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 3 курсу гр. ІО-42

Висоцький Н.С.

Перевірив:

Павлов В.Г.

Київ 2016 р.

Лабораторна робота №2

**Тема: Побудова і використання об’єктів вузлів деревоподібних та ієрархічних графів**

**Мета**: вивчення методів створення та використання вузлів графів автоматів, а також дерево­по­діб­них та ієрархічних графів, організації доступу до інформації, ре­кон­струкції вхідного тексту та скорочення графів через вилу­чення повторних вузлів.

Варіант:

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | while n<>0 do begin  n:=n-1; b:=b+a[n] end | Стани 0..9; 3->7(*dlm*),  5->8(*cfr*), 3->2(*ltr*) | Pascal |

Лістинг програми:

public class Main {

public static void main(String[] args) {

Lexeme[] lexemes = new Lexeme[19];

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

lexemes[i]=new Lexeme();

}

lexemes[0].setToken(Token.Do);

lexemes[0].setPrev(lexemes[1]);

lexemes[0].setNext(lexemes[5]); lexemes[1].setToken(Token.NotEquals);

lexemes[1].setPrev(lexemes[2]);

lexemes[1].setNext(lexemes[3]); lexemes[2].setToken(Token.variable);

lexemes[2].setLexeme('n');

lexemes[2].setPrev(lexemes[4]); lexemes[3].setToken(Token.constant);

lexemes[3].setLexeme(0); lexemes[4].setToken(Token.While); lexemes[5].setToken(Token.Begin);

lexemes[5].setNext(lexemes[6]); lexemes[6].setToken(Token.Assign);

lexemes[6].setPrev(lexemes[7]);

lexemes[6].setNext(lexemes[8]); lexemes[7].setToken(Token.variable);

lexemes[7].setLexeme('n'); lexemes[8].setToken(Token.Minus);

lexemes[8].setPrev(lexemes[7]);

lexemes[8].setNext(lexemes[9]); lexemes[9].setToken(Token.constant);

lexemes[9].setLexeme(1); lexemes[9].setNext(lexemes[10]); lexemes[10].setToken(Token.Semicolon); lexemes[10].setNext(lexemes[11]); lexemes[11].setToken(Token.Assign); lexemes[11].setPrev(lexemes[12]); lexemes[11].setNext(lexemes[13]); lexemes[12].setToken(Token.variable);

lexemes[12].setLexeme('b'); lexemes[13].setToken(Token.Plus); lexemes[13].setPrev(lexemes[12]); lexemes[13].setNext(lexemes[14]); lexemes[14].setToken(Token.LeftBracket); lexemes[14].setPrev(lexemes[15]); lexemes[14].setNext(lexemes[16]);

lexemes[15].setToken(Token.variable);

lexemes[15].setLexeme('a'); lexemes[16].setToken(Token.variable);

lexemes[16].setLexeme('n'); lexemes[16].setNext(lexemes[17]); lexemes[17].setToken(Token.RightBracket); lexemes[17].setNext(lexemes[18]); lexemes[18].setToken(Token.End);

lexemes[0].print();

Graph g =new Graph();

g.makeGraph();

}

}

public class Lexeme {

private Token token;

private Lexeme prev;

private Lexeme next;

private Object lexeme;

public Lexeme(){}

public Lexeme(Token token) {

this.token = token;

}

public Lexeme(Token token, Object lexeme) {

this.token = token;

this.lexeme = lexeme;

}

public Lexeme(Token token, Lexeme prev, Lexeme next) {

this.token = token;

this.prev = prev;

this.next = next;

}

public Lexeme(Token token, Lexeme prev, Lexeme next, Object lexeme) {

this.token = token;

this.prev = prev;

this.next = next;

this.lexeme = lexeme;

}

public void print(){

if(prev!=null){

prev.print();

} System.out.print(this.toString());

if(next!=null){

next.print();

}

}

@Override

public String toString() {

if(token==Token.variable){

return Character.toString(Token.variable.toChar((char)lexeme));

} if(token==Token.constant){//elif

return Integer.toString(Token.variable.toInt((int)lexeme));

}

return token.toString();

}

public Token getToken() {

return token;

}

public void setToken(Token token) {

this.token = token;

}

public Lexeme getPrev() {

return prev;

}

public void setPrev(Lexeme prev) {

this.prev = prev;

}

public Lexeme getNext() {

return next;

}

public void setNext(Lexeme next) {

this.next = next;

}

public Object getLexeme() {

return lexeme;

}

public void setLexeme(Object lexeme) {

this.lexeme = lexeme;

}

}

public enum Token {

variable,constant,

RightParentheses{

@Override

public String toString() {

return ")";}},

LeftParentheses{

@Override

public String toString() {

return "(";}},

RightBracket{

@Override

public String toString() {

return "]";}},

LeftBracket{

@Override

public String toString() {

return "[";}},

While{

@Override

public String toString() {

return "while ";}},

NotEquals{

@Override

public String toString() {

return " <> ";}},

Do{

@Override

public String toString() {

return " do";}},

Begin{

@Override

public String toString() {

return " begin ";}},

End{

@Override

public String toString() {

return " end";}},

Assign{

@Override

public String toString() {

return " := ";}},

Minus{

@Override

public String toString() {

return "-";}},

Semicolon{

@Override

public String toString() {

return "; ";}},

Plus{

@Override

public String toString() {

return "+";}};

public char toChar(char c) {

return c;}

public int toInt(int n) {

return n;

}

}

public class Graph {

public void makeGraph(){

System.out.println();

System.out.println();

String [] graph = {

"s0 -> s1",

"s1 -> s2",

"s2 -> s3",

"s3 -> s4",

"s4 -> s5",

"s5 -> s6",

"s6 -> s7",

"s7 -> s8",

"s8 -> s9",

};

String [] signals = {

"dlm",

"dlm",

"dlm",

"ltr",

"dlm",

"ltr",

"dlm",

"ltr",

"dlm",

"ctr",

"dlm",

"ctr",

"dlm"

};

int j = 0;

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

for(; j < graph.length; ){

if(j == 3){ if(i<signals.length){ if(signals[i].equals("dlm")){ System.out.println("s3 -> s7, (dlm)");

j = 7;

}

else if(signals[i].equals("ltr")){ System.out.println("s3 -> s2, (ltr)");

j = 2;

}

else {

System.out.println(graph[j]);

j++;

}

}

else break;

}

else if(j == 5){

if(i<signals.length){

if(signals[i].equals("ctr")){

System.out.println("s5 -> s8, (ctr)");

j = 8;

}

else {

System.out.println(graph[j]);

j++;

}

}

else break;

}

else{

if(i<signals.length) {

if (j < graph.length) {

System.out.println(graph[j]);

j++;

}

}

else break;

}

i++; }}}}