Міністерство освіти і науки, молоді та спорту України
Національний Технічний Університет України
“Київський Політехнічний Інститут ім.. Сікорського”
Факультет прикладної математики
Кафедра СПіСКС

Розрахунково-графічна робота

з дисципліни
“ІПЗ. Основи проектування трансляторів”

Тема: “Розробка синтаксичного аналізатора”

Виконав:
Студент групи КВ-82
Іваненко Олександр
Варіант 10

Київ 2021

Постановка задачі

1. Розробити програму синтаксичного аналізатора (СА) для підмножини мови програмування SIGNAL згідно граматики за варіантом.

2. Програма має забезпечувати наступне:

• читання рядка лексем та таблиць, згенерованих лексичним аналізатором, який було розроблено в лабораторній роботі «Розробка лексичного аналізатора»;

• синтаксичний аналіз (розбір) програми, поданої рядком лексем (алгоритм синтаксичного аналізатора вибирається за 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.

3. Для програмування може бути використана довільна алгоритмічна мова програмування високого рівня. Якщо обрана мова програмування має конструкції або бібліотеки для роботи з регулярними виразами, то використання цих конструкцій та/або бібліотек строго заборонено.

4. Входом синтаксичного аналізатора 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).

5. Виходом синтаксичного аналізатора має бути наступне:

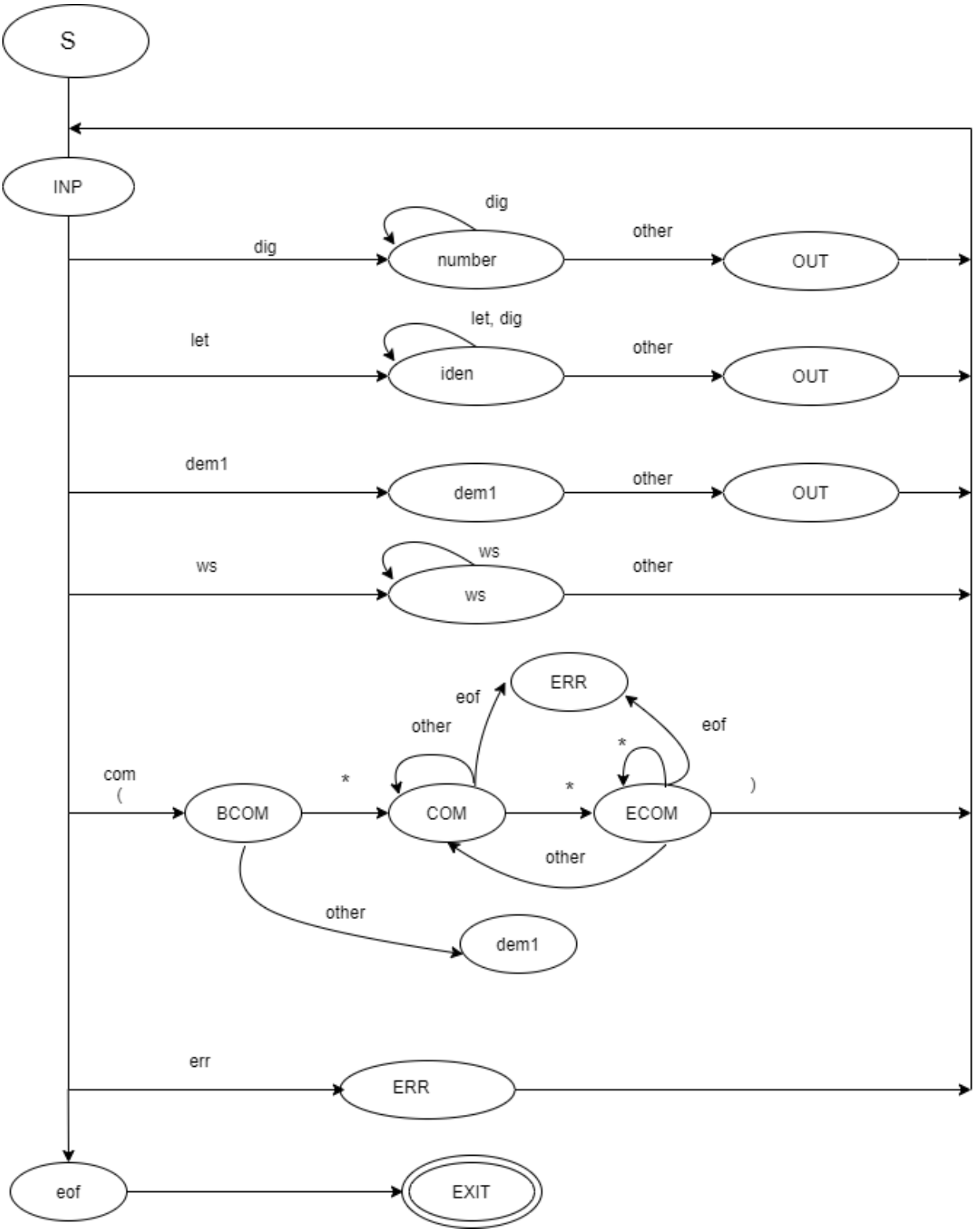
• дерево розбору вхідної програми;

