Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота № 2**

**з дисципліни**

**«Системне програмування»**

Виконала: студентка групи ІО-91

Косейкіна Ганна Сергіївна

2011 р.

Мета: Побудова та використання об’єктів вузлів деревовидних графів. Лексичний аналізатор.

Лістінг

**package** Ttable;

**public** **class** table {

**public** table(record head){

**if** (head != **null**){

**this**.head = **new** element(head);

**this**.last = **this**.head;

ln = 1;

}

**else**{

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

}

}

**public** table(){

**this**.ln=0;

**this**.head=**new** element();

**this**.last=**this**.head;

}

**public** **int** add(record rec){

**if** (rec != **null**){

last.next = **new** element(rec);

last = last.next;

ln++;

}

**else**{

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

}

**return** ln;

}

**public** **void** addLin(key k, record rec){

element e = head;

**while**(e.next != **null**){

**if**((e.rec.keys.ARG2 < k.ARG2) &&(e.next.rec.keys.ARG2 >= k.ARG2)){

element t = **new** element(rec);

t.next = e.next;

e.next = t;

ln++;

**break**;

}

e = e.next;

}

}

**public** **void** addBin(key k, record rec){

element e = findBin(k);

**if**(e == **null**)

add(rec);

**else**{

element t = **new** element(rec);

t.next = e.next;

e.next = t;

}

ln++;

}

**public** element findLin(key k){

element e = head;

**while**(e.next != **null**){

**if**(e.rec.keys.ARG1 == k.ARG1)

**return** e;

e = e.next;

}

**return** **null**;

}

**public** element findNum(**int** index){

element e = head;

**for**(**int** i = 1; i < index+1; i++)

**if**(e.next == **null**)

**return** **null**;

**else**

e = e.next;

**return** e;

}

**public** **int** find(String str ){

**int** ind=0;

element e=head;

**if**(str==**null**){

ind= -1;

}

**if**(str.length()<=0){

ind= -1;

}

**if** (e.rec.keys.ARG1.equals(str)){

ind=0;

}

**for** (**int** i=1; i<ln; i++){

**if** (e.next.rec.keys.ARG1.equals(str)){

ind=i;

}

}

**return** ind;

}

**public** element findBin(key k){

element t = head;

element e = t;

**int** ln = **this**.ln;

**while**(ln != 0){

e = t;

**for**(**int** i = 0; i <(ln/2); i++)

e = e.next;

**if**(e.rec.keys.ARG2 == k.ARG2)

**return** e;

**else** **if**(e.rec.keys.ARG2 < k.ARG2)

t = e;

ln /= 2;

}

e = e.next;

**if**(e.rec.keys.ARG2 == k.ARG2)

**return** e;

**return** **null**;

}

**public** **void** del(**int** index){

element e = findNum(index);

**if**(e != **null**)

e.rec.mode = -1;

**else**

System.*out*.println("----index incorrect----");

}

**public** **void** pack(){

element e = head;

**while**(e.next != **null**){

**if**(e.next.rec.mode == -1){

e.next = e.next.next;

ln--;

}

**else**

e = e.next;

}

**if**(head.rec.mode == -1){

head = head.next;

ln--;

}

}

**public** **void** showElement(element e){

**if**(e != **null**){

**if**(e.rec.mode == -1)

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

**else**

System.*out*.println(e.rec.keys.ARG1+" "+e.rec.keys.ARG2+" "+e.rec.fun.fun);

}**else**

System.*out*.println("record not found");

}

**public** **void** show(){

**int** i = 0;

element e = head;

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

**do**{

System.*out*.print(i+") ");

showElement(e);

e = e.next;

i++;

