LAB PREPARATION 2

Arithmetic Trees

Youssef Abouwarda (2502748)
Team 16 :
Youssef Abouwarda
M.N. Reeza Ahamed

1. Given Formula : *sin(4pi/3) exp(-(sqrt(2)-1)/8) / sqrt(6pi)*

$$\frac{sin\left(\frac{4\pi}{3}\right) \times e^{-\left(\frac{\sqrt{2}-1}{8}\right)}}{\sqrt{6\pi}}$$

- **postfix Expression**
- 4 pi 3 / sin -1 2 sqrt 1 - 8 / * exp * 6 pi sqrt /

- **Arithmetic Tree**

- **Postfix Expression**

By modifying the functions convert() of lab 1 code, the function convert() was called to get the postfix expression.

```
ArithmeticTerm expLab2Converted = new ArithmeticTerm(expLab2.convert());
System.out.println("The converted FPAE is " + expLab2Converted);
```

```
The converted FPAE is 4 pi 3 / sin -1 2 sqrt 1 - 8 / * exp * 6 pi sqrt /
```

- **Evaluation of the Expression**

By modifying the functions evaluate() of lab 1 code, the function evaluate() was called to get the evaluation of the expression.

```
System.out.println("The result of FPAE is " + expLab2Converted.evaluate());
```

```
The result of FPAE is -0.18940600202368948
```

Problem 4 (Member 1: Tree Construction from Postfix Expressions)

- The class BiNode was included inside the class Tree as an Inner Class. In order to be able to create instances of the inner class (BiNodes) , we made it static class.

- For the expression *"**5.1 9 8.88 + 4 sqrt 6 / ^ 7 - *"** .* The tree was constructed using the method Construct and tested In the main function, Please refer to the source code provided.

```
BiNode construct (String postfix){
           BiNode root;
         StringTokenizer tokenized = new StringTokenizer(postfix);

         StackLL <BiNode> stack = new StackLL<BiNode>();

         while(tokenized.hasMoreTokens())
         {
             String s = tokenized.nextToken();

             try{
                 Double d = Double.parseDouble(s);
                 BiNode node = new BiNode (s);
                 stack.push(node);

             }
             catch(Exception e)
             {
                 BiNode N  = stack.pop();
                 if(s.equals("+")| s.equals("-") || s.equals("*") || s.equals("/")||
s.equals("sqrt") || s.equals("^")){
                  stack.push(new BiNode(s,stack.pop(),N));
                 }
                 else if (s.equals("sqrt") || s.equals("sin") || s.equals("exp") ||
s.equals("pi"))
                 {
                  stack.push(new BiNode(s,N,null));
                 }

             }
         }
         return stack.pop();

     }
```

- The recursive method **private void postOrderTraversal(BiNode n)** was implemented as well as public method **postOrderTraversal** and both are working. Refer to source code
- The Tree constructor was tested on the example of lab 1 which is **5.1 9 8.88 + 4 sqrt 6 / ^ 7 - *** and it works, also the Post Order Traversal of the tree gives us the same expression from which the tree is constructed. Refer to source code , the output was

```
Traverse Node*
Traverse Node-
Traverse Node^
Traverse Node5.1
5.1
Traverse Node/
Traverse Nodesqrt
Traverse Node+
Traverse Node9
9
Traverse Node8.88
8.88
+
Traverse Node4
4
sqrt
Traverse Node6
6
/
^
Traverse Node7
7
-
*
```

- The program terminates with an appropriate error message when there is an error case, e.g. stack is empty or the element pushed is not an operator, refer to source code.