• таблиці ідентифікаторів та різних констант з повною інформацією, необхідною для генерування коду;

• лістинг вхідної програми з повідомленнями про лексичні та синтаксичні помилки

Варіант 10

1. <signal-program> --> <program>
2. <program> --> PROGRAM <procedure-identifier> ; <block> ;
3. <block> --> <declarations> BEGIN <statements-list> END
4. <statements-list> --> <empty>
5. <declarations> --> <procedure-declarations>
6. <procedure-declarations> --> <procedure> <procedure-declarations> | <empty>
7. <procedure> --> PROCEDURE <procedure-identifier><parameters-list> ;
8. <parameters-list> --> (<declarations-list>) | <empty>
9. <declarations-list> --> <declaration> <declarations-list> | <empty>
10. <declaration> --><variable-identifier><identifiers-list>:<attribute><attributes-list> ;
11. <identifiers-list> --> , <variable-identifier> <identifiers-list> | <empty>
12. <attributes-list> --> <attribute> <attributes-list> | <empty>
13. <attribute> --> SIGNAL | COMPLEX | INTEGER | FLOAT | BLOCKFLOAT | EXT
14. <variable-identifier> --> <identifier>
15. <procedure-identifier> --> <identifier>
16. <identifier> --> <letter><string>
17. <string> --> <letter><string> | <digit><string> | <empty>
18. <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
19. <letter> --> A | B | C | D | ... | Z


S
INP
dig
number
other
OUT
dig
let, dig
let
iden
other
OUT
dem1
dem1
other
OUT
ws
ws
ws
other
ERR
eof
other
*
eof
com
(
BCOM
*
COM
*
ECOM
)
other
other
dem1
err
ERR
eof
EXIT

