# CSCI 1120 (Low-Level Computing), Fall 2013

# Homework 3

**Credit:** 20 points.

## 1 Reading

Be sure you have read the assigned readings for classes through 9/25.

## 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 number and the assignment (e.g., "csci 1120 homework 3"). 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. (10 points) 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 sometime). 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 write 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 shows the distribution of results as in the sample output below.

   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) 173
(1) 150
(2) 179
(3) 181
(4) 155
(5) 162
counts using quotient method:
(0) 153
(1) 177
(2) 148
(3) 183
(4) 178
(5) 161
```

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) 173
(1) 150
(2) 179
(3) 181
(4) 155
(5) 162
min = 150, max = 181, difference 31
counts using quotient method:
(0) 153
(1) 177
(2) 148
(3) 183
(4) 178
(5) 161
min = 148, max = 183, difference 35
```

(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[1].

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? and we might eventually write a function that would make it easier.)

2. (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@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:*

- 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.

[1] `http://www.cs.trinity.edu/~bmassing/Classes/CS1120_2013fall/Homeworks/HW03/Problems/rands.c`