.Notes and Discussion on Makefiles and separate Compilation for Multiple Source Code Files

The makefile MUST seperately compile each source code file into a ".o" file and separately link the ".o" files together into an executable file. Review the makefile in the example below to see how this is done.

The command to create the .o file is:

gcc –c program1.c

The command to link the files program1.o, program2.o and program3.o into an executable file is:

gcc program1.o program2.o program3.o

The above command will just name the executable file using the default name, most often the –o option is given to provide a specific name for the executable file.

gcc program1.o program2.o program3.o –o program.exe

Example of Multiple Source Code Files

Consider the program contained in the following files:

- max3a.c
- max3b.c
- max3.h
- makefile

This example shows how to set up this simplest of multiple source code file program. Note that max3a.c and max3b.c just contain functions and a #include of max3.h. The file max3.h contains the prototypes (or forward declarations) for all of the functions that are called from outside its source code file and any "globally" needed information.

The header file (.h file) should contain all #include statements, any #define statements, any struct and/or typedef statements and the function prototypes. The job of the header file is to contain the information so the source code files can talk to each other. Please review the .h file in the example below.

The makefile is a special file that helps in the compilation of the source code into the object files into the executable file. A makefile is executed by the use of the **make** command. The syntax of a makefile can be strange. I have always found that it is easiest to modify an existing makefile rather than trying to create one from scratch. The makefile will contain multiple rules. Each rule has the following syntax:

target: dependencyList
commandLine

The multiple rules in the make file are separated by a blank line. Also note (this is VERY IMPORTANT) the commandLine must use a TAB for its indentation! An example of a rule is:

```
max3a.o: max3a.c max3.h
	gcc -c max3a.c
```

The **commandLine** is **gcc -c max3a.c**, which will compile the source code file of max3a.c into the object code file of max3a.o.

The **target** in the above example is the file **max3a.o**, which is also the name of the file created when the commandLine is executed. This relationship is what causes makefiles to work.

The **dependencyList** in the above example is the two files: **max3a.c** and **max3.h**, which are the files needed for the commandLine to properly run. Again, this relationship is what causes makefiles to work.

The make command uses the timestamps of the files in the target and the dependencyList. If any file in the dependencyList has a more recent timestamp than the target file, the commandLine is executed. The idea is that if the user has recently changed either **max3a.c** or **max3.h**, then the object file **max3a.o** needs to be re-compiled. Make is designed to help the programmer keep track of what needs to be compiled next.

Make and makefile tutorials can be found at:

- http://mrbook.org/tutorials/make/
- http://www.gnu.org/software/make/manual/make.html
- http://www.opussoftware.com/tutorial/TutMakefile.htm