Код програми

```
#ifndef LAB1_PARSER_H
#define LAB1_PARSER_H

#include "lexer.h"
#include "tree.h"

class Parser{
private:
    int curr_line = 1;
    int curr_col = 1;
public:
    Tree syn_tree;
    vector<Err> errors;
    vector<Lexeme> all_lexemes;
    vector <pair<string, int>> all_numbers;
    vector <pair<string, int>> all_ids;
    Parser();
    Parser(Lexer lex);
    void print_tree(TreeNode* parent, int tab);
    void startParser();
    void signalProgram();
    void program(TreeNode* parent);
    void block(TreeNode* parent);
    void statementList(TreeNode* parent);
    void declarations(TreeNode* parent);
    void procedureDeclarations(TreeNode* parent);
    void procedure(TreeNode* parent);
    void parametersList(TreeNode* parent);
    void declarationsList(TreeNode* parent);
    void declaration(TreeNode* parent);
    void identifiersList(TreeNode* parent);
    void attributesList(TreeNode* parent);
    void attribute(TreeNode* parent);
    void variableIdentifier(TreeNode* parent);
    void procedureIdentifier(TreeNode* parent);
    void identifier(TreeNode* parent);
    void empty(TreeNode* parent);

    void print_errors();


    void add_to_tree(TreeNode *parent, TreeNode* node);

    void fill_generated(TreeNode *parent, int tab, string test_folder);

    void print_errors(string test_folder);
};

#endif //LAB1_PARSER_H
```

```
#include "parser.h"

Parser::Parser() {
    syn_tree.root = new TreeNode();
    curr_line = 1;
}

Parser::Parser(Lexer lex) {
    all_lexemes = (lex.get_all_lexemes());
    syn_tree.root = new TreeNode();
    curr_line = 1;
    all_ids = lex.all_ids;
    all_numbers = lex.all_numbers;
}

void Parser::fill_generated(TreeNode* parent, int tab, string test_folder) {
    ofstream out;
    out.open(test_folder + "generated.txt", ios::app);

    for (int i = 0; i < tab; i++) {
        out << "....";
        cout << "....";
    }
    out << parent->rule << " " << parent->token << " " << endl;
    cout << parent->rule << " " << parent->token << " " << endl;
    int i = 0;
    while (i < parent->children.size()) {
        fill_generated(parent->children[i], tab + 1, test_folder);
        i++;
    }

}

void Parser::add_to_tree(TreeNode* parent, TreeNode* node){
    node->token = to_string(all_lexemes[0].get_code()) + " " +
all_lexemes[0].get_name();
    auto del =  all_lexemes.erase(all_lexemes.begin());
    syn_tree.insert(parent, node);
    if (!all_lexemes.empty()) {
        curr_line = all_lexemes[0].get_row();
        curr_col = all_lexemes[0].get_column();
    }
}

void Parser::startParser() {
    signalProgram();
}

void Parser::signalProgram() {
    syn_tree.root->rule = "<signal-program>";
    program(syn_tree.root);
}

void Parser::program(TreeNode* parent) {
    TreeNode *node = new TreeNode();
    node->rule = "<program>";
    syn_tree.insert(parent, node);
```

```
    if((!all_lexemes.empty()) && (all_lexemes[0].get_code() == 401)){
        TreeNode *new_node = new TreeNode();
        add_to_tree(node, new_node);
    }
    else if(errors.empty()){
        errors.emplace_back(curr_col, curr_line, "(parser) absent
'PROGRAM'");
        return;
    }

    procedureIdentifier(node);

    if((!all_lexemes.empty()) && (all_lexemes[0].get_code() == 59)){
        TreeNode *new_node = new TreeNode();
        add_to_tree(node, new_node);
    }
    else if(errors.empty()) {
        errors.emplace_back(curr_col, curr_line, "(parser) absent ';'");
        return;
    }

    block(node);

    if((!all_lexemes.empty()) && (all_lexemes[0].get_code() == 59)){
        TreeNode *new_node = new TreeNode();
        add_to_tree(node, new_node);
    }
    else if(errors.empty()){
        errors.emplace_back(curr_col, curr_line, "(parser) absent ';'");
        return;
    }
}

void Parser::block(TreeNode* parent) {
    if (!errors.empty()) return;
    TreeNode *node = new TreeNode();
    node->rule = "<block>";
    syn_tree.insert(parent, node);
    declarations(node);
    if((!all_lexemes.empty()) && (all_lexemes[0].get_code() == 402)){
        TreeNode *new_node = new TreeNode();
        add_to_tree(node, new_node);
        statementList(node);
        if((!all_lexemes.empty()) && (all_lexemes[0].get_code() == 403)) {
            TreeNode *new_node = new TreeNode();
            add_to_tree(node, new_node);
        }
        else if(errors.empty()){
            errors.emplace_back(curr_col, curr_line, "(parser) absent
'end'");
            return;
        }


    }
    else if(errors.empty()){
        errors.emplace_back(curr_col, curr_line, "(parser) absent 'begin'");
```

```
        return;
    }

}

void Parser::statementList(TreeNode* parent) {
    if (!errors.empty()) return;
    TreeNode *node = new TreeNode();
    node->rule = "<statement-list>";
    syn_tree.insert(parent, node);
    empty(node);
}

void Parser::declarations(TreeNode* parent) {
    if (!errors.empty()) return;
    TreeNode *node = new TreeNode();
    node->rule = "<declarations>";
    syn_tree.insert(parent, node);
    procedureDeclarations(node);
}

void Parser::procedureDeclarations(TreeNode* parent) {
    if (!errors.empty()) return;
    TreeNode *node = new TreeNode();
    node->rule = "<procedure-declarations>";
    syn_tree.insert(parent, node);
    if((!all_lexemes.empty()) && (all_lexemes[0].get_code() == 404)){
        procedure(node);
        procedureDeclarations(node);
    }
    else{
        empty(node);
    }
}

void Parser::procedure(TreeNode* parent) {
