CSC 228 Data Structures and Algorithms, Spring 2020
Instructor: Dr. Natarajan Meghanathan

Exam 1 (Take Home)

Submission (in Canvas; See instructions in the last page) **Due: Feb. 20th, by 11.59 PM**

Q1 - 18 pts) Consider the implementation of the List ADT using the code for Singly Linked List given to you for this question.
Add a member function (to the List class) called *recursivePrintForwardReverseOrders* that prints the contents of the list in a recursive fashion in both the forward order and reverse order.

For example, if the contents of the List are: 10 --> 4 --> 8 --> 12 --> 9, the recursive member function should print the List as follows:
10 4 8 12 9
9 12 8 4 10

Note that both the forward and reverse orders should be printed through an invocation of the *recursivePrintForwardReverseOrders* member function on the List object called from the main function. You are free to choose the parameter(s) that need to be passed to the *recursivePrintForwardReverseOrders* function. But, you are not supposed to pass more than three parameter(s). A suggestion for the parameter to pass is given in the main function of the code posted for Question 1.

To test your code (and take screenshot), create a List of at least 10 elements (with a maximum value of 100) and then call the *recursivePrintForwardReverseOrders* function on this List object by passing a pointer to the first node in the Linked List as an argument, as shown in the main function of the Singly Linked List code for Question 1.

Q2 - 22 pts)
For this question, you will implement the Stack ADT as a Singly Linked List without directly using the insertAtIndex, deleteElement, readIndex functions of the Singly Listed List. That is, the push, pop and peek operations should not call insertAtIndex(0, data), deleteElement(0) and readIndex(0) functions. The push, pop and peek operations should be directly implemented to insert an element in the beginning of the linked list, to delete an element (and return its value) from the beginning of the linked list and to read the element value from the beginning of the linked list. Your task is to implement the push, pop and peek functions (as explained above) without calling any other function. You can notice that the insertAtIndex, deleteElement and readIndex functions have been removed from the Stack class code assigned for this question and you should **not** include the code for these three functions (insertAtIndex, deleteElement and readIndex functions) to use them.

After implementing the three functions push, pop and peek, you will be comparing the actual run-times of the push and pop operations; the main function provided to you has the timers setup
for this purpose. Your task is to just run the main function and measure the average time taken (in microseconds) for the push and pop operations with the Stack as a Singly Linked List for the following values of the parameters: (i) # elements to be pushed = 10000,
100000, 1000000; (ii) maximum value for any element = 50000 and (iii) # trials = 50.

Q3 - 18 pts)
For this question, you will develop the code to determine the nodes that have the maximum data and minimum data values in a singly linked list-based implementation of a List (the code for this implementation is provided to you). The main function is setup to create a List of random integers. The main function has a comment indicating that you will write your code there to determine the addresses of the nodes with the maximum data and minimum data in the List and then print out the data of the corresponding nodes. You should not do any changes in the code provided for the List class.

Test run your code with inputs for the list size as 10 and the range of values for the elements is [1...50] and capture the screenshot.

Q4 - 19 pts)
You are given the code for a singly linked list-based implementation of the List ADT. You will add a function called isUnique( ) to the List class such that the function will return true if all the integer elements in the List are unique and return false otherwise. The main function is written in such a way that it will create 1000 Lists (each of size listSize) of random integers in a range [1...maxValue] and will call the isUnique function to determine how many of these 1000 Lists are unique. The main function is programmed to determine the fraction of the 1000 Lists that are unique and the fraction is referred to as the probability that a List of listSize with integers in the range [1...maxValue] will be unique. You will test your code with the following values for listSize and maxValue.

listSize: the last two digits of your J#. For example, if your J# is J00123456, then your listSize is 56. In case, the last two digits of your J# is a value less than 10, you will add 10 to that value and use the sum as the listSize. For example, if your J# is J00120005, then your listSize will be 10 + 05 = 15.

maxValue: the last four digits of your J#. For example, if your J# is J00123456, then your maxValue is 3456. In case, the last four digits of your J# is a value less than 1000, you will add 1000 to that value and use the sum as the maxValue. For example, if your J# is J00120005, then your maxValue will be 1000 + 0005 = 1005.

State your J# as part of the documentation you submit for this question.

Q5 - 23 pts)
Implement the Selection Sort algorithm discussed in class to sort a list of a certain size. The list is to be implemented **using a dynamic array** and you are given the startup code for this purpose.

The main function is also setup with the timers to evaluate the performance (sorting time) of your implementation. Note that the main function is written for you in such a way that the code will run for different values of list sizes ranging from 1000 to 20000, in increments of 1000. The inputs for the maximum value and number of trials are respectively 5000 and 50 for all cases. The code will output the average sorting time (in milliseconds) for each of value of list size for the implementation.

You are then required to tabulate the average sorting times observed for the implementation. Also, generate an Excel plot that displays the average sorting times (in Y-axis) for different values of the list size (in X-axis). Use the Add Trend Line option in Excel, choose the Power function option and display the equation and the R^2 value for the curve. The R^2 value is expected to be closer to 1.0. Make sure the equation and the R^2 value are clearly visible in your Excel plot. Compare the equation you obtained from your Excel plot with the theoretical time complexity of the algorithm.

Submission (in Canvas)

Submit separate C++ files for the codes as follows (so, a total of FIVE .cpp files as mentioned below):

Question 1 (16 pts) The complete code for the Node class, List class (including the recursivePrintForwardReverseOrders( ) function) and the main function.

Question 2 (20 pts) The complete code for the Node class, Stack class (including the code for the push, pop and peek functions) and the main function.

Question 3 (16 pts) The complete code for the Node class, List class and the main function (including the code to compute the addresses of the nodes with the minimum and maximum data).

Question 4 (17 pts) The complete code for the Node class, List class (including the isUnique( ) function) and the main function.

Question 5 (17 pts) The complete code for the dynamic array-based implementation of the list, the sorting algorithm and the main function.

Submit a single PDF file that includes your screenshots and analysis for all four questions put together as mentioned below (clearly label your screenshots/responses for each question):

Question 1 (2 pts) Screenshot of the execution of the code with at least 10 elements and maximum value of 100.

Question 2 (2 pts) Screenshots of the execution of the code for each of the three values for the number of elements to be pushed.

Question 3 (2 pts) Screenshots of the execution of the code with at least 10 elements and maximum value of 50.

Question 4 (2 pts) Screenshot of the execution of the code with the inputs for the number of elements and the maximum value for an element (both based on your J#) passed as described in the description of the question. Also, state your J#.

Question 5 (6 pts): (i) A table that displays the average sorting times (in milliseconds) from 1000 to 20000 (in increments of 1000); (ii) An Excel plot that displays the list size vs. average sorting time for the sorting algorithm as well as displays the equation of the fitted curve and the R^2 value. (iii) Compare the equation you obtained from your Excel plot with the theoretical time complexity of the algorithm.

IMPORTANT NOTE: If the instructor finds out (at the time of grading) that one or more students were involved in copying/cheating for even one question in the exam, all the students found to be involved in it will be assigned a ZERO for the whole exam.