Short Tutorial on `gcc` and `make`

Introduction

This tutorial intends to help you use `gcc` to compile your C code. It will also introduce *makefiles* to save you lots of typing. We will keep things as basic and brief as possible. If you need more help than is provided here, email us or come to office hours.

We are writing this tutorial for the RedHat Linux machines in the ENGR labs. ***We will grade your programs using those machines.*** You can develop your code wherever you want but before you submit it, make sure it will **compile and run without errors or warnings using gcc on linux** as described below.

The Terminal

Right click on the desktop and select "Open Terminal". You will be in your home directory. Here are a few basic commands for working in a terminal:

Commands and Arguments Description

`pwd`	Print the current directory path
`ls`	List files and directories in the current directory
`cd` <directory name>	Change working directory
`cd ..`	Change to parent directory
`mkdir`	Make a new directory
`cp` <src file name> <dst file name>	Create a copy of <src file> and name it <dst file>
`rm` <file name>	Remove the file
`emacs`	A nice text editor *Common commands:* Ctrl X Ctrl S: Save Ctrl X Ctrl C: Exit Ctrl G: used in situations that you get stuck
`emacs` <file name>	Open a particular file

Write Some Code

With a new terminal open, follow these commands:

```
1. mkdir cs261
2. cd cs261
3. mkdir assign1
4. cd assign1
5. emacs helloworld.c
6. Write your code and save the file. Start with just a main function that prints "Hello World".
7. gcc -Wall -std=c99 -o helloworld helloworld.c // See the next section for more
8. ./helloworld                                  // This will run your program!
```

Compiling with gcc

`gcc` is a command line compiler on Unix/Linux operating systems.
The most basic use is as follows:

1. Write a program in a file such as `helloWorld.c`
2. In the terminal, in the directory with the code file, type `gcc helloWorld.c`
    1. Errors and warnings will be printed to the terminal
        1. Errors are caused by syntax problems and the program cannot be compiled.
        2. Warnings mean the program can compile but the compiler thinks you might have done something wrong. ***Fix warnings too!***
    2. If there are no errors, the files `helloWorld.o` and `a.out` are created.
        1. helloWorld.o is an intermediate file that is compiled but not linked.
        2. `a.out` is the program that you can run by typing `./a.out`.

Options for unix programs are given on the command line and have a minus sign ('-') in front of them.

A few options for `gcc` are shown here:

Option	***Description***
`-Wall`	Show all warnings
`-std=c99`	Use the ANSI standardized version C99
`-o` <filename>	File name for the output program file
`-c`	Compile but don't link. Creates `*.o` files only.
`-lm`	Link the math library. Use this if you `#include <math.h>`

We will use `-Wall` and `-std=c99` when we grade so you should use them too!
In the previous example we only have 1 code file. What if the project has 3 code files: `code1.c`, `code2.c`, and `code3.c`?

To compile you'd type:

```
gcc -Wall -std=c99 -o prog code1.c code2.c code3.c
```

What if the files are big and you don't want to wait for them all to recompile because you made a change to one of them? You can do the following:

```
gcc -Wall -std=c99 -c code1.c
gcc -Wall -std=c99 -c code2.c
gcc -Wall -std=c99 -c code3.c
gcc -Wall -std=c99 -o prog code1.o code2.o code3.o
```

The first 3 commands compile each of the files and create code#.o files. If you change code1.c later, you only need to recompile that one file. The 4th command links the `code#.o` files to create the executable program `prog`.

This can be a lot of typing but there's a nice solution. See the next section.

Compiling with make

make is a program that reads a text file named "Makefile" and executes commands in that file. You can create a makefile in a text editor, just like your code. For our example with code1.c, code2.c and code3.c, we'd create a makefile that looks like this:

```
default: prog

code1.o: code1.c
	gcc -Wall -std=c99 -c code1.c

code2.o: code2.c
	gcc -Wall -std=c99 -c code2.c

code3.o: code3.c
	gcc -Wall -std=c99 -c code3.c

prog: code1.o code2.o code3.o
	gcc -Wall -std=c99 -o prog code1.o code2.o code3.o

clean:
	rm code1.o code2.o code3.o

cleanall: clean
	rm prog
```

That's a lot to type but it only has to be done once! Now, to compile the program, all you have to do is type `make`. The part of a line before a colon (':') is a target. The part of a line after a colon is a list of dependencies. If any dependency has changed since that target was executed, then it needs to execute again. Each indented line has the command that will execute for that target.

When you type `make` the default target is used. The default target depends on the prog target so it will cause it to execute. The prog target depends on the code#.o targets so they will be executed as well.

You can type `make target` to execute a particular target. This is useful with `make clean` which will delete the temporary code#.o files.

What About Header Files?

Let's extend the example and make it more realistic by pretending we have two header files: header1.h and header2.h. Let's say the code files `#include` the header files as follows:

- code1.c: header1.h
- code2.c: header2.h
- code3.c: header1.h header2.h

Then if `header1.h` changes, we want to recompile `code1.c` and `code3.c`. In the `makefile` add the appropriate `header#.h` files as dependencies.

```
code1.o: code1.c header1.h
code2.o: code3.c header2.h
code3.o: code2.c header1.h header2.h
```

Working at Home

As we mentioned before, you can write your programs anywhere you want. You don't have to write them on the lab machines. Before you submit a program, you should compile and run it on the lab (linux) machines to make sure there aren't any surprises when you get your grade.

You can access the lab machines from home using ssh. `dear115-##.engr.oregonstate.edu` or `kec1130-##.engr.oregonstate.edu` (where ## = 01, 02, ...)

You can only get to these machines from within the campus network though so you'll probably have to `ssh` to `flip.engr.oregonstate.edu` and connect to the lab machine from there.

If you're looking for a free C compiler to run at home, this website has a long list: http://www.thefreecountry.com/compilers/cpp.shtml

One option is Microsoft Visual C++ 2010 Express. You'll want to make sure the compiler is treating your files as C code and not C++ code. Many C/C++ compilers are capable of either and will likely default to C++.