

Національний технічний університет

«Київський політехнічний інститут»

Факультет інформатики і обчислювальної техніки,

Лабораторна робота № 4

|  |
| --- |
| Виконав: |
| студент 3 курсу ФІОТ |
| Групи ІО-91  Матвійчук Б.В. |
|  |

Київ-2011

package SyntaxicAnalizer;

import LexicalScaner.Token;

import tree.Node;

public class Element {

public Token token;

public String irrExpr;

public boolean isToken;

public Node tree;

public Element(Token token1){

if(token1==null){

throw new IllegalArgumentException();

}

token=token1;

isToken=true;

}

public Element(String irrExpr1){

if(irrExpr1==null){

throw new IllegalArgumentException();

}

irrExpr=irrExpr1;

isToken=false;

}

public boolean equals(Object element){

try{

Element e=(Element)element;

if(e.isToken==isToken && isToken==true){

if(e.token.equals(this.token)){

return true;

}

return false;

}else{

if(e.irrExpr.equals(irrExpr)){

return true;

}

return false;

}

}catch (Exception e){

return false;

}

}

public void printElement(){

if(isToken){

System.out.println("token element="+token.line);

}else{

System.out.println("expr element="+irrExpr);

}

}

}

package SyntaxicAnalizer;

import LexicalScaner.LexicalAnalizator;

import LexicalScaner.Token;

import com.sun.javaws.exceptions.InvalidArgumentException;

import tree.IncorectNodeException;

import tree.Node;

import tree.TablesOperator;

import java.util.ArrayList;

public class Rule {

public String name;

public ArrayList<Element> elements;

public int priority;

public static final String EXPR="EE";

public static final String ID="ID";

public static final String CS="CS";

public static final String TERMWORD="TM";

public static final String DELIM="DM";

public Rule(){

}

public Rule(String name1,ArrayList<Element> elements1,int priority1) throws InvalidArgumentException{

if(name1==null || elements1==null){

throw new IllegalArgumentException();

}

name=name1;

elements=elements1;

priority=priority1;

}

public boolean isExprForThisRule(ArrayList<Element> buf,int pos){

if(buf.size()<elements.size()+pos){

return false;

}

int n=0;

boolean isCorect;

for(int i=pos;i<buf.size();i++){

isCorect=false;

if(!buf.get(i).equals(elements.get(n))){

return false;

}

if(n ==elements.size()-1){

return true;

}

n++;

}

return true;

}

public Node executeRule(LexicalAnalizator lexicalAnalizator,StartRule rule,ArrayList<Element>buff,ArrayList<Node>){

Node head=new Node();

int n=1;

Node[] nodes=new Node[elements.size()];

try{

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

if(elements.get(i).isToken){

nodes[i]=new Node(null,null,elements.get(i).token.type,elements.get(i).token.index,Node.SPLIT\_CENTRE,lexicalAnalizator);

}

}

}catch (IncorectNodeException e){

e.printStackTrace();

}

if(name.equals("if")){

Node head=new Node(

}

Element newElement=new Element(EXPR+rule.rule.name);

newElement.tree=head;

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

buff.remove(rule.position);

}

buff.add(rule.position,newElement);

}

}

package SyntaxicAnalizer;

import LexicalScaner.LexicalAnalizator;

import LexicalScaner.Token;

import com.sun.javaws.exceptions.InvalidArgumentException;

import tree.IncorectNodeException;

import tree.Node;

import tree.TablesOperator;

import java.util.ArrayList;

public class Rule {

public String name;

public ArrayList<Element> elements;

public int priority;

public static final String EXPR="EE";

public static final String ID="ID";

public static final String CS="CS";

public static final String TERMWORD="TM";

public static final String DELIM="DM";

public Rule(){

}

public Rule(String name1,ArrayList<Element> elements1,int priority1) throws InvalidArgumentException{

if(name1==null || elements1==null){

throw new IllegalArgumentException();

}

name=name1;

elements=elements1;

priority=priority1;

}

public boolean isExprForThisRule(ArrayList<Element> buf,int pos){

if(buf.size()<elements.size()+pos){

return false;