}

**while**(e != **null**);

}

**public** **int** ln;

**public** element head;

**public** element last;

}

**package** Ttable;

**public** **class** record {

**public** record(String ARG1, **long** ARG2, **int** fun){

**this**.keys = **new** key(ARG1, ARG2);

**this**.fun = **new** func(fun);

**this**.mode = 0;

}

**public** record(){

**this**.keys=**new** key();

**this**.fun=**new** func();

**this**.mode=0;

}

**public** key keys;

**public** func fun;

**public** **int** mode;

}

**package** Ttable;

**public** **class** key {

**public** key(String ARG1, **long** ARG2){

**this**.ARG1 = ARG1;

**this**.ARG2 = ARG2;

}

**public** key(){

ARG1=**null**;

ARG2=0;

}

**public** String ARG1;

**public** **long** ARG2;

}

**package** Ttable;

**public** **class** index {

**public** index(**int** N){

ind = **new** index[N];

e = **new** element[N];

index = **new** **int**[N];

}

index[] ind;

element[] e;

**int**[] index;

}

**package** Ttable;

**public** **class** func {

**public** func(**int** fun){

**this**.fun = fun;

}

**public** func(){

fun=0;

}

**public** **int** fun;

}

**package** Ttable;

**public** **class** element {

**public** element(record rec){

**this**.rec = rec;

}

**public** element(){

rec=**new** record();

}

**public** record rec;

**public** element next = **null**;

}

**package** tree;

**public** **class** IncorectNodeException **extends** Exception {

**public** **void** printStackTrace(){

System.*out*.println("Exception Incorect node");

**super**.printStackTrace();

}

}

**package** tree;

**public** **class** IncorectParameters **extends** Exception {

**public** **void** printStackTrace(){

System.*out*.println("Exception Incorect parametrs");

**super**.printStackTrace();

}

}

**package** tree;

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

**public** Node(Node ch1,Node ch2,**int** type1,**int** index1,**int** concatType1, TablesOperator tablesOperator1) **throws** IncorectNodeException{

child1=ch1;

child2=ch2;

**if**(concatType1>3 || concatType1<0 || tablesOperator1==**null** || type1>3 || type1<0){

**throw** **new** IncorectNodeException();

}

type=type1;

index=index1;

concatType=concatType1;

tablesOperator=tablesOperator1;

}

**public** String calculateValue(){

String ch1="";

String ch2="";

**if**(child1!=**null**){

ch1=child1.calculateValue();

}

**if**(child2!=**null**){

ch2=child2.calculateValue();

}

String res="";

String value=tablesOperator.getValue(type,index);

**switch** (concatType){

**case** 0:res=value+ch1+ch2;**break**;

**case** 1:res=ch1+value+ch2;**break**;

**case** 2:res=ch1+ch2+value;**break**;

}

**return** res;

}

**public** Node child1;

**public** Node child2;

**public** **int** type;

**public** **int** index;

/\*\*

\* concatType=0 -- operation at left

\* concatType=1 -- operation at centre

\* concatType=2 -- operation at right

\*/

**public** **static** **int** *SPLIT\_LEFT*=0;

**public** **static** **int** *SPLIT\_CENTRE*=1;

**public** **static** **int** *SPLIT\_RIGHT*=2;

**public** **int** concatType;

**public** TablesOperator tablesOperator;

}

**package** tree;

**import** Ttable.\*;

**public** **class** TablesOperator {

**public** TablesOperator(){

identifiers=**new** table();

constants=**new** table();

operations=**new** table();

}

**public** String getValue(**int** type,**int** index){

String res=**null**;

**switch** (type){

**case** 0: res=identifiers.findNum(index).rec.keys.ARG1; **break**;

**case** 1: res=constants.findNum(index).rec.keys.ARG1; **break**;

**case** 2: res=operations.findNum(index).rec.keys.ARG1; **break**;

}

**if**(res==**null**){

System.*out*.print("Error!");

}

**return** res;

}

**public** **int** insertValue(**int** type,String value) {

**int** index=-1;

**switch** (type){

**case** 0: index=(**int**)identifiers.add(**new** record(value,0,0)); **break**;

**case** 1: index=(**int**)constants.add(**new** record(value,0,0)); **break**;

**case** 2: index=(**int**)operations.add(**new** record(value,0,0)); **break**;

}

**return** index;

}

/\*

type=0 - identifiers;

type=1 - constants;

type=2 - operations;

