CpSc 1111 Lab 9
Multi-Module Programs

Overview

This week, you will gain some experience with multiple file programs by using the program you wrote for lab 8 and putting each of the two functions into their own separate source file, and also creating a header file and a makefile for your program.

- `mainDriver.c` will contain your `main()` function
- `binToDec.c` will contain your `btod()` function
- `decToBin.c` will contain your `dtob()` function
- `defs.h` will be your header file and should contain the #include statements needed for your program, along with any prototypes
- `makefile`

Background Information

Multi-File Programs

In lecture, we talked about splitting up a file into multiple source files by moving a function into it's own file. Then to compile the program, when you use gcc, you list all the source files. For example, with this lab and the files named as above, to compile, you would type the following:

```
gcc-6 -Wall mainDriver.c binToDec.c decToBin.c -lm
```

If you want to change the name of the executabe to something other than a.out, you might do the following:

```
gcc-6 -Wall -o numberconverter mainDriver.c binToDec.c decToBin.c -lm
```

and then to run the program, you would type:

```
./numberconverter
```

- -

Header Files

User defined header files commonly contain things such as:

- #include statements needed in your source files
- prototypes for functions that appear in your source files
- shared variables, like constants or #define statements
- structure definitions

Header files do NOT contain function implementations (bodies).

You can name your header file whatever you want, but it must end with .h. In your source files, you would need to #include your header file, but don't forget that it needs to be in quotes instead of angle brackets. The angle brackets tell the compiler to look in the location where all the other C library header files are located. The quotes tell the compiler to look in the current directory for that file, which is where your header file will be located.

- -

makefile

A makefile (named `makefile` or `Makefile`) is a file that contains a list of "targets" and their dependencies, along with commands that you would otherwise type at the command-prompt. When you type `make` at the command-prompt, the system searches for a file called either `makefile` or `Makefile` and the command for the first target is run. If that first target contains the command to compile, then that is the command that is run. You can always type `make` followed by the target name to run whichever command is in your makefile that you want to run. An example is shown below:

```
lab8: lab8.c
      gcc -Wall -o lab8 lab8.c

# run has a dependency:  lab8
# lab8 comes from the  first line, the compilation
run: lab8
      ./lab8

clean:
      rm lab8
      rm -f output.txt
```

In the above makefile, there are 3 targets (in red font above). The dependencies are in purple font. If you want to add a comment, (those are in green), they are preceded with a #. The commands that will be run are in black font. To run any of those targets, you would type `make <target_name>` such as:

```
make lab8    // which would run the compile command
```

or

```
make run     // which would run the program; if it hasn't been compiled
             // yet it will compile it first and then run it since the
             // dependency of the executable named lab8 is listed -
             // in other words, it requires the existence of that executable
```

or

```
make clean   // will remove those files, if they exist
```

If you just type `make`, then the command for the first target is run, which in this case, is the command to compile.

By the way, the `-f` flag with the remove command says to remove those files listed without asking the user if they are sure. Be very careful if you use this flag. If, for example, you have a bunch of .o files that you want to remove, and you type

```
rm -f *.c
```

which means remove *all* files that end in .c (the * is a wildcard) without asking first if you're sure, instead of typing

```
rm -f *.o
```

then you will have just mistakenly removed all your source files and all your work is gone!

IMPORTANT NOTE:
The space before the command in the makefile **MUST BE** a tab. - - If it is not a tab, it won't work.

ONE OTHER NOTE:
If you have a lot of files on the compile line and you need to continue the list on a second line, use a backslash and then tab over on the next line, like this:

```
gcc -Wall -o newExName file1.c file2.c file3.c file4.c \
file5.c file6.c
```

Lab Assignment

For this week's lab assignment, you will write a multi-module program with the following files:

- `mainDriver.c` will contain your `main()` function
- `binToDec.c` will contain your `btod()` function
- `decToBin.c` will contain your `dtob()` function
- `defs.h` will be your header file and should contain the #include statements needed for your program, along with any prototypes
- `makefile`

Don't forget to do one step at a time and compile and run ***AFTER EACH STEP***. Perhaps try creating the header file first, #include it in your source file commenting out the things that are in the header file, and compile and run. Then create one of the source files listed, compile and run, moving on to the next step only after the current code works.

Also, don't forget that EACH source file should have a header comment at the top. You can put the function description in the header if the file contains only one function, along with the other required header comment items.

Reminder About Formatting and Comments

- The top of your file should have a header comment, which should contain:
    - Your name
    - Course and semester
    - Lab number
    - Brief description about what the program does
    - Any other helpful information that you think would be good to have.
- Variables should be declared at the top of the functions, and should have meaningful names.
- Always indent your code in a readable way. Some formatting examples may be found here: https://people.cs.clemson.edu/~chochri/Assignments/Formatting_Examples.pdf

Turn In Work

Show your ta that you completed the assignment. Then submit all 5 of your files using the handin page: http://handin.cs.clemson.edu

Grading Rubric

For this lab, points will be based on the following:

Functionality	70 (25 for general functionality according to specs 15 points for each source file for correct implementation)
defs.h	10
makefile	10
code formatting	10

Possible loss of points for other things, such as warnings when compile (-5), or other penalties for not following this lab's instructions.