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 the last value in the linked list whose first node is pointed to by front. (In other words, the added integer will become the second-to-last item in the resulting linked list.) The method should return a pointer/reference to the front node of the resulting linked list. If the input linked list is empty, the method should return null, without doing anything.

```
 public static IntNode addBeforeLast(IntNode front, int item) {
     /* COMPLETE THIS METHOD */
 }
```

---

2. Given the following definition of a StringNode class:

```
public class StringNode {
  public String data;
  public StringNode next;
  public StringNode(String data, StringNode next) {
     this.data = data; this.next = next;
  }
  public String toString() {
     return data;
  }
}
```

   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 */
```

```
}
```

---

3. * Assuming the IntNode class definition of problem 1, implement a method to delete EVERY OTHER item from an integer linked list. For example:

```
before: 3->9->12->15->21
 after: 3->12->21


before: 3->9->12->15
 after: 3->12


before: 3->9
 after: 3


before: 3
 after: 3
```

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

```
public static void deleteEveryOther(IntNode front) {
    /* COMPLETE THIS METHOD */
}
```

---

4. * 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:

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

---

5. * 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) {
  ...
}
```
