**DATA STRUCTURES & ALGORITHM  
FINAL PROJECT**

**Section: 1-BSCS-.2**

Marasigan, Vem Aiensi  
Lumabos, Nickie  
DelosSantos, Allen Chris

**----------------------------------------------------BINARY SEARCH TREE--------------------------------------------------------**

**SOURCE CODE**

package final\_Project;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.Scanner;

import java.util.concurrent.TimeUnit;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JTextField;

public class BSCS2\_Marasigan\_Lumabos\_DelosSantos\_FinalProject

{

public static Scanner in = new Scanner(System.in);

//A + ( ( B - C ) \* D ) / F //test sample

//A ^ B ^ C + D + F //test sample

//1 ^ 20 ^ 14 + 9 + -1 //test sample

public static void main(String[] args)

{

new GUI();

GUI.show();

}

static void UserVersion(String expression)

{

Element[] prefix = Converter.toPrefix(expression);

Tree binaryTree = new Tree(prefix);

Tools.PrintHorizontally("Creating the Tree, Please Wait", 20);

Tools.PrintHorizontally(".....\n", 800);

System.out.println();

Tools.PrintVertically(binaryTree.verticalTree, 100);

new PrefixCalculator(Converter.reversedPrefix);

Tools.end();

}

static void DeveloperVersion(String expression, int choice)

{

Element[] prefix = Converter.toPrefix(expression);

Tree xpTree = new Tree(prefix);

do

{

System.out.print("[1] Infix [2] Reversed Infix\n"

+ "[3] Prefix [4] Reversed Prefix\n"

+ "[5] TreeH [6] TreeV\n"

+ "[7] 2DArrInfo [8] Calculate\nChoice: ");

choice = in.nextInt();

switch (choice)

{

case 1: Tools.ArrayInformation(Converter.infix, "Element"); break;

case 2: Tools.ArrayInformation(Converter.reversedInfix, "Element"); break;

case 3: Tools.ArrayInformation(Converter.prefix, "Element"); break;

case 4: Tools.ArrayInformation(Converter.reversedPrefix, "Element"); break;

case 5: xpTree.printTreeHorizontal(); break;

case 6: xpTree.printTreeVertical();break;

case 7: Tools.ArrayInformation2D(xpTree.tree, "Classification"); break;

case 8: new PrefixCalculator(Converter.reversedPrefix); break;

}

System.out.println();

}while (choice != 9);

}

}

class Element

{

String element;

String classification;

int plvl;

//special characteristic for tree printing

int treeLevel;

double value;

Element(String element, String classification, int plvl)

{

this.element = element;

this.classification = classification;

this.plvl = plvl;

}

//information printing for testing purposes

void ShowCharacteristics(Element a)

{

System.out.println("Element name : " + a.element);

System.out.println("Classificaion: " + a.classification);

System.out.println("Precedence: " + a.plvl);

System.out.println("Tree Level: " + a.treeLevel);

}

static String classification(String element)

{

String classification = "Leaf";

if (element.equals("+") || element.equals("-") || element.equals("\*") || element.equals("/") || element.equals("^"))

classification = "Node";

if (element.equals(")") || element.equals("("))

classification = "Prnthss";

return classification;

}

static int plvl (String element)

{

int plvl = 0;

if(element.equals("+") || element.equals("-"))

plvl = 1;

if(element.equals("\*") || element.equals("/"))

plvl = 2;

if(element.equals("^"))

plvl = 3;

if(element.equals(")") || element.equals("("))

plvl = 4;

return plvl;

}

}

class Tree

{

Tree(Element[] array)

{

TreeAnalysis(array, 0);

TreeLevelCompiler(max\_level, array);

FormattedVerticalPrinting(tree, indentAndSpace(), 0);

}

int level = 1;

int scannedLeaf = 0;

int leaf = 0;

int node = 0;

int max\_level = 0;

//Holds data about the created tree

Element[][] tree = {};

String horizontalTree = "";

String verticalTree = "";

//ESSENTIAL TOOLS

void TreeAnalysis(Element[] array, int index)

{

if (index<array.length)

{

//the String (horizontalTree) records the events that happens in the recursion

horizontalTree += " " + level +"\t|";

