:



```
$g++ -c hello.cpp
$g++ -c first.cpp
$ll
total 48
-rw-r--r--@ 1 aychen2  staff   134 Sep 10 10:20 first.cpp
-rw-r--r--  1 aychen2  staff   680 Sep 10 12:29 first.o
-rw-r--r--@ 1 aychen2  staff   148 Sep 10 10:11 hello.cpp
-rw-r--r--@ 1 aychen2  staff   107 Sep 10 10:19 hello.h
-rw-r--r--  1 aychen2  staff  7288 Sep 10 12:29 hello.o
$
```

Note that in first.o, although hello.h is included in first.cpp, the detailed implementation of `helloWorld()` is missing. This member function is only declared in hello.h and the actual definition is in hello.cpp which forms hello.o, the other object file.

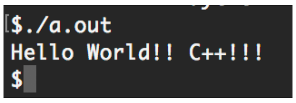
Then we can link the object files together into the one executable file with default name a.out:



```
$g++ first.o hello.o
$ll
total 80
-rwxr-xr-x  1 aychen2  staff  15876 Sep 10 12:31 a.out
-rw-r--r--@ 1 aychen2  staff    134 Sep 10 10:20 first.cpp
-rw-r--r--  1 aychen2  staff    680 Sep 10 12:29 first.o
-rw-r--r--@ 1 aychen2  staff    148 Sep 10 10:11 hello.cpp
-rw-r--r--@ 1 aychen2  staff    107 Sep 10 10:19 hello.h
-rw-r--r--  1 aychen2  staff   7288 Sep 10 12:29 hello.o
$
```

This linking provides the definition of `helloWorld()` in hello.o to first.o and thus our program can be executed.

You can then execute the program, with the a.out executable file:



```
$./a.out
Hello World!! C++!!!
$
```

## 4. Some more examples using C++

4.1. C++ 101: Write a program that sums from 0 to 9 using the `for` loop and print the sum.

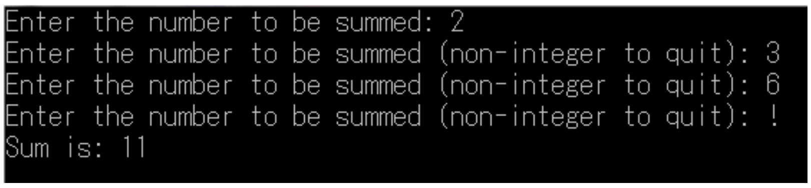
<u>Sum.cpp</u>

```
#include <iostream>

using namespace std;

int main()
{
    int sum = 0;
    for (int i = 0; i < 10; ++i)
        sum += i;
    cout << "Sum is: " << sum << endl;
    return 0;
}
```

4.2. Let's ask the user to input a set of numbers to sum. In this case, we won't know how many numbers to add. Instead, we'll keep reading numbers until there are no more numbers to read. Write a program and use `while` loop for the task.



```
Enter the number to be summed: 2
Enter the number to be summed (non-integer to quit): 3
Enter the number to be summed (non-integer to quit): 6
Enter the number to be summed (non-integer to quit): !
Sum is: 11
```

<u>AddSum.cpp</u>

**A:**

```
#include <iostream>

using namespace std;

int main()
{
    int sum = 0;
    int num;
    cout << "Enter the number to be summed: ";
    while (cin >> num) {
        sum += num;
        cout << "Enter the number to be summed (non-integer to 
quit): ";
    }
    cout << "Sum is: " << sum << endl;
    return 0;
}
```

Q: What is the effect to use an istream as a condition?

A:

– When we use an istream as a condition, the effect is to test the state of the stream. If the stream is valid, that is, if it is still possible to read another input then the test succeeds.

– An istream becomes invalid when we hit end-of- file (for file input) or encounter an invalid input, such as reading a value that is not an integer. An istream that is in an invalid state will cause the condition to fail.

<u>The coming week:</u>

- Variables
- Basic types
- References
- Pointers
- HW will be issued.
- Provide user name and password for Linux server before Tuesday night to TA.