Problem Set 3

Linked Lists, Recursion

---

1. *Given the following definition of a circular linked list (CLL) class:

```
public class Node {
  public String data;
  public Node next;
  public Node(String data, Node next) {
     this.data = data; this.next = next;
  }
}

public class LinkedList {
  private Node rear;  // pointer to last node of CLL
  ...
}
```

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 */
}
```

---

2. * 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.

```
public boolean addAfter(String newItem, String afterItem) {
   /* COMPLETE THIS METHOD */
}
```

---

3. 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 **/
}
```

---

4. **WORK OUT THE SOLUTION TO THIS PROBLEM ON PAPER, AND TURN IT IN AT RECITATION**

   With the same DLLNode definition as in the previous problem, implement a method to reverse the sequence of items in a DLL. Your code should NOT create any new nodes - it should simply resequence the original nodes. The method should return the front of the resulting list.

```
public static DLLNode reverse(DLLNode front) {
    /** COMPLETE THIS METHOD **/
}
```

---

5. 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.

```
public static Node deleteAll(Node front, String target) {
  /* COMPLETE THIS METHOD */
}
```

---

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) {
  ...
}
```

---

7. ** Implement a RECURSIVE method to reverse the sequence of items in a linked list. The nodes of the linked list are instances of the Node class defined in the previous problem. Your method should return a pointer to the front node of the reversed list. It should NOT create any new nodes, only resequence the nodes of the original list.

```
public static Node reverse(Node front) {
  /** COMPLETE THIS RECURSIVE METHOD **/
}
```