\*/

**public** **static** **int** *IDENTIFIERS*=0;

**public** **static** **int** *CONSTANTS*=1;

**public** **static** **int** *OPERATIONS*=2;

**private** table identifiers;

**private** table constants;

**private** table operations;

}

**package** LexicalScaner;

**public** **class** IncorectAnalizeException **extends** Exception {

**public** **void** printStackTrace(){

System.*out*.println("Exception Incorect data then analize is failed");

**super**.printStackTrace();

}

}

**package** LexicalScaner;

**public** **class** IncorectTokenException **extends** Exception {

**public** **void** printStackTrace(){

System.*out*.println("Exception Incorect token");

**super**.printStackTrace();

}

}

**package** LexicalScaner;

**public** **class** InvalidLine **extends** Exception {

**public** **void** printStackTrace(){

System.*out*.println("Exception Incorect line");

**super**.printStackTrace();

}

}

**package** LexicalScaner;

**import** Automat.AutomatesManager;

**import** Ttable.element;

**import** Ttable.record;

**import** Ttable.table;

**import** java.util.ArrayList;

**import** java.util.StringTokenizer;

**public** **class** LexicalAnalizator {

**public** LexicalAnalizator(table delimiters1,table terminalWords1,AutomatesManager automatesManager1) **throws** IncorectAnalizeException{

**if**(automatesManager1==**null** ||delimiters1==**null** || terminalWords1==**null**){

**throw** **new** IncorectAnalizeException();

}

automatesManager=automatesManager1;

delimiters=delimiters1;

terminalWords=terminalWords1;

identifiers=**new** table();

constants=**new** table();

}

**public** **void** scann(String line) **throws** InvalidLine{

**if**(line==**null**){

**throw** **new** InvalidLine();

}

**if**(line.length()==0){

**throw** **new** InvalidLine();

}

String delimSet=**new** String();

**for**(**int** i=0;i<delimiters.ln;i++){

element element=delimiters.findNum(i);

**if**(element!=**null**){

delimSet+=element.rec.keys.ARG1;

}

}

StringTokenizer tokenizer=**new** StringTokenizer(line,delimSet,**true**);

ArrayList<String> tokens=**new** ArrayList<String>();

**while** (tokenizer.hasMoreTokens()){

tokens.add(tokenizer.nextToken());

}

parts=**new** String[tokens.size()];

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

parts[i]=tokens.get(i);

}

}

**public** **void** analize(String line) **throws** IncorectAnalizeException{

**try**{

scann(line);

}**catch** (InvalidLine e){

**throw** **new** IncorectAnalizeException();

}

tokens=**new** Token[parts.length];

**try**{

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

Token currentToken;

**int** r=delimiters.find(parts[i]);

System.*out*.println("token line="+parts[i]);

**if**(r!=-1){

System.*out*.println(" type=delimiter index="+r);

currentToken=**new** Token(parts[i],Token.*DELIMITER*,r);

tokens[i]=currentToken;

**continue**;

}

r = terminalWords.find(parts[i]);

**if**(r!=-1){

System.*out*.println(" type=terminalwords index="+r);

currentToken=**new** Token(parts[i],Token.*TERMINALWORD*,r);

tokens[i]=currentToken;

**continue**;

}

System.*out*.println(" automat start work");

r=automatesManager.getTokenTypes(parts[i]);

**if**(r!=-1){

**if**(r==Token.*CONSTANT*){

System.*out*.println(" type=constant index="+r);

**int** index=constants.add(**new** record(parts[i],0,0));

currentToken=**new** Token(parts[i],Token.*CONSTANT*,index);

tokens[i]=currentToken;

**continue**;

}

**if**(r==Token.*IDENTIFIER*){

System.*out*.println(" type=identifier index="+r);

