CIT 594 Module 2 Programming Assignment - Part 1

In this part of the assignment, you will implement three methods that perform functions on a LinkedList, using an implementation similar to the one seen in this module's videos.

Learning Objectives
In completing this assignment, you will:

- Become familiar with the functionality and implementation of a LinkedList
- Apply what you have learned about how LinkedLists store data
- Develop algorithms for modifying LinkedList content

Getting Started
Download the **MyLinkedList.java** file, which contains the implementation of a LinkedList of Strings, as well as the unimplemented methods for the code that you will write in this assignment.

Activity
Implement the following methods in the **MyLinkedList.java** file:

reverse: This method reverses the order of the elements in the LinkedList. Hence the name. :-) Don't forget to update the LinkedList's *head* and *tail*!

removeMaximumValues: This method removes all instances of the *N* largest **values** in the LinkedList. Because the values are Strings, you will need to use the String class' **compareTo** method to compare the Strings; see the Java API for help with that method. If the input value *N* is zero or negative, this method should simply return without any modifications to the LinkedList. The other elements in the LinkedList should not be modified and their relative order must not be changed.

For instance, if the LinkedList contained the following:

DOG → BANANA → CAT → GORILLA → BEAR

Then removeMaximumValues(1) would remove the largest value, which is "GORILLA" and the LinkedList would then look like this:

DOG → BANANA → CAT → BEAR

However, given that same LinkedList, removeMaximumValues(2) would remove the two largest values, which are “GORILLA” and “DOG”, so the LinkedList would then look like this:

BANANA → CAT → BEAR

Keep in mind that if any of the *N* largest values appear more than once in the LinkedList, this method should return remove *all* instances, so it may remove more than *N* elements overall.

So for instance if we had:

KANGAROO → PLATYPUS → AARDVARK → KANGAROO → DONKEY → COYOTE

And called removeMaximumValues(2), it would remove *all* instances of the largest value (“PLATYPUS”) and *all* instances of the second-largest value (“KANGAROO”) and we would then have this:

AARDVARK → DONKEY → COYOTE

containsSubsequence: This method determines whether any part of the this LinkedList contains all elements of the input LinkedList in the same order with no other elements in the sequence, i.e. it should return true if the input LinkedList is a subsequence of this object, and false if it is not. The method should return false if either input is null or empty.

For instance, let’s say our LinkedList looked like this:

HIPPOPOTAMUS → GIRAFFE → LION → GAZELLE → ZEBRA → CHEETAH

Then containsSubsequence should return true for the following LinkedLists, all of which are considered subsequences:
HIPPOPOTAMUS → GIRAFFE → LION
ZEBRA
LION → GAZELLE → ZEBRA → CHEETAH

There are, of course, others besides these that should cause containsSubsequence to return true.

However, the following LinkedList inputs should cause containsSubsequence to return false:
GIRAFFE → LEOPARD → LION
LION → GIRAFFE
ZEBRA → CHEETAH → OSTRICH

And so on.

Please do not change the signatures of these three methods (their parameter lists, names, and return value types) and do not create any additional .java files for your solution. If you need additional classes, you can define them in *MyLinkedList.java*. Additionally, please be sure that your *MyLinkedList* class is in the default package, i.e. there is no "package" declaration at the top of the source code.

Helpful Hints

Keep in mind that you can use the *add*, *contains*, *remove*, and *get* methods of the *MyLinkedList* class as you implement the other three methods, and you can implement other helper methods as well. Try to avoid needing to manipulate the Nodes and links in the methods you write, but rather try to use the methods that are provided.

Before You Submit

Please be sure that:

- your *MyLinkedList* class is in the default package, i.e. there is no "package" declaration at the top of the source code
- your *MyLinkedList* class compiles and you have not changed the signature of the *reverse*, *removeMaximumValues*, or *containsSubsequence* methods
- you have not created any additional .java files

How to Submit

After you have finished implementing the *MyLinkedList* class, go to the "Module 2 Programming Assignment (Part 1) Submission" item and click the "Open Tool" button to go to the Codio platform.

Once you are logged into Codio, read the submission instructions in the README file. Be sure you upload your code to the "submit" folder.

To test your code before submitting, click the "Run Test Cases" button in the Codio toolbar.

Unlike the Module 1 Programming Assignment, **this will run some but not all of the tests that are used to grade this assignment.** That is, there **are** "hidden tests" on this assignment!

The test cases we provide here are "sanity check" tests to make sure that you have the basic functionality working correctly, but **it is up to you to ensure that your code satisfies all of the requirements described in this document.** Just because your code passes all the tests when you click "Run Test Cases" doesn't mean you'd get 100% if you submit the code for grading!

When you click “Run Test Cases,” you’ll see quite a bit of output, even if all tests pass, but at the bottom of the output you will see the number of successful test cases and the number of failed test cases.

You can see the name and error messages of any failing test cases by scrolling up a little to the “Failures” section.

Assessment

This assignment is scored out of a total of 48 points.

The reverse method is worth a total of 11 points, based on whether the elements are correctly reversed and the head and tail are correctly updated.

The removeMaximumValues method is worth a total of 20 points, based on whether the elements are correctly removed and other elements are in the correct positions.

The containsSubsequence method is worth a total of 17 points, based on whether the method correctly indicates that one LinkedList is a subsequence of the other.

As noted above, the tests that are executed when you click “Run Test Cases” are **not** all of the tests that are used for grading. There are “hidden” tests for each of the three methods described here.

After submitting your code for grading, you can go back to this assignment in Codio and view the “results.txt” file, which should be listed in the Filetree on the left. This file will describe any failing test cases.