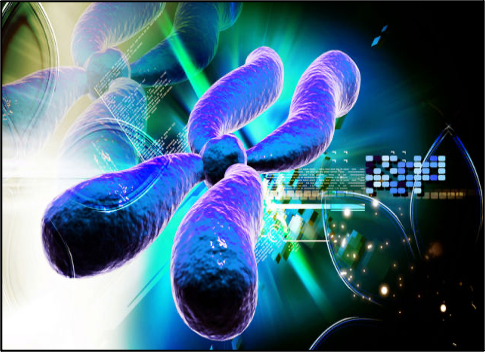
CS 46B

Spring 2023

Bonus Lab 9[[1]](#footnote-0)

7 bonus points​​



Transposons are short regions of chromosomes (DNA molecules) that are believed to play a role in the development of certain kinds of cancer, including lung and throat tumors.

In this lab you will finish the in class exercises on doubly linked lists including developing a set of test cases **(Note this part we didn’t do in class)**.

# Learning Outcomes

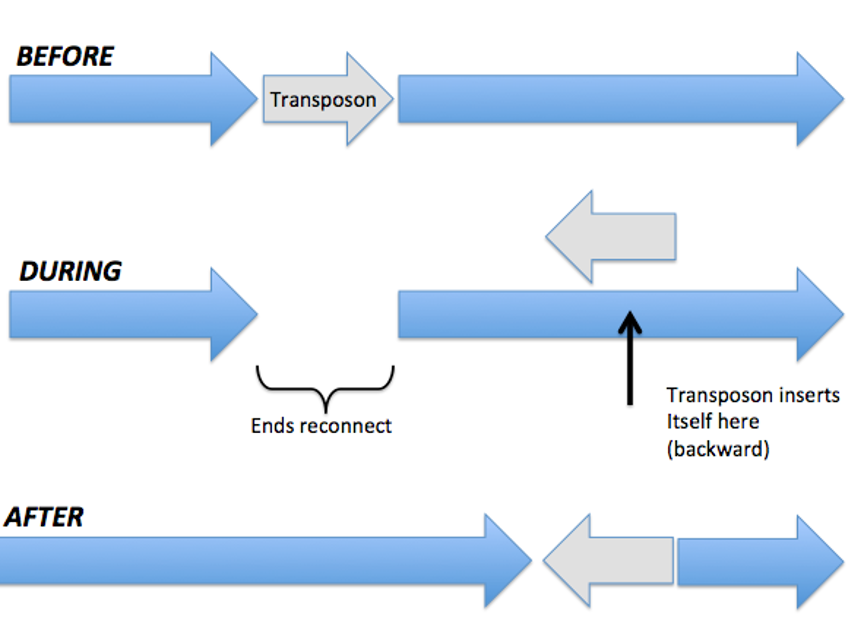
During this bonus lab you will learn how to do the following.

* Draw node diagrams for doubly linked lists
* Develop strategies for implementing code that manipulates chains of nodes
* Understand how to decompose a complex program into a set of simpler methods

# Background

Recall that a chromosome is a long molecule of DNA. DNA is a long chain composed of the chemical subunits (or “bases”) adenine, cytosine, guanine, and thymine, which are usually abbreviated as A, C, G, and T. In software, chromosomes are often modeled as strings of these characters. Since chromosomes can be hundreds of millions of bases long, the strings that model chromosomes can be hundreds of millions of characters long.

Sometimes a relatively short segment of DNA, called a transposon, drifts out of its chromosome for no apparent reason, turns itself around, and inserts itself backward into another part of the chromosome. The space left at the segment’s original location closes up again.



Often transposons do no harm. Sometimes they kill their cell, by inserting themselves in the middle of a unit of DNA (a gene) that programs some vital cell function. The function gets disrupted and the cell dies. But the death of 1 cell is no problem; you and I have around 60,000,000,000,000 cells. However, on rare occasions the transposon disrupts a gene that fights cancer. When this happens the cell doesn’t die, it mutates into a tumor cell. When that cell divides, the result is 2 identical cells that both have lost the ability to fight cancer. Then both of *those* cells divide and … well, you can do the math.

A tiny example: suppose a chromosome is ATCAGGGGG, where TC is the transposon and GGGGG is a sequence of instructions for anti-cancer defense. After transposition, the chromosome might be AAGGGGGCT, which would not be a problem; or it might be AAGGCTGGG, which would be a problem.

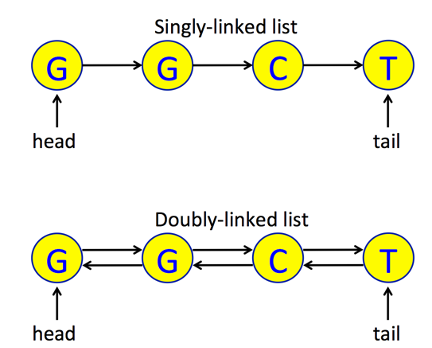
In order to understand transposons, we need to model them in software. Transposons are all about inserting and deleting. You have seen that for very long collections, inserting with a linked list can be significantly faster than inserting with an array list. The same is true for strings: manipulating a linked list is much faster than manipulating a string. To model transposons and battle cancer, we need to abandon array lists and strings, and write linked list code.