**int** index=identifiers.add(**new** record(parts[i],0,0));

currentToken=**new** Token(parts[i],Token.*IDENTIFIER*,index);

tokens[i]=currentToken;

**continue**;

}

}**else**{

**throw** **new** IncorectAnalizeException();

}

}

}**catch** (Exception e){

**throw** **new** IncorectAnalizeException();

}

}

**public** AutomatesManager automatesManager;

**public** table delimiters;

**public** String[] parts;

**public** Token[] tokens;

**public** table identifiers;

**public** table constants;

**public** table terminalWords;

}

**package** LexicalScaner;

**public** **class** Token {

**public** Token(String line1,**int** type1,**int** index1) **throws** IncorectTokenException{

**if**(line1==**null** || type1<1 || type1>4 || index1<0){

**throw** **new** IncorectTokenException();

}

line=line1;

type=type1;

index=index1;

}

**public** **static** **int** *CONSTANT*=3;

**public** **static** **int** *DELIMITER*=1;

**public** **static** **int** *TERMINALWORD*=2;

**public** **static** **int** *IDENTIFIER*=4;

**public** String line;

**public** **int** type; //1-delimiter, 2-terminal word, 3- constant,4-identifier

**public** **int** index;

}

**package** Automat;

**import** java.io.IOException;

**import** java.util.ArrayList;

**public** **class** AutomatesManager {

**public** ArrayList<AutomatTable> tables;

**public** AutomatesManager(ArrayList<AutomatTable> tables1) **throws** InvalidTableException{

**if**(tables1!=**null**){

tables=tables1;

}**else**{

**throw** **new** InvalidTableException();

}

}

**public** AutomatesManager(String[] fileNames) **throws** InvalidTableException,IncorectFileException,IOException{

**if**(fileNames!=**null**){

tables=**new** ArrayList<AutomatTable>();

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

tables.add(LoadAutomateTable.*load*(fileNames[i]));

}

}**else** {

**throw** **new** InvalidTableException();

}

}

**public** **int** getTokenTypes(String token){

**int** maxN=-1;

**int** max=-1;

**int** curr;

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

curr=tables.get(i).check(token);

System.*out*.println("curr="+curr);

**if**(curr>max){

max=curr;

maxN=i;

}

}

**if**(max>-1){

**return** tables.get(maxN).typeSolution;

}

**return** -1;

}

}

**package** Automat;

**public** **class** AutomatTable {

**public** **int** typeSolution;

**public** String[][] table;

**public** **int** startPosition;

**public** **int**[] endPositions;

**public** AutomatTable(String[][] table1,**int** startPosition1,**int**[] endPositions1,**int** typeSolution1) **throws** InvalidTableException{

**if**(table1==**null** || endPositions1==**null**){

System.*out*.println("table or end position null");

**throw** **new** InvalidTableException();

}

**if**(table1.length==0 || endPositions1.length==0){

System.*out*.println("table or end position size incorect");

**throw** **new** InvalidTableException();

}

**if**(table1.length!=table1[0].length){

System.*out*.println("table size incorect");

**throw** **new** InvalidTableException();

}

**if**(startPosition1<0 || startPosition1>table1.length){

System.*out*.println("start position incorect");

**throw** **new** InvalidTableException();

}

**for**(**int** i=0;i<endPositions1.length;i++){

**if**(endPositions1[i]<0 || endPositions1[i]>table1.length){

System.*out*.println("end position incorect");

**throw** **new** InvalidTableException();

}

}

typeSolution=typeSolution1;

table=table1;

startPosition=startPosition1;

endPositions=endPositions1;

}

**public** **int** check(String line){

**if**(line==**null**){

**return** -1;

}

**if**(line.length()==0){

**return** -1;

}

**int** stepCount=0;

**boolean** haveFind;

**int** currentPos=startPosition;

System.*out*.println("table size i="+table.length+" j="+table[0].length);

System.*out*.println("end pos="+endPositions[0]+" length="+endPositions.length);

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

haveFind=**false**;

