**Problem Set 2: Linked Lists**

1. **WORK OUT THE SOLUTION TO THIS PROBLEM (ON PAPER, YOU DON'T NEED TO COMPILE AND RUN IT) AND TURN IT IN AT NEXT WEEK'S RECITATION**

Assuming an **IntNode** class defined like this:

public class IntNode {

public int data;

public IntNode next;

public IntNode(int data, IntNode next) {

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

}

public String toString() {

return data + "";

}

Implement a method that will add a new integer before a target integer in the list. The method should return a pointer/reference to the front node of the resulting list. If the target is not found, it should return null:

public static IntNode addBefore(IntNode front, int target, int newItem) {

/\* COMPLETE THIS METHOD \*/

}

1. Given the following definition of a StringNode class:
3. public class StringNode {
4. public String data;
5. public StringNode next;
6. public StringNode(String data, StringNode next) {
7. this.data = data; this.next = next;
8. }
9. public String toString() {
10. return data;
11. }
12. }

Implement a method that will search a given linked list for a target string, and return the number of occurrences of the target:

public static int numberOfOccurrences(StringNode front, String target) {

/\* COMPLETE THIS METHOD \*/

}

1. \* Assuming the IntNode class definition of problem 1, implement a method to delete EVERY OTHER item from an integer linked list. For example:
2. before: 3->9->12->15->21  
    after: 3->12->21
3. before: 3->9->12->15  
    after: 3->12
4. before: 3->9  
    after: 3
5. before: 3  
    after: 3

If the list is empty, the method should do nothing.

public static void deleteEveryOther(IntNode front) {

/\* COMPLETE THIS METHOD \*/

}

1. \* With the same StringNode definition as in the previous problem, implement a method that will delete all occurrences of a given target string from a linked list, and return a pointer to the first node of the resulting linked list:
3. public static StringNode deleteAllOccurrences(StringNode front, String target) {
4. /\* COMPLETE THIS METHOD \*/
5. }
6. \*  Implement a (NON-RECURSIVE) method to find the common elements in two **sorted** linked lists, and return the common elements in **sorted** order in a NEW linked list. The original linked lists **should not** be modified. So, for instance,

l1 = 3->9->12->15->21  
 l2 = 2->3->6->12->19

should produce a new linked list:

3->12

You may assume that the original lists do not have any duplicate items.

Assuming an **IntNode** class defined like this:

public class IntNode {

public int data;

public IntNode next;

public IntNode(int data, IntNode next) {

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

}

public String toString() {

return data + "";

}

Complete the following method:

// creates a new linked list consisting of the items common to the input lists

// returns the front of this new linked list, null if there are no common items

public IntNode commonElements(IntNode frontL1, IntNode frontL2) {

...

}