НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«Київський політехнічний інститут ім. Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп’ютерних систем

##### Лабораторна робота №2

**«Розробка генератора коду»**

***з дисципліни "Інженерія програмного забезпечення1. Основи проектування трансляторів"***

**Варіант № 13**

Виконав: Микитенко С.С.

#### Група: КB-51

Номер залікової книжки: КВ-5113

Оцінка

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2 семестр 2017/2018

***Варіант 13***

1. <signal-program> --> <program>

2. <program> --> PROCEDURE <procedure-identifier> <parameters-list> ; <block> ;

3. <block> --> <declarations> BEGIN <statementslist> END

4. <declarations> --> <procedure-declarations>

5. <procedure-declarations> --> <procedure> <procedure- declarations> |<empty>

6. <procedure> --> PROCEDURE <procedureidentifier><parameters-list> ;

7. <parameters-list> --> ( <variable-identifier> <identifiers-list> ) | <empty>

8. <identifiers-list> --> , <variable-identifier> <identifiers-list> | <empty>

9. <statements-list> --> <statement> <statementslist> | <empty>

10. <statement> --> <procedure-identifier><actualarguments> ; | RETURN ;

11. <actual-arguments> --> ( <unsigned-integer><actual- arguments-list> ) | <empty>

12. <actual-arguments-list> --> , <unsigned-integer> <actual-arguments-list> | <empty>

13. <variable-identifier> --> <identifier>

14. <procedure-identifier> --> <identifier>

15. <identifier> --> <letter><string>

16. <string> --> <letter><string> | <digit><string> | <empty>

17. <unsigned-integer> --> <digit><digits-string>

18. <digits-string> --> <digit><digits-string> | <empty>

19. <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

20. <letter> --> A | B | C | D | ... | Z

**Таблиця переходів машини Кнута**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Адреса** | **Код операції** | **AT** | **AF** |
| 1 | <signal-program> | <program> |  | F |
| 2 |  | # | T | F |
| 3 | <program> | PROCEDURE |  | F |
| 4 |  | <procedure-identifier> |  | F |
| 5 |  | <parameters-list> |  | F |
| 6 |  | ; |  | F |
| 7 |  | <block> |  | F |
| 8 |  | ; | T | F |
| 9 | <block> | <declarations> |  | F |
| 10 |  | BEGIN |  | F |
| 11 |  | <statements-list> |  | F |
| 12 |  | END | T | F |
| 13 | <declarations> | <procedure-declarations> | T | F |
| 14 | <procedure-declarations> | <procedure> |  | T |
| 15 |  | <procedure- declarations> | T | F |
| 16 | <procedure> | PROCEDURE |  | F |
| 17 |  | <procedure-identifier> |  | F |
| 18 |  | <parameters-list> |  | F |
| 19 |  | ; | T | F |
| 20 | <parameters-list> | ( |  | T |
| 21 |  | <variable-identifier> |  | F |
| 22 |  | <identifiers-list> |  | F |
| 23 |  | ) | T | F |
| 24 | <identifiers-list> | , |  | T |
| 25 |  | <variable-identifier> |  | F |
| 26 |  | <identifiers-list> | T | F |
| 27 | <statements-list> | <statement> |  | T |
| 28 |  | <statements-list> | T | F |
| 29 | <statement> | <procedure-identifier> |  | ->32 |
| 30 |  | <actual-arguments> |  | F |
| 31 |  | ; | T | F |
| 32 |  | RETURN |  | F |
| 33 |  | ; | T | F |
| 34 | <actual-arguments> | ( |  | T |
| 35 |  | <unsigned-integer> |  | F |
| 36 |  | <actual- arguments-list> |  | F |
| 37 |  | ) | T | F |
| 38 | <actual-arguments-list> | , |  | T |
| 39 |  | <unsigned-integer> |  | F |
| 40 |  | <actual-arguments-list> | T | F |
| 41 | <variable-identifier> | <identifier> | T | F |
| 42 | <procedure-identifier> | <identifier> | T | F |

**Лістинг програми**

**public** **class** Main {

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

Tests test = **new** Tests();

test.parser("test1.txt");

test.parser("test2.txt");

test.parser("test3.txt");

test.parser("test4.txt");

test.parser("test5.txt");

test.parser("test6.txt");

test.parser("test7.txt");

test.parser("test8.txt");

