Compiling with g++

Executive Summary: *This document is a brief description of how to compile basic C++ programs using g++. It provides sample command lines for invoking the g++ compiler and a list of some common compiler options.*

What is g++?

g++ is your friendly Gnu C++ compiler. g++ does not handle templates well, but you can use them. This document serves as a very simple bunch of hints to start using g++, and is not meant to be complete. For all the gory details about g++'s options, check out its man page.

Compiling HelloWorld.C

Say you have a file helloworld.C as follows :

```
#include <stdio.h>

void main (){
    printf("Hello World\n");
}
```

You can compile and run it from the unix prompt as follows :

```
% g++ helloworld.C
```

This creates an executable called "a.out". You can run it by typing

```
% ./a.out
```

Since no executable name was specified to g++, a.out is chosen by default. Use the "-o" option to change the name :

```
% g++ -o helloworld helloworld.C
```

creates an executable called "helloworld".

Include Directories

Sometimes the header files that you write are not in the same directory as the .C file that #include's it. For example you might have a a file "foo.h" that resides in /homes/me/randomplace/include. If you want to include that file in helloworld.C, you could just give the full path name in the #include, *OR* you can do the following:

Add

```
#include <foo.h>
```

to helloworld.C and compile it with the -I option :

```
% g++ -o helloworld -I/homes/me/randomplace/include helloworld.C
```

This basically tells g++ to look for #include's in /homes/me/include in addition to other directories you specify with -I

Compiling multiple files

Most likely, you will be compiling separate modules and linking them into a single executable. Here's the basic idea: compile each .C file into a .o file, then link the .o files (along with any libraries) into an executable. Of course, one of these .C files has to define the main() or else the linker will complain. Suppose we have main.C, foo.C and bar.C and want to create an executable fubar, and suppose further that we need the math library:

```
% g++ -c -o foo.o foo.C
% g++ -c -o main.o main.C
% g++ -c -o bar.o bar.C
% g++ -o fubar foo.o main.o bar.o -lm
```

The first three commands generate foo.o, main.o and bar.o respectively. The last line links them together along with the math library, libm.a.

Some options

- **-g** - turn on debugging (so GDB gives more friendly output)
- **-Wall** - turns on most warnings
- **-O** or **-O2** - turn on optimizations
- **-o <*name*>** - name of the output file
- **-c** - output an object file (.o)
- **-I<*include path*>** - specify an include directory
- **-L<*library path*>** - specify a lib directory
- **-l<*library*>** - link with library lib<library>.a