

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 № 2**

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

**На тему: «Розробка синтаксичного аналізатора(парсера)»**

**Виконав:**

**Студент групи КВ-31**

**Черниш Андрій**

**Номер у списку групи : 23**

**Київ 2016**

**Постановка завдання**

Розробити програму синтаксичного аналізатора (СА) для підмножини мови програмування SIGNAL.

Програма має забезпечувати наступне:

* читання рядку лексем та таблиць, згенерованих лексичним аналізатором, який було розроблено в лабораторній роботі №1;
* синтаксичний аналіз (розбір) програми, поданої рядком лексем (алгоритм розбору вибирається згідно з варіантом);
* побудову дерева розбору;
* формування таблиць ідентифікаторів та різних констант з повною інформацією, необхідною для генерування коду;
* формування лістингу вхідної програми з повідомленнями про лексичні та синтаксичні помилки.

**Входом** СА має бути наступне:

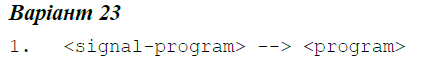
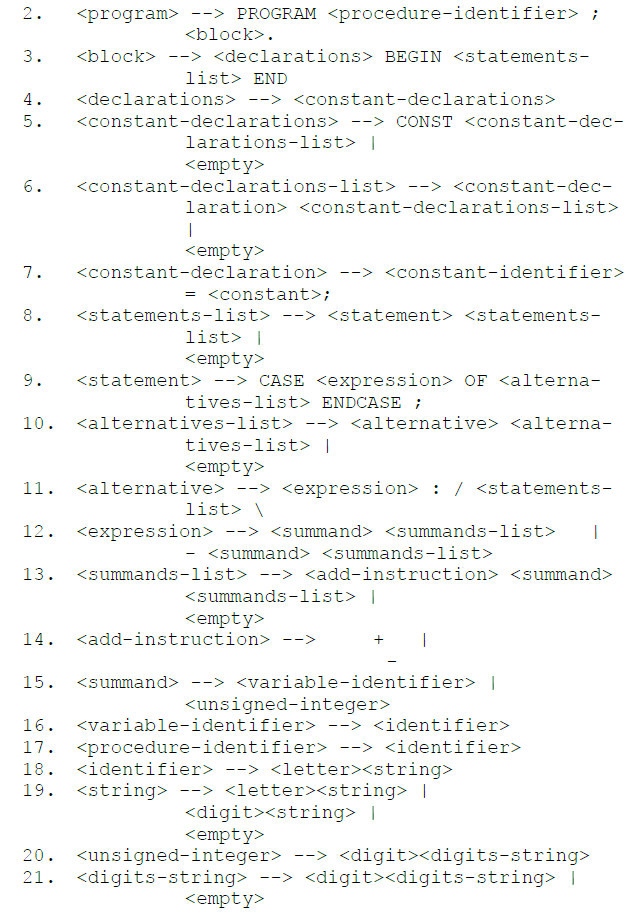
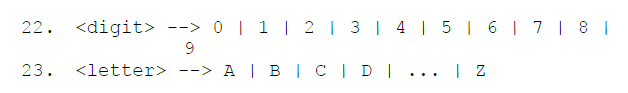
* закодований рядок лексем;
* таблиці ідентифікаторів, числових, символьних та рядкових констант (якщо це передбачено граматикою варіанту), згенеровані лексичним аналізатором;
* вхідна програма на підмножині мови програмування SIGNAL згідно з варіантом (необхідна для формування лістнигу програми).

**Виходом** СА має бути наступне:

* дерево розбору вхідної програми;
* таблиці ідентифікаторів та різних констант з повною інформацією, необхідною для генерування коду;
* лістинг вхідної програми з повідомленнями про лексичні та синтаксичні помилки.

***Метод синтаксичного розбору: аналізуюча машина Кнута:***

Задана граматика відповідає умовам даного методу ( не містить лівобічної рекурсії і відсутні правила з однаковою лівою частиною ), тому граматика залишається без змін.