    if (!errors.empty()) return;
    TreeNode *node = new TreeNode();
    node->rule = "<procedure>";
    syn_tree.insert(parent, node);
    if((!all_lexemes.empty()) && (all_lexemes[0].get_code() == 404)){
        TreeNode *new_node = new TreeNode();
        add_to_tree(node, new_node);
        procedureIdentifier(node);
        parametersList(node);
        if((!all_lexemes.empty()) && (all_lexemes[0].get_code() == 59)){
            TreeNode *new_node = new TreeNode();
            add_to_tree(node, new_node);
        }
        else if(errors.empty()){
            errors.emplace_back(curr_col, curr_line, "(parser) absent ';'");
            return;
        }
    }
    else if(errors.empty()) {
        errors.emplace_back(curr_col, curr_line, "(parser) absent
'procedure'");
```

```
        return;
    }
}

void Parser::parametersList(TreeNode* parent) {
    if (!errors.empty()) return;
    TreeNode *node = new TreeNode();
    node->rule = "<parameters-list>";
    syn_tree.insert(parent, node);
    if((!all_lexemes.empty()) && (all_lexemes[0].get_code() == 40)){
        TreeNode *new_node = new TreeNode();
        add_to_tree(node, new_node);
        declarationsList(node);
        if((!all_lexemes.empty()) && (all_lexemes[0].get_code() == 41)){
            TreeNode *new_node = new TreeNode();
            add_to_tree(node, new_node);
        }
        else if(errors.empty()){
            errors.emplace_back(curr_col, curr_line, "(parser) absent ')' in
parameters");
            return;
        }
    }
    else if(errors.empty()){
        empty(node);
    }
}

void Parser::declarationsList(TreeNode* parent) {
    if (!errors.empty()) return;
    TreeNode *node = new TreeNode();
    node->rule = "<declarations-list>";
    syn_tree.insert(parent, node);
    if((!all_lexemes.empty()) && (all_lexemes[0].get_code() >= 1000) &&
(all_lexemes[0].get_code() < 2000)){
        declaration(node);
        declarationsList(node);
    }else{
        empty(node);
    }

}

void Parser::declaration(TreeNode* parent) {
    if (!errors.empty()) return;
    TreeNode *node = new TreeNode();
    node->rule = "<declaration>";
    syn_tree.insert(parent, node);
    variableIdentifier(node);
    identifiersList(node);
    if ((!all_lexemes.empty()) && (all_lexemes[0].get_code() == 58)) {
        TreeNode *new_node = new TreeNode();
        add_to_tree(node, new_node);
    } else if(errors.empty()){
        errors.emplace_back(curr_col, curr_line, "(parser) absent ':' in
declaration");
        return;
```

```
    }
    attribute(node);
    attributesList(node);
    if ((!all_lexemes.empty()) && (all_lexemes[0].get_code() == 59)) {
        TreeNode *new_node = new TreeNode();
        add_to_tree(node, new_node);
    } else if(errors.empty()){
        errors.emplace_back(curr_col, curr_line, "(parser) absent ';' in
declaration");
        return;
    }
}


void Parser::identifiersList(TreeNode* parent) {
    if (!errors.empty()) return;
    TreeNode *node = new TreeNode();
    node->rule = "<identifiers-list>";
    syn_tree.insert(parent, node);
    if ((!all_lexemes.empty()) && (all_lexemes[0].get_code() == 44)) {
        TreeNode *new_node = new TreeNode();
        add_to_tree(node, new_node);
        variableIdentifier(node);
        identifiersList(node);
    } else empty(node);
}

void Parser::attributesList(TreeNode* parent) {
    if (!errors.empty()) return;
    TreeNode *node = new TreeNode();
    node->rule = "<attributes-list>";
    syn_tree.insert(parent, node);
    if ((!all_lexemes.empty()) && (all_lexemes[0].get_code() > 404) &&
(all_lexemes[0].get_code() < 411)){
        attribute(node);
        attributesList(node);
    }
    else empty(node);
}

void Parser::attribute(TreeNode* parent) {
    if (!errors.empty()) return;
    TreeNode *node = new TreeNode();
    node->rule = "<attribute>";
    syn_tree.insert(parent, node);
    if ((!all_lexemes.empty()) && (all_lexemes[0].get_code() > 404) &&
(all_lexemes[0].get_code() < 411)) {
        TreeNode *new_node = new TreeNode();
        add_to_tree(node, new_node);
    } else if(errors.empty()){
        errors.emplace_back(curr_col, curr_line, "(parser) bad attribute" +
all_lexemes[0].get_name());
        return;
    }
}

void Parser::variableIdentifier(TreeNode* parent) {
```

```
    if (!errors.empty()) return;
    TreeNode *node = new TreeNode();
    node->rule = "<variable-identifier>";
    syn_tree.insert(parent, node);
    identifier(node);
}

void Parser::procedureIdentifier(TreeNode* parent) {
    if(!errors.empty()) return;
    TreeNode* node = new TreeNode();
    node->rule = "<procedure-identifier>";
    syn_tree.insert(parent, node);
    identifier(node);
}

void Parser::identifier(TreeNode* parent) {
    TreeNode* node = new TreeNode();
    node->rule = "<identifier>";
    if((!all_lexemes.empty()) && (all_lexemes[0].get_code() >= 1000)){
        add_to_tree(parent, node);
    }
    else if(errors.empty()){
        errors.emplace_back(curr_col, curr_line, "(parser) absent
identifier");
        return;
    }
}

void Parser::empty(TreeNode* parent) {
    if (!errors.empty()) return;
    TreeNode *node = new TreeNode();
    node->rule = "<empty>";
    syn_tree.insert(parent, node);
}

void Parser::print_errors(string test_folder) {
    ofstream out;
    out.open(test_folder + "generated.txt", ios::app);

    for(Err error : errors){
        out << error.get_error();
        cout << error.get_error();
    }
}
```

