CSCI 1312 (Introduction to Programming for Engineering), Fall 2016

Homework 5

Credit: 40 points.

1 Reading

Be sure you have read (or at least skimmed) the assigned readings from chapter 6.

2 Honor Code Statement

Please include with each part of the assignment the Honor Code pledge or just the word "pledged", plus one or more of the following about collaboration and help (as many as apply).[1] Text *in italics* is explanatory or something for you to fill in. For written assignments, it should go right after your name and the assignment number; for programming assignments, it should go in comments at the start of your program.

- This assignment is entirely my own work.
- This assignment is entirely my own work, except for portions I got from the assignment itself *(some programming assignments include "starter code")* or sample programs for the course *(from which you can borrow freely — that's what they're for).*
- I worked with *names of other students* on this assignment.
- I got help with this assignment from *source of help — ACM tutoring, another student in the course, the instructor, etc.*
- I got significant help from *outside source — a book other than the textbook (give title and author), a Web site (give its URL), etc.. ("Significant" here means more than just a little assistance with tools — you don't need to tell me that you looked up an error message on the Web, but if you found an algorithm or a code sketch, tell me about that.)*
- I provided significant help to *names of students* on this assignment. *("Significant" here means more than just a little assistance with tools — you don't need to tell me about helping other students decipher compiler error messages, but beyond that, do tell me.)*

3 Programming Problems

Do the following programming problems. You will end up with at least one code file per problem. Submit your program source (and any other needed files) by sending mail to `bmassing@cs.trinity.edu` with each file as an attachment. Please use a subject line that mentions the course and the assignment (e.g., "csci 1312 hw 5" or "CS1 hw 5"). You can develop your programs on any system that provides the needed functionality, but I will test them on one of the department's

[1] Credit where credit is due: I based the wording of this list on a posting to a SIGCSE mailing list. SIGCSE is the ACM's Special Interest Group on CS Education.

Linux machines, so you should probably make sure they work in that environment before turning them in.

1. (20 points) *NOTE* that the point of this problem is for you to practice using `for` loops, so you must use at least one to get full credit, and I strongly recommend that you do all the needed repetition using `for`.

   Write a C program that gets a positive integer N from the user and prints an $N+2$ by $2N+4$ pattern of stars and spaces like the following:

   For $N = 4$:

   ```
   ************
   **  ********
   ****  ******
   ******  ****
   ********  **
   ************
   ```

   For $N = 7$:

   ```
   ******************
   **  **************
   ****  ************
   ******  **********
   ********  ********
   **********  ******
   ************  ****
   **************  **
   ******************
   ```

   Print an error message if what was entered is not a positive integer.

   You might find it useful to split your program into several functions, as a way of keeping the main program from being too complicated, and also as a way of not writing similar code over and over. Functions you might find useful in addition to the main one:

   - A function that, called with a character `c` and an integer `n`, prints `c` `n` times. Notice that you can use `printf` to print a single character, but it is simpler to just use `putchar`.
   - A function that, called with three integers (call them `stars1`, `spaces`, and `stars2`), prints a line consisting of `stars1` stars, then `spaces` spaces, then `stars2` spaces.

2. (20 points) *NOTE* that the point of this problem is for you to practice using `while` loops, so you must use at least one to get full credit, and I strongly recommend that you do all the needed repetition using `while`.

   Newton's method for computing the square root of a non-negative number x starts with an initial guess r_0 and then repeatedly refines it using the formula

   $$r_n = (r_{n-1} + (x/r_{n-1}))/2$$

Repetition continues until the absolute value of $(r_n)^2$ - x is less than some specified threshold value. An easy if not necessarily optimal initial guess is just x.

Write a C program that implements this algorithm and compares its results to those obtained with the library function `sqrt()`. Have the program prompt for x, the threshold value, and a maximum number of iterations; do the above-described computation; and print the result, the actual number of iterations, and the square root of x as computed using library function `sqrt()`. Also have the program print an error message if the input is invalid (non-numeric or negative).

Here are some sample executions:

```
[bmassing@dias04]$ ./a.out
enter values for input, threshold, maximum iterations
2 .0001 10
square root of 2:
with newton's method (threshold 0.0001):  1.41422 (3 iterations)
using library function:  1.41421
difference:  2.1239e-06

[bmassing@dias04]$ ./a.out
enter values for input, threshold, maximum iterations
2 .000001 10
square root of 2:
with newton's method (threshold 1e-06):  1.41421 (4 iterations)
using library function:  1.41421
difference:  1.59472e-12
```

Hints:

- While it may seem from the description of the problem that you will need a variable for each r_n, in fact you do not; all you need is one that represents the current guess (r_n) and the previous guess (r_{n-1}).
- You may find the library function `fabs()` useful.