НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

# Кафедра системного програмування та спеціалізованих комп’ютерних систем

**Розрахунково-графічна робота**

з дисципліни

«Основи проектування трансляторів»

Тема: **«Розробка синтаксичного аналізатора»**

Виконав: студент 3 курсу ФПМ

групи КВ-02

Костюков С. В.

Перевірив: Коваленко О. П.

Київ – 2023

**Постановка задачі**

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

Програма має забезпечувати наступні дії:

* читання рядка лексем та таблиць, згенерованих лексичним аналізатором, який було розроблено в лабораторній роботі «Розробка лексичного аналізатора»;
* синтаксичний аналіз (розбір) програми, поданої рядком лексем ;
* 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.

***Варіант 11***

**<signal-program> --> <program>  
<program> --> PROGRAM <procedure-identifier> ; <block> .  
<block> --> <declarations> BEGIN <statements-list> END  
<declarations> --> <label-declarations>  
<label-declarations> --> LABEL <unsigned-integer> <labels-list>; |  
 <empty>  
<labels-list> --> , <unsigned-integer> <labels-list> |  
 <empty>  
<statements-list> --> <statement> <statements-list> |  
 <empty>  
<statement> --> <unsigned-integer> : <statement> |  
 GOTO <unsigned-integer> ; |  
 LINK <variable-identifier> , <unsigned-integer> ; |  
 IN <unsigned-integer> ; |  
 OUT <unsigned-integer> ;  
<variable-identifier> --> <identifier>  
<procedure-identifier> --> <identifier>  
<identifier> --> <letter><string>  
<string> --> <letter><string> |  
 <digit><string> |  
 <empty>  
<unsigned-integer> --> <digit><digits-string>  
<digits-string> --> <digit><digit-string> |  
 <empty>  
<digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9  
<letter> --> A | B | C | D | ... | Z**

**Лістинг коду програми CА:**

***Main.cpp***

#include "lexical\_analyzer.h"

#include "syntactic\_analyzer.h"

#include "data.h"

/\*

\* Variant 11

<signal-program> --> <program>

<program> --> PROGRAM <procedure-identifier> ; <block> .

<block> --> <declarations> BEGIN <statements-list> END

<declarations> --> <label-declarations>

<label-declarations> --> LABEL <unsigned-integer> <labels-list>; |

<empty>

<labels-list> --> , <unsigned-integer> <labels-list> |

<empty>

<statements-list> --> <statement> <statements-list> |

<empty>

<statement> --> <unsigned-integer> : <statement> |

GOTO <unsigned-integer> ; |

LINK <variable-identifier> , <unsigned-integer> ; |

IN <unsigned-integer> ; |

OUT <unsigned-integer> ;

<variable-identifier> --> <identifier>

<procedure-identifier> --> <identifier>

<identifier> --> <letter><string>

<string> --> <letter><string> |

<digit><string> |

<empty>

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

<digits-string> --> <digit><digit-string> |

<empty>

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

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

\*/

/\*

',', ':', ';', '.'

44 58 59 46

\*/

**int** main() {

Data data;

data.root = **new** Tree();

data.curr = **new** Tree();

//if(!file\_parsing("Prog1.txt", data)) return 0;

//if(!file\_parsing("Prog2.txt", data)) return 0;

**if** (!file\_parsing("FalseTest1.txt", data)) **return** 0;

//if(!file\_parsing("FalseTest2.txt", data)) return 0;

signal\_program(data);

**return** 0;

}

***Syntactic\_analyzer.h***

#ifndef MAIN\_CPP\_SYNTACTIC\_ANALYZER\_H

#define MAIN\_CPP\_SYNTACTIC\_ANALYZER\_H

#include "data.h"

/\*

\* This file contains realization of syntactic analyzer for lex showed in main.cpp

\*/

**void** signal\_program(Data &);

#endif //MAIN\_CPP\_SYNTACTIC\_ANALYZER\_H

***Syntactic\_analyzer.cpp***

#include "syntactic\_analyzer.h"

#include "data.h"

#include <iomanip>

**using** std::cout;

**using** std::endl;

/\*

<signal-program> --> <program>

<program> --> PROGRAM <procedure-identifier> ; <block> .

