Assignment Prefix: Lab09

Points: 100

Due Date: Friday, March 24 @ 11:59pm

This is an individual assignment.

**Task:**

Write a Java project that:

* Implements the Shunting Yard algorithm to convert an in-fix expression into its corresponding post-fix notation.
* Uses the queue and stack approach to evaluate an expression that is in post-fix notation.
* Uses the queue and stack approach to build the binary expression tree for an expression that is in post-fix notation.

You must use your implementations of trees, stacks and queues from previous assignments.

Your program should:

* Ask the user to enter the absolute path and filename (as a single String) of the file that contains a list of arithmetic expressions. Each expression will be on a single line in the input text file delimited by an end of line character.
* Read arithmetic expressions from an input file until the EOF is reached.
  + See file format and example at end of assignment.
* For each expression your program should:
  + Print out the expression that was read from the file.
  + Determine if the expression is valid.
    - Print an invalid expression message for invalid expressions.
    - For each valid expression
      * Print the expression in post-fix notation
      * Represent the expression in a binary expression tree
      * Print the expression tree using the pre-order traversal
      * Print the expression tree using the in-order traversal
      * Print the expression tree using the post-order traversal
      * Evaluate the expression and display the results of the evaluation

**Input file format:**

Each token in the input file will be blank separated so the expressions should be easy to parse.

Tokens will be one of the following:

* Numeric value possibly includes negative numbers
  + The uniary negative operator will not have a blank space between the operator and its corresponding operand, e.g. -45
  + The binary subtraction operator will have blank space between the operator and its corresponding operands, e.g. 11 - 5
* Operators will be limited to:
  + Addition +
  + Subtraction -
  + Multiplication \*
  + Division /
* Parenthesis
  + In order to make expression more readable parenthesis, curly brackets and square brackets may be used.
  + For grouping and nesting purposes the symbols must match correctly.
  + For example:
    - ( 3 - [ { 4 / 3 } + 7 ] - 2 ) is correct grouping
    - ( { [ } ] ) is incorrect nesting
* There will be no “implied” multiplication
  + The expression 3 \* ( 4 - -5 ) is valid
  + The expression 3 ( 4 - -5 ) is not valid
* You do **not** need to check for invalid tokens.

**Turning in your assignment:**

* **Make sure that all of your code is properly documented.**
* Turn in your assignment using the standard method.
* Copy and paste each of your Java files into the document.
* Paste the screenshots showing the complete output of a complete run of your program after the Java code in your document.
* Export your NetBeans project to a zip archive.
* Turn in the Word document and zipped project as to separate files in a single Blackboard submission.
* You do not need to turn in your data files. We will test your program with a standard set of test files.

Example input file ( data.txt ):

3 \* -5

4 - 3 / 5

( 4 - 3 ) / 5

4 + ( 7 / 2 )

[ 4 + 7 ] \* { 8 - 11 }

4 + 7 8 - 11

( ( [ 3 + 1 ] \* 3 ) / ( ( 9 - 5 ) ) - ( ( 3 \* ( 7 - 4 ) ) + 6 ) )

( ( 3 + 1 ) \* 3 ) / ( ( 9 - 5 ) ) - ( ( 3 \* ( 7 - 4 ) ) + 6 ) )

3 + 1 \* 3 / 9 - 5 - 3 \* 7 - 4 + 6

42

8 \* 24 / ( 4 + 3

3 + 4 –

String exp = expressionList.get(0);

String exp2 = expressionList.get(1);

String exp

// testing valid expression

System.out.println(exp);

System.out.println(exp2);

System.out.println("+ is operand: " + isOperand("+") + " 10, 9, 25.3 operand?");

System.out.println(isOperand("10"));

System.out.println(isOperand("9"));

System.out.println(isOperand("25.3"));

System.out.println("exp1 valid? " + validateExpression(exp));

System.out.println("exp2 valid? " + validateExpression(exp2));

System.out.println("exp1 valid? " + validateExpression(exp));

/\* scanFile.useDelimiter(" "); // space as delimiter;

while (scanFile.hasNextLine()) {

Scanner inLine= new Scanner(scanFile.nextLine()); //

while (inLine.hasNext()) {

queue.enqueue(inLine.next());

}

int temp = queue.size();

for (int i = 0; i < temp; i++) {

System.out.println(queue.dequeue());

}\*/

}

https://96.3.163.217:8443/!/#rollvoda/view/head/rollvoda/Lab08/src/lab08/Lab08.java