CSC 228 Data Structures and Algorithms, Fall 2019
Instructor: Dr. Natarajan Meghanathan

Exam 1 (Take Home)

Submission (in Canvas; See instructions in next page) **Due: Oct. 1st, by 11.59 PM**

Q1 - 20 pts) Consider the implementation of the List ADT using 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 - 15 pts)
Consider the implementation of the Singly Linked List class (named: List) given to you for this question. Your task is to add a member function called pairwiseSwap( ) to the Singly Linked List class such that it can be called from the main function (as given in the code) to swap the elements of an integerList (an object of the class List) pairwise and print the updated list. For example, if the List before the pairwiseSwap is 4 -> 5 -> 2 -> 3 -> 1 -> 6, after the pairwiseSwap, the contents of the list should be: 5 -> 4 -> 3 -> 2 -> 6 -> 1. If the List has an odd number of elements, like: 4 -> 5 -> 2 -> 3 -> 1, then after the pairwiseSwap, the contents of the list should be: 5 -> 4 -> 3 -> 2 -> 1.

Test your code with 10 elements and 11 elements (with a maximum value of 25 in each case) and take screenshots of the List before and after pairwiseSwap, as printed in the main function.

Q3 - 15 pts)
The deleteElement(int deleteData) member function in the code for the Singly Linked List-based implementation of a List ADT deletes the first occurrence of deleteData in the List. Modify this member function in such a way that all occurrences of deleteData in the List are deleted with a single call to the deleteElement function from main. After you modify the *deleteElement* function, run the main function (as given in the startup code for this question) by creating a List of at least 15 elements (with a maximum value of 10 for any element so that certain elements repeat). Now ask for a value (deleteData) to delete from the user and call the deleteElement(deleteData) function to delete all occurrences of deleteData in the List. Capture the output of your program displaying the contents of the List before and after the call to the deleteElement function.

Question 4 (50 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 as well as a singly linked list. You are required to compare the performance (sorting time) for the dynamic array and singly linked list-based implementations.

You are given the startup codes for the dynamic array and singly-linked list based implementations. You are required to implement the Selection Sort function in each of these codes. 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 5000 and 50 for all cases. The codes will output the average sorting time (in milliseconds) for each of value of list size for the two implementations.

You are then required to tabulate the average sorting times observed for the two implementations and then plot the same in Excel. Generate one plot that displays (compares) the sorting times for both the implementations. Use the Add Trend Line option in Excel, choose the Power function option and display the equations and the R^2 values for the two curves. The R^2 values are expected to be closer to 1.0. Make sure the two equations and the R^2 values are clearly visible in your Excel plot.

Submission (in Canvas)

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

Question 1 (17 pts) The complete code for the Node class, List class (including the recursivePrintForwardReverseOrders( ) function) and the main function.

Question 2 (12 pts) The complete code for the Node class, List class (including the pairwiseSwap( ) function) and the main function.

Question 3 (12.5 pts) The complete code for the Node class, List class (including the modified version of the deleteElement(deleteData) function) and the main function.

Question 4-(i: 18 pts) The complete C++ code for the dynamic array-based implementation to sort the contents of a list

Question 4-(ii: 18 pts) The complete C++ code for the singly linked list-based implementation to sort the contents of a list

Submit a <u>single PDF file</u> 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 (3 pts) Screenshot of the execution of the code with at least 10 elements and maximum value of 100.

Question 2 (3 pts) Screenshots of the execution of the code with 10 elements and 11 elements (with a maximum value of 25 in each case).

Question 3 (2.5 pts) Screenshot of the execution of the code as mentioned above (list of 15 elements, with a maximum value of 10 so that some elements repeat and you try to delete one of such repeating elements).

Question 4-(iii: 4 pts) A table that displays the average sorting times (in milliseconds) from 1000 to 20000 (in increments of 1000) for the two implementations

Question 4-(iv: 8 pts) An Excel plot that displays the list size vs. average sorting times for the two implementations (in a single chart) as well as displays the equations of the fitted curves and the R^2 values.

Question 4-(v: 2 pts) Based on the results observed in (iii)/(iv), what is your conclusion? Which implementation (dynamic array or singly linked list or either of them give comparable performance) would you suggest for the Selection Sort algorithm?