**Таблиця, що задає алгоритм програми АМК**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Адреса операції** | **Код операції** | **АТ** | **AF** |
| **0** | <signal-program> | <program> | 1 | -1 |
| **1** | <program> | PROGRAM | 2 | -1 |
| **2** |  | <Procedure-identifier> | 52 | -1 |
| **3** |  | ; | 4 | -1 |
| **4** |  | <block> | 6 | -1 |
| **5** |  | . | -2 | -1 |
| **6** | <block> | <declarations> | 10 | -1 |
| **7** |  | BEGIN | 8 | -1 |
| **8** |  | <statements-list> | 21 | -1 |
| **9** |  | END | -2 | -1 |
| **10** | <declarations> | <constant-declarations>| | 11 | -1 |
| **11** | <constant-declarations> | CONST | 12 | 13 |
| **12** |  | <constant-declarations-list> | 14 | -1 |
| **13** |  | <empty> | -2 | -2 |
| **14** | <constant-declarations-list> | <constant-declaration> | 17 | 16 |
| **15** |  | <constant-declarations-list>| | 14 | -1 |
| **16** |  | <empty> | -2 | -2 |
| **17** | <constant-declaration> | <constant-identifier> | 53 | -10 |
| **18** |  | = | 19 | -1 |
| **19** |  | <constant> | 54 | -1 |
| **20** |  | ; | -2 | -1 |
| **21** | <statements-list> | <statement> | 24 | 23 |
| **22** |  | <statements-list>| | 21 | -1 |
| **23** |  | <empty> | -2 | -2 |
| **24** | <statement> | CASE | 25 | -1 |
| **25** |  | <expression> | 38 | -1 |
| **26** |  | OF | 27 | -1 |
| **27** |  | <alternatives-list> | 30 | -1 |
| **28** |  | ENDCASE | 29 | -1 |
| **29** |  | ; | -2 | -1 |
| **30** | <alternatives-list> | <alternative> | 33 | 32 |
| **31** |  | <alternatives-list>| | 30 | -1 |
| **32** |  | <empty> | -2 | -2 |
| **33** | <alternative> | <expression> | 38 | -10 |
| **34** |  | : | 35 | -1 |
| **35** |  | / | 36 | -1 |
| **36** |  | <statements-list> | 21 | -1 |
| **37** |  | \ | -2 | -1 |
| **38** | <expression> | <summand> | 49 | 40 |
| **39** |  | <summands-list>| | 43 | -1 |
| **40** |  | - | 41 | -10 |
| **41** |  | <summand> | 49 | -1 |
| **42** |  | <summands-list>| | 43 | -1 |
| **43** | <summands-list> | <add-instruction> | 47 | 46 |
| **44** |  | <summand> | 49 | -1 |
| **45** |  | <summands-list>| | 43 | -1 |
| **46** |  | <empty> | -2 | -2 |
| **47** | <add-instruction> | + | -2 | 48 |
| **48** |  | - | -2 | -10 |
| **49** | <summand> | <variable-identifier> | 51 | 50 |
| **50** |  | <unsigned-integer>| | 55 | 55 |
| **51** | <variable-identifier> | Search in identifiers table | -2 | -1 |
| **52** | <procedure-identifier> | Search in identifiers table | -2 | -1 |
| **53** | <constant-identifier> | Search in identifiers table | -2 | -1 |
| **54** | <constant> | Search in constants table | -2 | -1 |
| **55** | <unsigned-integer> | Search in constants table | -2 | -1 |