TabH(level);

if(array[index].classification.equals("Node"))

{

leaf = 0;

node++;

horizontalTree+="(" + array[index].element + ")\n";

array[index].treeLevel = level;

level++;

}

else

{

scannedLeaf++;

leaf++;

array[index].treeLevel = level;

horizontalTree += "(" + array[index].element + ")\n";

}

if (leaf==2)

{

leaf--;

level--;

if(node-1 == scannedLeaf && array[index-scannedLeaf].classification == "Leaf")

{

node = 1;

level--;

scannedLeaf = 0;

}

}

if (level>max\_level)

max\_level = level;

TreeAnalysis(array, index+1);

}

return;

}

void TreeLevelCompiler(int level, Element[] array)

{

tree = Tools.ArrayMaker(level);

for(int index1 = 0; index1<tree.length; index1++)

{ //Level stage

int scanned = 0; int arrayIndex = 0;

for(int index2 = 0; index2<tree[index1].length; index2++)

{ //array stage

for (; arrayIndex<array.length && scanned<tree[index1].length; arrayIndex++)

{ //Scans array details

if(array[arrayIndex].treeLevel == index1+1)// index+1 refers to the level

{

scanned++;

tree[index1][index2] = array[arrayIndex];

arrayIndex++;

break;

}

}

}

}

//The loop only records the data to the 2Darray based only on the tree levels

tree = TreeFix(tree);

}

static Element[][] TreeFix(Element[][] array)

{

Element[][] fixed = array;

for (int count =1; count<array.length; count++)

{

Element[] adjust = new Element[array[count].length];

int adjustIndex = 0;

for(int count2 =0, index =0; count2<array[count-1].length; count2++)

{

if (array[count-1][count2] != null)

if (array[count-1][count2].classification == "Node")

{//record 2 child -=Binary Tree Rule=-

adjust[adjustIndex] = array[count][index];

adjust[adjustIndex+1] = array[count][index+1];

index += 2;

}

adjustIndex += 2;

}

array[count] = adjust;

}

return fixed;

}

//PRINTING METHODS (for Testing)

void printTreeHorizontal()

{//Horizontal Manner for level provider tester

HorizontalHeader(max\_level);

System.out.println(horizontalTree);

}

//only for horizontal tree printing as test

static void HorizontalHeader(int maxlvl)

{

System.out.print(" LEVEL |");

for (int level = 1; level <= maxlvl; level++)

{

System.out.print(" " + level + "\t");

}

System.out.print("\n--------+");

for (; maxlvl>0; maxlvl--)

{

System.out.print("--------");

}

System.out.println();

}

void TabH(int count)

{

for (; count>1; count--)

{

horizontalTree += "\t";

}

}

//Tree printing 2.0 ragh!

void printTreeVertical()

{

System.out.println(verticalTree);

}

void FormattedVerticalPrinting(Element[][] array, int[][] levelIS, int level)

{

if (level < max\_level)

{

int indent = levelIS[level][0], space = levelIS[level][1];

//System.out.println("level " + (level+1) + ": " + levelIS[level][0]);

//connection(array[level], levelIS[level], 3, 2);

TabV(indent);

for (int index = 0; index<array[level].length; index++)

{

if (array[level][index] != null)

{

this.verticalTree += "(" + array[level][index].element + ")";

}

else

{

this.verticalTree +=" ";

}

TabV(space);

}

this.verticalTree += "\n";

if (level != max\_level-1)

{

connection(array[level], levelIS[level+1], 3, 2);

}

FormattedVerticalPrinting(array, levelIS, level+1);

}

else

this.verticalTree +="\n";

return;

}

void TabV(int count)

{

for (; count>0; count--)

{

this.verticalTree += "\t";

}

}

int[][] indentAndSpace()

{

int[][] levelIS = new int[max\_level][2];

int indent = 0, space = 2;

for (int level = max\_level-1; level > -1; level--)

{

levelIS[level][0] = indent;// Indent of that level

//System.out.println("level " + (level+1) + ": " + levelIS[level][0]);

levelIS[level][1] = space; // spacing of that level

//System.out.println("level " + (level+1) + ": " + levelIS[level][1]);

indent = indent + space/2;

space = space\*2;

