Lab 05: Dice Game

CSci 515 Spring 2015

<2015-02-11 Wed>

Dates:

Due: In Lab, Wednesday February 18, by 4:05 pm (lab end time)

Objectives

- More practice writing C functions
- Use standard library rand() functions to implement random simulations
- Practice creating user defined functions that break problems into smaller pieces
- Learn about rescaling ranges of values.

Description

In this lab we will write a program to answer a simple question from probability. The question is, if I roll 2 dices, what is the probability of rolling a sum of either a 7 or 11 if I add up the faces of the 2 dice rolled.

You will first of all create a function that will return a random number between 1 and 6 when called that simulates the rolling of a single dice. Then you will write another function to implement the simulation of the dice game. The simulation should roll N pairs of dice, and keep track of the number of times M out of the N rolls that the sum of the two dice was either a 7 or an 11. The function will return this result M/N as a floating point ratio of the result (make sure you avoid integer division problems when returning the result). For example, if we roll 1000 pairs of dice, and it comes up with a 7

or an 11 on 676 out of those 1000 trials, then the function will return 676 / 1000 = 0.676

Perform the following tasks:

1. Write a function named `rollDice()`. This function takes no parameters as input. It will return a single integer value in the range from 1 to 6. The function should use the rand() system call to generate a random number, and then scale the number to the appropriate range [1,6] and return it.

2. Write a function call `simulateDiceGame()`. This function will take a single integer called `numTrials` as the input. This is the number of pairs of dice it is to simulate rolling. You should implement rolling `numTrials` pairs of dice by writing a for loop that uses an index that ranges from 1 up to `numTrials`. You need to call your `rollDice()` function two times, to simulate rolling 2 pairs of dice. You then need to sum up the values of the two random dice rolls you receive. You need to keep track of the number of trials whose sum is either 7 or 11, by for example declaring a variable called `successes` that is initially 0, and that is incremented by 1 each time the sum of the two dice rolls is as indicated. The result of your `simulateDiceGame()` function should be a single floating point number, between [0.0, 1.0], that is the result of dividing the number of successes, by the total number of simulated dice throwing trials that were performed.

3. In your `main()` function, prompt the user to enter two values. First of all, ask the user to enter a random number to use as the seed for the srand() function. Use the value entered to seed the random number generator. Then ask the user to enter the number of trials to run. Using this number of trials values, call your `simulateDiceGame()` with this number of trials, and get back the resulting ratio. This ratio should represent an estimate of the probability of throwing a 7 or an 11 when rolling 2 dice.

Your program output should look something close to the following when I run your program:

```
Enter a seed with which to initialize the random number generator: 42

I will simulate rolling a pair of dice and estimate the probability
of rolling a 7 or an 11.  Enter the number of trials to run: 1000000
```

```
I tried 1000000 experiments.  The estimated probability
of rolling a 7 or an 11 is: 0.2224999964237213
```

If you implement your program correctly, then giving the same value for the seed will end up resulting in the same estimated probability on subsequent runs of your program. However different seeds will give slightly different results. The true probability of getting a sum of 7 or 11 when rolling 2 dice is $\frac{8}{36} \approx 0.22222222$, so any experiment you run should end up with a value somewhere around that true probability.

NOTE: Now that our programs have more functions than just the `main()` function, the use of the function headers becomes meaningful and required. Make sure that all of your functions (`main`, `rollDice`, `simulateDiceGame`) have function headers preceding them that document the purpose of the functions, and the input values and return value of the function.

Lab Submission

An eCollege dropbox has been created for this lab. You should upload your version of the lab by the end of lab time to the eCollege dropbox named `Lab 05 Dice Game`. Work submitted by the end of lab will be considered, but after the lab ends you may no longer submit work, so make sure you submit your best effort by the lab end time in order to receive credit.

Requirements and Grading Rubrics

Program Execution, Output and Functional Requirements

1. Your program must compile, run and produce some sort of output to be graded. 0 if not satisfied.

2. 40+ pts. Your program must have the 2 required named functions, that accept the required input parameters and return the required values (if any).

3. 20+ pts. Your dice rolling function must return a random value within the correct range each time it is called.

4. 20+ pts. Your dice simulation function must correctly perform the number of indicated trials, and count up the successful trials from all

of the trials performed, and return the correct probability ratio. Your ratio must be correct.

5. 20+ pts. You should prompt the user for the number of trials to perform in your `main()` function, and display the results. The interaction with your program should be as shown in the example output above.

Program Style

Your programs must conform to the style and formatting guidelines given for this course. The following is a list of the guidelines that are required for the lab to be submitted this week.

1. The file header for the file with your name and program information and the function header for your main function must be present, and filled out correctly.
2. A function header must be present for all functions you define. You must document the purpose, input parameters and return values of all functions.
3. You must indent your code correctly and have no embedded tabs in your source code. (Don't forget about the Visual Studio Format Selection command).
4. You must not have any statements that are hacks in order to keep your terminal from closing when your program exits.
5. You must have a single space before and after each binary operator.
6. You must have a single blank line after the end of your declaration of variables at the top of a function, before the first code statement.
7. You must have a single blank space after , and ; operators used as a separator in lists of variables, parameters or other control structures.
8. You must have opening `{` and closing `}` for control statement blocks on their own line, indented correctly for the level of the control statement block.

Failure to conform to any of these formatting and programming practice guidelines for this lab will result in at least 1/3 of the points (33) for the assignment being removed for each guideline that is not followed (up to 3

before getting a 0 for the assignment). Failure to follow other class/textbook programming guidelines may result in a loss of points, especially for those programming practices given in our Deitel textbook that have been in our required reading so far.