**for** (**int** j=0;j<table.length;j++){

**if**(table[currentPos][j]!=**null**){

System.*out*.println("Current pos="+currentPos+" symb="+line.charAt(i));

**if**(table[currentPos][j].indexOf(line.charAt(i))>-1){

System.*out*.println("have jump to "+j);

currentPos=j;

stepCount++;

haveFind=**true**;

**break**;

}

}

}

**if**(!haveFind){

**for** (**int** j=0;j<endPositions.length;j++){

**if**(endPositions[j]==currentPos){

**return** stepCount;

}

}

**return** -1;

}

}

**return** stepCount;

}

}

**package** Automat;

**public** **class** IncorectFileException **extends** Exception {

**public** **void** printStackTrace(){

System.*out*.println("Exception Incorect data int file");

**super**.printStackTrace();

}

}

**package** Automat;

**public** **class** InvalidTableException **extends** Exception {

**public** **void** printStackTrace(){

System.*out*.println("Exception Incorect table");

**super**.printStackTrace();

}

}

**package** Automat;

**import** java.io.BufferedReader;

**import** java.io.FileInputStream;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.util.ArrayList;

**public** **class** LoadAutomateTable {

**public** **static** AutomatTable load(String fileName) **throws** IOException,IncorectFileException{

BufferedReader in;

AutomatTable automatTable;

ArrayList<String> lines=**new** ArrayList<String>();

String type="";

**try**{

in = **new** BufferedReader(**new** InputStreamReader(**new** FileInputStream(fileName)));

**boolean** fl=**false**;

**while** (in.ready()) {

String s = in.readLine();

**if**(fl){

lines.add(s);

}**else**{

fl = **true**;

type=s;

}

}

in.close();

}**catch** (IOException e) {

**throw** **new** IOException("Cann't read file");

}

**if**(lines.size()<0){

**throw** **new** IncorectFileException();

}

String[][] table=**new** String[lines.size()][lines.size()];

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

String s=lines.get(i);

**int** st=0;

**while** (**true**){

**int** stBr=s.indexOf("(");

**int** stFBr=s.indexOf(" )");

String num=s.substring(st,stBr);

**int** n=Integer.*valueOf*(num);

String value=s.substring(stBr+1,stFBr);

table[i][n]=value;

s=s.substring(stFBr+2,s.length());

**if**(s.length()==0){

**break**;

}

}

}

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

**for**(**int** j=0;j<table[0].length;j++){

System.*out*.print(table[i][j]+" ");

**if**(table[i][j]!=**null**){

System.*out*.print("length="+table[i][j].length());

}

}

System.*out*.println();

}

String paternStrart=" start position=";

String paternEnd=" end position=";

**int** stPos=type.indexOf(paternStrart);

**int** typeInt=Integer.*valueOf*(type.substring(0,stPos));

type=type.substring(stPos+paternStrart.length(),type.length());

**int** endPos=type.indexOf(paternEnd);

**int** startPosition=Integer.*valueOf*(type.substring(0, endPos));

type=type.substring(endPos+paternEnd.length(),type.length());

ArrayList<Integer> positions=**new** ArrayList<Integer>();

**while** (**true**){

**int** colPos=type.indexOf(",");

**if**(colPos!=-1){

positions.add(Integer.*valueOf*(type.substring(0,colPos)));

type=type.substring(colPos+1,type.length());

}

System.*out*.println("type="+type);

**if**(type.indexOf(",")==-1){

positions.add(Integer.*valueOf*(type));

**break**;

}

}

**int**[] endposVar=**new** **int**[positions.size()];

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

endposVar[i]=positions.get(i);

System.*out*.println("position="+endposVar[i]);

}

**try**{

automatTable=**new** AutomatTable(table,startPosition,endposVar,typeInt);