}

return levelIS;

}

void connection(Element[] array, int[] IS, int stairsUp, int stairsDown)

{

// - - E

// - - E

// - - E

if (stairsUp > 0)

{

TabV(IS[0]);

for(int index = 0; index < array.length; index++)

{

if (array[index] != null && array[index].classification == "Node")

{

for(int count = stairsUp\*IS[1]; count>0; count--)

{

this.verticalTree += " ";

}

this.verticalTree += "-";

for(int count = stairsDown\*IS[1]; count>0; count--)

{

this.verticalTree += " ";

}

this.verticalTree += "-";

for(int count = stairsUp\*IS[1]; count>2; count--)

{

this.verticalTree += " ";

}

}

else

{

for(int count = stairsUp\*IS[1]; count>0; count--)

{

this.verticalTree += " ";

}

this.verticalTree += " ";

for(int count = stairsDown\*IS[1]; count>0; count--)

{

this.verticalTree += " ";

}

this.verticalTree += " ";

for(int count = stairsUp\*IS[1]; count>2; count--)

{

this.verticalTree += " ";

}

}

TabV(IS[1]);

}

this.verticalTree += "\n";

connection(array, IS, stairsUp-1, stairsDown+2);

}

return;

}

}

//A + B - C \* D / F

class Converter

{

static Element[] stack;

static Element[] infix;

static Element[] reversedInfix;

static Element[] prefix;

static Element[] reversedPrefix;

static Element[] toPrefix(String expression)

{

infix = ArrayConverter(expression);

reversedInfix = reverse(infix);

stack = new Element[Tools.count(reversedInfix, "Operator")];

reversedPrefix = new Element[reversedInfix.length - Tools.count(reversedInfix, "Parenthesis")];

stack[0] = new Element(" ", " ", 0);

int top = -1, reversedPrefixIndex = 0, raiseLvl = 0, rIndex = -1;

int[] reference = new int[Tools.count(reversedInfix, "Parenthesis")/2];

//marks the index that the parenthesis started

for (int index = 0; index<reversedInfix.length; index++)

{

//increase precedence level for parenthesis

if (reversedInfix[index].element.equals(")"))

{

rIndex++;

reference[rIndex] = top;

raiseLvl += 5;//make sure to prioritize those inside parenthesis

index++;

//System.out.println("Reference index: " + top);

}

if (reversedInfix[index].element.equals("("))

{

raiseLvl = -5;//decreases level raiser depending on the ( encountered

index++;

//System.out.println("currentTop: " + top);

while (top > reference[rIndex]) //pops all operators within the parenthesis

{

reversedPrefix[reversedPrefixIndex] = stack[top];

reversedPrefixIndex++;

top--;

}

rIndex--;

}

if (index == reversedInfix.length)//breaks loop immediately after all elements are scanned

break;

if (reversedInfix[index].classification == "Node")//if operator

{

//condition in reversedPrefix changes when multiple carets are encountered

if (reversedInfix[index].element.equals("^"))

{

if (top == -1 || reversedInfix[index].plvl+raiseLvl > stack[top].plvl)

{

top++;

stack[top] = reversedInfix[index];

}

else

{

while (top != -1 && reversedInfix[index].plvl+raiseLvl <= stack[top].plvl )

{

reversedPrefix[reversedPrefixIndex] = stack[top];//pop

reversedPrefixIndex++;

top--;

}

top++;

stack[top] = reversedInfix[index];

}

}

else //if not "^"

{

//stack empty or scanned is greater than top

if (top == -1 || reversedInfix[index].plvl+raiseLvl >= stack[top].plvl)

{

top++;

stack[top] = reversedInfix[index];

}

else

{

while (top != -1 && reversedInfix[index].plvl+raiseLvl < stack[top].plvl )

{

reversedPrefix[reversedPrefixIndex] = stack[top];//pop

reversedPrefixIndex++;

top--;

}

top++;

stack[top] = reversedInfix[index];

}

}

}

else if (reversedInfix[index].classification == "Leaf")//if operand

{

reversedPrefix[reversedPrefixIndex] = reversedInfix[index];//pop

reversedPrefixIndex++;

}

}

//insert all remaining elements in the stack

while (top > -1)

