**Problem Set 3 – Solution: Linked Lists, Recursion**

1. \*Given the following definition of a circular linked list (CLL) class:
3. public class Node {
4. public String data;
5. public Node next;
6. public Node(String data, Node next) {
7. this.data = data; this.next = next;
8. }
9. }
10. public class LinkedList {
11. private Node rear; // pointer to last node of CLL
12. ...
13. }

The class keeps a circular linked list, with a rear pointer to the last node.

Implement the following method in the LinkedList class, to delete the *first* occurrence of a given item from the linked list. The method returns true if the item is deleted, or false if the item is not found.

public boolean delete(String target) {

/\* COMPLETE THIS METHOD \*/

}

1. \* Implement a method in the circular linked list class of problem 1, to add a new item *after* the first occurrence (from the front) of a specified item. If the item does not exist in the list, the method should return false, otherwise true.
3. public boolean addAfter(String newItem, String afterItem) {
4. /\* COMPLETE THIS METHOD \*/
5. }
6. **WORK OUT THE SOLUTION TO THIS PROBLEM ON PAPER, AND TURN IT IN AT RECITATION**

A *doubly linked list* (DLL) is a linked list with nodes that point both forward and backward. Here's an example:

3 <---> 5 <---> 7 <---> 1

Here's a DLL node definition:

public class DLLNode {

public String data;

public DLLNode prev, next;

public DLLNode(String data, DLLNode next, DLLNode prev) {

this.data = data; this.next = next; this.prev = prev;

}

}

The next of the last node will be null, and the prev of the first node will be null.

Implement a method to move a node (given a pointer to it) to the front of a DLL.

// moves target to front of DLL

public static DLLNode moveToFront(DLLNode front, DLLNode target) {

/\*\* COMPLETE THIS METHOD \*\*/

}

1. Implement a RECURSIVE method to delete all occurrences of an item from a (non-circular) linked list. Use the Node class definition of problem 1. Return a pointer to the first node in the updated list.
3. public static Node deleteAll(Node front, String target) {
4. /\* COMPLETE THIS METHOD \*/
5. }
6. \*  Implement a RECURSIVE method to merge two **sorted** linked lists into a single **sorted** linked list WITHOUT duplicates. No new nodes must be created: the nodes in the result list are a subset of the nodes in the original lists, rearranged appropriately. You may assume that the original lists do not have any duplicate items.

For instance:

l1 = 3->9->12->15  
 l2 = 2->3->6->12

should result in the following:

2->3->6->9->12->15

Assuming a **Node** class defined like this:

public class Node {

public int data;

public Node next;

}

Complete the following method:

public static Node merge(Node frontL1, Node frontL2) {

...

}