**CSC220 Assignment08**

**Linked Lists**

The goal of this week’s assignment is:

1. Work with linked structure

2. Learn about the importance of debugging

**Things you must do:**

1. There are many details in this assignment. Make sure you read the whole thing carefully before writing any code and closely follow this instruction.

2. You must complete your assignment **individually**.

3. Always remember Java is case sensitive.

4. Your file names, class names, package name, and method signatures must match exactly as they are specified here.

**Things you must not do:**

1. You must not change the file names, class names, package names.

2. You must not change the signature of any of these methods (name, parameters, …).

**3.** **DO NOT modify the implementation of the methods provided.**

**Important Note: You are going to write a few functions this week but there are many details involved. Start early! We are providing exhaustive test cases here for you. Focus on getting your code to work for all these test cases and watch for infinite loops!**

**Part 0**

• You first must make sure that you have already finished the lab successfully and have all methods in the lab instruction working properly.

• **DO NOT** start your assignment unless you have all the features in the labworking.

**Part 1 – The problem description**

For this lab, you are going to work on implementing various methods for a linked list that holds integer values. The main functionality of the class has been implemented for you. You are responsible for implementing the remaining methods based on the instruction below. You MUST follow the instruction exactly and you are not allowed to change the method signatures.

Your first task is to familiarize yourself with the current methods provided to you. Make sure to go through each and every method that is implemented in **ListNode.java** and **LinkedIntList.java** and make sure you know what it does, and how you can use them.

**Part 2 – Implementation**

For this assignment, you are required to write two functions:

* public void stutter()

* This method will double the size of the list by replacing every integer in the list with two of that integer. If the list is empty, simply return.
* For example, suppose a variable list stores the values [1, 8, 19, 4, 17], after a call of list.stutter(), it should store [1, 1, 8, 8, 19, 19, 4, 4, 17, 17].

* public void shift()

* This method will rearrange the elements of a list of integers by moving to the end of the list all values that are in odd-numbered positions and otherwise preserve the list order. Remember, it does not matter whether the value itself is odd or even; what matters is whether the value appears in an odd index (index 1, 3, 5, etc). Also, the original order of the elements of the list should otherwise be preserved. You may not construct any new nodes nor use any auxiliary data structures to solve this problem. You also may not change any data fields of the nodes; you MUST solve this problem by rearranging the links of the list.

* For example, suppose that a variable list stores the values [10, 31, 42, 23, 44, 75, 86]. The call of list.shift(), should rearrange the list to store [10, 42, 44, 86, 31, 23, 75].

Again, you need to accomplish both tasks by changing the references. Do not create copies of the linked list. In this assignment, you should be very careful and mindful of infinite loops, (see the next section).

**Part 3 – Test your code**

As usual you need to test the functionality of the methods you have implemented. A set of comprehensive test has been provided for you as part of **LinkedIntListTester.java**. Uncomment the assignment portion of the tests and run the main function. If you see any red text that says “TEST FAILED”, you need to debug your code.

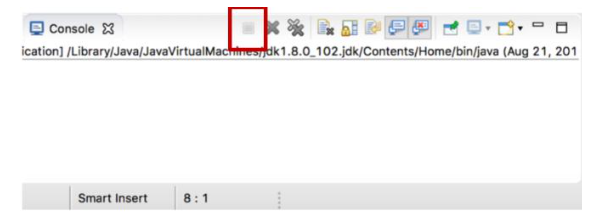
How to debug your code?

1. A toString method has been provided for you, use it to inspect the structure of your linked list.

2. Use the Eclipse debugger you learned about during the first lab

3. If you see JavaStackOverflow, that means that you have an infinite recursive call and your recursive call is filling up the “call stack” (we talked about this concept in class). Go back and debug the method that is causing the problem.

4. Infinite loops! How would you know you have an infinite loop? As you should know from CSC120, if you have an infinite loop in your code, your code will not stop running. An easy way to inspect that in Eclipse is to look at your console window, if your code is done running the console should look like the following



If the little square marked above is red and continues to stay red, that means your code has an infinite loop!

**Remarks**

* Make sure to submit your assignment by (re-)uploading your **Lab08** folder into your **csc220-cXXXX** folder by the deadline (note: 10/20 @ 11:59pm)
* **For all your assignments, please start early and seek help early (either from the instructor or the TAs).**