}

int n=0;

boolean isCorect;

for(int i=pos;i<buf.size();i++){

isCorect=false;

if(!buf.get(i).equals(elements.get(n))){

return false;

}

if(n ==elements.size()-1){

return true;

}

n++;

}

return true;

}

public Node executeRule(LexicalAnalizator lexicalAnalizator,StartRule rule,ArrayList<Element>buff,ArrayList<Node>){

Node head=new Node();

int n=1;

Node[] nodes=new Node[elements.size()];

try{

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

if(elements.get(i).isToken){

nodes[i]=new Node(null,null,elements.get(i).token.type,elements.get(i).token.index,Node.SPLIT\_CENTRE,lexicalAnalizator);

}

}

}catch (IncorectNodeException e){

e.printStackTrace();

}

if(name.equals("if")){

Node head=new Node(

}

Element newElement=new Element(EXPR+rule.rule.name);

newElement.tree=head;

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

buff.remove(rule.position);

}

buff.add(rule.position,newElement);

}

}

package SyntaxicAnalizer;

import LexicalScaner.LexicalAnalizator;

import LexicalScaner.Token;

import tree.Node;

import java.util.ArrayList;

public class SyntaxAnalizer {

public ArrayList<Element> buff;

public ArrayList<StartRule> activeRules;

public ArrayList<Rule> rules;

public LexicalAnalizator lexicalAnalizator;

public Node tree;

public SyntaxAnalizer(ArrayList<Rule> rules1){

if(rules1==null){

throw new IllegalArgumentException();

}

rules=rules1;

}

public void analize(LexicalAnalizator lexicalAnalizator1){

if(lexicalAnalizator1==null){

throw new IllegalArgumentException();

}

buff=new ArrayList<Element>();

activeRules=new ArrayList<StartRule>();

lexicalAnalizator=lexicalAnalizator1;

for(int i=0;i<lexicalAnalizator.tokens.length+1;i++){

if(i==lexicalAnalizator.tokens.length){

try{

activeRules.add(new StartRule(new Rule("nop",new ArrayList<Element>(),0),lexicalAnalizator.tokens.length));

}catch (Exception e){

e.printStackTrace();

}

}else {

buff.add(new Element(lexicalAnalizator.tokens[i]));

System.out.println("Add token="+lexicalAnalizator.tokens[i].line+" type="+lexicalAnalizator.tokens[i].type+" pos="+(buff.size()-1));

activeRules.addAll(getRuleForExpresstion(new Element(lexicalAnalizator.tokens[i]),buff.size()-1));

}

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

//System.out.println("active rule="+activeRules.get(j).rule.name+" pos="+activeRules.get(j).position);

}

while (checkExitindRules(lexicalAnalizator)){};

}

}

public ArrayList<StartRule> getRuleForExpresstion(Element element,int pos){

ArrayList<StartRule> rulesForCurrentPos=new ArrayList<StartRule>();

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

if(rules.get(i).elements.get(0).equals(element)){

rulesForCurrentPos.add(new StartRule(rules.get(i),pos));

}

}

return rulesForCurrentPos;

}

public boolean checkExitindRules(LexicalAnalizator lexicalAnalizator){

ArrayList<StartRule> rulesByPriority=new ArrayList<StartRule>();

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

boolean isCanStart=activeRules.get(i).rule.isExprForThisRule(buff,activeRules.get(i).position);

System.out.println("iscanstart="+isCanStart+" rulename="+activeRules.get(i).rule.name+" pos="+activeRules.get(i).position);

if(isCanStart){

rulesByPriority.add(activeRules.get(i));

}

}

boolean[] isCorrectOrder=new boolean[rulesByPriority.size()];

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

isCorrectOrder[i]=false;

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

if(activeRules.get(j).position>rulesByPriority.get(i).position && activeRules.get(j).rule.priority<rulesByPriority.get(i).rule.priority){

isCorrectOrder[i]=true;

break;

}

}

}

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

System.out.println("isCorectOrder="+isCorrectOrder[i]);