<block> --> <declarations> BEGIN <statements-list> END

<declarations> --> <label-declarations>

<label-declarations> --> LABEL <unsigned-integer> <labels-list>; |

<empty>

<labels-list> --> , <unsigned-integer> <labels-list> |

<empty>

<statements-list> --> <statement> <statements-list> |

<empty>

<statement> --> <unsigned-integer> : <statement> |

GOTO <unsigned-integer> ; |

LINK <variable-identifier> , <unsigned-integer> ; |

IN <unsigned-integer> ; |

OUT <unsigned-integer> ;

<variable-identifier> --> <identifier>

<procedure-identifier> --> <identifier>

\*/

**enum** Errors {

INCORRECT\_SYMBOL = 0, // Incorrect symbol : expected 'symbol'

INCORRECT\_KEYWORD, // Keyword 'word' expected

INCORRECT\_VALUE, // Expected <unsigned\_integer>

INCORRECT\_IDENTIFIER, // No such identifier in tables

};

**bool** program(**auto** &, Data &);

**bool** block(**auto** &, Data &);

**bool** declarations(**auto** &, Data &);

**bool** label\_declarations(**auto** &, Data &);

**bool** labels\_list(**auto** &, Data &);

**bool** statement\_list(**auto** &, Data &);

**bool** statement(**auto** &, Data &);

**bool** variable\_identifier(**auto** &, Data &);

**bool** procedure\_identifier(**auto** &, Data &);

**bool** identifier(**auto** &, Data &);

**bool** unsigned\_integer(**auto** &, Data &);

**void** Error\_message(Errors, **auto** &, **const** std::string &);

**void** signal\_program(Data &data) {

**auto** it = data.lexemes.begin();

data.root->name = "<signal\_program>";

**if** (program(it, data)) {

data.root->ShowInfo();

}

}

**bool** program(**auto** &it, Data &data) {

data.curr = **new** Tree(data.root, "<program>");

Tree \*elem = **nullptr**;

**if** ((\*it).code == 301) { // "PROGRAM"

elem = **new** Tree(data.curr, (\*it).code, (\*it).lexem);

**if** (!procedure\_identifier(++it, data)) { **return** **false**; } // <procedure\_identifier>

**if** ((\*it).code == 59) { // ";"

elem = **new** Tree(data.curr, (\*it).code, (\*it).lexem);

**if** (!block(++it, data)) { **return** **false**; } // <block>

**if** ((\*it).code == 46) { // "."

elem = **new** Tree(data.curr, (\*it).code, (\*it).lexem);

} **else** {

Error\_message(INCORRECT\_SYMBOL, it, ".");

**return** **false**;

}

} **else** {

Error\_message(INCORRECT\_SYMBOL, it, ";");

**return** **false**;

}

} **else** {

Error\_message(INCORRECT\_KEYWORD, it, "'PROGRAM'");

**return** **false**;

}

data.curr = data.curr->parent;

**return** **true**;

}

**bool** block(**auto** &it, Data &data) {

data.curr = **new** Tree(data.curr, "<block>");

**if** (!declarations(it, data)) { **return** **false**; }

Tree \*elem = **nullptr**;

**if** ((\*it).code == 302) { // "BEGIN"

elem = **new** Tree(data.curr, (\*it).code, (\*it).lexem);

it++;

**if** (!statement\_list(it, data)) { **return** **false**; }

**if** ((\*it).code == 303) { // "END"

elem = **new** Tree(data.curr, (\*it).code, (\*it).lexem);

it++;

} **else** {

Error\_message(INCORRECT\_KEYWORD, it, "'END'");

**return** **false**;

}

} **else** {

Error\_message(INCORRECT\_KEYWORD, it, "'BEGIN'");

**return** **false**;

}

data.curr = data.curr->parent;

**return** **true**;

}

**bool** declarations(**auto** &it, Data &data) {

data.curr = **new** Tree(data.curr, "<declarations>");

**if** (!label\_declarations(it, data)) { **return** **false**; }

data.curr = data.curr->parent;

**return** **true**;

}

**bool** label\_declarations(**auto** &it, Data &data) {

data.curr = **new** Tree(data.curr, "<label-declaration>");

Tree \*elem = **nullptr**;

**if** ((\*it).code != 302 && (\*it).code != 308) {

Error\_message(INCORRECT\_KEYWORD, it, "'LABEL'");