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

\*

\*/

*/\*\*  
 \* Scanning hashmap. Find key by value. Its needed by tree  
 \*  
 \** ***@param value*** *- value for hashmap, where we will find key(Strig for tree)  
 \** ***@return*** *String with key  
 \*/***private** String getKeyToValue(**int** value) {  
 **if** (value > 39 && value < 100) {  
 **return** String.*valueOf*((**char**) value);  
 } **else if** (value > 400 && value <= 500) {  
 **for** (Map.Entry<String, Integer> entry : **constants**.entrySet()) {  
 **if** (entry.getValue().equals(value)) **return** entry.getKey();  
 }  
 } **else if** (value > 700 && value <= 707) {  
 **for** (Map.Entry<String, Integer> entry : **keywords**.entrySet()) {  
 **if** (entry.getValue().equals(value)) **return** entry.getKey();  
 }  
 } **else if** (value > 707 && value <= 10000) {  
 **for** (Map.Entry<String, Integer> entry : **identificators**.entrySet()) {  
 **if** (entry.getValue().equals(value)) **return** entry.getKey();  
 }  
 }  
 **return "Error"**;  
}  
  
*/\*\*  
 \* Find node, which has no such name, as previous node have  
 \*  
 \** ***@param node*** *: its a start node, from which we will go up  
 \** ***@return*** *highest node with different parent's name  
 \*/***private** DefaultMutableTreeNode findNode(DefaultMutableTreeNode node) {  
 DefaultMutableTreeNode parentNode = (DefaultMutableTreeNode) node.getParent();  
 **while** (parentNode.getUserObject().equals(node.getUserObject())) {  
 parentNode = (DefaultMutableTreeNode) parentNode.getParent();  
 node = (DefaultMutableTreeNode) node.getParent();  
 }  
 **return** node;  
}  
  