}**catch** (InvalidTableException e){

**throw** **new** IncorectFileException();

}

**return** automatTable;

}

}

**package** UnitTests;

**import** LexicalScaner.LexicalAnalizator;

**import** LexicalScaner.Token;

**import** junit.framework.TestCase;

**import** junit.framework.TestSuite;

**import** junit.textui.TestRunner;

**import** Ttable.record;

**import** Ttable.table;

**import** tree.Node;

**import** tree.TablesOperator;

**import** Automat.\*;

**public** **class** TestClass **extends** TestCase{

**public** TestClass(String testName) {

**super**(testName);

}

**public** **void** testTree() {

Node head=**null**;

String formula="f=a\*b+c-d";

String cFormula=**null**;

**try**{

TablesOperator tablesOperator=**new** TablesOperator();

**int** a=tablesOperator.insertValue(TablesOperator.*IDENTIFIERS*,"a");

**int** f=tablesOperator.insertValue(TablesOperator.*IDENTIFIERS*,"f");

**int** b=tablesOperator.insertValue(TablesOperator.*IDENTIFIERS*,"b");

**int** c=tablesOperator.insertValue(TablesOperator.*IDENTIFIERS*,"c");

**int** d=tablesOperator.insertValue(TablesOperator.*IDENTIFIERS*,"d");

**int** op1=tablesOperator.insertValue(TablesOperator.*OPERATIONS*,"=");

**int** op2=tablesOperator.insertValue(TablesOperator.*OPERATIONS*,"+");

**int** op3=tablesOperator.insertValue(TablesOperator.*OPERATIONS*,"\*");

**int** op4=tablesOperator.insertValue(TablesOperator.*OPERATIONS*,"-");

Node A=**new** Node(**null**,**null**,TablesOperator.*IDENTIFIERS*,a,Node.*SPLIT\_CENTRE*,tablesOperator);

Node B=**new** Node(**null**,**null**,TablesOperator.*IDENTIFIERS*,b,Node.*SPLIT\_CENTRE*,tablesOperator);

Node C=**new** Node(**null**,**null**,TablesOperator.*IDENTIFIERS*,c,Node.*SPLIT\_CENTRE*,tablesOperator);

Node D=**new** Node(**null**,**null**,TablesOperator.*IDENTIFIERS*,d,Node.*SPLIT\_CENTRE*,tablesOperator);

Node F=**new** Node(**null**,**null**,TablesOperator.*IDENTIFIERS*,f,Node.*SPLIT\_CENTRE*,tablesOperator);

Node OP4=**new** Node(C,D,TablesOperator.*OPERATIONS*,op4,Node.*SPLIT\_CENTRE*,tablesOperator);

Node OP3=**new** Node(A,B,TablesOperator.*OPERATIONS*,op3,Node.*SPLIT\_CENTRE*,tablesOperator);

Node OP2=**new** Node(OP3,OP4,TablesOperator.*OPERATIONS*,op2,Node.*SPLIT\_CENTRE*,tablesOperator);

head=**new** Node(F,OP2,TablesOperator.*OPERATIONS*,op1,Node.*SPLIT\_CENTRE*,tablesOperator);

cFormula=head.calculateValue();

}**catch** (Exception e){

e.printStackTrace();

}

*assertEquals*(formula,cFormula);

}

**public** **void** test1Tree() {

Node head=**null**;

String formula="while(i>0)i--";

String cFormula=**null**;

**try**{

TablesOperator tablesOperator=**new** TablesOperator();

**int** o1=tablesOperator.insertValue(TablesOperator.*OPERATIONS*,"(");

**int** i=tablesOperator.insertValue(TablesOperator.*IDENTIFIERS*,"i");

**int** o2=tablesOperator.insertValue(TablesOperator.*CONSTANTS*,"0");

**int** op1=tablesOperator.insertValue(TablesOperator.*OPERATIONS*,">");

**int** op2=tablesOperator.insertValue(TablesOperator.*OPERATIONS*,"while");

**int** op3=tablesOperator.insertValue(TablesOperator.*OPERATIONS*,"--");

**int** op4=tablesOperator.insertValue(TablesOperator.*OPERATIONS*,")");

Node A=**new** Node(**null**,**null**,TablesOperator.*IDENTIFIERS*,i,Node.*SPLIT\_CENTRE*,tablesOperator);

