

Міністерство освіти і науки, молоді та спорту України

Національний технічний університет України

«Київський Політехнічний Інститут»

Факультет прикладної математики

Кафедра спеціалізованих комп’ютерних систем та системного програмування

**Лабораторна робота № 1**

**З дисципліни «Інженерія програмного забезпечення. Основи проектування трансляторів»**

**На тему: «Розробка лексичного аналізатора»**

**Виконав:**

**Студент групи КВ-31**

**Черниш Андрій**

**Номер у списку групи : 23**

**Київ 2015**

**Постановка завдання**

Розробити програму лексичного аналізатора (ЛА) для підмножини мови програмування SIGNAL.

Програма має забезпечувати наступне (якщо це передбачається граматикою варіанту):

* згортання ідентифікаторів;
* згортання ключових слів;
* згортання цілих десяткових констант;
* згортання дійсних десяткових констант;
* згортання строкових констант, якщо вони визначені в заданій мові;

Також у всіх варіантах необхідно забезпечити:

* видалення коментарів, заданих у вигляді (\*<текст коментарю>\*)

Для кодування лексем необхідно використовувати числові діапазони, вказані в Таблиці 1.

***Діапазони кодування лексем***

|  |  |
| --- | --- |
| **Вид лексеми** | **Числовий діапазон** |
| односимвольні роздільники та знаки операцій (: / ; + і т.д.) | 0 – 255,  Тобто коди ASCII |
| багатосимвольні роздільники ( := ,<= ,<=, і т.д.) | 301 - 400 |
| цілі десяткові константи | 401 - 500 |
| символьні константи | 501 - 600 |
| рядкові константи | 601 - 700 |
| ключові слова (BEGIN, END, FOR) та ідентифікатори | 701 – 1000 |

**Входом** ЛА має бути наступне:

вихідна програма, написана підмножиною мови SIGNAL відповідно до варіанту;

таблиця кодів ASCII з атрибутами для визначення токенів;

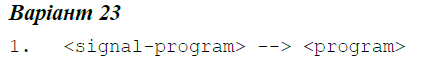
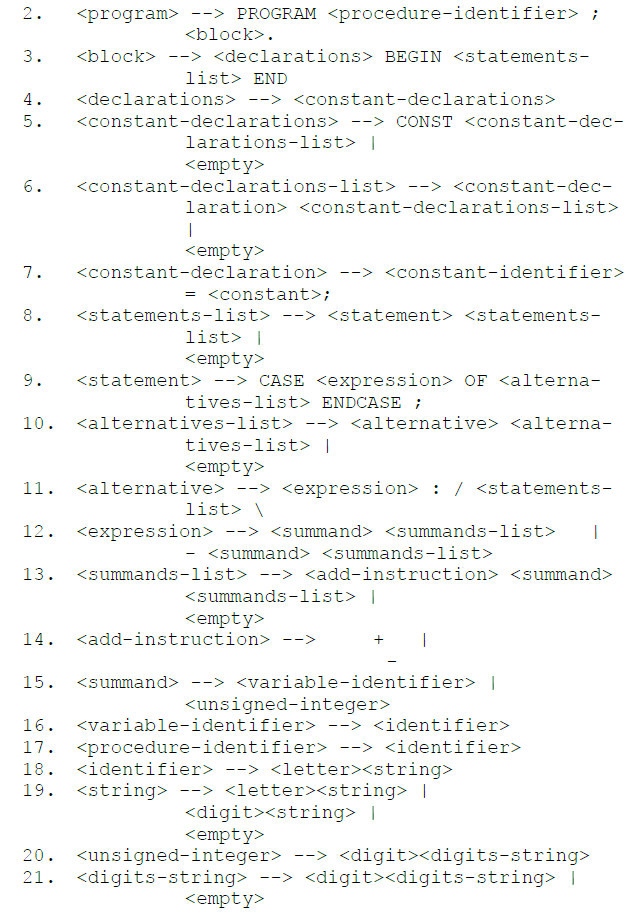
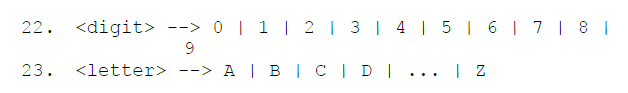
таблиця багато символьних роздільників;

