CS 403: Lab 3: Building with Make

Getting Started

1. Boot into Linux and log in.

2. Download a fresh copy of Cbasics.tar from the course website[1]. This is essentially the same code as before, but I've added newlines to eliminate the warning messages and deleted one #include statement that was redundant. Remember, to untar the file, type tar svf Cbasics.tar.

Building with make

1. Open the .c and .h files in Cbasics in an editor (nedit *.c *.h &). For each row in the table below, place an "X" in column two if the file in first column includes system.h. Place an "X" in the second column if the file includes io.h.

	system.h	io.h
system.c		
io.c		
main.c		

2. We're now ready to start building a Makefile. First, open a blank file in your editor (in nedit, select "New" from the file menu). On the first few lines, place a # in the first column and describe what we're doing (in case you're wondering, we're creating a Makefile for Cbasics, which will end up being similar to the Makefile for the model problem).

3. Now we'll define some variables (macros). I'll give you the first one: put "CC = gcc" on a new line. This lets us define the compiler for our system. You should probably place a comment after "gcc" to remind yourself that this is where you set the compiler. Create a variable CFLAGS and set it equal to the compiler flags we need (hint: we don't need any). Finally, create a variable called PROGRAM and set it equal to the application name (we used cbasic last time, feel free to change it).

[1] www.cs.cornell.edu/Courses/cs403/2002sp

4. We're ready for our first dependency. The first dependency in any Makefile must define how the program is built–this is the last command we would enter to build the executable. Because we have a multi-file program, we're going to take advantage of the -c option. Thus, cbasic will depend on the object files for each .c file. We write this `$(PROGRAM): main.o io.o system.o`. This tells make that if any of the .o files have changed (if their modification dates are newer than the program's) to execute the commands on the next line(s). Move to the line immediately below the dependency statement. HIT THE TAB KEY ONCE! The tab is critical. `make` is very fussy and absolutely requires a tab on the lines following the dependency statement. Then type `$(CC) $(CFLAGS) main.o io.o system.o -o $(PROGRAM)`. Note: `make` replaces $(Variable) by the value of Variable.

5. To finish the Makefile, work backwards from our last command and determine the dependency statements and commands to build main.o, io.o, and system.o. Hint1: an object file depends on its .c file any any .h files that are included (check the table above) . Hint2: remember to use the .c so that you build object files. When you're done, save your changes.

6. Delete any object files (`rm *.o`) and type `make` at the command line. If everything is correct, you should see your build statesments scroll by and you should see your program (cbasic) when you type `ls`. If you get an error message, try to figure out your error.

7. We'll now see exactly why Makefiles and the -c option are useful. Place some checkpoint statements (something like `printf("I'm now entering routine X\n")`) in main.c, save your changes, and type `make` again. What happens? If your Makefile is correct, then only main.c should be compiled. Try adding some checkpoints to io.c and system.c. What happens when you make them?

8. Last but not least, suppose you want to have PrintArray place a line of text (a comment line) at the start of the output file (C.txt). To do this, you will need to edit the description of PrintArray in io.c to something like

```
void PrintArray(char name[], char comment[], double C[], int m)
```

You will also need to change the prototype in io.h to match, and you will have to change the call in main.c to something like

```
PrintArray("C.txt", "this is a comment", C, m);
```

Now run make. Which statements were executed? What do you think would happen if you edited system.c (add a blank line or a comment and try it)? What about if you edit system.h?