}

if(rulesByPriority.size()==0){

return false;

}

int n=0;

int max=-1;

int p=-1;

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

if(isCorrectOrder[i]){

if(max<rulesByPriority.get(i).rule.priority){

max=rulesByPriority.get(i).rule.priority;

n++;

p=i;

}

}

}

if(p==-1){

return false;

}

rulesByPriority.get(p).rule.executeRule(lexicalAnalizator,rulesByPriority.get(p),buff);

System.out.println("Start execute rule="+rulesByPriority.get(p).rule.name+" pos="+rulesByPriority.get(p).position);

int pos=rulesByPriority.get(p).position;

Element el=buff.get(pos);

activeRules.addAll(getRuleForExpresstion(el,pos));

activeRules.remove(rulesByPriority.get(p));

return true;

}

}

package SyntaxicAnalizer;

public class Tuple {

public int type;

public int index;

public Tuple(int type1,int index1){

type=type1;

index=index1;

}

}

package UnitTests;

import LexicalScaner.IncorectTokenException;

import LexicalScaner.LexicalAnalizator;

import LexicalScaner.Token;

import SyntaxicAnalizer.Element;

import SyntaxicAnalizer.Rule;

import SyntaxicAnalizer.SyntaxAnalizer;

import com.sun.javaws.exceptions.InvalidArgumentException;

import junit.framework.TestCase;

import junit.framework.TestSuite;

import junit.textui.TestRunner;

import table.Record;

import table.Table;

import tree.Node;

import tree.TablesOperator;

import Automat.\*;

import java.util.ArrayList;

public class TestClass extends TestCase{

public TestClass(String testName) {

super(testName);

}

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.insert(new Record(" ",null));//0

delimiters.insert(new Record(",",null));//1

delimiters.insert(new Record(".",null));//2

delimiters.insert(new Record("(",null));//3

delimiters.insert(new Record(")",null));//4

delimiters.insert(new Record("+",null));//5

delimiters.insert(new Record("-",null));//6

delimiters.insert(new Record("\*",null));//7

delimiters.insert(new Record("/",null));//8

delimiters.insert(new Record("<",null));//9

delimiters.insert(new Record(">",null));//10

delimiters.insert(new Record("=",null));//11

delimiters.insert(new Record(";",null));//12

delimiters.insert(new Record("{",null));//13

delimiters.insert(new Record("}",null));//14

delimiters.insert(new Record("&",null));//17 \

delimiters.insert(new Record("|",null));//18

Table terminalWords=new Table();

terminalWords.insert(new Record("if", null));//0

terminalWords.insert(new Record("while",null));//1

terminalWords.insert(new Record("int",null));//2

terminalWords.insert(new Record("bool",null));//3

terminalWords.insert(new Record("true",null));//4

terminalWords.insert(new Record("false",null));//5

terminalWords.insert(new Record("else",null));//6

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

}catch (Exception e){

e.printStackTrace();

}

return lexicalAnalizator;

}

public SyntaxAnalizer createSyntaxAnalizer(){

ArrayList<Rule> rules=new ArrayList<Rule>();

try{

ArrayList<Element> elements=new ArrayList<Element>();

elements.add(new Element(Rule.EXPR+"expr"));

elements.add(new Element(Rule.EXPR+"unop"));

Rule expr1=new Rule("expr",elements,89);

rules.add(expr1);

elements=new ArrayList<Element>();

elements.add(new Element(Rule.EXPR+"expr"));

elements.add(new Element(Rule.EXPR+"op"));

elements.add(new Element(Rule.EXPR+"expr"));

Rule expr2=new Rule("expr",elements,87);

rules.add(expr2);

elements=new ArrayList<Element>();

elements.add(new Element(new Token("(",Token.DELIMITER,3)));

elements.add(new Element(Rule.EXPR+"expr"));

elements.add(new Element(new Token(")",Token.DELIMITER,4)));

Rule expr3=new Rule("expr",elements,88);

rules.add(expr3);

elements=new ArrayList<Element>();

elements.add(new Element(new Token("",Token.IDENTIFIER,0)));