Node B=**new** Node(**null**,**null**,TablesOperator.*CONSTANTS*,o2,Node.*SPLIT\_CENTRE*,tablesOperator);

Node F=**new** Node(**null**,**null**,TablesOperator.*OPERATIONS*,op2,Node.*SPLIT\_CENTRE*,tablesOperator);

Node C=**new** Node(A,B,TablesOperator.*OPERATIONS*,op1,Node.*SPLIT\_CENTRE*,tablesOperator);

Node D=**new** Node(F,C,TablesOperator.*OPERATIONS*,o1,Node.*SPLIT\_CENTRE*,tablesOperator);

Node OP3=**new** Node(A,**null**,TablesOperator.*OPERATIONS*,op3,Node.*SPLIT\_CENTRE*,tablesOperator);

head=**new** Node(D,OP3,TablesOperator.*OPERATIONS*,op4,Node.*SPLIT\_CENTRE*,tablesOperator);

cFormula=head.calculateValue();

}**catch** (Exception e){

e.printStackTrace();

}

*assertEquals*(formula,cFormula);

}

**public** **void** testLoadAutomate(){

String[][] t=**new** String[4][4];

t[0][1]="ab";

t[1][0]="d";

t[1][3]="c";

t[2][2]="e";

t[3][3]="e";

**int**[] endpos=**new** **int**[2];

endpos[0]=1;

endpos[1]=2;

AutomatTable etalon=**null**;

AutomatTable table=**null**;

**try**{

etalon=**new** AutomatTable(t,0,endpos,1);

table=LoadAutomateTable.*load*("C:\\automats\\automat2.txt");

}**catch**(Exception e){

e.printStackTrace();

}

*assertEquals*(table,etalon);

}

**public** **void** testAutomate(){

**try**{

String[] tableFiles={"C:\\automats\\automat1.txt","C:\\automats\\automat3.txt"};

AutomatesManager automatesManager=**new** AutomatesManager(tableFiles);

String line1="i0de123ntif";

String line2="3";

**int** res1=automatesManager.getTokenTypes(line1);

**int** res2=automatesManager.getTokenTypes(line2);

System.*out*.println("res1="+res1+" res2="+res2);

*assertEquals*(res1,Token.*IDENTIFIER*);

*assertEquals*(res2,Token.*CONSTANT*);

}**catch** (Exception e){

e.printStackTrace();

}

}

**public** **void** testAutomateMethodCheck(){

**try**{

String[] tableFiles={"C:\\automats\\automat4.txt"};

AutomatesManager automatesManager=**new** AutomatesManager(tableFiles);

String line2="+0.34.5";

**int** res2=automatesManager.tables.get(0).check(line2);

System.*out*.println("result="+res2);

*assertEquals*(res2,Token.*CONSTANT*);

}**catch** (Exception e){

e.printStackTrace();

}

}

**public** LexicalAnalizator createLexicalAnalizator(){

String[]automateFiles=**new** String[2];

automateFiles[0]="C:\\automats\\automat1.txt";

automateFiles[1]="C:\\automats\\automat3.txt";

LexicalAnalizator lexicalAnalizator=**null**;

**try**{

AutomatesManager automatesManager=**new** AutomatesManager(automateFiles);

table delimiters=**new** table();

delimiters.add(**new** record(" ",0,0));

delimiters.add(**new** record(",",0,0));

delimiters.add(**new** record(".",0,0));

delimiters.add(**new** record("(",0,0));

delimiters.add(**new** record(")",0,0));

delimiters.add(**new** record("+",0,0));

delimiters.add(**new** record("-",0,0));

delimiters.add(**new** record("\*",0,0));

delimiters.add(**new** record("/",0,0));

delimiters.add(**new** record("<",0,0));

delimiters.add(**new** record(">",0,0));

delimiters.add(**new** record("=",0,0));

delimiters.add(**new** record(";",0,0));

delimiters.add(**new** record("{",0,0));

delimiters.add(**new** record("}",0,0));

table terminalWords=**new** table();

terminalWords.add(**new** record("if", 0,0));

terminalWords.add(**new** record("for",0,0));

terminalWords.add(**new** record("while",0,0));

lexicalAnalizator=**new** LexicalAnalizator(delimiters,terminalWords,automatesManager);

