CS 0445 Spring 2023
Midterm Exam Projects on Linked Lists

Project 1

Introduction:

Recently in lecture we have discussed the list interface and the author's LList<T> class. In this project you will add some useful functionality to the LList<T> class that is not part of the list interface. Specifically, you will add a copy constructor and an equals() method to the LList<T> class. You should be familiar with these concepts from previous assignments. However, you implemented these in an array-based class before. Now you will implement them in a linked-list based class.

Specifically, you will implement the following methods:

```
// Creates a new list that contains the same data in the
// same order that was contained in old.  The lists should
// be separated from each other (so you must make new
// Nodes in the copy) but the data values within the lists
// can be shared (so the copy is not totally deep).
public LList(LList<T> old)

// Return true if the current LList<T> and the argument
// LList<T> contain the same data (based on the equals() method
// for class T) in the same order; return false otherwise
public boolean equals(LList<T> rhs)
```

Once you have completed your modified LList<T> class, test it with the main program MidProj1.java. Your output should match that shown in file P1Out.txt. To run this program you will need the following files:

LList.java (author's original class)

ListInterface.java (even though we are not using ListInterface<T> in the main program, since LList<T> implements ListInterface<T>, we still need the interface file.

MidProj1.java (main program to test your class)

P1Out.txt (output to use for comparison)

Hints:

For the copy constructor you will need to iterate through the nodes of your old LList<T>, making a new Node corresponding to each old Node (and containing the same data). This can be done with a loop, adding each new Node to the end of the list. Be careful to make sure that you are linking the Nodes correctly. Also be careful about "one off" errors – don't miss any Nodes (ex: first or last) and don't iterate too far (could give a NullPointerException).

For the equals() method you will need to iterate through both lists, keeping separate Node pointers as you proceed down each one. You should first test the lengths of the lists, since clearly if they are not the same the answer will be false – this will save you the trouble of having to deal with lists of different lengths in your loop.

Project 2

Introduction:

Now that we have discussed linked lists, it is a good idea to get some practical experience working with them. In this project you will implement a commonly used type in a somewhat uncommon way – using a circular doubly-linked list. You should be familiar with the **StringBuilder** class in Java. Recall that the String class in Java produces immutable objects, or, in plain terms, strings that are fixed once they have been created. This can be inefficient if operations such as appending to the end or inserting into the middle of a string are necessary. The StringBuilder class, on the other hand, allows for strings that can be modified. The predefined StringBuilder class is implemented with an underlying array of characters. In this project, you will implement your own class, called MyStringBuilder, which is similar in functionality to the standard StringBuilder, but is implemented using a **linked list of characters** rather than an array.

Hints:

I have provided the specifications for the MyStringBuilder class in the file MyStringBuilder.java. Download this file and use this as the starting point for your coding. **Read these specifications over very carefully, paying particular attention to the comments**. Note that your data must be as specified (you **may not** add any additional instance variables) and you must implement your class using your own **circular doubly-linked** list with the inner class CNode as it is written in MyStringBuilder.java.

You may not use any predefined linked list within your class, you may not use any predefined Java String-ish class (such as StringBuilder or StringBuffer) anywhere in your code, and you may not copy any code from the Internet. You may only use the String class when it is either an argument or a return type in one of the MyStringBuilder methods. Furthermore, your methods must act directly on the linked list (so, for example, you may not convert your MyStringBuilder to an array of chars and then perform the method on that array). You must also not do a lot of unnecessary work / extra traversals of your list or arguments in order to perform your methods. For example, in the replace() method, one inefficient approach is to first do a delete() of the specified characters and then do an insert() of the new String. This is not allowed, since it traverses the list twice, when only a single traversal is necessary. As another example, for the copy constructor, you cannot call toString() of the argument and then use the constructor that takes a String, since toString() will traverse the argument list and you will then also need to traverse the resulting String. Think about this issue for all of your methods.

Some of the methods will be relatively simple to implement while some will be more complicated. I recommend working on the methods one at a time, only proceeding to the next one when you are sure that the previous one works correctly. This way, any compilation or logic errors that occur will be fairly easy to locate and will be less difficult to fix. For most of the methods, be very careful to **handle special cases**, as we discussed in lecture. You may find the special cases to be the most challenging part of some methods.

To get you started, **I have implemented one of the constructors and the toString() method for you. See the code and the comments in MyStringBuilder.java. These methods may help you to see how to approach the other methods in your class.** Project 1 will also be very helpful – it is a different class and a singly- linked list, but once you understand how the copy constructor and equals() method for the LList<T> class work, you should better be able to implement some of the MyStringBuilder methods.

After you have completed your MyStringBuilder class, you will test it in two ways:

1) You will compile and run the program **MidProj2.java**. This program is provided to you and **must be used as is.** The idea is similar to that of the test files for previous assignment – the program is a simple driver that tests most of the functionality of the MyStringBuilder class. **MidProj2.java** will definitely test special cases of your operations, so be sure that these are handled. The output to this program is shown in file **P2Out.txt**. Your output should be **identical** to the data in this file.
2) You will compile and run the program **MidProj2B.java**. This program is also provided to you and demonstrates the efficiencies of some of the MyStringBuilder operations. The overall run-times when run with your MyStringBuilder class **will depend a lot of the computer / system in which the program is run**, but the trend demonstrated should be similar to that of the sample output shown in file **P2BOut.txt**. **Note that this program may take a few minutes to run.**