*/\*\*  
 \* Knuth parser(modified by Marchenko O.I)  
 \* It get identifiers, keywords, constants and errors tables. Also it get lexical analyser result  
 \* It parse input code and verify it. Also it builds a tree.  
 \*  
 \** ***@return*** *model of tree, which GUI will build  
 \*/***private** DefaultTreeModel syntax() {  
 **long** time = System.*nanoTime*();  
 **boolean** flag = **true**; *// AT or AF* **boolean** identifiersFlag = **false**; *// Before BEGIN and after* Stack<Integer> stack = **new** Stack<>(); *//Stack, which contains position of iterator in the Knuth table* Stack<DefaultMutableTreeNode> nodeStack = **new** Stack<>(); *//Stack, which contains useful tree nodes* **int** i = 0; *//iterator of Knuth table* DefaultMutableTreeNode treeNode = **new** DefaultMutableTreeNode(); *//Node of the tree* **for** (Lexeme lex : **result**) { *//Main cycle, goes for lexeme vector* **try** {  
 **if** (lex.getLexCode() < 0) *//Lexical errors* **return new** Error(**errors**.get(lex.getLexCode()), lex.getRow(), lex.getCol(), treeNode.getRoot());  
 **if** (lex.getLexCode() == 706) identifiersFlag = **true**; *//After BEGIN* **do** {  
 **while** (**syntaxTable**.get(i).getOperationCode().startsWith(**"<"**)) { *//Non-Terminals* **if** (!**syntaxTable**.get(i).getOperationCode().equals(**"<empty>"**)) {  
 **if** (!**syntaxTable**.get(i).getOperationCode().endsWith(**"|"**) && flag)  
 stack.push(i);  
 **if** (flag) {  
 **if** (**syntaxTable**.get(i).getOperationAddress() != **null**) { *// Make new node* treeNode.add(**new** DefaultMutableTreeNode(  
 **syntaxTable**.get(i).getOperationAddress()));  
 treeNode = (DefaultMutableTreeNode) treeNode.getLastChild();  
 nodeStack.push(treeNode);  
 }  
 i = **syntaxTable**.get(i).getAt(); *//Goes to AT* } **else** {  
 i = **syntaxTable**.get(i).getAf(); *//Goes to AF* }  
 **if** (i == -10) {  
 treeNode = (DefaultMutableTreeNode) treeNode.getParent();  
 i = stack.pop();  
 }  
 } **else** { *//Empty grammar* i = stack.pop() + 1;  
 treeNode.removeAllChildren();  
 **do** {  
 treeNode = nodeStack.pop(); *//Return to upper node* } **while** (!treeNode.getUserObject().equals(**syntaxTable**.get(i - 1).getOperationCode()));  
 treeNode = findNode(treeNode);  
 treeNode = (DefaultMutableTreeNode) treeNode.getParent();  
 }  
 }  
 **if** (**syntaxTable**.get(i).getOperationAddress() != **null**) { *//Non-Terminals. Its for tree* treeNode.add(**new** DefaultMutableTreeNode(  
 **syntaxTable**.get(i).getOperationAddress()));  
 treeNode = (DefaultMutableTreeNode) treeNode.getLastChild();  
 }  
 **if** (**syntaxTable**.get(i).getOperationCode().equals(**"SI"**)) { *//Search identifiers (51 - 53 in the Knuth table)* **if** (!identifiersFlag)  
 **if** (!**identificators**.containsValue(lex.getLexCode()) &&  
 !**identificators**.containsValue(lex.getLexCode() << 2))  
 **return new** Error(**"Identifier expected"**, lex.getRow(), lex.getCol(), treeNode.getRoot());  
 **else if** (**identificators**.replace(getKeyToValue(lex.getLexCode()), lex.getLexCode() << 2) == **null**) {  
 **return new** Error(**"Override variable "** + getKeyToValue(lex.getLexCode() << 2),  
 lex.getRow(), lex.getCol(), treeNode.getRoot()); *// Checking for variable rewriting* }  