}

}

**public** **class** Lexeme {

**public** **int** code;

**public** **int** row;

**public** **int** pos;

**public** **void** set(**int** new\_code, **int** new\_row, **int** new\_pos) {

code = new\_code;

row = new\_row;

pos = new\_pos;

}

**public** Lexeme() {

}

**public** Lexeme(**int** new\_code, **int** new\_row, **int** new\_pos) {

code = new\_code;

row = new\_row;

pos = new\_pos;

}

}

**public** **class** ProcedureDeclaration {

**public** **int** id;

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

}

**public** **class** TreeNode {

**public** **int** level;

**public** **int** code = -1;

**public** String node;

TreeNode(**int** \_level, String \_node){

level = \_level;

node = \_node;

}

TreeNode(**int** \_level, **int** \_code){

level = \_level;

code = \_code;

}

}

**public** **class** Tests {

**public** **void** parser(String fileName)

{

Tables table = **new** Tables();

Lexer lexer = **new** Lexer();

lexer.analize(table, fileName);

table.printTables();

**if** (table.errors.size()==0) {

Parser parser = **new** Parser();

parser.analize(table);

table.printTree();

}

table.printErrors();

}

**public** Tests() {

}

}

**public** **class** Tables {

//lexer

ArrayList<String> keys = **new** ArrayList<String>(4);// 400 - 499

ArrayList<String> consts = **new** ArrayList<String>();// 500 - 999

ArrayList<String> ids = **new** ArrayList<String>();// 1000 - ...

ArrayList<Integer> attrs = **new** ArrayList<Integer>(256);

ArrayList<Lexeme> lexemes = **new** ArrayList<Lexeme>();

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

//parser

ArrayList<ProcedureDeclaration> procedureDeclarations = **new** ArrayList<ProcedureDeclaration>();

ArrayList<ProcedureDeclaration> statements = **new** ArrayList<ProcedureDeclaration>();

ArrayList<TreeNode> tree = **new** ArrayList<TreeNode>();

//lexer

**public** String getTokken(**int** code){

**if** (code < 128) {

**return** String.*valueOf*((**char**)code);

}

**if** (code < 500) {

**return** keys.get(code-400);

}

**if** (code < 1000) {

**return** consts.get(code-500);

}

**return** ids.get(code-1000);

}

**public** **void** printErrors(){

**int** i;

**if** (errors.size()>0) {

System.***out***.println("\nErrors:");

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

System.***out***.format(errors.get(i)+"%n");

}

} **else** {

System.***out***.println("\nNo errors");

}

}

//...

//parser

**public** **void** addParserError(String str, **int** index) {

**if** (index<lexemes.size()){

Lexeme buff = lexemes.get(index);

addError("Lexer: Error (line "+buff.row+", column "+buff.pos +"): expected {"+str+"} but found {"+getTokken(buff.code)+"}");

} **else** {

addError("Lexer: Error (on eof): expected {#} but found EOF");

}

}

**void** addTreeNode(**int** \_level, String \_node) {

TreeNode buff = **new** TreeNode(\_level,\_node);

tree.add(buff);

}

**void** addTreeNode(**int** \_level, **int** \_code) {

TreeNode buff = **new** TreeNode(\_level,\_code);

tree.add(buff);

}

String getTreeNode(**int** id) {

**if** (id<tree.size()) {

TreeNode node = tree.get(id);

String buff = "";

**for** (**int** i=0;i<node.level\*2;i++) {

buff += '.';

}

**if** (node.code>0) {

buff += node.code +" "+ getTokken(node.code) +"\n";

} **else** {

buff += node.node +"\n";

}

**return** buff;

}

**return** "OutOfRange";

}

**void** printTree() {

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

System.***out***.print(getTreeNode(i));

}

}

}

**public** **class** Parser {

**int** index=0;

Tables table;

ProcedureDeclaration buffer;

**private** **boolean** identifier(**int** level) {

table.addTreeNode(level, "<identifier>");

**if** (index >= table.lexemes.size()) {

table.addTreeNode(level + 1, "<empty>");

**return** **false**;

}

**int** buff = table.lexemes.get(index).code;

**if** (buff < 1000) {

**return** **false**;

}

table.addTreeNode(level + 1, buff);

index++;