**return** **false**;

}

**bool** is\_any\_label = **false**;

**while** ((\*it).code == 308) { // "LABEL"

is\_any\_label = **true**;

elem = **new** Tree(data.curr, (\*it).code, (\*it).lexem);

it++;

**if** (!unsigned\_integer(it, data)) {

Error\_message(INCORRECT\_VALUE, it, "");

**return** **false**;

}

**if** (!labels\_list(it, data)) { **return** **false**; }

**if** ((\*it).code == 59) { // ";"

elem = **new** Tree(data.curr, (\*it).code, (\*it).lexem);

} **else** {

Error\_message(INCORRECT\_SYMBOL, it, ";");

**return** **false**;

}

it++;

}

**if** (!is\_any\_label) {

elem = **new** Tree(data.curr, "<empty>");

}

data.curr = data.curr->parent;

**return** **true**;

}

**bool** labels\_list(**auto** &it, Data &data) {

data.curr = **new** Tree(data.curr, "<labels-list>");

Tree \*elem;

**while** ((\*it).code == 44) { // ","

elem = **new** Tree(data.curr, (\*it).code, (\*it).lexem);

it++;

**if** (!unsigned\_integer(it, data)) {

Error\_message(INCORRECT\_VALUE, it, "");

**return** **false**;

}

}

elem = **new** Tree(data.curr, "<empty>");

data.curr = data.curr->parent;

**return** **true**;

}

**bool** statement\_list(**auto** &it, Data &data) {

data.curr = **new** Tree(data.curr, "<statement-list>");

**while** ((\*it).code != 303) { // END

**if** (!statement(it, data)) {

**return** **false**;

}

}

Tree \*elem = **new** Tree(data.curr, "<empty>");

data.curr = data.curr->parent;

**return** **true**;

}

**bool** statement(**auto** &it, Data &data) {

data.curr = **new** Tree(data.curr, "<statement>");

Tree \*elem = **nullptr**;

**if** (unsigned\_integer(it, data)) {

**if** ((\*it).code == 58) { // ":"

elem = **new** Tree(data.curr, (\*it).code, (\*it).lexem);

it++;

} **else** {

Error\_message(INCORRECT\_SYMBOL, it, ":");

**return** **false**;

}

}

**if** ((\*it).code == 304 || (\*it).code == 306 || (\*it).code == 307) { // "GOTO", "IN", "OUT"

elem = **new** Tree(data.curr, (\*it).code, (\*it).lexem);

it++;

**if** (!unsigned\_integer(it, data)) {

Error\_message(INCORRECT\_VALUE, it, "");

**return** **false**;

}

**if** ((\*it).code == 59) { // ";"

elem = **new** Tree(data.curr, (\*it).code, (\*it).lexem);

it++;

} **else** {

Error\_message(INCORRECT\_SYMBOL, it, ";");

**return** **false**;

}

} **else** **if** ((\*it).code == 305) { // "LINK"

elem = **new** Tree(data.curr, (\*it).code, (\*it).lexem);

it++;

**if** (!variable\_identifier(it, data)) {

**return** **false**;

}

**if** ((\*it).code == 44) { // ","

elem = **new** Tree(data.curr, (\*it).code, (\*it).lexem);

it++;

**if** (!unsigned\_integer(it, data)) {

Error\_message(INCORRECT\_VALUE, it, "");

**return** **false**;

}

**if** ((\*it).code == 59) { // ";"

elem = **new** Tree(data.curr, (\*it).code, (\*it).lexem);

it++;

} **else** {

Error\_message(INCORRECT\_SYMBOL, it, ";");

**return** **false**;

}

} **else** {

Error\_message(INCORRECT\_SYMBOL, it, ",");

**return** **false**;

}

} **else** {

Error\_message(INCORRECT\_KEYWORD, it, "one of: 'LINK', 'GOTO', 'IN', 'OUT', 'END'");

**return** **false**;

}

data.curr = data.curr->parent;

**return** **true**;

}

**bool** variable\_identifier(**auto** &it, Data &data) {

data.curr = **new** Tree(data.curr, "<variable-identifier>");

**if** (!identifier(it, data)) {

**return** **false**;

}

data.curr = data.curr->parent;