Rule expr4=new Rule("expr",elements,97);

rules.add(expr4);

elements=new ArrayList<Element>();

elements.add(new Element(new Token("",Token.CONSTANT,0)));

Rule expr5=new Rule("expr",elements,97);

rules.add(expr5);

elements=new ArrayList<Element>();

elements.add(new Element(new Token("+",Token.DELIMITER,5)));

Rule op1=new Rule("op",elements,90);

rules.add(op1);

elements=new ArrayList<Element>();

elements.add(new Element(new Token("-",Token.DELIMITER,6)));

Rule op2=new Rule("op",elements,91);

rules.add(op2);

elements=new ArrayList<Element>();

elements.add(new Element(new Token("\*",Token.DELIMITER,7)));

Rule op3=new Rule("op",elements,92);

rules.add(op3);

elements=new ArrayList<Element>();

elements.add(new Element(new Token("/",Token.DELIMITER,8)));

Rule op4=new Rule("op",elements,97);

rules.add(op4);

elements=new ArrayList<Element>();

elements.add(new Element(new Token("+",Token.DELIMITER,5)));

elements.add(new Element(new Token("+",Token.DELIMITER,5)));

Rule unop1=new Rule("unop",elements,100);

rules.add(unop1);

elements=new ArrayList<Element>();

elements.add(new Element(new Token("-",Token.DELIMITER,6)));

elements.add(new Element(new Token("-",Token.DELIMITER,6)));

Rule unop2=new Rule("unop",elements,99);

rules.add(unop2);

elements=new ArrayList<Element>();

elements.add(new Element(new Token(">",Token.DELIMITER,10)));

Rule boolOp1=new Rule("boolop",elements,94);

rules.add(boolOp1);

elements=new ArrayList<Element>();

elements.add(new Element(new Token("<",Token.DELIMITER,9)));

Rule boolOp2=new Rule("boolop",elements,95);

rules.add(boolOp2);

elements=new ArrayList<Element>();

elements.add(new Element(new Token("=",Token.DELIMITER,11)));

elements.add(new Element(new Token("=",Token.DELIMITER,11)));

Rule boolOp3=new Rule("boolop",elements,96);

rules.add(boolOp3);

elements=new ArrayList<Element>();

elements.add(new Element(new Token("&",Token.DELIMITER,17)));

elements.add(new Element(new Token("&",Token.DELIMITER,17)));

Rule logicOp1=new Rule("logicop",elements,83);

rules.add(logicOp1);

elements=new ArrayList<Element>();

elements.add(new Element(new Token("|",Token.DELIMITER,18)));

elements.add(new Element(new Token("|",Token.DELIMITER,18)));

Rule logicOp2=new Rule("logicop",elements,82);

rules.add(logicOp2);

elements=new ArrayList<Element>();

elements.add(new Element(Rule.EXPR+"expr"));

elements.add(new Element(new Token("=",Token.DELIMITER,11)));

elements.add(new Element(Rule.EXPR+"expr"));

elements.add(new Element(new Token("",Token.DELIMITER,12)));

Rule assignment=new Rule("assignment",elements,86);

rules.add(assignment);

elements=new ArrayList<Element>();

elements.add(new Element(Rule.EXPR+"expr"));

elements.add(new Element(Rule.EXPR+"boolop"));

elements.add(new Element(Rule.EXPR+"expr"));

Rule boolExpr1=new Rule("boolexpr",elements,81);

rules.add(boolExpr1);

elements=new ArrayList<Element>();

elements.add(new Element(Rule.EXPR+"boolexpr"));

elements.add(new Element(Rule.EXPR+"logicop"));

elements.add(new Element(Rule.EXPR+"boolexpr"));

Rule boolExpr2=new Rule("boolexpr",elements,80);

rules.add(boolExpr2);

elements=new ArrayList<Element>();

elements.add(new Element(new Token("",Token.TERMINALWORD,4)));