**return** **true**;

}

**private** **boolean** variableIdentifier(**int** level) {

table.addTreeNode(level, "<variable-identifier>");

**if** (identifier(level+1) == **false**) {

**return** **false**;

}

buffer.parameters.add(table.lexemes.get(index-1).code);

**return** **true**;

}

**private** **boolean** procedureIdentifier(**int** level) {

table.addTreeNode(level, "<procedure-identifier>");

**if** (identifier(level+1) == **false**) {

**return** **false**;

}

buffer.id = table.lexemes.get(index-1).code;

**return** **true**;

}

**private** **boolean** unsignedInteger(**int** level) {

table.addTreeNode(level, "<unsigned-integer>");

**if** (index >= table.lexemes.size()) {

**return** **false**;

}

**int** buff = table.lexemes.get(index).code;

**if** ((buff < 500) || (buff > 999)) {

**return** **false**;

}

buffer.parameters.add(buff);

table.addTreeNode(level + 1, buff);

index++;

**return** **true**;

}

**private** **boolean** actualArgumentsList(**int** level) {

table.addTreeNode(level, "<actual-arguments-list>");

**if** ((index >= table.lexemes.size()) || (table.lexemes.get(index).code != ',')) {

table.addTreeNode(level + 1, "<empty>");

**return** **true**;

}

table.addTreeNode(level + 1, table.lexemes.get(index).code);

index++;

**if** (unsignedInteger(level+1) == **false**) {

**return** **false**;

}

**if** (actualArgumentsList(level+1) == **false**) {

**return** **false**;

}

**return** **true**;

}

**private** **boolean** actualArguments(**int** level) {

table.addTreeNode(level, "<actual-arguments>");

**if** ((index >= table.lexemes.size()) || (table.lexemes.get(index).code != '(')) {

table.addTreeNode(level + 1, "<empty>");

**return** **true**;

}

table.addTreeNode(level + 1, table.lexemes.get(index).code);

index++;

**if** (unsignedInteger(level+1) == **false**) {

**return** **false**;

}

**if** (actualArgumentsList(level+1) == **false**) {

**return** **false**;

}

**if** ((index >= table.lexemes.size()) || (table.lexemes.get(index).code != ')')) {

**return** **false**;

}

table.addTreeNode(level + 1, table.lexemes.get(index).code);

index++;

**return** **true**;

}

**private** **boolean** statement(**int** level) {

table.addTreeNode(level, "<statement>");

**int** oldSize = table.tree.size();

**int** buff = index;

buffer = **new** ProcedureDeclaration();

**if** (procedureIdentifier(level+1) == **false**) {

index = buff;

**for** (**int** i = table.tree.size() - 1; i >= oldSize; i--) {

table.tree.remove(i);

}

**if** ((index >= table.lexemes.size()) || (table.lexemes.get(index).code != 403)) {

**return** **false**;

}

table.addTreeNode(level + 1, table.lexemes.get(index).code);

index++;

**if** ((index >= table.lexemes.size()) || (table.lexemes.get(index).code != ';')){

index = buff;

**return** **false**;

}

table.addTreeNode(level + 1, table.lexemes.get(index).code);

buffer.id = 403;

table.statements.add(buffer);

index++;

**return** **true**;

}

**if** (actualArguments(level+1) == **false**) {

index = buff;

**return** **false**;

}

**if** ((index >= table.lexemes.size()) || (table.lexemes.get(index).code != ';')) {

index = buff;

**return** **false**;

}

table.addTreeNode(level + 1, table.lexemes.get(index).code);

index++;

table.statements.add(buffer);

**return** **true**;

}

**private** **boolean** statementsList(**int** level) {

table.addTreeNode(level, "<statements-list>");

**int** oldSize = table.tree.size();

**if** (statement(level+1) == **false**) {

**for** (**int** i = table.tree.size() - 1; i >= oldSize; i--) {

table.tree.remove(i);

}

table.addTreeNode(level + 1, "<empty>");

**return** **true**;

}

**if** (statementsList(level+1) == **false**) {

**return** **false**;

}

**return** **true**;

}

**private** **boolean** identifiersList(**int** level) {

table.addTreeNode(level, "<identifiers-list>");

**if** ((index >= table.lexemes.size()) || (table.lexemes.get(index).code != ',')) {

table.addTreeNode(level + 1, "<empty>");

