НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«Київський політехнічний інститут ім. Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп’ютерних систем

##### Лабораторна робота №1

**«Розробка лексичного аналізатора»**

***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>

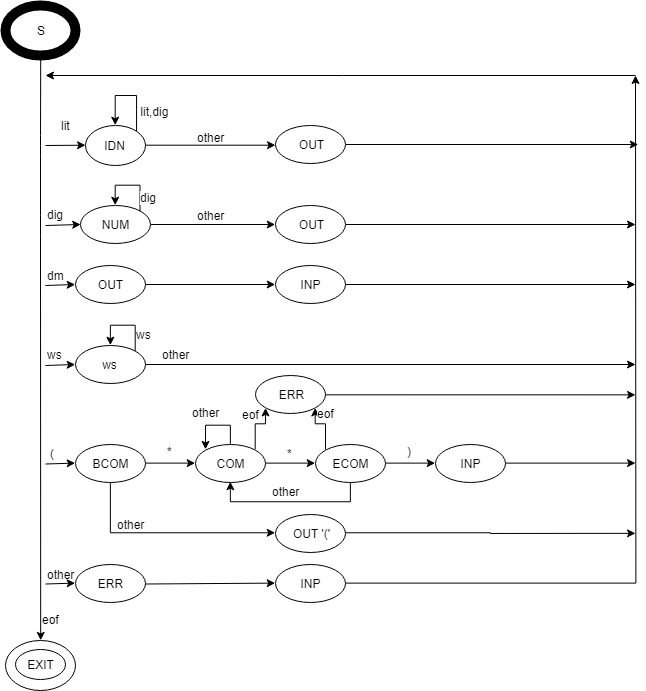
17. <unsigned-integer> --> <digit><digits-string>

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

Граф автомату



Лістинг програми

**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** Tests {

**public** **void** lexer(String fileName) {

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

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

lexer.analize(table, fileName);

table.printTables();

table.printErrors();

}

**public** Tests() {

}

}

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

**public** **static** **void** main() {

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

test.lexer("test1.txt");

test.lexer("test2.txt");

test.lexer("test3.txt");

}

}

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

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>();

**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** printTables(){

**int** i;

Lexeme lexeme;

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

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

lexeme = lexemes.get(i);

System.***out***.format("%04d %04d %05d "+ getTokken(lexeme.code)+"%n", lexeme.row, lexeme.pos, lexeme.code);

}

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

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

System.***out***.format("%05d "+ keys.get(i)+"%n", i+400);

}

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

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

System.***out***.format("%05d "+ consts.get(i)+"%n", i+500);

}

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

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

System.***out***.format("%05d "+ ids.get(i)+"%n", i+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");

}

}

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

**int** id = consts.indexOf(str);

**if** (id == -1) {

**if**(consts.size()<=500) {

id = consts.size();

consts.add(str);

} **else** {

**return** -1;

}

}

**return** id + 500;

}

**private** **int** getKeyId(String str) {

//return index of element in keys(+400) or ids(+1000)

**int** id = keys.indexOf(str);

**if** (id == -1) {

id = ids.indexOf(str);

**if** (id == -1) {

id = ids.size();

ids.add(str);

}

**return** id + 1000;

}

**return** id + 400;

}

**public** **void** addConst(String str, **int** j, **int** i) {

**int** constsId = getConstsId(str);

**if** (constsId == -1){

errors.add("Lexer: Error (line "+j+", column "+i +"): more than 500 consts");

} **else** {

addLexeme(constsId, j, i);

}

}

**public** **void** addIds(String str, **int** j, **int** i) {

addLexeme(getKeyId(str), j, i);

}

**public** **void** addError(String str) {

errors.add(str);

}

**public** **void** addLexeme(**int** code, **int** j, **int** i) {

Lexeme lexeme = **new** Lexeme(code, j, i);

lexemes.add(lexeme);

}

**public** **int** getAttrs(**int** digit) {

**if** ((digit >= 0)&&(digit <=127)){

**return** attrs.get(digit);

}

**return** 6;

}

**public** Tables() {

**for** (**int** i = 0; i <= 127; i++) {

attrs.add(6);