Rule boolExpr3=new Rule("boolexpr",elements,85);

rules.add(boolExpr3);

elements=new ArrayList<Element>();

elements.add(new Element(new Token("",Token.TERMINALWORD,5)));

Rule boolExpr4=new Rule("boolexpr",elements,84);

rules.add(boolExpr4);

elements=new ArrayList<Element>();

elements.add(new Element(Rule.EXPR+"operator"));

Rule operatorList1=new Rule("operatorlist",elements,75);

rules.add(operatorList1);

elements=new ArrayList<Element>();

elements.add(new Element(Rule.EXPR+"operator"));

elements.add(new Element(Rule.EXPR+"operatorlist"));

Rule operatorLis2=new Rule("operatorlist",elements,74);

rules.add(operatorLis2);

elements=new ArrayList<Element>();

elements.add(new Element(new Token("if",Token.TERMINALWORD,0)));

elements.add(new Element(new Token("(",Token.DELIMITER,3)));

elements.add(new Element(Rule.EXPR+"boolexpr"));

elements.add(new Element(new Token(")",Token.DELIMITER,4)));

elements.add(new Element(Rule.EXPR+"operator"));

elements.add(new Element(new Token(";",Token.DELIMITER,12)));

Rule operator1=new Rule("operator",elements,78);

rules.add(operator1);

elements=new ArrayList<Element>();

elements.add(new Element(new Token("if",Token.TERMINALWORD,0)));

elements.add(new Element(new Token("(",Token.DELIMITER,3)));

elements.add(new Element(Rule.EXPR+"boolexpr"));

elements.add(new Element(new Token(")",Token.DELIMITER,4)));

elements.add(new Element(Rule.EXPR+"operator"));

elements.add(new Element(new Token("else",Token.TERMINALWORD,6)));

elements.add(new Element(Rule.EXPR+"operator"));

elements.add(new Element(new Token(";",Token.DELIMITER,12)));

Rule operator2=new Rule("operator",elements,79);

rules.add(operator2);

elements=new ArrayList<Element>();

elements.add(new Element(new Token("while",Token.TERMINALWORD,1)));

elements.add(new Element(new Token("(",Token.DELIMITER,3)));

elements.add(new Element(Rule.EXPR+"boolexpr"));

elements.add(new Element(new Token(")",Token.DELIMITER,4)));

elements.add(new Element(Rule.EXPR+"operator"));

elements.add(new Element(new Token(";",Token.DELIMITER,12)));

Rule operator3=new Rule("operator",elements,77);

rules.add(operator3);

elements=new ArrayList<Element>();

elements.add(new Element(Rule.EXPR+"assignment"));

Rule operator4=new Rule("operator",elements,76);

rules.add(operator4);

elements=new ArrayList<Element>();

elements.add(new Element(new Token("{",Token.DELIMITER,13)));

elements.add(new Element(Rule.EXPR+"operatorlist"));

elements.add(new Element(new Token("}",Token.DELIMITER,14)));

Rule operator5=new Rule("operator",elements,73);

rules.add(operator5);

elements=new ArrayList<Element>();

elements.add(new Element(Rule.EXPR+"expr"));

elements.add(new Element(new Token(";",Token.DELIMITER,12)));

Rule operator6=new Rule("operator",elements,73);

rules.add(operator6);

SyntaxAnalizer syntaxAnalizer=new SyntaxAnalizer(rules);

return syntaxAnalizer;

}catch (InvalidArgumentException ex){

return null;

}

catch (IncorectTokenException e){

return null;

}

}

public void checkSyntaxAnalizer(){

SyntaxAnalizer analizer=this.createSyntaxAnalizer();

LexicalAnalizator lexicalAnalizator=this.createLexicalAnalizator();

try{

lexicalAnalizator.analize("c=(d+5)\*a");

}catch (Exception ex){

System.out.println("Lexical analizer exception");

}

analizer.analize(lexicalAnalizator);

}

public static void main(String[] args) {

TestRunner runner = new TestRunner();

TestSuite suite = new TestSuite();

suite.addTest(new TestClass("checkSyntaxAnalizer"));

runner.doRun(suite);

}

}