 i = stack.pop() + 1;  
 **if** (**identificators**.containsValue(lex.getLexCode() << 2)) { *//If it is identifier* flag = **true**;  
 treeNode.add(**new** DefaultMutableTreeNode(getKeyToValue(lex.getLexCode() << 2)));  
 **if** (**syntaxTable**.get(i).getOperationCode().endsWith(**"|"**)) {  
 i = stack.pop() + 1; *// for 49-50* treeNode = (DefaultMutableTreeNode) treeNode.getParent();  
 }  
 treeNode = (DefaultMutableTreeNode) treeNode.getParent();  
 } **else** { *//If not identifier* **if** (**identificators**.containsValue(lex.getLexCode()))  
 **return new** Error(**"Identifier not defined"**, lex.getRow(), lex.getCol(), treeNode.getRoot());  
 **if** (!**syntaxTable**.get(i).getOperationCode().startsWith(**"<"**)) i -= 1;  
 DefaultMutableTreeNode temp = treeNode;  
 treeNode = (DefaultMutableTreeNode) treeNode.getParent();  
 temp.removeFromParent();  
 flag = **false**;  
 }  
 } **else if** (**syntaxTable**.get(i).getOperationCode().equals(**"SC"**)) { *//Search constants (54 - 55 in the Knuth table)* i = stack.pop() + 1;  
 **if** (**constants**.containsValue(lex.getLexCode())) {  
 flag = **true**;  
 treeNode.add(**new** DefaultMutableTreeNode(getKeyToValue(lex.getLexCode())));  
 treeNode = (DefaultMutableTreeNode) treeNode.getParent();  
 **if** (**syntaxTable**.get(i).getOperationCode().endsWith(**"|"**))  
 treeNode = (DefaultMutableTreeNode) treeNode.getParent();  
 } **else** {  
 **if** (**syntaxTable**.get(i).getOperationCode().endsWith(**"|"**)) i -= 1;*// for 49-50* treeNode = (DefaultMutableTreeNode) treeNode.getParent();  
 DefaultMutableTreeNode temp = treeNode;  
 treeNode = (DefaultMutableTreeNode) treeNode.getParent();  
 temp.removeFromParent();  
 flag = **false**;  
 }  
 } **else if** (lex.getLexCode() == Integer.*parseInt*(**syntaxTable**.get(i).getOperationCode())) { *//Terminals processing* treeNode.add(**new** DefaultMutableTreeNode(getKeyToValue(lex.getLexCode())));  
 **if** (**syntaxTable**.get(i).getAt() == -2) {  
 i = stack.pop() + 1;  
 treeNode = (DefaultMutableTreeNode) treeNode.getParent();  
 } **else** i = **syntaxTable**.get(i).getAt();  
 flag = **true**;  
 } **else if** (**syntaxTable**.get(i).getAf() == -10) { *//Return to upper rules* flag = **false**;  
 **if** (i == 40) {  
 treeNode = (DefaultMutableTreeNode) treeNode.getParent();  
 i = stack.pop();  
 }  
 treeNode = (DefaultMutableTreeNode) treeNode.getParent();  
 i = stack.pop();  
 } **else if** (**syntaxTable**.get(i).getAf() == -1) { *//Error processing* **return new** Error(getKeyToValue(Integer.*parseInt*(**syntaxTable**.get(i).getOperationCode())) + **" expected!!!"**,  
 lex.getRow(), lex.getCol(), treeNode.getRoot());  
 } **else** { *//Go to AF* i = **syntaxTable**.get(i).getAf();  
 flag = **false**;  
 }  
 } **while** (!flag);  
 } **catch** (NullPointerException e) {  
 System.***out***.println(**"Check your code!!! NULL Pointer exception in treeNode"**);  
 e.printStackTrace();  
 }  
 }  
 treeNode = (DefaultMutableTreeNode) treeNode.getRoot();  
 System.***out***.printf(**"time(ms) : %f"**, (**float**)(System.*nanoTime*() - time) / 1000000);  
 **return new** DefaultTreeModel(treeNode.getNextNode());  
}  
  
