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

<u>Exam 1 (Take Home)</u>

Submission: Submit everything together as one PDF file in Canvas. Due: March 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.

You need to submit the following as part of your answer for this question:
(i) the complete code for the Node class, List class (including the recursivePrintForwardReverseOrders( ) function) and the main function.
(ii) Snapshot of the execution of the code with at least 10 elements and maximum value of 100.

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.

You need to submit the following as part of your answer for this question:
(i) the complete code for the Node class, List class (including the pairwiseSwap( ) function) and the main function.
(ii) Snapshots of the execution of the code with 10 elements and 11 elements (with a maximum value of 25 in each case).

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.

You need to submit the following as part of your answer for this question:
(i) the complete code for the Node class, List class (including the modified version of the deleteElement(deleteData) function) and the main function.
(ii) 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).

Q4 - 25 pts)
Implement Stack ADT as a Singly Linked List without 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 from the beginning of the linked list and to read the element value from the beginning of the linked list. To help you out, the implementation of the push function is given in the startup code provided. Your task is to implement the peek and pop functions like this without calling any other function. You can notice that the insertAtIndex, deleteElement and readIndex functions have been removed from the Stack class and you should not use them.

After implementing the pop and peek functions, you will be comparing the actual run-time of the push and pop operations of a Stack implemented as a Singly Linked List (with insertions and deletions in the beginning of the Linked List) with that of a Stack implemented as a Doubly Linked List (with insertions and deletions at the tail/end of the Linked List). The main function provided to you has the timers setup for this purpose. Your task is to just run the main functions and measure the average time taken (in microseconds) for the push and pop operations with the Stack as a Singly Linked List and with the Stack as a Doubly Linked List for the following values of the parameters: (i) # elements to be pushed = 1000, 10000, 100000, 1000000; (ii) maximum value for any element = 50000 and (iii) # trials = 50.

You need to include the following as part of your answer to this question:
(i) the code for the Singly Linked List-based implementation of the Stack class
(ii) a table presenting the actual run-times for the parameters mentioned above
(iii) snapshots of the actual run-times for the above cases.
(iv) interpretation of the difference/similarity in the actual run-times for the push and pop operations between the Singly Linked List and Doubly Linked List implementation of the Stack ADT.

Q5 - 25 pts)
Design and implement an algorithm to determine the next greater element of an element in an array in $\Theta(n)$ time, where 'n' is the number of elements in the array. You could use the Stack ADT for this purpose.

The next greater element (NGE) for an element at index i in an array A is the element that occurs at index j ($i < j$) such that $A[i] < A[j]$ and $A[i] \geq A[k]$ for all $k \in [i+1..., j-1]$. That is, index j ($j > i$) is the first index for which A[j] > A[i]. If no such index j exists for an element at index i, then the NGE for the element A[i] is considered to be -1.

For example: if an array is {1, 15, 26, 5, 20, 17, 36, 28}, then the next greater element (NGE) of the elements in the array are as follows:

1	15
15	26
26	36
5	20
20	36
17	36
36	-1
28	-1

Note: Your code need not determine and print the elements and their NGE values in the same order of the appearance of the elements in the array. An out of order print is also acceptable as long as the correct NGE for an element is printed out. For example, the following out of order print for the above array is also acceptable.

1	15
15	26
5	20
17	36
20	36
26	36
28	-1
36	-1

You are given a code file to run the algorithm on a smaller array (of size 10 integers) and print the output (i.e., the NGE for each element in the array, in the order determined by your algorithm). You should implement the algorithm in the main function itself and make use of the **Stack stack** object as part of the processing needed by your algorithm.

You are given the doubly linked list-based implementation code for Stack and there is no need to make any changes in the Stack class.

You need to submit the following as part of your answer for this question:
(i) Write a pseudo code of the algorithm designed and implemented for this problem.
(ii) Discuss the time complexity of the algorithm and justify that it is **Θ(n) with respect to the number of pop operations**.
(iii) Include the main function that has the implementation of the algorithm) that you come up with. There is no need to submit the doubly-linked list implementation code for the Stack class.
(iv) Show a screenshot of the execution of your algorithm for an array of 10 elements whose maximum value could be 50.