CS141 Homework 2

~~Due Friday Sept 19 at 3pm.~~

Due Monday Sept 22 at 3pm.

You will write two C programs and implement some utility C functions for this assignment.

Objectives:

- Practice with devising looping mechanisms for a variety of situations (including user interaction).
- Writing and testing C functions according to given specifications.
- Writing and modifying your own makefile

[9/11/14; 4:45pm: Part III complete…]

[9/10/14; 5:30pm: Parts I and II being released now; update including all of Part III and final submission instructions to be available within 24 hours]

Overview:

There are 3 parts to this assignment:

I. An "ASCII histogram" program. You will write **hist.c**
II. An interactive arithmetic quiz program. You will write **quiz.c**
III. Implementation of four C "utility" functions. You will complete the functions in a file **util.c**. You will be given util.h

Logistics:

- Everybody must submit the following:
  - hist.c
  - quiz.c
  - util.c
  - util.h (unmodified)
  - util_tst.c
  - makefile
    the makefile must contain targets hist, quiz and util.o

We should be able to compile all of your work with:

```
$make hist
$make quiz
$make util.o
$make util_tst
```

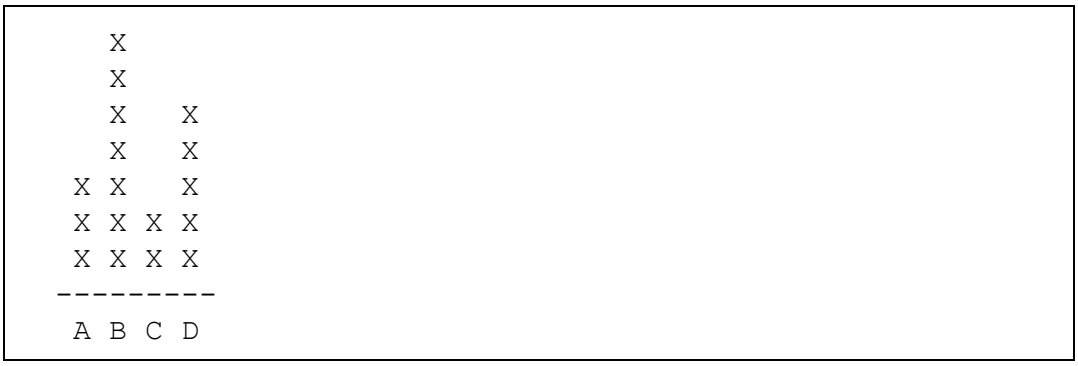
- User Input: for parts I and II, you are welcome/encouraged to use the simpleio functions; but it is not required -- e.g., if you are familiar with the C library function scanf, you may use it.

  However, if you use the simpleio functions, you must also include simpleio.h and simpleio.c in your submission. *In other words, when we unpack your directory, we should be able to compile everything without adding any files.*

Part I: A histogram program

This program will "draw" an ASCII histogram for 4 user-entered values A, B, C and D.

Example: if A=3, B=7, C=2 and D=5, the output should look like this:



Behavior/Specifications/Details:

- Your program must be in a file called **hist.c**

- Behavior:
    - When run the program prompts the user four times - once for each of A..D (example below).
    - If the user gives a negative value, you must re-prompt and ask for a non-negative value until the user gives a correct value.
    - Once all of A, B, C and D have been entered, you produce the ASCII histogram and terminate.

Sample run:

```
$./hist
  enter A:  3
  enter B:  -7
    only non-negative values
  enter B:  7
  enter C:  2
  enter D:  5


     X
     X
     X   X
     X   X
   X X   X
   X X X X
   X X X X
  ---------
   A B C D
```

Note 1: the histogram must be "vertical" as in the example above. The following would be considered wrong:

```
WRONG!!!!

A: XXX
B: XXXXXXX
C: XX
D: XXXXX
```

Note 2: You may NOT use arrays for this program!!!

Part II: An arithmetic quiz program

This program will quiz the user on basic arithmetic problems.

Filename: quiz.c

Behavior: when the program runs, the user will be given a menu of three choices:

1. Give me an addition problem.
2. Give me a subtraction problem.
3. Give me a multiplication problem.
4. Quit.

The user's choice is specified with the corresponding integer. (Note: I've left out division so we can just work with integers.) If the user gives an integer outside of 1..4, the program prints an error message and re-prints the menu.

For options 1-3, your program will generate a problem for the user to solve. For example an addition problem might look like:

Complete: 12 + 22 =

After displaying the problem, the program reads an answer from the user and reports if it is correct or not.

If the answer *is* correct, the program congratulates the user and re-prints the menu and awaits the user's next choice.

If the answer is *incorrect* the program will ask the user to try again, but only up to a **maximum of 5 attempts**. If the user gives 5 incorrect answers, the program reports the correct answer with an appropriate message ***and the program re-prints the menu and awaits user input.***

Your program will use non-negative numbers in the range [0..100] (inclusive) for the problems given to users. To generate such numbers at "random" you can use the rand() and srand() library functions. The rand() function returns an integer. To get an integer in the specified range, you can use the mod operator:

```
a = rand() % 101;  // % 101 gives a number in 0..100
```

The rand() function produces a "pseudo-random" number. To "seed" the generator so it doesn't produce the same sequence each time the program is run, add this magic code to the top of main:

```
srand( (unsigned)time(NULL));
```

You will need to include the time.h header file as well:

```
#include <time.h>
```

Part III: Some utility functions.

You will complete four C functions in a file called **util.c**

The function declarations are already created in a header file **util.h.** Below are the function declarations:

```
double abs_diff(double x, double y);

int median3(int a, int b, int c);

unsigned  int sqrt_floor(unsigned int n);

int num_ones(unsigned int n);
```

See the comments for specifications of each function's behavior.

Your Job:

- Write implementations of each function in util.c
- Modify util_tst.c so that it does more extensive testing of your implementations. Start by reading util_tst.c in its current form and think about its limitations!
- Make sure your makefile allows us to compile your test program like this:

  $ make util_tst

Details/Reminders:

- Your implementations must be self-contained. Particularly with respect to sqrt_floor(n), you cannot call an external function **(e.g., sqrt() function from the C math library!).**
- Your functions should NOT print anything to the terminal!
- Your test program does not have to be perfect -- the goal is to start thinking more rigorously about testing and correctness.

Things to remember:

- You will make **one submission** for all three parts.
- You will include a **makefile** in your submission which supports the following:
  - $ make hist
  - $ make quiz
  - $ make util_tst
- The files util.c, util.h and util_tst.c are available in the src directory in the same sub-directory as this handout (link).
- Specifications for the behavior of the util functions appear as comments in util.h
- If you use the simpleio functions, you must also include the source files in your submission (i.e., so the above make commands work)
- Where will you be writing code:
  - You will write **hist.c** from scratch
  - You will write **quiz.c** from scratch
  - You will create a **makefile** from scratch (you may of course use previous makefiles as a model).
  - You will modify **util.c** -- i.e., adding the bodies of the functions.
  - You will modify/extend **util_tst.c** to test your utility functions more extensively.
- You may **NOT** modify **util.h**
- You may **NOT** use arrays.