Lab 1 – Linear Data Structures

Release: 14 Sep 2020 (Mon, Week 5)
Due: Online submission in eLearn by Sunday, 20 Sep 2020, 11pm

Instructions:

1. This is an individual lab.
2. If need be, you can discuss the general approach to solve the questions within your assigned study groups but you should not be discussing or sharing your codes with your team members or anyone, including in Piazza. Piazza should only be used for the purpose of clarifying the requirements (if any).
3. Unless specifically stated in the question, you are allowed to create additional methods in the classes.
4. Please ensure that all your codes can be compiled.
5. Please include meaningful comments at the block of codes that you have written to explain your logic.
6. We may use additional test cases other than the ones provided to test the submissions.
7. Multiple submissions are allowed and we will take the submission closest to the deadline as the final submission.
8. Please zip up your submission as named the zip file as <StudentName>.zip.
9. The deadline is final, late submissions will not be accepted.

Grading Guidelines : (5% of final grade)

Grade	Expectations
A and above	Complete and working solutions with clear and concise comments to support the implementation. Implementations are efficient with the appropriate use of data structures, algorithms and considerations.
B and above	Complete and working solutions with clear and concise comments to support the implementation
C and below	Incomplete submissions, or some codes cannot be compiled.

Q1 – Swapping Elements in a Single Linked List

An implementation of a Single Linked List has been given in SinglyLinkedList.java. Study the codes and complete the following requirements in SinglyLinkedList.java to extend the functionality of the linked list. Please ensure the head and tail nodes are set accordingly if they are impacted by the method. Compile and run SinglyLinkedListTest.java to test your implementation.

- Write a method named **swap** that swap the sequence of all the elements whereby the biggest value will be swapped with the lowest value, the second biggest value will be swapped with the second lowest value….
- You can assume all the values are unique.

Before Swap :
Linked List : 3 2 1
First Element : 3
Last Element : 1

After Swap :
Linked List : 1 2 3
First Element : 1
Last Element : 3

Before Swap :
Linked List : 5 1 2 3
First Element : 5
Last Element : 3

After Swap :
Linked List : 1 5 3 2
First Element : 1
Last Element : 2

Before Swap :
Linked List : 5 3 4 1 2
First Element : 5
Last Element : 2

After Swap :
Linked List : 1 3 2 5 4
First Element : 1
Last Element : 4

Q2 – Grouping Nodes within a Double Linked List

An implementation of a Double Linked List has been given in SinglyLinkedList.java. Study the codes and complete the following requirements in DoublyLinkedList.java to extend the functionality of the linked list. Please ensure the header and trailer nodes are set accordingly if they are impacted by the method. Compile and run DoublyLinkedListTest.java to test your implementation.

- Write a method named **group** that will group all the nodes with a null values together while preserving the order of the non-null values. Refer to the examples on the order of the expected output.

Before Group :
Linked List : 1 null 2
First Element : 1
Last Element : 2

After Group :
Linked List : null 1 2
First Element : null
Last Element : 2

Before Group :
Linked List : 4 null 1 null 3
First Element : 4
Last Element : 3

After Group :
Linked List : null null 4 1 3
First Element : null
Last Element : 3

Q3 – Sorting a Stack with recursion

An implementation of an ArrayStack has been given in ArrayStack.java. You are not supposed to modify the ArrayStack class and the Stack interface. Compile and run StackTest.java to test your implementation.

- Write a method named **sort** in StackTest.java to sort the stacks in descending order
- You are supposed to use recursion for this implementation
- You are allowed to create additional methods in StackTest.java

Before Sorting :
(3, 5, 2, 4, 1)
After Sorting :
(5, 4, 3, 2, 1)

Q4 – Checking for Palindrome using different data structures

Palindrome are words that follow the same sequence in both directions e.g. civic, madam, refer… Do note that it's not case sensitive thus Civic and civic are both considered as palindrome. Explore the use of the different structures (stacks and queues) to check for Palindrome words. Compile and run PalindromeTest.java to test your implementation.

- Write a method named **checkPalindromeStack** in PalindromeTest.java to implement this with the use of stack
- Write a method named **checkPalindromeQueue** in PalindromeTest.java to implement this with the use of queue
- Write a method named **checkPalindromeStackAndQueue** in PalindromeTest.java to implement this with the combined use of stack and queue.

  abc
  Palindrome(Stack) : false
  Palindrome(Queue) : false
  Palindrome(Stack&Queue) : false

  refer
  Palindrome(Stack) : true
  Palindrome(Queue) : true
  Palindrome(Stack&Queue) : true

  Madam
  Palindrome(Stack) : true
  Palindrome(Queue) : true
  Palindrome(Stack&Queue) : true

Q5 – Extension of Music App

In Lab 0, there were requirements given to implement a Music App application which makes use of a Double Linked List. Extend the functionalities of the Music App player by building a new feature that makes use of Stack, Queue, Circular Queue, Priority Queue or Deque. (Choose any 1). Please include comments at the start of your MyMusicApp class to describe the new feature which uses the additional data structure.

```
public class MyMusicApp {

    // Please include comments here to describe the feature
    // which uses the additional data structure

    public static void main(String[] args) {

    }
}
```