**Michael Tompkins**

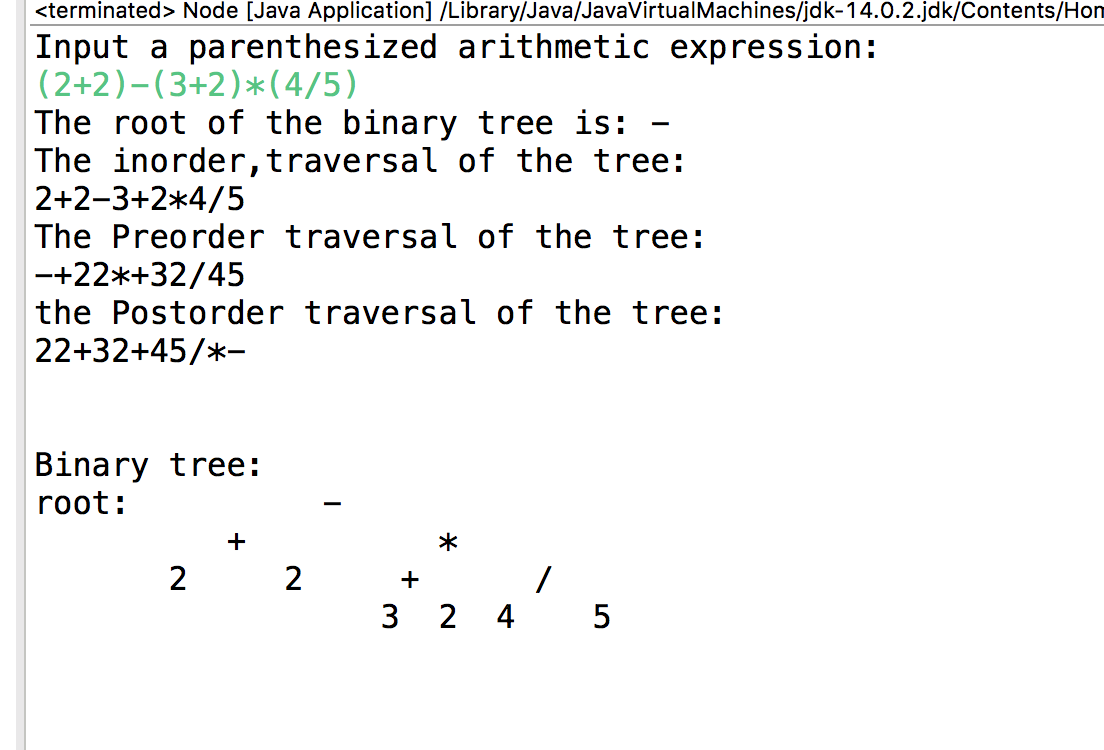
**Project 4 due Nov 30th.**

p-8.65 Write a program that takes as input a fully parenthesized , arithmetic expression and convert it to a binary expression tree. Your program should display the tree in some way and also print the value associated with the root. For an additional challenge , allow the leaves to store variables of X1, X2 , X3 and so on, which are initially 0 and which can be updated interactively by your program, with the corresponding update in the printed value of the root of the expression tree.

I have Included five different inputs and outputs of my program, the inputs are in green text, and the outputs are in black. Below I will put my code. Thank you ☺

Graphical user interface, text, application

Description automatically generated



Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**package** ugh;

**import** java.util.Scanner;

**import** java.util.Stack;

**public** **class** Node {

Node left;

Node right;

**char** data;

**int** key;

Node(**char** c){

**this**.left = **null**;

**this**.right = **null**;

data = c;

}

**public** **static** **boolean** isOperator(**char** c) {

**if** (c == '+' || c == '-'

|| c == '\*' || c == '/'

|| c == '^' || c == '('|| c == ')') {

**return** **true**;

}

**return** **false**;

}

**static** **int** getPriority(**char** C)

{

**if** (C == '-' || C == '+')

**return** 1;

**else** **if** (C == '\*' || C == '/')

**return** 2;

**else** **if** (C == '^')

**return** 3;

**return** 0;

}

**public** **static** String infixtoPostfix(String exp)

{

String result = **new** String("");