*/\*\*  
 \** ***@throws*** *IOException Make graphical user interface and put tables and result there  
 \*/***void** getInfo() **throws** IOException {  
 GUI frame = **new** GUI();  
 frame.initTable(**keywords**, **identificators**, **constants**, **result**, **signalProgram**, syntax());  
 frame.drawFrame();  
}

**StandartTables.java**

package com.itcherry.translators;

import java.util.Arrays;

import java.util.HashMap;

public abstract class StandartTables {

**private void** fillSyntax() {  
 **syntaxTable**.add(**new** KnuthItem(**"<signal-program>"**, **"<program>"**, 1, -1));  
 **syntaxTable**.add(**new** KnuthItem(**"<program>"**, **keywords**.get(**"PROGRAM"**).toString(), 2, -1));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"<procedure-identifier>"**, 52, -1));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"59"**, 4, -1)); *// ;* **syntaxTable**.add(**new** KnuthItem(**null**, **"<block>"**, 6, -1));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"46"**, -2, -1));*// .* **syntaxTable**.add(**new** KnuthItem(**"<block>"**, **"<declarations>"**, 10, -1));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **keywords**.get(**"BEGIN"**).toString(), 8, -1));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"<statements-list>"**, 21, -1));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **keywords**.get(**"END"**).toString(), -2, -1));  
 **syntaxTable**.add(**new** KnuthItem(**"<declarations>"**, **"<constant-declarations>|"**, 11, -1));  
 **syntaxTable**.add(**new** KnuthItem(**"<constant-declarations>"**, **keywords**.get(**"CONST"**).toString(), 12, 13));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"<constant-declarations-list>|"**, 14, -1));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"<empty>"**, -2, -2));  
 **syntaxTable**.add(**new** KnuthItem(**"<constant-declarations-list>"**, **"<constant-declaration>"**, 17, 16));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"<constant-declarations-list>|"**, 14, -1));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"<empty>"**, -2, -2));  
 **syntaxTable**.add(**new** KnuthItem(**"<constant-declaration>"**, **"<constant-identifier>"**, 53, -10));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"61"**, 19, -1)); *// =* **syntaxTable**.add(**new** KnuthItem(**null**, **"<constant>"**, 54, -1));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"59"**, -2, -1)); *// ;* **syntaxTable**.add(**new** KnuthItem(**"<statements-list>"**, **"<statement>"**, 24, 23));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"<statements-list>|"**, 21, -1));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"<empty>"**, -2, -2));  
 **syntaxTable**.add(**new** KnuthItem(**"<statement>"**, **keywords**.get(**"CASE"**).toString(), 25, -10));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"<expression>"**, 38, -1));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **keywords**.get(**"OF"**).toString(), 27, -1));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"<alternatives-list>"**, 30, -1));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **keywords**.get(**"ENDCASE"**).toString(), 29, -1));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"59"**, -2, -1)); *// ;* **syntaxTable**.add(**new** KnuthItem(**"<alternatives-list>"**, **"<alternative>"**, 33, 32));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"<alternatives-list>|"**, 30, -1));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"<empty>"**, -2, -2));  
 **syntaxTable**.add(**new** KnuthItem(**"<alternative>"**, **"<expression>"**, 38, -10));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"58"**, 35, -1)); *// :* **syntaxTable**.add(**new** KnuthItem(**null**, **"47"**, 36, -1)); *// /* **syntaxTable**.add(**new** KnuthItem(**null**, **"<statements-list>"**, 21, -1));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"92"**, -2, -1)); *// \* **syntaxTable**.add(**new** KnuthItem(**"<expression>"**, **"<summand>"**, 49, 40));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"<summands-list>|"**, 43, -1));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"45"**, 41, -10)); *// -* **syntaxTable**.add(**new** KnuthItem(**null**, **"<summand>"**, 49, -1));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"<summands-list>|"**, 43, -1));  
 **syntaxTable**.add(**new** KnuthItem(**"<summands-list>"**, **"<add-instruction>"**, 47, 46));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"<summand>"**, 49, -1));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"<summands-list>|"**, 43, -1));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"<empty>"**, -2, -2));  
 **syntaxTable**.add(**new** KnuthItem(**"<add-instruction>"**, **"43"**, -2, 48));*// +* **syntaxTable**.add(**new** KnuthItem(**null**, **"45"**, -2, -10)); *// -* **syntaxTable**.add(**new** KnuthItem(**"<summand>"**, **"<variable-identifier>"**, 51, 50));  
 **syntaxTable**.add(**new** KnuthItem(**null**, **"<unsigned-integer>|"**, 55, 55));  
 **syntaxTable**.add(**new** KnuthItem(**"<variable-identifier>"**, **"SI"**, -2, -1));  
 **syntaxTable**.add(**new** KnuthItem(**"<procedure-identifier>"**, **"SI"**, -2, -1));  
 **syntaxTable**.add(**new** KnuthItem(**"<constant-identifier>"**, **"SI"**, -2, -1));  
 **syntaxTable**.add(**new** KnuthItem(**"<constant>"**, **"SC"**, -2, -1));  
 **syntaxTable**.add(**new** KnuthItem(**"<unsigned-integer>"**, **"SC"**, -2, -1));  
}

