НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФАКУЛЬТЕТ ІНФОРМАТИКИ І ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Лабораторна робота №2

з дисципліни **«**Системне програмування 2**»**

Виконав:

студент групи ІО-42, ФІОТ

Коваленко В’ячеслав Сергійович

ЗК : 4209

м. Київ 2016 р.

Варіант

|  |  |  |  |
| --- | --- | --- | --- |
| № вар. | Вираз, який відтворюється в графі внутрішнього подання | Настроювання графа автомата з послідовними станами | Мова відтво­рення |
| 9 | b=c?d:2\*a[n]; | Стани 1..9; 3->3(*dlm*), 5->9(*ltr*) | С |

Лістинг програми

**Class Main**

public class Main {

private static ArrayList<LxNode> lxNodeList = new ArrayList<LxNode>();

public static void main(String[] args) {

// reverse fill lxNodeList

lxNodeList.set(10, new LxNode(Token.RIGHT\_SQUARE\_BRACKET)); //]

lxNodeList.set(9, new LxNode(Token.LEFT\_SQUARE\_BRACKET)); //[

lxNodeList.set(8, new LxNode('n', lxNodeList.get(9), lxNodeList.get(10))); //n

lxNodeList.set(7, new LxNode('a')); //a

lxNodeList.set(6, new LxNode(Token.MUL, lxNodeList.get(7), lxNodeList.get(8))); //\*

lxNodeList.set(5, new LxNode('d')); //d

lxNodeList.set(4, new LxNode('c')); //c

lxNodeList.set(3, new LxNode(Token.QUESTION\_MARK, lxNodeList.get(4), lxNodeList.get(5))); //?

lxNodeList.set(2, new LxNode(Token.COLON, lxNodeList.get(3), lxNodeList.get(6))); //:

lxNodeList.set(1, new LxNode('b')); // b

lxNodeList.set(0, new LxNode(Token.SET, lxNodeList.get(1), lxNodeList.get(2))); //=

if (areBracketsNormas()) outputResultTree(lxNodeList.get(0));

else System.out.println("Something went wrong");

// Possible combinations

//String[] signalsSequence = {"sgn1", "sgn2", "sgn3", "sgn4", "ltr"}; // s5 => s8

//String[] signalsSequence = {"sgn1", "sgn2", "sgn3", "sgn4", "sgn5", "sgn6", "sgn7", "sgn8"}; // in order

String[] signalsSequence = {"sgn1", "sgn2", "dlm", "sgn4", "sgn5", "sgn6", "sgn7", "sgn8"}; // recursion

GraphCreator createdGraph = new GraphCreator();

createdGraph.outputGraph(signalsSequence);

}

private static void outputResultTree(LxNode lxNode){

if(lxNode.getLeftNode() !=null){

outputResultTree(lxNode.getLeftNode());}

System.out.print(lxNode.toString());

if(lxNode.getRightNode() !=null){

outputResultTree(lxNode.getRightNode());}

}

private static boolean areBracketsNormas(){

if(!lxNodeList.isEmpty()){

int bracketsNumberDifference = 0;

for (int i = 0; i < lxNodeList.size(); i++) {

if(lxNodeList.get(i).getToken() == Token.LEFT\_SQUARE\_BRACKET) bracketsNumberDifference++;

else if (lxNodeList.get(i).getToken() == Token.RIGHT\_SQUARE\_BRACKET) bracketsNumberDifference--;

}

if(bracketsNumberDifference != 0) return false;

return true;

}

return false;

}

}

**Class LxNode**

public class LxNode {

private Token token;

private LxNode leftNode;

private LxNode rightNode;

private Object lex;

public LxNode(Token token, LxNode prevNode, LxNode nextNode) {

this.token = token;

this.leftNode = prevNode;

this.rightNode = nextNode; }

public LxNode(Token token) {

this.token = token;

}

public LxNode(char c, LxNode prevNode, LxNode nextNode) {

this.token = Token.var;

this.lex = c;

this.leftNode = prevNode;

this.rightNode = nextNode;

}

public LxNode() {

}

public LxNode getLeftNode() {

return leftNode;

}

public LxNode getRightNode() {

return rightNode;

}

public Token getToken() {

return token;

}

@Override

public String toString() {

if(token==Token.var){

return String.valueOf(toChar((char)lex));

}

return token.toString();

}

public char toChar(char c) {

return c;

}

}

**Enum Token**

public enum Token {

SET{

@Override

public String toString() {

return "=";

}

},

COLON{

@Override

public String toString() {

return ":";

}

},

QUESTION\_MARK{

@Override

public String toString() {

return "?";

}

},

MUL{

@Override

public String toString() {

return "\*";

}

},

LEFT\_SQUARE\_BRACKET{

@Override

public String toString() {

return "[";

}

},

RIGHT\_SQUARE\_BRACKET{

@Override

public String toString() {

return "]";

}

},

var

}

**Class Combination**

public class Combination {

String currentVertex;

String nextVertex;

String signal;

public Combination(String currentVertex, String nextVertex, String signal) {

this.currentVertex = currentVertex;

this.nextVertex = nextVertex;

this.signal = signal;

}

public String getSignal() {

return signal;

}

public String getCurrentVertex() {

return currentVertex;

}

public String getNextVertex() {

return nextVertex;

}

}

**Class GraphCreator**

public class GraphCreator {

List<Combination> combinationList;

public GraphCreator() {

combinationList = new ArrayList<Combination>();

combinationList.add(new Combination("s1", "s2", "sgn1"));

combinationList.add(new Combination("s2", "s3", "sgn2"));

combinationList.add(new Combination("s3", "s3", "dlm"));

combinationList.add(new Combination("s3", "s4", "sgn3"));

combinationList.add(new Combination("s4", "s5", "sgn4"));

combinationList.add(new Combination("s5", "s9", "ltr"));

combinationList.add(new Combination("s5", "s6", "sgn5"));

combinationList.add(new Combination("s6", "s7", "sgn6"));

combinationList.add(new Combination("s7", "s8", "sgn7"));

combinationList.add(new Combination("s8", "s9", "sgn8"));

}

public boolean outputGraph(String[] signalsSequence) {

System.out.println("\n \nResult");

for (int i = 0; i < signalsSequence.length; i++) {

String currentSignal = signalsSequence[i];

for (int j = 0; j < combinationList.size(); j++) {

if (combinationList.get(j).getSignal().equals(currentSignal)) {

System.out.println(combinationList.get(j).currentVertex + "=>" + combinationList.get(j).getNextVertex() + " |" + combinationList.get(j).getSignal());

if (combinationList.get(j).getNextVertex().equals(combinationList.get(j).getCurrentVertex())) {

System.out.println("Recursion");

return true;

}

if(combinationList.get(j).getNextVertex().equals("s9")){

System.out.println("End");

return true;

} break; }}}

return true; }}

Результати програми