}

//white space

attrs.set(8, 0);//backspace

attrs.set(9, 0);//tab

attrs.set(10, 0);//next line

attrs.set(11, 0);//tab

attrs.set(13, 0);//carriage return

attrs.set(32, 0);//space

**for** (**int** i = 48; i <= 57; i++) {

attrs.set(i, 1);//digits

}

**for** (**int** i = 65; i <= 90; i++) {

attrs.set(i, 2);//letters

}

attrs.set(59, 3);//;

attrs.set(40, 5);//( ==> (\*

attrs.set(41, 3);//)

attrs.set(44, 3);//,

keys.add("PROCEDURE");//0

keys.add("BEGIN");//1

keys.add("END");//2

keys.add("RETURN");//3

}

}

**public** **class** Lexer {

**public** **void** prnt() {

**char** c='?';

**int** i=63;

System.***out***.println((**int**)c);

System.***out***.println((**char**)i);

System.***out***.println(i==c);

}

**public** **void** analize(Tables table, String fileName) {

**try**(FileReader reader = **new** FileReader("src/"+fileName))

{

**int** digit= reader.read();

**int** i = 1;//position in row

**int** j = 1;//row number

String str;

**while**((digit) != -1){

**switch**(table.getAttrs(digit)) {

**case** 0: //whitespace

**if** (digit == 10) {

j++;

i=1;

digit = reader.read();//just read next symbol

} **else** **if** (digit == 13) {

digit = reader.read();//just read next symbol

} **else** {

i++;

digit = reader.read();//just read next symbol

}

**break**;

**case** 1: //digit

str = "";

**while**(table.getAttrs(digit) == 1) {

str+=(**char**)digit;

i++;

digit = reader.read();//read next symbol

**if** (digit == -1) {

**break**;

}

}

table.addConst(str,j,i - str.length());

**break**;

**case** 2: //Letter

str = "";

**while** ((table.getAttrs(digit) == 1) || (table.getAttrs(digit) == 2)) {

str+=(**char**)digit;

i++;

digit = reader.read();//read next symbol

**if** (digit == -1) {

**break**;

}

}

table.addIds(str,j,i - str.length());

**break**;

**case** 3: //Delimiter

table.addLexeme(digit,j,i);

i++;

digit = reader.read();//read next symbol

**break**;

**case** 5: //"(" - may be beginning of "(\*" - comments

digit = reader.read();//read next symbol

i++;

**if** (digit != '\*') {

table.addLexeme(((**int**)'('), j, i - 1);

**break**;

}

Boolean previous=**false**;

**while** (digit != -1) {

**if** (digit == 10) {

j++;

i=1;

digit = reader.read();//just read next symbol

} **else** **if** (digit == 13) {

digit = reader.read();//just read next symbol

} **else** {

i++;

digit = reader.read();//read next symbol

**if** (digit == '\*') {

previous = **true**;

} **else** **if** (previous) {

**if** (digit == ')') {

**break**;

} **else** {

previous = **false**;

}

}

}

}

**if** (digit == -1) {

table.addError("Lexer: Error (line "+j+", column "+i +"): expected '\*)' but found EOF");

} **else** {

**if** (previous) {

i++;

digit = reader.read();//read next symbol

}

}

**break**;

**case** 6: // wrong character

table.addError("Lexer: Error (line "+j+", column "+i +"): wrong character [ "+((**char**)digit)+" ] ");

i++;

digit = reader.read();//read next symbol

**break**;

**default**:

System.***out***.print("wrong number in attrs for " + digit);

i++;

digit = reader.read();//read next symbol

**break**;

}

}

reader.close();

}

**catch**(IOException ex){

System.***out***.println(ex.getMessage());

}

}

**public** Lexer() {

}

}

**Тести**

1) True- тест

PROCEDURE ABADA(A,B,D);

PROCEDURE KENTAVRA;

PROCEDURE MAMBA(A);

BEGIN

KENTAVRA;

MAMBA(7);

RETURN;

END

;

Lexemes:

0001 0001 00400 PROCEDURE

0001 0011 01000 ABADA

0001 0016 00040 (

0001 0017 01001 A

0001 0018 00044 ,

0001 0019 01002 B

0001 0020 00044 ,

0001 0021 01003 D

0001 0022 00041 )