{

reversedPrefix[reversedPrefixIndex] = stack[top];

reversedPrefixIndex++;

top--;

}

prefix = reverse(reversedPrefix);

return prefix;

}

static Element[] reverse(Element[] expression)

{

Element[] reverse = new Element[expression.length];

for (int count = expression.length-1, index = 0; count>=0; count--, index++)

{

reverse[index] = expression[count];

}

return reverse;

}

static Element[] ArrayConverter(String expression)

{

String reference[] = Tools.stringToArray(expression);

Element[] elementArray = new Element[reference.length];

for (int index = 0; index<reference.length; index++)

{

elementArray[index] = new Element(reference[index], Element.classification(reference[index]), Element.plvl(reference[index]));

}

return elementArray;

}

}

class PrefixCalculator

{

Scanner in = new Scanner(System.in);

Element[] expression;

Element[] stack;

Element[] calculated;

PrefixCalculator(Element[] reversedPrefix)

{

expression = reversedPrefix;

for (int index = expression.length-1; index>-1; index--)

{

if (expression[index].classification == "Leaf")

{

System.out.print(" Enter value of " + expression[index].element + ": ");

expression[index].value = in.nextDouble();

expression[index].element = Double.toString(expression[index].value);

}

}

//System.out.println(Tools.count(expression, "Operand"));

stack = new Element[Tools.count(expression, "Operand")];

int recurse = Tools.count(reversedPrefix, "Operator")-1;

Tree binaryExpression = new Tree(Converter.reverse(expression));

Tools.PrintVertically(binaryExpression.verticalTree, 100);

Element[] calculatedPrefix = Calculate(expression);

binaryExpression = new Tree(calculatedPrefix);

Tools.PrintVertically(binaryExpression.verticalTree, 100);

calculatedPrefix = Converter.reverse(calculatedPrefix);

for (; recurse > 0; recurse--)

{

calculatedPrefix = Calculate(calculatedPrefix);

binaryExpression = new Tree(calculatedPrefix);

Tools.PrintVertically(binaryExpression.verticalTree, 100);

calculatedPrefix = Converter.reverse(calculatedPrefix);

}

Tools.PrintHorizontally(" FINAL ANSWER: " + calculatedPrefix[0].element, 50);

}

Element[] Calculate(Element[] expression)

{

int top = -1;

int index = 0;

for (; index<expression.length; index++)

{

if (expression[index].classification == "Leaf")

{

top++;

stack[top] = expression[index];

}

else

{

if (expression[index].element.equals("+"))

{

expression[index].value = stack[top].value + stack[top-1].value;

}

else if (expression[index].element.equals("-"))

{

expression[index].value = stack[top].value - stack[top-1].value;

}

else if (expression[index].element.equals("\*"))

{

expression[index].value = stack[top].value \* stack[top-1].value;

}

else if (expression[index].element.equals("/"))

{

expression[index].value = stack[top].value / stack[top-1].value;

}

else if (expression[index].element.equals("^"))

{

double result=stack[top].value;

for (double limit = stack[top-1].value; limit>1; limit--)

{

result = result\*stack[top].value;

}

expression[index].value = result;

}

expression[index].classification = "Leaf";

expression[index].element = Double.toString(expression[index].value);

top -=2;

break;

}

}

//System.out.println("Index : " + index);

calculated = new Element[expression.length - 2];

//System.out.println("size: " + (expression.length - 2));

int newIndex = 0;

for (int stackIndex=0; stackIndex<top+1; newIndex++, stackIndex++)

{

calculated[newIndex] = stack[stackIndex];

}

for (; index < expression.length && newIndex < calculated.length; index++, newIndex++)

{

calculated[newIndex] = expression[index];

}

return Converter.reverse(calculated);

//Tools.ArrayInformation(calculated, "Value");

//System.out.println("\nNewIndex: " + newIndex);

//System.out.println(); //for tests only

}

}

class Tools

{

static String[] stringToArray(String expression)

{

String array[];

array = expression.split(" ");

return array;

}

static void ArrayInformation(Element[] array, String characteristic)

{

switch (characteristic)

{

case "Element":

System.out.print("Element name:\t");

for (int index = 0; index<array.length; index++ )

{

if (array[index] == null)