таблиця ідентифікаторів, в яку попередньо занесені ключові слова з атрибутом ключового слова;

**Вихід** ЛА має бути наступним:

закодований рядок лексем;

таблиці ідентифікаторів, числових, символьних та рядкових констант, сформовані для конкретної програми;



**Граф автомату, що задає алгоритм ЛА**

**ws**

**other**

**eof**

**eof**

**other**

**)**

**other**

**\***

**other**

**\***

**(**

**other**

**lt**

**ws**

**other**

**lt, dg**

**dm(not “(”)**

**\***

**eof**

Код програми

**Analyzer.java**

package com.itcherry.translators;

import java.io.BufferedInputStream;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

/\*\*

\*

\* @author Chernysh Andrii, NTUU "KPI", FAM, KV-31

\* date : 03-Mar-2016

\* Class analyzer is first part of translator language signal.

\* This class is responsible for lexical analyze some code.

\* Result of analyzing we put into ArrayList<Integer> result.

\* Method getInfo starts graphical interface to show all tables and result

\*

\*/

public class Analyzer extends StandartTables {

private ArrayList<Integer> result;

private FileInputStream fileInputStream;

private BufferedInputStream reader;

private boolean eof;

private char ch;

private StringBuffer buf;

private int lexCode;

private boolean supressOutput;

/\*\*

\*

\* @param fileName - name of file

\*

\* Method responsible for initializing some file from storage

\*

\*/

private void initFile(String fileName) {

File file = new File("src/res/" + fileName);

try {

fileInputStream = new FileInputStream(file);

} catch (IOException x) {

System.err.format("IOException: %s%n", x);

}

}

/\*\*

\*

\* @return 1 symbol from file

\*/

private char getChar() {

char current = (char) 0;

try {

current = (char) fileInputStream.read();

} catch (IOException e) {

e.printStackTrace();

}

return current;

}

/\*\*

\* Closing the opened file

\*/

private void closeFile() {

try {

fileInputStream.close();

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* Identifying keyword or identificator

\*/

private void processLetter() {

while (!eof

&& (attributes[(int) ch] == StandartTables.DIGIT || attributes[(int) ch] == StandartTables.LETTER)) {

buf.append(ch);

ch = getChar();

}

if (keywords.containsKey(buf.toString())) {

lexCode = keywords.get(buf.toString());

} else if (identificators.containsKey(buf.toString())) {

lexCode = getIdentificator(buf.toString());

} else {

putIdentificator(buf.toString());

lexCode = getIdentificator(buf.toString());

}

}

/\*\*

\* Identifying constants

\*/

private void processDigit() {

while (!eof && (attributes[(int) ch] == StandartTables.DIGIT)) {

buf.append(ch);

ch = getChar();

}

if (constants.containsKey(buf.toString())) {

lexCode = constants.get(buf.toString());

} else {

putConst(buf.toString());

lexCode = constants.get(buf.toString());

}

}

/\*\*

\* Identifying whitespaces

\*/

private void processWhitespaces() {

while (!eof && (attributes[(int) ch] == StandartTables.WHITESPACE)) {

ch = getChar();

}

supressOutput = true;

}

/\*\*

\*

\* @throws IOException

\* Handle with commentaries

\*/

private void processComment() throws IOException {

eof = fileInputStream.available() <= 0;

if (eof)

lexCode = (int) '(';

else {

ch = getChar();

if (ch == '\*') {

if (eof) {

lexCode = -99; // Error : \*) expected but end of file found

} else {

ch = getChar();

do {

while (fileInputStream.available() > 0 && ch != '\*') {

ch = getChar();