0001 0023 00059 ;

0002 0001 00400 PROCEDURE

0002 0011 01004 KENTAVRA

0002 0019 00059 ;

0003 0001 00400 PROCEDURE

0003 0011 01005 MAMBA

0003 0016 00040 (

0003 0017 01001 A

0003 0018 00041 )

0003 0019 00059 ;

0004 0001 00401 BEGIN

0005 0001 01004 KENTAVRA

0005 0009 00059 ;

0006 0001 01005 MAMBA

0006 0006 00040 (

0006 0007 00500 7

0006 0008 00041 )

0006 0009 00059 ;

0007 0001 00403 RETURN

0007 0007 00059 ;

0008 0001 00402 END

0009 0001 00059 ;

Keys:

00400 PROCEDURE

00401 BEGIN

00402 END

00403 RETURN

Consts:

00500 7

Identifier:

01000 ABADA

01001 A

01002 B

01003 D

01004 KENTAVRA

01005 MAMBA

No errors

2)True- тест

PROCEDURE 123 PRO(\* lalala \*) 266 PTU 123 PRO (

BEGIN END ;

(\*\*)END

(\* \*);

(\* \*( \*\*PRO\*\*\* :(\* \*\*)END;

(\* enter

ф\*)START;

(\* what if \*

)HAHA(\*

lalala\*) END;;;

Lexemes:

0001 0001 00400 PROCEDURE

0001 0011 00500 123

0001 0015 01000 PRO

0001 0031 00501 266

0001 0035 01001 PTU

0001 0039 00500 123

0001 0043 01000 PRO

0001 0047 00040 (

0002 0001 00401 BEGIN

0002 0007 00402 END

0002 0011 00059 ;

0003 0005 00402 END

0004 0006 00059 ;

0005 0026 00402 END

0005 0029 00059 ;

0007 0004 01002 START

0007 0009 00059 ;

0010 0010 00402 END

0010 0013 00059 ;

0010 0014 00059 ;

0010 0015 00059 ;

Keys:

00400 PROCEDURE

00401 BEGIN

00402 END

00403 RETURN

Consts:

00500 123

00501 266

Identifier:

01000 PRO

01001 PTU

01002 START

No errors

3)False- тест

PROCaEDURE ABADA(A,B,D);

PROCEDURE KENTAVRA;

PROCEDURE MAMBA(A);?

BEGIN

KENфTAVRA;

MAMBA(7);

RETURN;

END

;(\* end of file:

Lexemes:

0001 0001 01000 PROC

0001 0006 01001 EDURE

0001 0012 01002 ABADA

0001 0017 00040 (

0001 0018 01003 A

0001 0019 00044 ,

0001 0020 01004 B

0001 0021 00044 ,

0001 0022 01005 D

0001 0023 00041 )

0001 0024 00059 ;

0002 0001 00400 PROCEDURE

0002 0011 01006 KENTAVRA

0002 0019 00059 ;

0003 0001 00400 PROCEDURE

0003 0011 01007 MAMBA

0003 0016 00040 (

0003 0017 01003 A

0003 0018 00041 )

0003 0019 00059 ;

0004 0001 00401 BEGIN

0005 0001 01008 KEN

0005 0005 01009 TAVRA

0005 0010 00059 ;

0006 0001 01007 MAMBA

0006 0006 00040 (

0006 0007 00500 7

0006 0008 00041 )

0006 0009 00059 ;

0007 0001 00403 RETURN

0007 0007 00059 ;

0008 0001 00402 END

0009 0001 00059 ;

Keys:

00400 PROCEDURE

00401 BEGIN

00402 END

00403 RETURN

Consts:

00500 7

Identifier:

01000 PROC

01001 EDURE

01002 ABADA

01003 A

01004 B

01005 D

01006 KENTAVRA

01007 MAMBA

01008 KEN

01009 TAVRA

Errors:

Lexer: Error (line 1, column 5): wrong character [ a ]

Lexer: Error (line 3, column 20): wrong character [ ? ]

Lexer: Error (line 5, column 4): wrong character [ ф ]

Lexer: Error (line 9, column 17): expected '\*)' but found EOF