Problem Set 2 - Solution

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 */
}
```

   SOLUTION

```
public static IntNode addBeforeLast(IntNode front, int item) {
  if (front == null) {
    return null;
  }
  IntNode prev=null, ptr=front;
  while (ptr.next != null) {
     prev = ptr;
     ptr = ptr.next;
  }

  IntNode temp = Intnew Node(item, ptr); // next of new node should point to last
  if (prev == null) { // added item is first, so new node will be new front
    return temp;
  }
  prev.next = temp;
  return front;  // front is unchanged
}
```

---

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 */
}
```

SOLUTION

```
public static int numberOfOccurrences(StringNode front, String target) {
  int count=0;
  for (StringNode ptr=front;ptr != null;ptr=ptr->next) {
     if (target.equals(ptr.data)) {
        count++;
  }
  return count;
}
```

---

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 */
}
```

SOLUTION

```
public static void deleteEveryOther(IntNode front) {
  if (front == null) {
    return;
  }
  Node prev=front, ptr=front.next;
  boolean tbd=true;
  while (ptr != null) {
    if (tbd) {
      ptr = ptr.next;   // advance to after item to be deleted
      prev.next = ptr;  // bypass item to be deleted
      tbd = false;      // next item should not be deleted
    } else {
      prev = ptr;       // don't delete this (ptr) item, advance prev and ptr
      ptr = ptr.next;
      tbd = true;       // but mark next item for deletion
    }
  }
}
```

---

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 */
}
```

SOLUTION

```
public static StringNode deleteAllOcurrences(StringNode front, String target) {

 if (front == null) {
   return null;
 }

 StringNode curr=front, prev=null;

 while (curr != null) {
   if (curr.data.equals(target)) {
     if (prev == null) {      // target is the first element
       front = curr.next;
     } else {
       prev.next = curr.next;
     }
   } else {
     prev = curr;
   }
   curr = curr.next;
```

```
 }

 return front;
}
```

---

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) {
     ...
   }
   ```

   SOLUTION

   ```
   public IntNode commonElements(IntNode frontL1, IntNode frontL2) {
     IntNode first=null, last=null;
     while (frontL1 != null && frontL2 != null) {
       if (frontL1.data < frontL2.data) {
         frontL1 = frontL1.next
       } else if (frontL1.data > frontL2.data) {
         frontL2 = frontL2.next;
       } else {
         IntNode ptr = new IntNode(frontL1.data, null);
         if (last != null) {
           last.next = ptr;
         } else {
   ```

```
            first = ptr;
          }
          last = ptr;
          frontL1 = frontL1.next;
          frontL2 = frontL2.next;
        }
      }
      return first;
    }
```