}

if (fileInputStream.available() <= 0) {

lexCode = -99; // Error : \*) expected but end of

// file found

return;

} else

ch = getChar();

} while (ch != ')');

}

} else

lexCode = (int) '(';

}

ch = getChar();

supressOutput = true;

}

/\*\*

\* Identifying delimiters

\*/

private void processDelimiter() {

lexCode = (int) ch;

ch = getChar();

}

/\*\*

\* Identifying forbidden symbols

\*/

private void processForbidden() {

lexCode = -100; // Error : forbidden symbol in line

ch = getChar();

}

/\*\*

\*

\* @param fileName - name of file

\* @throws IOException

\*

\* Analyzing some file and get result array

\*/

public void analyse(String fileName) throws IOException {

initFile(fileName);

result = new ArrayList<>();

eof = fileInputStream.available() <= 0;

if (eof) {

System.out.println("Empty file!!");

return;

}

buf = new StringBuffer();

lexCode = 0;

ch = getChar();

boolean lastSymbol = false;

do {

supressOutput = false;

switch (attributes[(int) ch]) {

case StandartTables.DIGIT:

processDigit();

break;

case StandartTables.COMMENT\_START:

processComment();

break;

case StandartTables.FORBIDDEN:

processForbidden();

break;

case StandartTables.LETTER:

processLetter();

break;

case StandartTables.SINGLE\_DELIMITER:

processDelimiter();

break;

case StandartTables.WHITESPACE:

processWhitespaces();

break;

default:

break;

}

buf.delete(0, buf.length());

if (!supressOutput)

result.add(lexCode);

eof = fileInputStream.available() <= 0;

} while (((int) ch != 65535));

closeFile();

}

/\*\*

\*

\* @throws IOException

\* Make graphical user interface and put tables and result there

\*/

public void getInfo() throws IOException {

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

/\*for (Integer i : result) {

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

}\*/

GUITables frame = new GUITables();

frame.initTable(keywords, identificators, errors, constants, result);

frame.drawFrame();

}

}

**StandartTables.java**

package com.itcherry.translators;

import java.util.Arrays;

import java.util.HashMap;

public abstract class StandartTables {

protected HashMap<String, Integer> keywords; //Table of keywords

protected HashMap<String, Integer> identificators; //Table of identificators

protected HashMap<Integer, String> errors; //Table of errors

protected HashMap<String, Integer> constants; //Table of constants

public StandartTables() {

keywords= new HashMap<>();

identificators= new HashMap<>();

errors = new HashMap<>();

constants = new HashMap<>();

fillAttribute();

fillKeywords();

fillErrors();

}

/\* Initializing hashmap of keywords \*/

private void fillKeywords() {

keywords.put("PROGRAM", 701);

keywords.put("CONST", 702);

keywords.put("CASE", 703);

keywords.put("OF", 704);

keywords.put("ENDCASE", 705);

keywords.put("BEGIN", 706);

keywords.put("END", 707);

}

/\* End keywords initializing \*/

/\* Initializing hashmap of errors \*/

private void fillErrors() {

errors.put(-100, "Error : forbidden symbol in line");

errors.put(-99, "Error : \*) expected but end of file found");

}

/\* End errors initializing \*/

/\* Initializing hashmap of identificators \*/

private static final int IDENTIFICATOR\_MIN = 708;

private static final int IDENTIFICATOR\_MAX = 1000;

protected int identificatorIterator = IDENTIFICATOR\_MIN;

protected void putIdentificator(String ident) {

if (identificatorIterator <= IDENTIFICATOR\_MAX) {

identificators.put(ident, identificatorIterator);

identificatorIterator++;

}else {

System.err.println("Too much identificators");

}

}

protected int getIdentificator(String ident) {

return identificators.get(ident);

}

/\* End identificators initializing \*/

/\* Initializing hashmap of constants \*/

private static final int CONST\_MIN = 401;

private static final int CONST\_MAX = 500;

public int constIterator = CONST\_MIN;

protected void putConst(String constant) {

if (constIterator <= CONST\_MAX) {

constants.put(constant, constIterator);

constIterator++;

