CS3305-03 Assignment #2 Due: 02/25/2020 at 11:59pm

Instructions:

1. ALL programs must be coded in C++ using Object Oriented Programming style.
2. Submit ALL the programming questions as C++ (*.cpp) programs. ALL programs must be executed and tested for valid outputs before submission.
3. Provide necessary indentation in the code for readability.
4. Use constructors to initialize member variables in the programs.
5. Provide comments to briefly explain the purpose of the program and the functions in it. Please include your name, class and assignment number in the comments at the start of the program.
6. Submit your assignment as ZIP file (including CODELITE project folder) through D2L using the appropriate assignment link.
7. The ZIP file name format is '<yourname>_assignment2.zip'
8. All assignment problems are from the required text for CS3305. Kindly refer syllabus.

Objective:

The goal of assignment #2 is to reinforce linear data structure concepts, and its implementation in C++.

Assignment Problems:

PROGRAMMING PROJECTS (page - 288 in text)

1. Problem 8 (20 points)

Write a program for keeping a course list for each student in a college. The information about each student should be kept in an object that contains the student's name and a list of 8 courses completed by the student. The courses taken by a student are stored as a linked list in which each node contains the name of a course, the number of units for the course, and the course grade. The program gives a menu with choices that include adding a student's record, deleting a student's record, adding a single course record to a student's record, deleting a single course record from a student's record, and printing a student's record to the screen. The program input should accept the student's name in any combination of upper- and lowercase letters. A student's record should include the student's GPA (grade point average) when displayed on the screen. When the user is through with the program, the program should store the records in a file. The next time the program is run, the records should be read back out of the file and the list should be reconstructed. (Ask your instructor if there are any rules about what type of file you should use.)

PROGRAMMING PROJECTS (page - 351 in text)

2. Problem 6 (20 points)

Rewrite the polynomial class of Section 4.6 so that the data type of the coefficients is a template parameter. This data type can be any type that has operators for addition, subtraction, multiplication, and assignment. The class should also have a default constructor, which results in a zero value. For example, our template class would allow us to build polynomials where coefficients are complex numbers (using the complex<double> type from <complex>).

PROGRAMMING PROJECTS (page - 390 in text)

3. Problem 8 (20 points)

Write a program that evaluates an arithmetic expression in postfix notation. The basic algorithm is contained in "Evaluating Postfix Expressions" on page 377. Assume the input contains numbers (but no variables) as well as the arithmetic operations +, -, *, and /. Your program should allow the user to evaluate additional expressions until the user wants to end the program. You might also enhance your program so that the expression need not be well formed; if it is not well formed, then the user must reenter the expression.

4. Problem 12 (10 points)

Choose one of your stack implementations and write a friend function to display the contents of a stack from top to bottom. Then, implement a friend function to display the stack bottom to top.

PROGRAMMING PROJECTS (page - 434 in text)

5. Problem 9 (30 points)

Write a simulation program for a small airport that has only one runway. There will be a queue of planes waiting to land and a queue of planes waiting to take off. However, only one plane can use the runway at a time. So there can be only one takeoff or one landing in progress at any one time. Assume that all takeoffs take the same amount of time. Assume that all landings take the same amount of time, although this does not need to be the same as the takeoff time. Assume that planes arrive for landing at random times, but with a specified probability of a plane arriving during any given minute. Similarly, assume that planes arrive at the takeoff queue at random times, but with a (possibly different) specified probability of a departure. (Despite the fact that takeoffs and landings are scheduled, delays make this a reasonable assumption.) Since it is more expensive and more dangerous to keep a plane waiting to land than it is to keep a plane waiting to take off, landings will have priority over takeoffs. Thus, as long as some plane is waiting to land, no plane can take off. Use a clock that is an unsigned integer variable that counts the number of minutes simulated. Use the cstdlib random number generator to simulate arrival and departure times of airplanes. This simulation can be used, among other things, for deciding when the air traffic has become so heavy that a second runway must be built. Hence, the simulation will simulate conditions that would be a disaster, in which planes crash because they run out of fuel while waiting too long in the landing queue. By examining the simulated situation, the airport authority hopes to avoid the real tragedy. Assume all planes can remain in the queue the same amount of time before they run out of fuel. If a plane runs out of fuel, your simulation will not discover this until the simulated plane is removed from the queue; at that point, the fact that the plane crashed is recorded, that plane is discarded, and the next plane is processed. A crashed plane is not considered in the calculation of waiting time. At the end of the simulated time, the landing queue is examined to see whether any of the planes in the simulated queue have crashed. You can disregard the planes left in the queue at the end of the simulation, other than those that crashed for lack of sufficient fuel. Use the following input and output specification:

Input: (1) The amount of time needed for one plane to land; (2) the amount of time needed for one plane to take off; (3) the average amount of time between arrival of planes to the landing queue; (4) the average amount of time between arrival of planes to the takeoff queue; (5) the maximum amount of time that a plane can stay in the landing queue without running out of fuel and crashing; (6) the total length of time to be simulated.

Output: (1) The number of planes that took off in the simulated time; (2) the number of planes that landed in the simulated time; (3) the number of planes that crashed because they ran out of fuel before they could land; (4) the average time that a plane spent in the takeoff queue; (5) the average time that a plane spent in the landing queue.