CSCI 1312 (Introduction to Programming for Engineering), Fall 2015

Homework 4

Credit: 35 points.

1 Reading

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

2 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 homework 4" or "CS1 hw4"). You can develop your programs on any system that provides the needed functionality, but I will test them on one of the department's Linux machines, so you should probably make sure they work in that environment before turning them in.

1. (5 points) Write a C program that defines and tests a function for converting Fahrenheit temperatures to Celsius. (You should be able to reuse your code from Homework 2, perhaps with some slight modifications.) Your program should include three functions:

   - A conversion function to perform the actual conversion, declared as follows:

     ```
     double f_to_c(double f);
     ```

     where the single parameter `f` represents a Fahrenheit temperature and the return value represents the equivalent Celsius temperature. (So, for example, `f_to_c(32.0)` should evaluate to 0.0.0

   - A convert-and-print function declared as follows:

     ```
     void convert_and_print(double f);
     ```

     that calls the conversion function and prints its input and output nicely. So for example `convert_and_print(32.0);` might print

     ```
     32 degrees Fahrenheit is 0 degrees Celsius
     ```

     (Don't worry too much about printing appropriate numbers of digits after the decimal point; you can do whatever is easy, or read the man page or other documentation for `printf` to find out how to get more control.)

   - A main function that calls `convert_and_print()` at least four times with different inputs that you think will illustrate that the conversion is working right. (So this program is self-contained and doesn't prompt the user for anything!)

     NOTE that the point of this problem is for you to practice defining and using functions, so you will not get full credit unless your program includes functions as described.

2. (10 points) Write a C program that asks the user for two non-negative integers, call them a and b, not both zero, and computes and prints $\gcd(a, b)$, the greatest common divisor of a and b, using a recursive version of Euclid's algorithm. Print an error message if what was entered is not two integers, or either input is negative, or both are zero.

   Euclid's algorithm can be described recursively thus: For non-negative integers a and b, not both zero, with $a \geq b$,

$$\gcd(a, b) = \begin{cases} a & \text{if } b = 0 \\ \gcd(b, a \bmod b) & \text{otherwise} \end{cases}$$

   where $a \bmod b$ is the remainder when a is divided by b. (You don't actually have to understand this algorithm to turn it into code, but if you want to and don't, a Web search will likely turn up some good explanations of how/why it works.)

   NOTE that the point of this problem is for you to practice defining and using a recursive function, so you will not get full credit unless you do. I recommend putting all the error checking in the main program and having a recursive function declared as

```
int gcd(int a, int b);
```

3. (10 points) Write a C program that gets a positive integer N from the user and prints an N by $2N$ pattern of stars and spaces like the following:

   For $N = 4$:

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

   For $N = 7$:

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

   CORRECTION: In the above examples, the output is actually $N + 2$ by $2N + 2$, which was my intent. Solutions that produce the above output for $N = 6$ and $N = 9$ respectively will be considered correct.

   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.

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`.

4. (10 points) 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
```

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`.

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.