}else {

System.err.println("Too much constants");

}

}

protected int getConst(String constant) {

return identificators.get(constant);

}

/\* End constants initializing \*/

/\* Initializing array of attributes \*/

public static final int WHITESPACE = 1;

public static final int SINGLE\_DELIMITER = 2;

public static final int DIGIT = 4;

public static final int LETTER = 5;

public static final int COMMENT\_START = 6;

public static final int FORBIDDEN = 7;

private static final int ATTRIBUTE\_MIN = 0;

private static final int ATTRIBUTE\_MAX = 255;

private static final int ATTRIBUTES\_NUMBER = 256;

public int[] attributes;

private void fillAttribute() {

attributes = new int[ATTRIBUTES\_NUMBER];

Arrays.fill(attributes, FORBIDDEN); // forbidden symbols

attributes[10] = WHITESPACE; //whitespaces

attributes[9] = WHITESPACE;

attributes[11] = WHITESPACE;

attributes[13] = WHITESPACE;

attributes[32] = WHITESPACE; // space symbol

attributes[61] = SINGLE\_DELIMITER; // '='

attributes[58] = SINGLE\_DELIMITER; // ':'

attributes[59] = SINGLE\_DELIMITER; // ';'

attributes[41] = SINGLE\_DELIMITER; // ')'

attributes[43] = SINGLE\_DELIMITER; // '+'

attributes[45] = SINGLE\_DELIMITER; // '-'

attributes[47] = SINGLE\_DELIMITER; // '/'

attributes[92] = SINGLE\_DELIMITER; // '\'

attributes[46] = SINGLE\_DELIMITER; // '.'

attributes[40] = COMMENT\_START; // '('

Arrays.fill(attributes, 48, 58, DIGIT); // '0', ..., '9'

Arrays.fill(attributes, 65, 91, LETTER); // 'A', ...,'Z'

}

public int getAttribute(int ch) {

int result = FORBIDDEN;

if (ch >= ATTRIBUTE\_MIN && ch <= ATTRIBUTE\_MAX) {

result = attributes[ch];

}

return result;

}

/\* End attributes initializing \*/

}

**GUITables.java**

package com.itcherry.translators;

import java.awt.Font;

import java.awt.SystemColor;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.HashMap;

import java.util.Map;

import javax.swing.JFrame;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.JTextArea;

import javax.swing.JTextField;

public class GUITables extends JFrame {

private JTable identificators;

private JTable constants;

private JTable keywords;

private JTable errors;

private JTextField textField;

private String[][] keywordsFields;

private String[][] identificatorsFields;

private String[][] errorsFields;

private String[][] constantsFields;

private String[] columnNames = {

"Value",

"Code",

};

private ArrayList<Integer> result;

public void initTable(HashMap<String, Integer> keywords, HashMap<String, Integer> identificators,

HashMap<Integer, String> errors, HashMap<String, Integer> constants, ArrayList<Integer> result) {

this.result = result;

keywordsFields = new String[keywords.size()][keywords.size()];

int i = 0;

for (Map.Entry<String, Integer> pair : keywords.entrySet()){

keywordsFields[i][0] = pair.getKey();

keywordsFields[i][1] = pair.getValue().toString();

i++;

}

identificatorsFields = new String[identificators.size()][identificators.size()];

i = 0;

for (Map.Entry<String, Integer> pair : identificators.entrySet()){

identificatorsFields[i][0] = pair.getKey();

identificatorsFields[i][1] = pair.getValue().toString();

i++;

}

errorsFields = new String[errors.size()][errors.size()];

i = 0;

for (Map.Entry<Integer, String> pair : errors.entrySet()){

errorsFields[i][0] = pair.getKey().toString();

errorsFields[i][1] = pair.getValue();

i++;