**return** **true**;

}

table.addTreeNode(level + 1, table.lexemes.get(index).code);

index++;

**if** (variableIdentifier(level+1) == **false**) {

**return** **false**;

}

**if** (identifiersList(level+1) == **false**) {

**return** **false**;

}

**return** **true**;

}

**private** **boolean** paramatersList(**int** level) {

table.addTreeNode(level, "<paramaters-list>");

**if** ((index >= table.lexemes.size()) || (table.lexemes.get(index).code != '(')) {

table.addTreeNode(level + 1, "<empty>");

**return** **true**;

}

table.addTreeNode(level + 1, table.lexemes.get(index).code);

index++;

**if** (variableIdentifier(level+1) == **false**) {

//err expected id

**return** **false**;

}

**if** (identifiersList(level+1) == **false**) {

table.addParserError("<identifier>",index);

**return** **false**;

}

**if** ((index >= table.lexemes.size()) || (table.lexemes.get(index).code != ')')) {

table.addParserError(")",index);

**return** **false**;

}

table.addTreeNode(level + 1, table.lexemes.get(index).code);

index++;

**return** **true**;

}

**private** **boolean** procedure(**int** level) {

table.addTreeNode(level, "<procedure>");

**int** buff = index;

**if** ((index >= table.lexemes.size()) || (table.lexemes.get(index).code != 400)) {

**return** **false**;

}

table.addTreeNode(level + 1, table.lexemes.get(index).code);

index++;

buffer = **new** ProcedureDeclaration();

**if** (procedureIdentifier(level+1) == **false**) {

index = buff;

**return** **false**;

}

**if** (paramatersList(level+1) == **false**) {

index = buff;

**return** **false**;

}

**if** ((index >= table.lexemes.size()) || (table.lexemes.get(index).code != ';')) {

index = buff;

**return** **false**;

}

table.addTreeNode(level + 1, table.lexemes.get(index).code);

index++;

table.procedureDeclarations.add(buffer);

**return** **true**;

}

**private** **boolean** procedureDeclarations(**int** level) {

table.addTreeNode(level, "<procedure-declarations>");

**int** oldSize = table.tree.size();

**if** (procedure(level+1) == **false**) {

**for** (**int** i = table.tree.size() - 1; i >= oldSize; i--) {

table.tree.remove(i);

}

table.addTreeNode(level + 1, "<empty>");

**return** **true**;

}

**if** (procedureDeclarations(level+1) == **false**) {

**return** **false**;

}

**return** **true**;

}

**private** **boolean** declarations(**int** level) {

table.addTreeNode(level, "<declarations>");

**if** (procedureDeclarations(level+1) == **false**) {

**return** **false**;

}

**return** **true**;

}

**private** **boolean** block(**int** level) {

table.addTreeNode(level, "<block>");

**if** (declarations(level+1) == **false**) {

**return** **false**;

}

**if** ((index >= table.lexemes.size()) || (table.lexemes.get(index).code != 401)) {

table.addParserError("BEGIN",index);

**return** **false**;

}

table.addTreeNode(level + 1, table.lexemes.get(index).code);

index++;

**if** (statementsList(level+1) == **false**) {

**return** **false**;

}

**if** ((index >= table.lexemes.size()) || (table.lexemes.get(index).code != 402)) {

table.addParserError("END",index);

**return** **false**;

}

table.addTreeNode(level + 1, table.lexemes.get(index).code);

index++;

**return** **true**;

}

**private** **boolean** program(**int** level){

table.addTreeNode(level, "<program>");

**if** ((index >= table.lexemes.size()) || (table.lexemes.get(index).code != 400)) {

table.addParserError("PROCEDURE",index);

**return** **false**;

}

table.addTreeNode(level + 1, table.lexemes.get(index).code);

index++;

buffer = **new** ProcedureDeclaration();

**if** (procedureIdentifier(level+1) == **false**) {

table.addParserError("<identifier>",index);

**return** **false**;

}

**if** (paramatersList(level+1) == **false**) {

**return** **false**;

}

table.procedureDeclarations.add(buffer);

**if** ((index >= table.lexemes.size()) || (table.lexemes.get(index).code != ';')) {

table.addParserError(";",index);

**return** **false**;

}

table.addTreeNode(level + 1, table.lexemes.get(index).code);

index++;