System.out.print("\t");

else

System.out.print(array[index].element + "\t");

}

System.out.println();

break;

case "PLVL":

System.out.print("Precedence lvl:\t");

for (int index = 0; index<array.length; index++ )

{

if (array[index] == null)

System.out.print("\t");

else

System.out.print(array[index].plvl + "\t");

}

System.out.println();

break;

case "Classification" :

System.out.print("Classification:\t");

for (int index = 0; index<array.length; index++ )

{

if (array[index] == null)

System.out.print("\t");

else

System.out.print(array[index].classification + "\t");

}

System.out.println();

break;

case "TLVL":

System.out.print("\nTreeLevel:\t");

for (int index = 0; index<array.length; index++ )

{

if (array[index] == null)

System.out.print("\t");

else

System.out.print(array[index].treeLevel + "\t");

}

break;

case "Value":

System.out.print("\nValue:\t");

for (int index = 0; index<array.length; index++ )

{

if (array[index].value == 0.0)

System.out.print(array[index].element + "\t");

else

System.out.print(array[index].value + "\t");

}

break;

}

}

static void ArrayInformation2D(Element[][] array, String characteristic)

{

for(int index1 = 0; index1<array.length; index1++)

{

System.out.print("LVL " + (index1+1) + ": ");

for(int index2 = 0; index2<array[index1].length; index2++)

{

if (array[index1][index2] == null)

{

System.out.print("\t");

}

else

{

switch (characteristic)

{

case "Element":

System.out.print(array[index1][index2].element + "\t");

break;

case "PLVL":

System.out.print(array[index1][index2].plvl + "\t");

break;

case "Classification":

System.out.print(array[index1][index2].classification + "\t");

break;

case "TLVL":

System.out.print(array[index1][index2].treeLevel + "\t");

break;

}

}

}

System.out.println();

}

}

static int count(Element[] array, String characteristic)

{

int counter = 0;

for(int index = 0; index<array.length; index++)

{

switch(characteristic)

{

case "Operator":

if (array[index].classification == "Node")

counter++;

break;

case "Operand":

if (array[index].classification == "Leaf")

counter++;

break;

case "Parenthesis":

if (array[index].classification == "Prnthss")

counter++;

break;

}

}

return counter;

}

static Element[][] ArrayMaker(int level)

{

Element[][] lines = new Element[level][];

int size2D = 1;

for(int index = 0; index<level; index++)

{

lines[index] = new Element[size2D];

size2D = size2D\*2;

}

return lines;

}

static void PrintVertically(String s, int speed)

{

try

{

for (int index = 0; index<s.length(); index++)

{

if (s.charAt(index) == '\n')

{

TimeUnit.MILLISECONDS.sleep(speed);

}

System.out.print(s.charAt(index));

}

}

catch (Exception e) {}

}

static void PrintHorizontally(String s, int speed)

{

try

{

for (int index = 0; index<s.length(); index++)

{

TimeUnit.MILLISECONDS.sleep(speed);

System.out.print(s.charAt(index));

}

}

catch (Exception e) {}

}

static void end()

{

String end = "\n\n\n\n\n\n\n\n\n\n\n\n"

+ "\t \* ,1111111. \*\r\n"

+ "\t 11111111111 .\r\n"

+ "\t 1111111111111\r\n"

+ "\t \* 1111111111111\r\n"

+ "\t 1111111111111\r\n"

+ "\t '11111111111'\r\n"

+ "\t '1111111' \*\r\n"

+ "\t |\\\_\_\_/| -\r\n"

+ "\t ) ( . '\r\n"

+ "\t =\\ /=\r\n"

+ "\t )===( \*\r\n"

+ "\t / \\\r\n"

+ "\t | |\r\n"

+ "\t / \\\r\n"

+ "\t \\ /\r\n"

+ "\t \_/\\\_/\\\_/\\\_\_ \_/\_/\\\_/\\\_/\\\_/\\\_/\\\_/\\\_/\\\_/\\\_/\\\_\r\n"

+ "\t | | | |( ( | | | | | | | | | |\r\n"

+ "\t | | | | ) ) | | | THANK YOU PO | | |\r\n"

+ "\t | | | |(\_( | | | | | | | | | |\r\n"

