Lab Report 04

Nathan Bickel

Problem

The problem was to write a class that takes in a String that is an arithmetic problem written in Reverse Polish Notation. It should check to see if it is formatted correctly and doesn't create errors like dividing by 0—if it passes, it should return the result, and otherwise it should print the error to standard system output and return 0. The class should also use a flexible stack that is implemented in the form of a linked list.

Solution

The first step was to create the stack interface and the corresponding linked list implementation. Both took a generic type T, and the linked list class had an internal class called ListNode to store data and links together, as well as a ListNode head to keep track of the top of the stack and an int size to keep track of the size. The interface had 5 method signatures: **push**, **pop**, **peek**, **print**, and **size**, and the linked list class implemented them. **push** added data in: it took data as a parameter and added it to the list. If there was data, it created a new ListNode with the link pointing to the head, and then moved head to point to the new node and incremented the size. Otherwise, it pointed the head to the new ListNode. **pop** took data out: it created a variable pointing to the data in head, pointed head to its own link, decremented the size, and returned the variable. **peek** simply returned the data at the head without changing the list, **print** printed the data in every ListNode in the list out, and **size** returned the size.

Now that a stack was available to use in the calculator class, one using type Integer was declared as an instance variable and reinitialized every time the **calculate** method was called in the driver. To aid in calculation, three helper methods were created: **isInteger**, **isOperator**, and **calculateGivenInputs**. **isInteger** attempts to parse the parameter String as an integer—if it succeeds, it returns true, and it returns false if an exception is thrown. **isOperator** compares the parameter String to a regex containing the operators—if they match, it returns true, and it returns false otherwise. **calculateGivenInputs** is used once an operation is determined to be possible with given inputs, and it simply runs the calculation using a switch statement to determine which operation to run.

The most important method in the ReversePolishCalculator class was the **calculate** method, which takes a parameter String and returns an output (or 0 if there are errors). It first takes the String and splits it into a String array. Then, it iterates through the array, checking each String to see if it is an integer or operator (or neither, in which case it returns 0). If it is an integer, it pushes it onto the stack. If it is an operator, it checks that there are at least two values on the stack and pops them off, checks that it won't be dividing by 0, and then pushes the return of **calculateGivenInputs** given the two values and the operator back onto the stack. It continues doing this until the last element in the array is reached. If there is exactly one value left in the stack, it pops this off and returns it. If any of these checks fail, 0 is returned and the error is printed.

Implementation Problems Encountered

One issue I had was with the check that there are two values left on the stack for an operator to act on. It took me a while to debug this, but I eventually realized that I was checking the size after I had already popped off the two values, which clearly was the wrong order and needed to be switched.

Another problem I had was where to initialize the stack. At first, I wrote a constructor that initialized the stack there, but I realized that if something went wrong with a calculation, there could potentially still be values left in the stack when the **calculate** method is called on another String. I noticed this when I ran the String "2 + +" and it returned 11 because of leftover values. I then changed it to reinitialize the stack every time the **calculate** method was called, which fixed the problem.

Lab Report Questions

1. **Head** is indicated in bold and underlined

A.

10
9
8
7
6
5
4
3
2
1

B.

8
7
6
5
4
3
2
1

C.

15
14
13
12
11
8
7
6
5
4
3
2
1

D.

12
11
8
7
6
5
4
3
2
1

2. **Head** is indicated in bold and underlined

A.

1
2
3
4
5
6
7
8
9
10

B.

3
4
5
6
7
8
9
10

C.

3
4
5
6
7
8
9
10
11
12
13
14
15

D.

6
7
8
9
10
11
12
13
14
15