# Setup

Create an Eclipse workspace and a Java project in the workspace. Create a package called linked. Copy the starter file DNALinkedList.java into your package.

# Let’s Program

The code you write today will be a little different from the code you saw in previous lectures, where each node had a “next” variable. Today’s nodes will have both “next” and “previous” variables, and they will be chained together to form what are called *doubly-linked* lists. You will see how taking care of nodes is more work than with an ordinary (*singly-linked*) lists; the benefit is that certain operations on the lists, which would be hard or slow with a singly-linked list, are fast and easy with a doubly-linked list.



## DNALinkedList.java

This class contains a generic Node inner class like the examples you’ve seen in lecture, except it has a prev (short for “previous”) instance variable in addition to next. Note that the data instance variable has the generic type T.

This class uses the generic Node class. Note that every Node declaration is “Node<Character>” because every node’s data is one of the characters A, C, G, or T.

The constructors for this class are already written. Your job is to finish writing 7 methods in this class plus a main method for testing.

* append(Node<Character> n)
* matches(Node<Character> startNode, String target)
* find(String target)
* extract(Node<Character> firstExtractedNode, Node<Character> lastExtractedNode)
* insert(DNALinkedList insertMe, Node<Character> insertionPoint)
* reverse()
* transpose(String transposon, String target)
* main(String[] args). The comments in the methods will tell you what to do.

*Look for “???”, which you will replace with code that is correct and efficient. As you work, be aware that whenever 2 nodes are linked together or disconnected, you need to take care of the “next” of one of the nodes, and the “prev” of the other node.*

## Start Here for Lab

First, try to discuss and complete the first four methods (append, matches, find, extract)we talked about in class. Make sure that all of your group mates are at the same place and then begin working on implementing the last three methods (insert, reverse, and transpose).

**STOP**

*Make sure everyone in your group has successfully implemented the first five methods before moving on to the reverse method.*

### Reverse Method

The reverse method reverses the order of the nodes. Note that you are doing a similar problem for your homework, but on a singly linked list. Do you think the problem on your homework is harder or easier?

**STOP**

*Make sure everyone in your group has successfully implemented the reverse method before moving on to the transposon method.*

### Transposon Method

This method removes sequence matching transposon, reverses it, and inserts it back into this list immediately before target. It throws IllegalArgumentException if it can't find transposon or target. **Hint**: Use the methods you just wrote.

**STOP**

*Make sure everyone in your group has successfully implemented the transposon method before moving on to the main method.*

### Main Method

In order to complete the main() method, you will need to create your own test case. The code prompts you to specify a chromosome string, a transposon string, and a target string. These will be used to test the transpose() method, which calls all the others. One way to test would be to use a real chromosome. The shortest human chromosome is 48 million bases long. ***Why would that be a bad first test case?***

Think about what features a good first test case should have. ***What are these?*** (Hint: one feature is that they should be short enough that you can draw the diagrams described below.)

Now create the 3 strings chromosome, transposon, and target. ***What 3 strings did you choose? What do you expect the DNALinkedList to look like after the call to transpose()?***

On a piece of paper, draw diagrams like the yellow “Doubly-linked list” diagram above, showing the expected structure of your test case list at various points in transpose():

* At the start of the method.
* After finding firstNodeOfTransposon, firstNodeOfTarget, and lastNodeOfTransposon. Draw arrows pointing to each of these nodes.
* After extracting the transposon (also draw the extracted transposon).
* After the transposon is reversed (only the transposon list will change).
* At the end of transpose(), after the reversed transposon has been inserted back into the chromosome.

Now test your code. If you don’t see the expected result, look at the transpose() code. You can use the debugger, logging statements, or println statements, whatever method you prefer to look at the list and the transposon. That should give you all the information that you need. Which option did you choose and why?

Look at the extract() and insert() methods. ***For each method, would implementation with a singly linked list be a little harder, a lot harder, or about the same?***

Lastly: notice how you wrote simple methods, then more sophisticated methods that called the simple methods. The instructions made it easy. In real life you won't have instructions, you’ll just have a problem description. A strategy to ensure that you have internalized the concepts is to start again with the starter code for DNALinkedList, completely delete the starter code for all the methods you wrote, and write them without instructions.

### Optional Challenge

For the extract and insert methods, write code to handle the corner cases when you are extracting and/or inserting at the head or tail of the list.

# Saving your work

# It can be a good idea to back up your work in case you ever accidentally delete your eclipse-workspace. You can export your work as a jar file. You can import jar files directly into Eclipse. Later this semester we will learn better ways to backup your work using version control and github.

1. Modified from material provided by Dr. Philip Heller and Dr. Chakarov [↑](#footnote-ref-0)