+ "\t | | | | | | BY: VEM AIENSI MARASIGAN |\r\n"

+ "\t | | | | | | | | NICKIE LUMABOS | |\n"

+ "\t | | | | | | ALLEN CHRIS DELOS SANTOS |\n"

+ "\n\n\n\n\n\n\n\n\n";

String credits = "\t ASCII art is from: \n"

+ "\t http://user.xmission.com/~emailbox/ascii\_cats.htm\n";

Tools.PrintVertically(end, 100);

Tools.PrintHorizontally(credits, 20);

}

}

class GUI extends BSCS2\_Marasigan\_Lumabos\_DelosSantos\_FinalProject

{

static JFrame mainWindow = new JFrame();

static String expression;

static String Mode;

static JFrame ask = new JFrame();

static JTextField infix = new JTextField();

static void show()

{

ActionListener pass = new ActionListener()

{

public void actionPerformed(ActionEvent e)

{

JLabel note = new JLabel("Note: Please include spaces in between elements");

note.setBounds(80, 80, 300, 20);

JLabel label = new JLabel("Please Type Infix Expression: ");

label.setBounds(20, 10, 300, 50);

infix.setBounds(50, 60, 400, 20);

JButton submit = new JButton("SUBMIT");

submit.setBounds(150, 110, 170, 20);

ActionListener proceed = new ActionListener() {

public void actionPerformed(ActionEvent e)

{

expression = infix.getText();

mainWindow.setVisible(false);

if (Mode == "DEV")

DeveloperVersion(expression, 0);

else

UserVersion(expression);

}

};

submit.addActionListener(proceed);

JPanel panel2 = new JPanel();

panel2.setLayout(null);

panel2.add(infix);

panel2.add(label);

panel2.add(submit);

panel2.add(note);

mainWindow.add(panel2);

mainWindow.setTitle("Binary Search Tree");

mainWindow.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

mainWindow.setSize(500, 210);

mainWindow.setResizable(false);

mainWindow.setVisible(true);

}

};

JPanel panel = new JPanel();

JLabel welcome = new JLabel("Welcome to BINARY SEARCH TREE!");

welcome.setBounds(90, 15, 300, 20);

JLabel mode = new JLabel("CHOOSE MODE");

mode.setBounds(150, 30, 150, 20);

JButton user = new JButton("USER VERSION");

user.setBounds(20, 80, 150, 20);

ActionListener User = new ActionListener() {

public void actionPerformed(ActionEvent e)

{

Mode = "USER";

ask.setVisible(false);

}

};

user.addActionListener(User);

user.addActionListener(pass);

JButton dev = new JButton("DEVELOPER MODE");

dev.setBounds(200, 80, 150, 20);

ActionListener developer = new ActionListener() {

public void actionPerformed(ActionEvent e)

{

Mode = "DEV";

ask.setVisible(false);

}

};

dev.addActionListener(developer);

dev.addActionListener(pass);

panel.setLayout(null);

panel.add(mode);

panel.add(user);

panel.add(dev);

panel.add(welcome);

ask.setTitle("Mode Selection");

ask.setVisible(true);

ask.add(panel);

ask.setSize(400, 200);

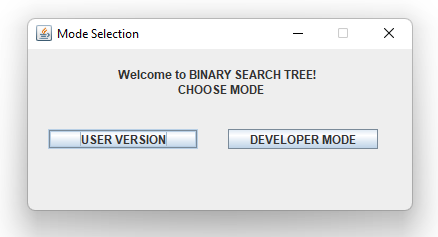
ask.setDefaultCloseOperation(1);

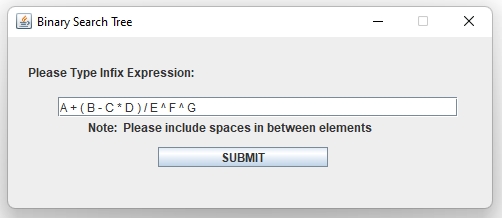
ask.setResizable(false);