}**catch** (Exception e){

e.printStackTrace();

}

**return** lexicalAnalizator;

}

**public** **void** testMethodScan(){

**try**{

LexicalAnalizator lexicalAnalizator=**this**.createLexicalAnalizator();

lexicalAnalizator.scann("if(abc<b){c=b-a;}");

String[] res={"if","(","abc","<","b",")","{","c","=","b","-","a",";","}"};

**for**(**int** i=0;i<lexicalAnalizator.parts.length;i++){

*assertEquals*(res[i],lexicalAnalizator.parts[i]);

}

}**catch** (Exception e){

e.printStackTrace();

}

}

**public** **void** testLexicalAnalizator(){

**try**{

LexicalAnalizator lexicalAnalizator=**this**.createLexicalAnalizator();

lexicalAnalizator.analize("if(abc<b){c=b-5;}");

String[] res={"if","(","abc","<","b",")","{","c","=","b","-","5",";","}"};

Token[] resT=**new** Token[res.length];

resT[0]=**new** Token(res[0],Token.*TERMINALWORD*,1);

resT[1]=**new** Token(res[1],Token.*DELIMITER*,1);

resT[2]=**new** Token(res[2],Token.*IDENTIFIER*,1);

resT[3]=**new** Token(res[3],Token.*DELIMITER*,1);

resT[4]=**new** Token(res[4],Token.*IDENTIFIER*,1);

resT[5]=**new** Token(res[5],Token.*DELIMITER*,1);

resT[6]=**new** Token(res[6],Token.*DELIMITER*,1);

resT[7]=**new** Token(res[7],Token.*IDENTIFIER*,1);

resT[8]=**new** Token(res[8],Token.*DELIMITER*,1);

resT[9]=**new** Token(res[9],Token.*IDENTIFIER*,1);

resT[10]=**new** Token(res[10],Token.*DELIMITER*,1);

resT[11]=**new** Token(res[11],Token.*CONSTANT*,1);

resT[12]=**new** Token(res[12],Token.*DELIMITER*,1);

resT[13]=**new** Token(res[13],Token.*DELIMITER*,1);

System.*out*.println("size rest="+resT.length+" lex="+lexicalAnalizator.tokens.length);

**for**(**int** i=0;i<lexicalAnalizator.tokens.length;i++){

**if**(lexicalAnalizator.tokens[i]!=**null**)

System.*out*.println("token i="+i+" v="+lexicalAnalizator.tokens[i].line);}

**for**(**int** i=0;i<lexicalAnalizator.tokens.length;i++){

System.*out*.println("i="+i);

System.*out*.println("resT="+resT[i].line);

System.*out*.println("lexical="+lexicalAnalizator.tokens[i].line);

*assertEquals*(resT[i].line,lexicalAnalizator.tokens[i].line);

*assertEquals*(resT[i].type,lexicalAnalizator.tokens[i].type);}

}**catch** (Exception e){

e.printStackTrace();} }

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

TestRunner runner = **new** TestRunner();

TestSuite suite = **new** TestSuite();

suite.addTest(**new** TestClass("testTree"));

suite.addTest(**new** TestClass("test1Tree"));

suite.addTest(**new** TestClass("testLoadAutomate"));

suite.addTest(**new** TestClass("testAutomate"));

suite.addTest(**new** TestClass("testMethodScan"));

suite.addTest(**new** TestClass("testLexicalAnalizator"));

suite.addTest(**new** TestClass("testAutomateMethodCheck"));

runner.doRun(suite);

}}