**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

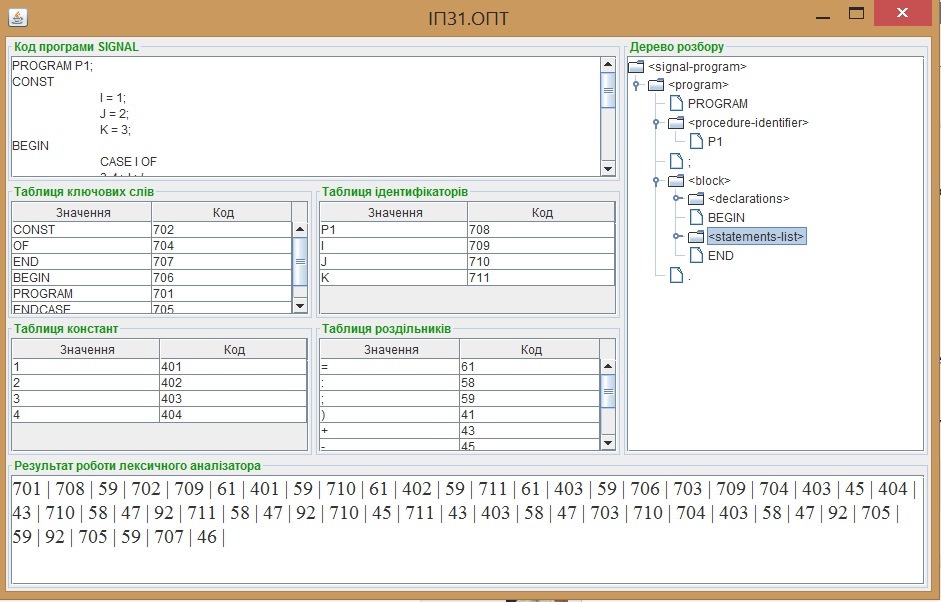
3 : /\

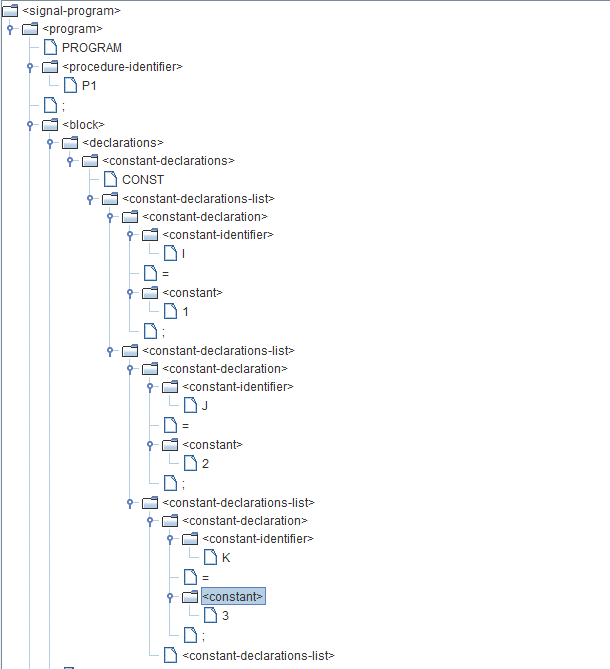
ENDCASE;

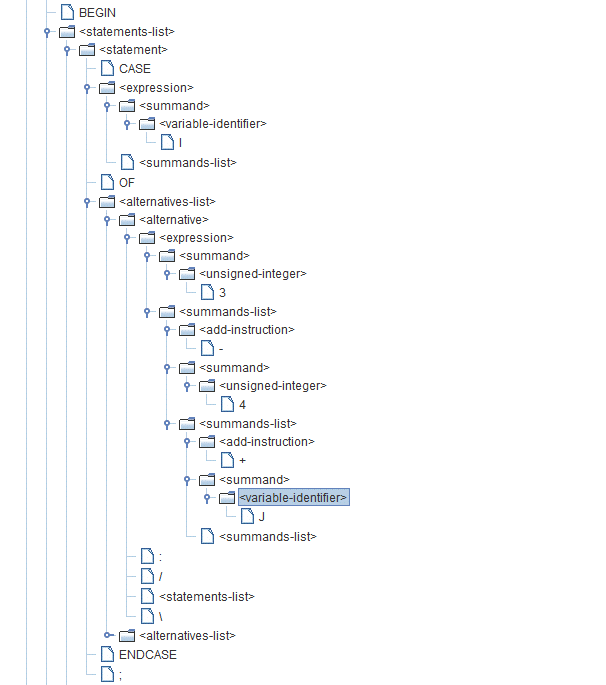
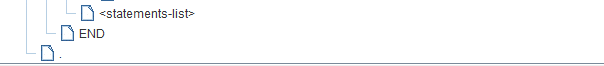
\

ENDCASE;

END.