Stack<Character> stack = **new** Stack<>();

**for** (**int** i = 0; i<exp.length(); ++i)

{

**char** c = exp.charAt(i);

**if** (Character.*isLetterOrDigit*(c))

result += c;

**else** **if** (c == '(')

stack.push(c);

**else** **if** (c == ')')

{

**while** (!stack.isEmpty() &&

stack.peek() != '(')

result += stack.pop();

stack.pop();

}

**else**

{

**while** (!stack.isEmpty() && *getPriority*(c)

<= *getPriority*(stack.peek())){

result += stack.pop();

}

stack.push(c);

}

}

**while** (!stack.isEmpty()){

**if**(stack.peek() == '(')

**return** "Invalid Expression";

result += stack.pop();

}

**return** result;

}

**public** **static** **void** main(String[] args) {

Stack<Node> Stack1 = **new** Stack<>();

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Input a parenthesized arithmetic expression:");

String userexpression = scan.nextLine();

String expression = **new** String(*infixtoPostfix*(userexpression));

**for** (**int** i = 0; i <= expression.length() - 1; i++) {

**if**(Character.*isLetterOrDigit*(expression.charAt(i))){

Node x = **new** Node(expression.charAt(i));

Stack1.push(x);

}

**else** **if**(*isOperator*(expression.charAt(i))) {

Node r = Stack1.pop();

Node l = Stack1.pop();

Node x = **new** Node(expression.charAt(i));

x.left = l;

x.right = r;

Stack1.push(x);

}

}

Node root = Stack1.peek();

System.***out***.println("The root of the binary tree is: " + root.data);

System.***out***.println("The inorder,traversal of the tree: ");

*printInorder*(root);

System.***out***.println("");

System.***out***.println("The Preorder traversal of the tree:");

*printPreorder*(root);

System.***out***.println("");

System.***out***.println("the Postorder traversal of the tree:");

*printPostorder*(root);

System.***out***.println("");

System.***out***.println(" ");

System.***out***.println(" ");

System.***out***.println("Binary tree:");

System.***out***.println("root: " + root.data);

**if**(root.left!= **null** && root.right != **null**) {

System.***out***.println(" " + root.left.data + " " + root.right.data);

}

**if**(root.left.left != **null** && root.right.right != **null** && root.left.right != **null** && root.right.left != **null**) {

System.***out***.println(" " +root.left.left.data +" " +root.left.right.data + " " + root.right.left.data + " " + root.right.right.data);

}

**if**(root.left.left != **null** && root.right.right == **null** && root.left.right != **null** && root.right.left == **null**) {

System.***out***.println(" " +root.left.left.data +" " +root.left.right.data );

}

**if**(root.left.left == **null** && root.right.right != **null** && root.left.right == **null** && root.right.left != **null**) {

System.***out***.println(" "+" " + root.right.left.data + " " +root.right.right.data);

}

**if**(root.left.left.left != **null** && root.left.right.right != **null**) {

System.***out***.println(" " + root.left.left.left.data + " " + root.left.left.right.data + " " +root.left.right.left.data + " " + root.left.right.right.data);

}

**if**(root.right.left.left != **null** && root.right.right.right != **null**) {

System.***out***.println(" " + root.right.left.left.data + " " +root.left.right.data + " " + root.right.right.left.data + " " + root.right.right.right.data);

}

scan.close();

}

**static** **void** printInorder(Node node)

{

**if** (node == **null**)

**return**;

*printInorder*(node.left);

System.***out***.print(node.data);

*printInorder*(node.right);

}

**static** **void** printPreorder(Node node)

{

**if** (node == **null**)

**return**;

System.***out***.print(node.data);

*printPreorder*(node.left);

*printPreorder*(node.right);

}

**static** **void** printPostorder(Node node)

{

**if** (node == **null**)

**return**;

*printPostorder*(node.left);

*printPostorder*(node.right);

System.***out***.print(node.data);

}

}