CMSC 106 Project #2 Fall 2010

Date assigned: Saturday, September 25, 2010
Date due: Tuesday, October 5, 10:00 p.m.

1 Introduction

The purpose of this project is to become familiar compilation of multiple files, user defined functions, and conditional statements.

2 Project specifications

The provided mail should create output that is identical to what is shown in the sample execution in Section 8): But instead of looking at the whole project, you should work on implementing each function to do exactly what is described in the `functions.h` file. Each function needed has a prototype and a description in the `functions.h` file. All of your code should be placed in the `functions.c` file. Each function must do exactly what is described in the `functions.h` description for that one function.

3 Step by Step Directions

1. Copy the file named `proj2.tgz` from the `projects` directory that is in your `106public` directory. Place your copy of this file into your `106` directory.

   ```
   cp ~/106public/projects/proj2.tgz ~/106
   ```

2. Change into your 106 directory.

   ```
   cd ~/106
   ```

3. Then extract the contents of the file.

   ```
   tar -xvzf proj2.tgz
   ```

4. Then, you will see that a new directory has been created. That new directory is in your `106` directory and is named Proj2. Change into that directory.

   ```
   cd Proj2
   ```

5. Then, edit the file named functions.c. **DO NOT MAKE ANY CHANGES TO functions.h OR TO main.c.**

6. Submit after you are confident that your functions are all written according to the specifications given in the functions.h file.

4 Development suggestions

Many people like to write their whole program, I have provided the main method for you and you will be writing functions called by this main method or possibly others. Your process should be that you take the main and comment out the lines you do not want to test right now - then write one function and uncomment the line(s) that deal with that function. Test it completely (on different inputs) before declaring that building block done. Once you are convinced that the one function you just wrote gives correct values for every single case, then go to writing the next function. Do this process for every one of the functions you must implement.

5 Project requirements

1. All of your projects for this course must have a comment near their top which contains your name, your Glue ID, your section number, your TA's name, and your university ID number (not your Social Security number).

2. All your C programs in this course should be written in ANSI C, which means they must compile and run correctly when compiled with the compiler `gcc`, with the options `-Wall`, `-Werror`, `-ansi` and `-pedantic-errors`. This will be much easier if you ran the setup program discussed in the lab section when you did the setup.

3. Please review the project grading policy in the course syllabus. Note that you will **lose credit** during grading if your program produces any warning messages when it is compiled. Note again that not only does your program have to compile correctly, it must also produce correct output as well, so you should check the results it prints carefully.

4. You **may not** use any C language features except those introduced **up through Chapter 3** of your textbook, plus those presented in lecture and discussion section while the material in those chapters was being described. Note that using any features of C other than these will result in **losing credit** when your project is graded. For this project you may not use arrays or conditionals or loops.

5. The Campus Senate has adopted a policy asking students to include the following statement on each assignment in every course: "I pledge on my honor that I have not given or received any unauthorized assistance on this assignment." Consequently you're asked to include this pledge in a comment near the top of your program. See below for important information regarding violations of academic integrity.

6. Your program should be written using good programming style and formatting, which for this project is considered to consist of:
   - having an initial block comment describing the purpose of and task solved by the program,
   - use of adequate descriptive comments throughout, to indicate what your program is doing and how,
   - neat and readable indentation and formatting, including consistent alignment of braces and indentation of subsidiary statements, and avoiding writing any statements or comments which are wider than the standard terminal window width of 80 columns,
   - writing clear and readable code,
   - use of meaningful and descriptive variable names,
   - use of symbolic constants for any special values which don't change and which are used in several places in your code, and
   - avoiding disallowed C language features as discussed above.

6 Submitting your project

1. Turn in your program using the "submit" program provided by running "submit". **This submits only the .c file containing your source code, not the executable version of your program**!

2. Your project must be electronically submitted by the date and time above to avoid losing credit. No projects more than three days late will be accepted for credit without prior permission and a valid, documented excuse, as described in the course syllabus.

   Note there is **no grace period** for project submissions (the submission deadline is enforced exactly at the time above, the 24–hour late deadline is enforced exactly 24 hours later, etc.) and exceptions cannot be made; see the course syllabus for details.

3. Only the version(s) of the project which you electronically submit can be graded; it is your responsibility to test the program you are submitting and verify that it works properly before submitting.

4. **Do not delay or wait until the last minute to submit your program.** If you do, and you have any trouble, it will be too late to come to office hours, and you'll lose credit on the project. But if you start early and have questions or run into any difficulty, you can come to the TAs' office hours for assistance.

   It is recommended that you complete and submit your project **at least one day prior** to the due date and then reread this project assignment to insure that you haven't missed anything important which could cause you to lose credit on your project. If you have, this will give you ample time to correct it.

7 Academic integrity statement

Please **carefully read** the academic honesty section of the course syllabus. **Any evidence** of impermissible cooperation on projects, use of disallowed materials or resources, or unauthorized use of computer accounts, **will be submitted** to the Student Honor Council, which could result in an XF for the course, or suspension or expulsion from the University. Be sure you understand what you are and what you are not permitted to do in regards to academic integrity when it comes to project assignments. These policies apply to all students, and the Student Honor Council does not consider lack of knowledge of the policies to be a defense for violating them. Full information is found in the course syllabus– please review it at this time.

8 Sample output

Here is a sample output assuming the executable version of their program is in a file named "proj2.x" and the prompt on the linux.grace.umd.edu system is shown as as ">". Underlined text is typed as input when the program is run, while everything else is its output.

```
> proj2.x
Give Positive Integer: 17
Give Positive Integer: -4
Give Positive Integer: 0
values: 17 4 1
largest: 17
larger = 4.25
smaller = 0.23529
Give Shape: p
Give Integer: 5
Played: you win
>
```

Or an example of a run when you do not win:

```
Give Positive Integer: 9
Give Positive Integer: -29
Give Positive Integer: 44
values: 9 29 44
largest: 44
larger = 3.22
smaller = 0.31034
Give Shape: a
Could Not Play:not a good character
```

In addition to this information about how the supplied main should run. The tar file also contains two of the files used on the submit server for grading. The files named `public00.c` and `public01.c` are smaller main functions where each tests only one function (from your functions.c file). The directions for compiling these are provided in comments at the top of each file. I suggest you write similar small main functions to test each of the other functions that you implement.