ACMS 20210 Exam 3

Due: 11:59 PM on Wednesday, December 5

Submit the C++ programs below using Sakai. If you are compiling a single file from the command line on the CRC or another Linux system, I suggest using the command

```
g++ your_program_name.cpp -std=c++1y -o your_executable_name
```

For this exam, to simplify grading, except for Problem 4, **use the following naming scheme for organizing your programs. If you do not adhere to this scheme (including captialization, underscores, etc.), you may incur a small penalty for each incorrectly named file.** For a given problem number i, use only one .cpp file, with the name exam_3_problem_i.cpp. For example, your submission for problem 1 should be in a single file named exam_3_problem_1.cpp.

You are allowed to work with others, to use notes, and to use outside sources, but you must cite any sources that you use. You may look at code that others have written for similar problems, but you must write your own implementation for the final version that you submit.

1. An integer is a palindrome in base 10 if reversing its decimal representation yields the same integer (i.e. if reading the number forwards and backwards gives the same value). For example, 3124213 is a palindrome in base 10, since reversing the digits yields 3124213. However 34546 is not a palindrome, since reversing the digits yields 64543, a distinct number.

   Write a C++ program that takes any number of file names, passed by command line argument. The files passed to your program are guaranteed to consist of positive integers, separated by spaces (or line breaks), with an unknown number of integers per line. Your program should report the total number of (base 10) palindrome numbers encountered among all files, the sum of the palindrome numbers, the number of distinct palindrome numbers, and the sum of all distinct palindrome numbers to the user.

   Among the three sample files on Sakai, there are 8646 palindromes, and the total of these is 392536868. The are 1098 distinct palindromes, and the total of these is 50045040.

   (You may wish to use long long ints for this problem to avoid potential problems with the maximum size of the sum of all the integers.)

2. The Gambler's Ruin is a well-known problem in probability theory. One version of the problem is as follows:

   A gambler starts with an initial fortune of S dollars , and has a goal of a obtaining a final value of G dollars (both assumed to be positive integers). The gambler bets in 1 dollar increments. As long as the gambler is not ruined (has 0 dollars) and has not reached his goal, he continues to bet. On each bet, he either wins a dollar with fixed probability p, or loses a dollar with probability $q = 1 - p$. We are interested in approximating his probability of obtaining his goal before being ruined.

   Write a C++ Monte Carlo simulation that takes four command line arguments: the gambler's goal G, his starting fortune S, his probability of winning each bet p, and a number of trials N. On each trial, the gambler starts with S dollars and places bets until either he reaches his goal or is ruined. Report to the user the percentage of trials on which the gambler succeeded in reaching his goal (which should approximate his probability of success).

   The exact value is given by:

$$\text{Probability of Success} = \begin{cases} \frac{1-\left(\frac{q}{p}\right)^S}{1-\left(\frac{q}{p}\right)^G} & \text{if } p \neq \frac{1}{2} \\ \\ \frac{S}{G} & \text{if } p = \frac{1}{2} \end{cases}$$

   where $q = 1 - p$.

3. Consider an $M \times M$ chessboard, with $M \geq 2$. Two kings, from opposing sides, are placed in opposite corners of the chessboard. The usual rule that the kings cannot be adjacent to one another is suspended, and the rule that they cannot capture one another is also suspended. The kings alternate turns, choosing from among their available legal moves uniformly at random (any legal move has the same probability of being selected as any other legal move). Write a C++ program that uses a Monte Carlo simulation to determine the expected number of moves before one of the kings captures the other (moves to the same space as the other).

   Your program should take two command line arguments: the first for the size M of each side of the board, and the second for the number of trials to conduct. (If $M < 2$, the program should terminate early with a message to the user that the board is too small.) Your program should report the approximate average number of moves required for one of the kings to capture the other.

   For a 2×2 chessboard, the expected number of moves until one of the kings captures the other is exactly 3.
   For a 5×5 chessboard, the expected number of moves is approximately 36.7593.
   For an 8×8 chessboard, the expected number of moves is approximately 120.3046.

   More challenging alternative problem (not due for submission):

   Write a C++ program that uses the theory of Markov chains to calculate this quantity deterministically.

4. On Sakai, you will find two files: exam_3_problem_4.cpp (where a main routine for this problem is located), and Vector3D.h. Your task is to implement the functions declared in the header file Vector3D.h, placing your implementation in a file named Vector3D.cpp. For this problem, submit your Vector3D.cpp file. The output (to the terminal) for the fully compiled program should mimic that in the example output on Sakai. (Note that you do not need to submit exam_3_problem_4.cpp, as I am providing that for you.)

   The Vector3D class implements a mathematical vector with 3 components. Some of the supported functions involve: checking for equality, vector arithmetic (including addition, subtraction, and scalar multiplication), calculating dot products, cross products, and norms, and finding the angle between two vectors.

   Below are the formulas for the dot product, cross product, and the angle between two vectors:

   If $u = (u_1, u_2, u_3)$ and $v = (v_1, v_2, v_3)$, with θ the angle between u and v, then:

$$|u| = \sqrt{u_1^2 + u_2^2 + u_3^2}$$

$$u \cdot v = u_1 v_1 + u_2 v_2 + u_3 v_3$$

$$u \times v = (u_2 v_3 - u_3 v_2, u_3 v_1 - u_1 v_3, u_1 v_2 - u_2 v_1)$$

$$\theta = \cos^{-1}\left(\frac{u \cdot v}{|u|\,|v|}\right)$$

   (Note that if either u or v is the zero vector, the angle between u and v is not defined. For the purposes of this problem, you need not worry about this detail.)

   One issue with computing the angle between vectors that you might encounter is that, due to floating point arithmetic, the quantity inside the inverse cosine in the expression for θ might evaluate to a number that is slightly greater than 1.0 or slightly less than -1.0, for which the inverse cosine is undefined. Should either of these cases occur, before computing the inverse cosine, you should replace the expression with 1.0 or -1.0 as appropriate.

   Note that some of the functions have both a member function and non-member function implementation.

   On Sakai, I have provided you with the definition of the overloaded insertion operator in the Vector3D.cpp file.