Department of Electrical and Computer Engineering
ECSE 202 – Introduction to Software Development
Assignment 4 – Interpreting Infix Expressions

due date March 26, 2018 at 5:00 pm.

Problem Description

In the previous assignment, you wrote a Java program that prompted the user for an infix expression and converted the result to postfix, displaying the result in the console window. In this assignment, instead of simply displaying the postfix, you will implement a simple interpreter that evaluates the expression, and returns an ordered sequence on one and two operand expressions that can subsequently be evaluated by a simple processor. The point of this assignment is to understand the Stack Interpreter outlined in the Appendix of Assignment 3.

Consider the following example: a+c*y-d*x

The corresponding Postfix expression is: a c y * + d x * -

Following the convention of Assignment 3, the expression would be evaluated as follows:

Token	Stack
a	a
c	a c
y	a c y
*	a <c*y>
+	<a + <c*y>>
d	<a + <c*y>> d
x	<a + <c*y>> d x
*	<a + <c*y>> <d*x>
-	<a + <c*y>> - <d*x>

The <> indicate the result of an operation which is subsequently placed back on the stack for further evaluation until all operators have been processed.

Your program should list these operations as follows:

```
Infix to Postfix Interpreter
Enter expression (blank line to exit): a+c*y-d*x
Eval1:<c*y>
Eval2:<a+Eval1>
Eval3:<d*x>
Eval4:<Eval2-Eval3>
```

where `EvalN` indicates a particular operation on two operands, with N specifying the order of the operation. In this example, 4 operations are required to interpret the input expression.

Method

The simplest approach is to keep your existing implementation for Assignment 3 and add in the additional code to implement the interpreter and print the result, as shown above. The basic algorithm for the stack interpreter is not too hard to figure out (this will be discussed in the tutorials).

Let's go through the example in detail with input: `a c y * + d x * -`

Notice that this example contains no unary operators (this is not required, but can of course be implemented). Tokens are pushed left to right until an operator is encountered. At this point, the last two tokens are pushed from the stack:

```
Operator 1      =    c
Operator 2      =    y
Operand         =    *
Eval1           =    <c*y>
```

The result is pushed onto the stack which is now: `a <c*y>`

The next token is an operator, so the stack is popped again and the expression evaluated:

```
Operator 1      =    a
Operator 2      =    <c*y>
Operand         =    +
Eval2           =    <a+<c*y>>  or  <a+Eval1>
```

Continuing on, the next 2 operands are pushed onto the stack: `<a+Eval1> d x`

The next token is an operator, so the stack is popped again and the expression evaluated:

```
Operator 1      =    d
Operator 2      =    x
Operand         =    *
Eval3           =    <d*x>
```

And the result placed on the stack: `<a+Eval1> <d*x>`

The final token is an operator, so the stack is popped and the final expression evaluated:

```
Operator 1      =    <a+Eval1>
Operator 2      =    <d*x>
Operand         =    -
Eval4           =    <<a+Eval1>-<d*x>> or <Eval2-Eval3>
```

Test Cases:

Enter expression (blank line to exit): a+c*y-d*x
Eval1:<c*y>
Eval2:<a+Eval1>
Eval3:<d*x>
Eval4:<Eval2-Eval3>

Enter expression (blank line to exit): 34/5+16*2
Eval1:<34/5>
Eval2:<16*2>
Eval3:<Eval1+Eval2>

Enter expression (blank line to exit): 5+9.27/1.4*3+2/3
Eval1:<9.27/1.4>
Eval2:<Eval1*3>
Eval3:<5+Eval2>
Eval4:<2/3>
Eval5:<Eval3+Eval4>

Enter expression (blank line to exit): 6/7+3/4+1/2
Eval1:<6/7>
Eval2:<3/4>
Eval3:<Eval1+Eval2>
Eval4:<1/2>
Eval5:<Eval3+Eval4>

Enter expression (blank line to exit): 9.8+3*6.7/2-4
Eval1:<3*6.7>
Eval2:<Eval1/2>
Eval3:<9.8+Eval2>
Eval4:<Eval3-4>

Notes:

1. Your program must replicate each of the test cases shown below using the identical notation. The program should run in a loop until a blank line is entered.
2. Although the interpreter is not difficult to implement given the instructions in the Appendix to Assignment 3, you will need to pick apart the resulting expression in order to render the output as shown. You can do this in any way you choose, so long as the result matches the examples shown.
3. Since this is the last Java programming assignment, feel free to use any classes that you wish. Pay particular attention to the String and Collection Classes a these can be particularly useful.
4. There are no bonus marks awarded for this assignment.

Max grade on this assignment: 100/100.

Instructions

Write the program as described in the preceding sections. It should operate interactively and be able to replicate the output from the test cases shown above. To obtain full marks your code must be fully documented and correctly replicate the test cases.

Your submission should consist of 5 files:

1. Stack.java - stack class
2. Queue.java - queue class
3. listNode.java - node object
4. JCalcS.java - program
5. JCalcS.txt - output

Upload your files to myCourses as indicated.

Note: You may decide to include other classes in your program, but the main class must be called Jcalc.

fpf/March 8, 2018