CSE143X Section #8 Problems

For these problems, assume that we are using the standard ListNode class:

```
    public class ListNode {
        public int data;       // data stored in this node
        public ListNode next;  // link to next node in the list

        // post: constructs a node with data 0 and null link
        public ListNode() {
            this(0, null);
        }

        // post: constructs a node with given data and null link
        public ListNode(int data) {
            this(data, null);
        }

        public ListNode(int data, ListNode next) {
            this.data = data;
            this.next = next;
        }
    }
```

And that we are writing methods for a class called LinkedIntList that has a single data field of type ListNode called front:

```
    public class LinkedIntList {
        private ListNode front;

        <methods>
    }
```

In solving these problems, you may not call any other methods of the class.

1. Write a method hasTwoConsecutive that returns whether or not a list of integers has two adjacent numbers that are consecutive integers (returning true if such a pair exists and returning false otherwise). For example, if a variable called list stores the following sequence of values:

```
     [1, 18, 2, 7, 8, 39, 18, 40]
```

   then the call:

```
     list.hasTwoConsecutive()
```

   should return true because the list contains the adjacent numbers (7, 8) which are a pair of consecutive numbers. If instead the list had stored this sequence of values:

```
     [1, 18, 17, 2, 7, 39, 18, 40, 8]
```

   then the method should return false. This sequence contains some pairs of numbers that could represent consecutive integers (e.g., 1 and 2, 7 and 8, 39 and 40), but those pairs of numbers are not adjacent in the sequence. The list also has a pair of adjacent numbers (18, 17) that are not in the right order to be considered consecutive. You may not make any assumptions about how many elements are in the list.

2. Write a method isSorted that returns true if the list is in sorted (nondecreasing) order and that returns false otherwise. An empty list is considered to be sorted.

3. Write a method stutter that doubles the size of the list by replacing every integer in the list with two of that integer. For example, if the list stores the following sequence of integers when the method is called:

```
[1, 8, 19, 4, 17]
```

It should store the following sequence of integers after stutter is called:

```
[1, 1, 8, 8, 19, 19, 4, 4, 17, 17]
```

4. Write a method reverse3 that reverses each successive sequence of 3 values in a list of integers. For example, suppose that a variable list stores:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

and we make the following call:

```
list.reverse3();
```

Afterwards the list should store the following sequence of values:

```
[3, 2, 1, 6, 5, 4, 9, 8, 7, 12, 11, 10, 15, 14, 13]
```

The first sequence of 3 values (1, 2, 3) has been reversed to be (3, 2, 1). The second sequence of 3 values (4, 5, 6) has been reverse to be (6, 5, 4). And so on. If the list has extra values that are not part of a sequence of 3, those values are unchanged. For example, if the list had instead stored:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]
```

The result would have been:

```
[3, 2, 1, 6, 5, 4, 9, 8, 7, 12, 11, 10, 15, 14, 13, 16, 17]
```

Notice that the values (16, 17) are unchanged in position.

The list will not always contain sequential integers. The following list:

```
[3, 8, 19, 42, 7, 26, 19, -8, 193, 204, 6, -4, 99]
```

would be rearranged as follows:

```
[19, 8, 3, 26, 7, 42, 193, -8, 19, -4, 6, 204, 99]
```

Your method should not change the list if it has fewer than three values.

5. Write a method removeAll that removes all occurrences of a particular value. For example, if the list contains the following values:

```
[3, 9, 4, 2, 3, 8, 17, 4, 3, 18]
```

The method call:

```
list.removeAll(3);
```

removes all occurrences of the value 3 from the current list, yielding:

```
[9, 4, 2, 8, 17, 4, 18]
```

If the list is empty or the value doesn't appear in the list at all, then the list should not be changed by your method. You must preserve the original order of the list.

6. Write a method removeEvens that removes the values in even-numbered positions from a list, returning those values in their original order as a new list. For example if a variable called list1 stores these values:

```
list1: [8, 13, 17, 4, 9, 12, 98, 41, 7, 23, 0, 92]
```

Then the following call:

```
LinkedIntList list2 = list1.removeEvens();
```

Should result in list1 and list2 storing the following values:

```
list1: [13, 4, 12, 41, 23, 92]
list2: [8, 17, 9, 98, 7, 0]
```

Notice that the values stored in list2 are the values that were originally in even-valued positions (index 0, index 2, index 4, and so on) and that these values appear in the same order as in the original list. Also notice that the values left in list1 also appear in the same order as in the original list.

Recall that LinkedIntList has a zero-argument constructor that returns an empty list. You may not call any methods of the class other than the constructor to solve this problem. You are not allowed to create any new nodes or to change the values stored in data fields to solve this problem. You MUST solve it by rearranging the links of the list.

7. Write a method doubleList that doubles the size of a list by appending a copy of the original sequence to the end of the list. For example, if a variable called list stores this sequence of values:

```
[1, 3, 2, 7]
```

and we make the following call:

```
list.doubleList();
```

then it should store the following values after the call:

```
[1, 3, 2, 7, 1, 3, 2, 7]
```

Notice that it has been doubled in size by having the original sequence appearing two times in a row.

You may not make assumptions about how many elements are in the list. You may not call any methods of the class to solve this problem. If the original list contains n nodes, then you should construct exactly n nodes to be added to the list. You may not use any auxiliary data structures to solve this problem (no array, ArrayList, stack, queue, String, etc). Your method should run in O(n) time where n is the number of nodes in the list.

8. Write a method reverse that reverses the order of the elements in the list. For example, if the list initially stores this sequence of integers:

```
[1, 8, 19, 4, 17]
```

It should store the following sequence of integers after reverse is called:

```
[17, 4, 19, 8, 1]
```