CSC 112 – Primes Random Numbers

Purpose and Goals

By the end of the lab, you will have:

- Additional experience with the Zylab and CLion programming environments
- Written 2 programs in C++, making use of the fundamental aspects of the C++ language, with a focus on arrays, random number generation, and file output, and following the desired style guidelines
- Dealt with special case programming scenarios where a solution needs to be arrived at differently due to some property of the data provided
- Explored how computers can be used to solve complex math problems (finding prime numbers) and to perform simulations (die rolls).

Reading and Background

Part 1 of this assignment is to continue to build familiarity with C++ and CLion and to demonstrate that you can translate information and pseudo-code about an algorithm into C++ code. There are also some tricky loop limits to deal with.

Note: You may start with the templates provided, but your final code **must** follow the style guide.

The first two parts deal with prime numbers.

A **prime number** (or a **prime**) is a natural number greater than 1 that has no positive divisors other than 1 and itself.

Prime numbers can be determined using the Sieve of Eratosthenes. Information on this algorithm, including pseudo-code can be found at the link below. You are not expected to become an expert on the math, although it is interesting. However, you will have to translate the pseudo-code to c++.

https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

For motivation of understanding primes, you can check the following or doing your own investigation if so inclined:

https://en.wikipedia.org/wiki/RSA_(cryptosystem)

http://world.mathigon.org/Prime_Numbers

The template code uses the bool type. You can find this in section 3.6 of the zyBook.

You may also need the c++ modulus operator. I am sure you can find this on your own.

The template for all parts is in Part 1. There may be variables you don't need, and you may need to add variables. But the template is a start.

Part 1

Code using clion but start with the template in the zyLab Part 1. Input will be an integer in the closed interval [0, kN], where kN will not exceed 10000. Using the Sieve of Eratosthenes algorithm, determine the prime numbers in [0, kN] as well as the number of primes and the max prime. Below is sample output for kN=20. Test your code in clion, then cut and paste into the zyLab and run the tests. Submission is your zyLab inline code.

```
Enter kN:
2 3 5 7 11 13 17 19
#Primes: 8, Max_prime: 19
```

1) Determine the average difference (larger minus smaller) between the primes found. You may want to make an array of the primes – that is one way to handle it. Example for kN = 20.

```
Enter kN:
2 3 5 7 11 13 17 19
#Primes: 8, Max_prime: 19
Avg prime diff: 2
```

2) Print the number of twin prime pairs in your list. As a point of interest, you may want to check the twin prime conjecture. Maybe you can prove it. Link below.

https://en.wikipedia.org/wiki/Twin_prime

3) Put the actual prime numbers in an array, primes, in locations primes[0] through primes[#primes-1] in numerical order. Iterate through the array primes. If the digit in the unit's place position primes[i] > digit in the units position in primes[i+1], then swap the primes in these two locations. E.g., 19 would swap with 23. Continue from[i+1] to the end. Output the modified array primes.

4) Swap the positions of the two largest values in the modified primes. For full credit, you must keep track of these values while doing any of the previous tasks. I .e., you may **not** add any additional loops other than to output the newly modified array primes. Output this newly modified array primes.

Example output for kN=55 follows.

```
Enter kN:
55
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53
#Primes: 16, Max_prime: 53
Avg prime diff: 3
#Twin prime pairs: 6
2 3 5 11 13 7 17 23 19 31 37 41 43 47 53 29
2 3 5 11 13 7 17 23 19 31 37 41 43 53 47 29
```

Part 2

Simulation represents a vital and widespread application for computers. In particular, simulating events that have a random component is an excellent way to understand and predict various kinds of systems that could include drought, weather, nuclear weapons, power usage, network traffic, disease spread, and a lot more.

This lab simulates dice rolling and also gives experience outputting data to a file for analysis by other software, Excel (or similar spreadsheet programs).

For this lab, an event is the roll of two identical but independent dice. Each die can have from 3 to 9 sides. A trial is some number of dice rolls. Example output is at the end. Here are the specifications for this lab:

- Input the following three unsigned int values:
  - rSeed seed for the random number generator
  - numTrials number of trials to run in the interval [1, 1000]
  - numSides number of sides per die in the interval [1, 9]
- Loop for numTrials (a while loop might be useful here.) For each trial:
  - roll each die until doubles are rolled. Doubles means that both die show the same number.
  - If doubles are not rolled, (doubles means that both die show the same number) add the value rolled on the two dice together and keep a running score for this trial. The roll again
  - If doubles are rolled, output to a file the number of rolls and the score for the trial. Note, if doubles are rolled on the first roll, then that counts one roll with a total equal zero. The output, on one line, should be: the number of rolls, a comma, the score. Output file in your ClionProjects folder. Name the file: DiceSimOut.csv. Note: this is a comma delimited file.

- For all trials in total, keep track of the total number of rolls as well as how many times a particular sum of two die appears. E.g., for 6-sided die, there are 10 possible sums. Output the total number of rolls to the console. Output the count of each sum to the console. This represents a histogram of the various possible sums.
- Run a trial with rSeed = 199873; numTrials = 1000; and numSides = 6. Load the output file into Excel. Make a scatter plot from the data. The x-axis should be the number of rolls. The y-axis should be the score. An example is below for 6-side die and 1,000 trials. The trendline and statistics are not required, but do this if you can.
- **Submit your code to the zyLab to test for console output. Note for this you may have to comment out or otherwise inactivate file access and output.**
- **Submit the following to Sakai:**
  - **main.cpp**
  - **jpeg screenshot of your scatter plot (for PC, you can use the "snipping tool")**
  - **For two, fair, 6-sided die, what are the expected percentage for each of the following sums: (2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)? Write your answer in Sakai. Hint: How many possible ways can two die show up?**

Example run

Console output and input

Enter seed: 199977
Enter number of trials: 10
Enter number of sides per die: 6
Total rolls: 87
Counts for two-die roll counts.
1 6 8 7 15 11 15 11 6 4 3

File output

3, 13
28, 202
3, 15
1, 0
13, 85
5, 22
13, 89
2, 6
16, 94
3, 13

Example scatter plot (6-sided die; 1,000 trials)