

CIS 121 - Fall 2009

Homework 3 – Friday, October 9

Due Friday, October 16, 10am (Online submission)

Note: It you're using Eclipse, remove all the project files and declarations from your folders and only turn in the requested .java files!

2 Required Problems (100 points)

Problem 1: 40 Points; files to submit `SinglyLinkedList.java`, `MyDeque.java`

In this problem you will write a simple Deque (double-sided queue) using a Singly-Linked List. Deques are generally implemented using a doubly-linked list. A deque is a queue that allows removal and insertion from both sides of the queue. You can consult Wikipedia or your textbook for more information. We provide skeleton code for both the **SinglyLinkedList** and **MyDeque** (found at <http://www.seas.upenn.edu/~cis121/hws/hw03/hw03files.zip>); you must implement all of the methods in both classes. The methods are self-explanatory. A **SinglyLinkedList** object is just an element in a linked list, so it should have a reference to the next element in the list (another **SinglyLinkedList** object). The constructor for this object takes a generic element which it will store.

The **MyDeque** constructor must take the head of a linked list as an argument to the constructor. This is to force you to use a singly-linked list object to implement your Deque, otherwise you could use any other object (such as the Java implementation of Deque) to write your solution. Just to make this completely clear, you *MUST* use this object to implement your Deque. Our tester will feed this object to your program, and will be able to check that elements are in the correct place. Since we have given you this warning now, not using **SinglyLinkedList** to implement **MyDeque** is not a valid reason for a regrade. However, you may use anything youd like to test, including the Java implementation.

Note: Please make sure that your `prefix.java` can handle poorly formed input strings. If there is a character that is not an operator or not a number, or the string is poorly formed for another reason, please throw *IllegalArgumentException*

Problem 2: 60 Points; file to submit `prefix.java`

As you have already seen, our classical method of expressing algebraic expressions has the small

problem of being ambiguous in some cases.

$$2 * 3 + 4$$

could be interpreted in a couple of ways, either $6 + 4$ or $2 * 12$. There does also exist other methods for expressing algebra which is never ambiguous. Prefix notation is one of these techniques. Rather than putting the operators inbetween the operands (“infix”) in prefix the operator comes first. Thus our two interpretations of the first infix expression can be represented as:

$$+ * 234 = (2 * 3) + 4$$

$$*2 + 34 = (2 * (3 + 4))$$

Your task will be to take a string in prefix notation, and using stacks and queues evaluate its final value.

```
public static int calcPrefix(String input){  
  
}
```

Note: You only need to implement these five binary operators: +, -, *, /, ^. In addition, you can assume one-digit numbers, to preserve unambiguity in prefix expressions.