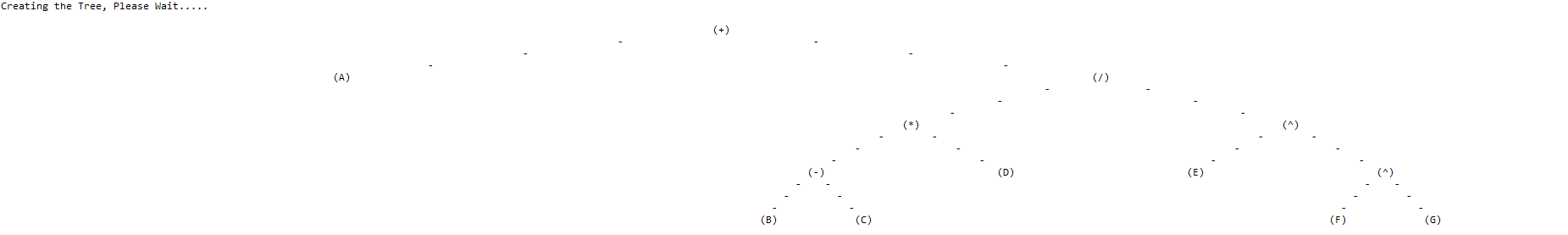
}

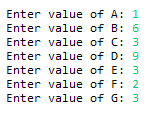
}

**SCREENSHOTS**

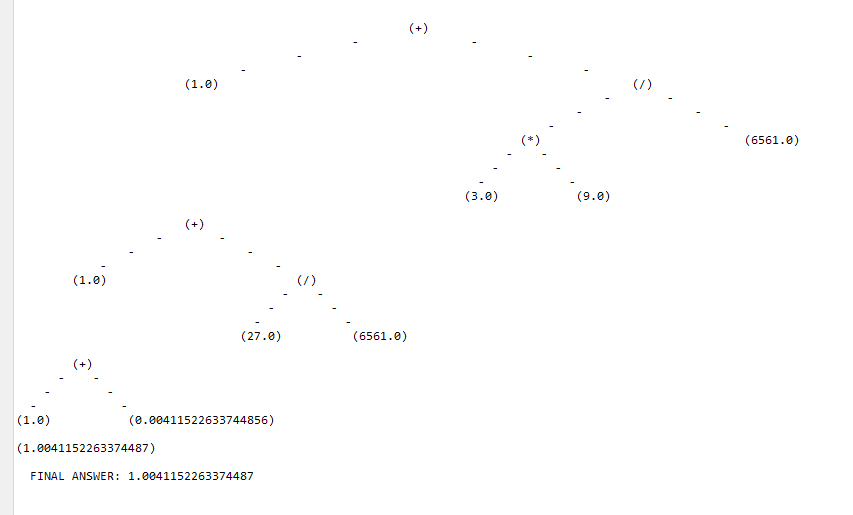
****

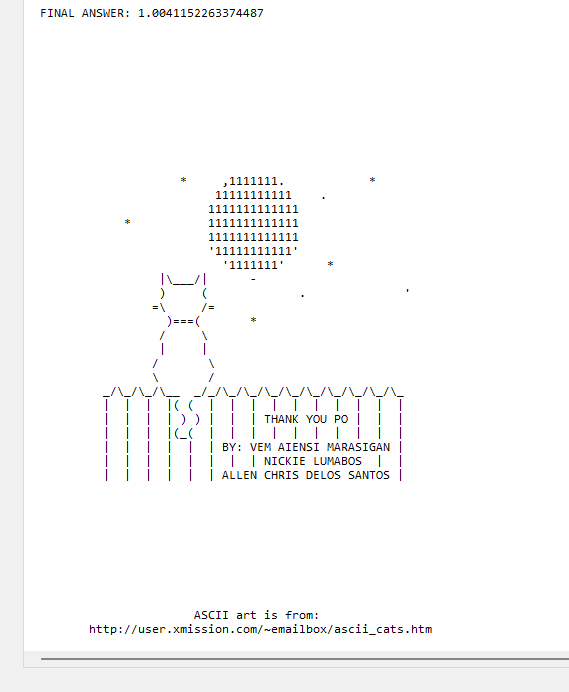
****

****

****

****

****

****

**Running Video:**

<https://drive.google.com/file/d/1mRU4HnHHgwU5I030kondw2lYZHObXVBB/view?usp=sharing>