CSCI 1120 (Low-Level Computing), Fall 2017

Homework 4

Credit: 20 points.

1 Reading

Be sure you have read, or at least skimmed, the assigned readings for classes through 9/13.

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(s).

- This assignment is entirely my own work. *(Here, "entirely my own work" means that it's your own work except for anything you got from the assignment itself — some programming assignments include "starter code", for example — or from the course Web site. In particular, for programming assignments you can copy freely from anything on the "sample programs page".)*
- 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. (Here, "help" means significant help, beyond a little assistance with tools or compiler errors.)*
- I got help from *outside source — a book other than the textbook (give title and author), a Web site (give its URL), etc.. (Here too, you only need to mention significant help — 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 help to *names of students* on this assignment. *(And here too, you only need to tell me about significant help.)*

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 1120 hw 4" or "LL hw 4"). 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] 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.

1. (10 points) ***NOTE*** *that this problem is meant as an opportunity to get some practice with* `while` *loops in C, so you will only get full credit if you use at least one.*

   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 — but notice that zero is okay).

   Here are some sample executions:

   ```
   [bmassing@diasw04]$ ./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@diasw04]$ ./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:

   - For this problem I recommend that you resist any impulse to split up your program into several functions; I think it's simplest and clearest to just put all the needed computation in `main()`.
   - To use the library function `sqrt()` you need not only the appropriate `#include` line (as documented in its `man` page) but also the compile flag `-lm` (also documented in the `man` page).
   - You may find the library function `fabs()` useful.

2. (10 points) ***NOTE*** *that this problem is meant as an opportunity to get some practice with* `for` *loops in C, so you will only get full credit if you use at least one.*

   C, like many programming languages, has a library function (`rand()`) that can be used to generate a "random" sequence of numbers (quotes because it's not truly random — more in

class). This function can be used to generate a value in a specified range (e.g., between 0 and 5 inclusive if you're trying to simulate rolling a 6-sided die). `rand()` itself generates a number between 0 and the library-defined constant value `RAND_MAX`, so to get a value in a smaller range you have to somehow map the larger range to the smaller one. The somewhat obvious way to do this is by computing a remainder (see starter program below), but with some implementations of `rand()` this gives results that aren't very good. The conventional wisdom is therefore to instead try to do a more-direct map (e.g., to map to just two possible values, assign values from 0 through `RAND_MAX/2` to 0 and the remaining values to 1).

Your mission for this problem is to complete a C program that, given a number of samples N and a number of "bins" B generates a sequence of N "random" numbers, uses both methods to map each generated number to a number between 0 and *B-1* inclusive, and counts for each method how many elements of the sequence fall into each bin (e.g., for each method bin 0 is how many elements of the sequence map to 0), and prints the result, as in the sample output below. To help you (I hope!) I'm providing a starter program, link below, which you should use as your starting point.

One other thing to know about `rand()` is that by default it always starts with the same value (and produces the same sequence). To make it start with a different value, you can call `srand()` with an integer "seed", so your program should prompt for one of those too.

Sample execution:

```
[bmassing@diasw04]$ ./a.out
seed?
5
how many samples?
1000
how many bins?
6
counts using remainder method:
(0) 154
(1) 188
(2) 171
(3) 161
(4) 155
(5) 171
counts using quotient method:
(0) 172
(1) 175
(2) 183
(3) 150
(4) 168
(5) 152
```

If you feel ambitious you could also have the program print maximum and minimum counts and the difference between them, as a crude measure of how uniform the distribution is:

```
[bmassing@diasw04]$ ./a.out
```

```
seed?
5
how many samples?
1000
how many bins?
6
counts using remainder method:
(0) 154
(1) 188
(2) 171
(3) 161
(4) 155
(5) 171
min = 154, max = 188, difference 34
counts using quotient method:
(0) 172
(1) 175
(2) 183
(3) 150
(4) 168
(5) 152
min = 150, max = 183, difference 33
```

(You will get an extra-credit point for doing this.)

Here is a starter program that prompts for the seed, generates a few "random" numbers, and illustrates the two methods of mapping to a specified range: rands.c[2].

Of course, your program should check to make sure all the inputs are positive integers. (Yes, error checking is a pain, but it's an incentive to get better at copy-and-paste?)

[2] `http://www.cs.trinity.edu/~bmassing/Classes/CS1120_2017fall/Homeworks/HW04/Problems/rands.c`