}

constantsFields = new String[constants.size()][constants.size()];

i = 0;

for (Map.Entry<String, Integer> pair : constants.entrySet()){

constantsFields[i][0] = pair.getKey();

constantsFields[i][1] = pair.getValue().toString();

i++;

}

}

GUITables() {

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

this.setTitle("Syntax analyser");

this.setSize(780, 500);

this.setResizable(false);

this.setLocationRelativeTo(null);

}

public void drawFrame(){

this.setLayout(null);

identificators = new JTable(identificatorsFields,columnNames);

JScrollPane scrollpane1 = new JScrollPane(identificators);

scrollpane1.setBackground(SystemColor.controlHighlight);

scrollpane1.setBounds(12, 13, 175, 292);

this.add(scrollpane1);

constants = new JTable(constantsFields,columnNames);

JScrollPane scrollpane2 = new JScrollPane(constants);

scrollpane2.setBackground(SystemColor.controlHighlight);

scrollpane2.setBounds(199, 13, 178, 292);

this.add(scrollpane2);

keywords = new JTable(keywordsFields,columnNames);

JScrollPane scrollpane3 = new JScrollPane(keywords);

scrollpane3.setBackground(SystemColor.controlHighlight);

scrollpane3.setBounds(389, 13, 173, 292);

this.add(scrollpane3);

errors = new JTable(errorsFields,columnNames);

JScrollPane scrollpane4 = new JScrollPane(errors);

scrollpane4.setBackground(SystemColor.controlHighlight);

scrollpane4.setBounds(574, 13, 173, 292);

this.add(scrollpane4);

JTextArea text = new JTextArea();

JScrollPane scrollpane5 = new JScrollPane(text);

text.setEnabled(true);

scrollpane5.setBounds(12, 318, 735, 102);

text.setText(Arrays.toString(result.toArray()));

text.setVisible(true);

text.setLineWrap(true);

text.setWrapStyleWord(true);

text.setFont(new Font("Times New Roman", Font.PLAIN, 20));

this.add(scrollpane5);

this.setVisible(true);

}

}

**Main.java**

import java.io.File;

import java.io.IOException;

public class Main {

public static void main(String[] args) throws IOException {

Analyzer analyser = new Analyzer();

analyser.analyse("test2.dat");

analyser.getInfo();

}

}

**Контрольні приклади**

**Приклад 1 :**

PROGRAM P1;

CONST

I = 1;

J = 2;

K = 3;

BEGIN

CASE I OF

3-4+J : /

\

k : /

\

J - K + 3 : /

CASE J OF

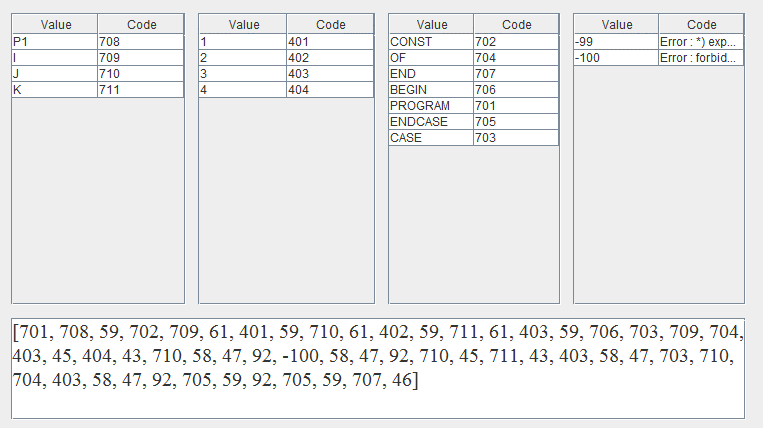
3 : /\

ENDCASE;

\

ENDCASE;

END.



**Приклад 2 :**

PROGRAM ProG1;

CONST (\*Some Commentary\*)

I = 1;

J = 2;

K = 3;

BEGIN

CASE I OF

3-4+J : / \

k : /

\

J-K + 3 : /

CASE J OF

3 : /\

ENDCASE;

\

ENDCASE;

END.