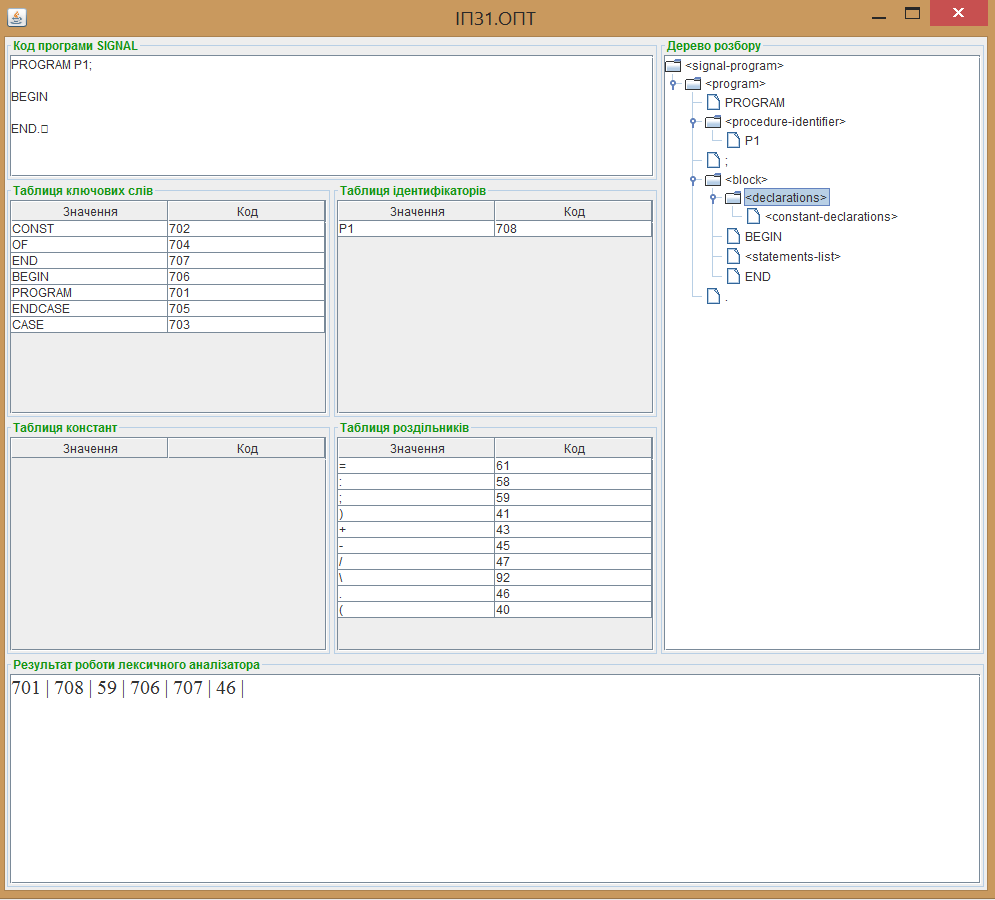


**Приклад 2 :**

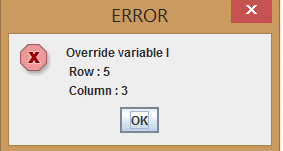
PROGRAM ProG1;

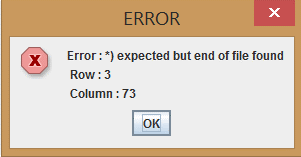
BEGIN

END.

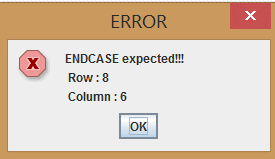


**Приклад 3 :**

PROGRAM P1;  
CONST  
 I = 1;  
 K = 3;  
 I = 4;  
BEGIN  
 CASE K OF  
 3(\*fsdfsdfg\*\*\*)+K: /\  
 ENDCASE;  
END.

**Приклад 4:**

PROGRAM P1;  
CONST  
 (\*I = 1;  
 K = 3;  
BEGIN  
 CASE K OF  
 3+K: /\  
 ENDCASE;  
END.

**Приклад 5:**

PROGRAM P1;  
CONST  
 I = 1;  
 K = 3;  
BEGIN  
 CASE K OF  
 3+K: /\  
 ;  
END.