**if** (block(level+1) == **false**) {

**return** **false**;

}

**if** ((index >= table.lexemes.size()) || (table.lexemes.get(index).code != ';')) {

table.addParserError(";",index);

**return** **false**;

}

table.addTreeNode(level + 1, table.lexemes.get(index).code);

index++;

**return** **true**;

}

**private** **boolean** signalProgram(){

**int** level = 0;

table.addTreeNode(level, "<signal-program>");

**if** (program(level+1) == **false**) {

**return** **false**;

}

**if** ((index < table.lexemes.size()) && (table.lexemes.get(index).code == '#')) {

**return** **true**;

}

table.addParserError("#",index);

**return** **false**;

}

**public** **void** analize(Tables inTable) {

table = inTable;

**if** (table.lexemes.size()>1) {

Lexeme buff = table.lexemes.get(table.lexemes.size()-1);

Lexeme lexeme = **new** Lexeme('#',buff.row,buff.pos+1);

table.lexemes.add(lexeme);

signalProgram();

}

}

**public** Parser() {

}

}

**True-Тести**

PROCEDURE ABADA;

PROCEDURE KENTAVRA;

BEGIN

KENTAVRA;

RETURN;

END;<signal-program>

..<program>

....400 PROCEDURE

....<procedure-identifier>

......<identifier>

........1000 ABADA

....<paramaters-list>

......<empty>

....59 ;

....<block>

......<declarations>

........<procedure-declarations>

..........<procedure>

............400 PROCEDURE

............<procedure-identifier>

..............<identifier>

................1001 KENTAVRA

............<paramaters-list>

..............<empty>

............59 ;

..........<procedure-declarations>

............<empty>

......401 BEGIN

......<statements-list>

........<statement>

..........<procedure-identifier>

............<identifier>

..............1001 KENTAVRA

..........<actual-arguments>

............<empty>

..........59 ;

........<statements-list>

..........<statement>

............403 RETURN

............59 ;

..........<statements-list>

............<empty>

......402 END

....59 ;

No errors

PROCEDURE ABADA(A,B,D);

BEGIN

END;<signal-program>

..<program>

....400 PROCEDURE

....<procedure-identifier>

......<identifier>

........1000 ABADA

....<paramaters-list>

......40 (

......<variable-identifier>

........<identifier>

..........1001 A

......<identifiers-list>

........44 ,

........<variable-identifier>

..........<identifier>

............1002 B

........<identifiers-list>

..........44 ,

..........<variable-identifier>

............<identifier>

..............1003 D

..........<identifiers-list>

............<empty>

......41 )

....59 ;

....<block>

......<declarations>

........<procedure-declarations>

..........<empty>

......401 BEGIN

......<statements-list>

........<empty>

......402 END

....59 ;

No errors

PROCEDURE ABADA;

PROCEDURE KEN(A,B);

PROCEDURE TAR(A);

BEGIN

RETURN;

TAR(3,2);

KEN(4);

ABADA;

END;

<signal-program>

..<program>

....400 PROCEDURE

....<procedure-identifier>

......<identifier>

........1000 ABADA

....<paramaters-list>

......<empty>

....59 ;

....<block>

......<declarations>

........<procedure-declarations>

..........<procedure>

............400 PROCEDURE

............<procedure-identifier>

..............<identifier>

................1001 KEN

............<paramaters-list>

..............40 (

..............<variable-identifier>

................<identifier>

..................1002 A

..............<identifiers-list>

................44 ,

................<variable-identifier>

..................<identifier>

....................1003 B

................<identifiers-list>

..................<empty>

..............41 )

............59 ;

..........<procedure-declarations>

............<procedure>

..............400 PROCEDURE

..............<procedure-identifier>

................<identifier>

..................1004 TAR

..............<paramaters-list>

................40 (

................<variable-identifier>

..................<identifier>

....................1002 A

................<identifiers-list>

..................<empty>

................41 )

..............59 ;

............<procedure-declarations>

..............<empty>

......401 BEGIN

......<statements-list>

........<statement>

..........403 RETURN

..........59 ;

........<statements-list>

..........<statement>

............<procedure-identifier>

..............<identifier>

................1004 TAR

............<actual-arguments>

..............40 (

..............<unsigned-integer>

................500 3

..............<actual-arguments-list>

................44 ,

................<unsigned-integer>

..................501 2

................<actual-arguments-list>

..................<empty>

..............41 )