**return** **true**;

}

**bool** procedure\_identifier(**auto** &it, Data &data) {

data.curr = **new** Tree(data.curr, "<procedure-identifier>");

**if** (!identifier(it, data)) {

**return** **false**;

}

data.curr = data.curr->parent;

**return** **true**;

}

**bool** identifier(**auto** &it, Data &data) {

data.curr = **new** Tree(data.curr, "<identifier>");

Tree \*elem = **nullptr**;

**if** (data.identifiers.contains((\*it).lexem)) {

elem = **new** Tree(data.curr, (\*it).code, (\*it).lexem);

it++;

} **else** {

Error\_message(INCORRECT\_IDENTIFIER, it, "");

**return** **false**;

}

data.curr = data.curr->parent;

**return** **true**;

}

**bool** unsigned\_integer(**auto** &it, Data &data) {

**if** ((\*it).code == 304 || (\*it).code == 305 || (\*it).code == 306 || (\*it).code == 307) {

**return** **false**;

}

data.curr = **new** Tree(data.curr, "<unsigned-integer>");

Tree \*elem = **nullptr**;

**if** (data.labels.contains((\*it).lexem)) {

elem = **new** Tree(data.curr, (\*it).code, (\*it).lexem);

it++;

} **else** {

data.curr = data.curr->parent;

**return** **false**;

}

data.curr = data.curr->parent;

**return** **true**;

}

**void** Error\_message(Errors error, **auto** &it, **const** std::string &description) {

cout << "Parser: Error (line " << (\*it).line << ", column " << (\*it).column << "): ";

**if** (error == INCORRECT\_SYMBOL) {

cout << "Incorrect symbol '" << (\*it).lexem << "', " << "expected '" << description << "'" << endl;

} **else** **if** (error == INCORRECT\_KEYWORD) {

cout << "Incorrect KeyWord '" << (\*it).lexem << "', " << "expected " << description << endl;

} **else** **if** (error == INCORRECT\_IDENTIFIER) {

cout << "Incorrect identifier '" << (\*it).lexem << "'";

} **else** **if** (error == INCORRECT\_VALUE) {

cout << "Incorrect value '" << (\*it).lexem << "'";

} **else** {

cout << "Other Error" << endl;

}

}

**Тестування програми:**

*True-тестування:*

***Текст програми:***

Prog1.txt

PROGRAM PROG;  
(\*jqwj\*)  
LABEL 12, 13;  
(\*  
\*\*\*\*ewfwef\*\*\*\*  
  
\*)  
  
BEGIN  
12 : GOTO 13 ;  
 LINK VERIDENT, 13 ;  
13 : IN 12 ;  
 OUT 12 ;  
END.

***Отриманий результат:***

<signal\_program>

<program>

PROGRAM

<procedure-identifier>

<identifier>

PROG

;

<block>

<declarations>

<label-declaration>

LABEL

<unsigned-integer>

12

<labels-list>

,

<unsigned-integer>

13

<empty>

;

BEGIN

<statement-list>

<statement>

<unsigned-integer>

12

:

GOTO

<unsigned-integer>

13

;

<statement>

LINK

<variable-identifier>

<identifier>

VERIDENT

,

<unsigned-integer>

13

;

<statement>

<unsigned-integer>

13

:

IN

<unsigned-integer>

12

;

<statement>

OUT

<unsigned-integer>

12

;

<empty>

END

.

*False-тестування:*

***Текст програми:***

FalseTest1.txt

PROGRAM PROG;  
(\*jqwj\*)  
LABEL 12, 13;  
  
BEIN  
12 : GOTO 13 ;  
 LINK VERIDENT, 13 ;  
13 : IN 12 ;  
 OUT 12 ;  
END.  
(\*knf\*)

***Отриманий результат:***

Parser: Error (line 5, column 1): Incorrect KeyWord 'BEIN', expected 'BEGIN'

***Текст програми:***

FalseTest2.txt

PROGRAM PROG;  
(\*jqwj\*)  
LABEL 12, 13;  
  
BEIN  
12 : GOTO 13 ;  
 LINK VERIDENT, 13 ;  
13 : IN 12 ;  
 OUT 12 ;  
END,  
(\*knf\*)

***Отриманий результат:***

Parser: Error (line 10, column 4): Incorrect symbol ',', expected '.'