How to Compile C++ with gcc

Dr. Manuel A. Pérez-Quiñones

Introduction

There is a different between compiling and linking your program, a difference that if you learned programming from integrated environments you might not be aware of it. Compilation involves converting your source code to object form, but not joining multiple object files into an executable. So, in C/C++, each source file can be compiled independently of other source files that go into a project. When all individual source files are compiled, then you can "join" them into an executable, this is called linking the project.

So the steps are:

Compilation:	Source (*.cpp) -> Object (*.o)
Linking:	Object (*.o) -> Executable (.exe in DOS)

To explain how to do each of these steps, follow the directions below. For this discussion I assume that $ this is the prompt of your operating system. Also, the gcc compiler compiles C and C++. It makes the determination of which compiler to use depending on the file name extension. For that purpose, I use .cpp to let the compiler know that the file is a C++ file. There are other extension that the compiler recognizes as C++ also, but this is the most clear of all. In some environments, there is a symbol defined as either g++ or c++ that just runs the gcc compiler directly. I use g++ in the discussions here, but we are really running the gcc compiler.

Single source file programs

Some projects are self contained in a single source file. If that is the case, then compilation and linking can occur in a single step because gcc calls the linker (ld) when it is done compiling a single source project.

The command line is:

```
$ g++ file.cpp
```

This will compile file.cpp, and then link the executable. The resulting executable is place in a.out. You can run this by typing:

```
$ a.out
```

or

```
$ ./a.out
```

if your path variable does not include the current directory (.) in it.

You also have the choice of specifying the name of the file that will contain the executable by using the -o option.

```
$ g++ file.cpp –o prog1
```

This will compile file.cpp, link the executable and place it in the file named prog1.

Multiple source file programs

The previous explanation is ok for very small programs, but rarely we work with small

programs. Most software projects have lots of files in which case you want to take advantage of the separate compilation facilities of C/C++.

The following command will only compile the file.cpp. It will not try to run the linker to create an executable. The idea is that if you have multiple source files, you can compile them one by one, and recompile only those that have changed since the last compilation. This command uses the -c option which tells the compiler, compile only the file. The object file that results is placed in file.o

```
$ g++ –c file.cpp
```

Imagine that your project consists of two files, file.cpp and other.cpp. You can build the full executable in any of the following ways:

Compile them both at the same time and place result in a.out

```
$ g++ file.cpp other.cpp
```

Compile them both at the same time and place results in prog2

```
$ g++ file.cpp other.cpp –o prog2
```

Compile each separately and then link them into a.out

```
$ g++ –c file.cpp
$ g++ –c other.cpp
$ g++ file.o other.o
```

Compile each separately and then link them into prog2

```
$ g++ –c file.cpp
$ g++ –c other.cpp
$ g++ file.o other.o –o prog2
```

Structure of your source and header files

Another source of confusion is the structure of your source and header files. Here are some very simple guidelines of how you should structure your code. I strongly recommend you follow these guidelines, even if they are different from what you are used to. If you have questions, contact your me or the TA to help you understand these.

File Extensions

- always use cpp as the extension for your source files
- always use h as the extension for your header files
- use something other than cpp for your template files (you might use tcc)

What to include in your files

- never include a cpp into another file. Files with cpp extensions are meant to be separately compilable units (source) using either g++ or g++ -c as explained above. If you include them elsewhere, you run the risk of multiple definition errors coming from the linking step.

- always put #ifndef around your header files to avoid double definitions and to break circular definitions. Your header files will look like this, assume that the file name is data.h, then:

```
// data.h
// other header information
#ifndef __DATA_H__
#define __DATA_H__

your header declarations goes here

#endif
```

Make sure that the symbol used in the `ifndef` is ***unique*** in your whole projects. The best thing to do is to use the file name with the "." replaced by an underscore and adding other underscores around to make it more unique.

Be careful if you copy a header file to use as a template to create another one. If you do this, make sure you change the symbol, having two header files with the same `ifndef` symbol will in effect hide the second you include in every compilation.

- include in your header file only those other header files that are needed in the header itself, not for the source associated with it.

Enjoy C/C++ programming.

Last updated on Sept 20, 2002