

CSC205 In-Class Lab Assignment – Expression Parsing Lab

11/01/2006

Objective for this Lab

This in-class lab assignment demonstrates how we can use a set of stacks to parse a simple arithmetic expression. The intent of this lab is to create a class that parses a string expression using the Java Stack class. You are going to create two small Java classes; a driver class that has the static main method and a regular Java class named MyStack as follows:

Driver	Your test harness class with the static main method
MyStack	Your work class for the stack methods that parse expressions

Required Methods

Your application needs the following two methods in your MyStack class.

1. public boolean isBalanced(String exp)
2. public void parse(String exp)

Import Statements

Below is the import statement you will need for the MyStack class.

```
import java.util.Stack;
```

The Test Harness Class

All this class does is to make a call to the empty MyStack constructor and to drive all the public methods. It should very similar to this:

```
public class Driver
{
    public Driver()
    {
    }

    public static void main(String[] args)
    {
        MyStacks s = new MyStacks();

        String exp="( (1-2)*(5+3)*(9+1)*(5-3)"; // unbalanced test
        if (s.isBalanced(exp))
            s.parse(exp);

        exp="( (1-2)*(5+3)*(9+1)*(5-3) )"; // balanced test
        if (s.isBalanced(exp))
            s.parse(exp);

        exp="( (9-2)+(5+9)+(9+1)*(5-1) )"; // balanced test
        if (s.isBalanced(exp))
            s.parse(exp);
    }
}
```

The MyStack Class

This class does the heavy lifting for the expression parser and has all of the public methods for this assignment. It has one empty default constructor.

```
import java.util.Stack;

public class MyStack
{
    Stack number        = new Stack();
    Stack operator      = new Stack();

    public MyStack()
    {
    }

    public boolean isBalanced(String exp)
    {
        // student code goes here to see if the parens,
        // braces and brackets are balanced
    }

    public void parse(String exp)
    {
        // student code goes here to process the expression
    }
} // end of class
```

Notes and Hints

Recall, that we need two stacks to parse an expression

1. A number stack
2. An operator stack

We read up to the first right parenthesis; the numbers we encounter along the way are pushed onto the number stack. Operators we encounter along the way are pushed onto the operator stack.

Whenever we reach a right parenthesis, we combine the top two numbers on the number stack, using the topmost operator on the operator stack

Bear in mind that we are reading character data and need to convert to numeric for the mathematics. Also, keep in mind that we are using simple integers so any division operations will be integer division.

The expressions for this test should be simple. By that, I mean no embedded spaces and just single digit constants.

Generated Output

```
((1-2)*(5+3)*(9+1)*(5-3))
Snap:1 - 2 = -1
Snap:5 + 3 = 8
Snap:9 + 1 = 10
Snap:5 - 3 = 2
```

CSC205 In-Class Lab Assignment – Expression Parsing Lab

11/01/2006

```
Snap:10 * 2 = 20
Snap:8 * 20 = 160
Snap:-1 * 160 = -160
-160
```

```
((9-2)+(5+9)+(9+1)*(5-1))
Snap:9 - 2 = 7
Snap:5 + 9 = 14
Snap:9 + 1 = 10
Snap:5 - 1 = 4
Snap:10 * 4 = 40
Snap:14 + 40 = 54
Snap:7 + 54 = 61
61
```

Test Data

Here are the two test expressions:

```
((1-2)*(5+3)*(9+1)*(5-3))
((1-2)*(5+3)*(9+1)*(5-3))
((9-2)+(5+9)+(9+1)*(5-1))
```

Development Environment

The classroom has several Integrated Development Environments. We have BlueJ, JBuilder, Eclipse and TextPad. Use the one in which you are most comfortable.

Deliverables

Turn in the source code from the classes plus the generated output plus any questions that are required to be answered.

Conclusion

This is a great example of a small class that would need millions of combinations to test thoroughly. How much more complexity would need to be introduced to parse more complex expressions like these:

```
(( (6+9) / 3 ) * (6-4) ); // numbers only are easy
( ( ( 6 + 9 ) / 3 ) * ( 6 - 4 ) ); // white space
(( (x+y) / q ) * (6-w) ); // variables
( ( ( x + y ) / q ) * ( 6 - w ) ); // variables & white space
((( x + y ) / q ) * (6 - pow(w))); // embedded methods
```