............59 ;

..........<statements-list>

............<statement>

..............<procedure-identifier>

................<identifier>

..................1001 KEN

..............<actual-arguments>

................40 (

................<unsigned-integer>

..................502 4

................<actual-arguments-list>

..................<empty>

................41 )

..............59 ;

............<statements-list>

..............<statement>

................<procedure-identifier>

..................<identifier>

....................1000 ABADA

................<actual-arguments>

..................<empty>

................59 ;

..............<statements-list>

................<empty>

......402 END

....59 ;

No errors

**false-Тести**

PROCEDURE ABADA;

PROCEDURE KEN(A,B);

PROCEDURE TAR(A);

BEGIN

RETURN;

;

KEN(4);

ABADA;

END;

<signal-program>

..<program>

....400 PROCEDURE

....<procedure-identifier>

......<identifier>

........1000 ABADA

....<paramaters-list>

......<empty>

....59 ;

....<block>

......<declarations>

........<procedure-declarations>

..........<procedure>

............400 PROCEDURE

............<procedure-identifier>

..............<identifier>

................1001 KEN

............<paramaters-list>

..............40 (

..............<variable-identifier>

................<identifier>

..................1002 A

..............<identifiers-list>

................44 ,

................<variable-identifier>

..................<identifier>

....................1003 B

................<identifiers-list>

..................<empty>

..............41 )

............59 ;

..........<procedure-declarations>

............<procedure>

..............400 PROCEDURE

..............<procedure-identifier>

................<identifier>

..................1004 TAR

..............<paramaters-list>

................40 (

................<variable-identifier>

..................<identifier>

....................1002 A

................<identifiers-list>

..................<empty>

................41 )

..............59 ;

............<procedure-declarations>

..............<empty>

......401 BEGIN

......<statements-list>

........<statement>

..........403 RETURN

..........59 ;

........<statements-list>

..........<empty>

Errors:

Parser: Error (line 6, column 1): expected {END} but found {;}

PROCEDURE ABADA;

PROCEDURE KEN(A,B);

PROCEDURE TAR(A);

RETURN;

TAR(3,2);

KEN(4);

ABADA;

END;

<signal-program>

..<program>

....400 PROCEDURE

....<procedure-identifier>

......<identifier>

........1000 ABADA

....<paramaters-list>

......<empty>

....59 ;

....<block>

......<declarations>

........<procedure-declarations>

..........<procedure>

............400 PROCEDURE

............<procedure-identifier>

..............<identifier>

................1001 KEN

............<paramaters-list>

..............40 (

..............<variable-identifier>

................<identifier>

..................1002 A

..............<identifiers-list>

................44 ,

................<variable-identifier>

..................<identifier>

....................1003 B

................<identifiers-list>

..................<empty>

..............41 )

............59 ;

..........<procedure-declarations>

............<procedure>

..............400 PROCEDURE

..............<procedure-identifier>

................<identifier>

..................1004 TAR

..............<paramaters-list>

................40 (

................<variable-identifier>

..................<identifier>

....................1002 A

................<identifiers-list>

..................<empty>

................41 )

..............59 ;

............<procedure-declarations>

..............<empty>

Errors:

Parser: Error (line 4, column 1): expected {BEGIN} but found {RETURN}

PROCEDURE ABADA;

PROCEDURE KEN(A,B);

PROCEDURE TAR(A;

BEGIN

RETURN;

TAR(3,2);

KEN(4);

ABADA;

END;

<signal-program>

..<program>

....400 PROCEDURE

....<procedure-identifier>

......<identifier>

........1000 ABADA

....<paramaters-list>

......<empty>

....59 ;

....<block>

......<declarations>

........<procedure-declarations>

..........<procedure>

............400 PROCEDURE

............<procedure-identifier>

..............<identifier>

................1001 KEN

............<paramaters-list>

..............40 (

..............<variable-identifier>

................<identifier>

..................1002 A

..............<identifiers-list>

................44 ,

................<variable-identifier>

..................<identifier>

....................1003 B

................<identifiers-list>

..................<empty>

..............41 )

............59 ;

..........<procedure-declarations>

............<empty>

Errors:

Parser: Error (line 3, column 1): expected {BEGIN} but found {PROCEDURE}