Тести

Приклади без помилок

```
program qw;
procedure check1(float1: float; (**));(**)
procedure check (float1,float1 : float;);
begin
(**8)*)
end ;
(**)

<signal-program>
....<program>
........ 401 program
........<procedure-identifier>
............<identifier> 1000 qw
........ 59 ;
........<block>
............<declarations>
..............<procedure-declarations>
....................<procedure>
.......................... 404 procedure
.......................<procedure-identifier>
.........................<identifier> 1001 check1
.......................<parameters-list>
........................... 40 (
.........................<declarations-list>
.............................<declaration>
...................................<variable-identifier>
.....................................<identifier> 1002 float1
...................................<identifiers-list>
.....................................<empty>
.................................... 58 :
...................................<attribute>
..................................... 407 float
...................................<attributes-list>
.....................................<empty>
.................................... 59 ;
.............................<declarations-list>
...................................<empty>
........................... 41 )
......................... 59 ;
...................<procedure-declarations>
.......................<procedure>
........................... 404 procedure
.........................<procedure-identifier>
.............................<identifier> 1003 check
```

```
.........................<parameters-list>
.............................. 40 (
.............................<declarations-list>
.................................<declaration>
.......................................<variable-identifier>
.........................................<identifier> 1002
float1
.....................................<identifiers-list>
.......................................... 44 ,
.........................................<variable-identifier>
...........................................<identifier> 1002
float1
.........................................<identifiers-list>
...............................................<empty>
..................................... 58 :
.....................................<attribute>
........................................ 407 float
.....................................<attributes-list>
.........................................<empty>
..................................... 59 ;
.................................<declarations-list>
.....................................<empty>
.............................. 41 )
........................... 59 ;
.....................<procedure-declarations>
.........................<empty>
............ 402 begin
............<statement-list>
..............<empty>
............ 403 end
........ 59 ;
All ids:
   1   1   401 program
   1   9  1000 qw
   1  11    59 ;
   2   1   404 procedure
   2  11  1001 check1
   2  17    40 (
   2  18  1002 float1
   2  24    58 :
   2  26   407 float
   2  31    59 ;
   2  37    41 )
   2  38    59 ;
   3   1   404 procedure
   3  11  1003 check
   3  17    40 (
   3  18  1002 float1
   3  24    44 ,
```

```
   3  25  1002 float1
   3  32    58 :
   3  34   407 float
   3  39    59 ;
   3  40    41 )
   3  41    59 ;
   4   1   402 begin
   6   1   403 end
   6   5    59 ;


All ids:
        qw 1000
    check1 1001
    float1 1002
     check 1003


All separators:
    ; 59
    : 58
    ) 41
    , 44
```

```
program qw;
procedure check1();(**)
procedure check2;
(*/1
*/
/*)
begin
end ;
(**)


<signal-program>
....<program>
........ 401 program
........<procedure-identifier>
............<identifier> 1000 qw
........ 59 ;
........<block>
............<declarations>
..............<procedure-declarations>
....................<procedure>
.......................... 404 procedure
.......................<procedure-identifier>
.............................<identifier> 1001 check1
.......................<parameters-list>
............................. 40 (
............................<declarations-list>
..............................<empty>
............................. 41 )
.......................... 59 ;
....................<procedure-declarations>
.......................<procedure>
............................. 404 procedure
............................<procedure-identifier>
..............................<identifier> 1002 check2
............................<parameters-list>
..............................<empty>
............................. 59 ;
.......................<procedure-declarations>
............................<empty>
............ 402 begin
............<statement-list>
..............<empty>
............ 403 end
........ 59 ;
```

```
All ids:
   1   1   401 program
   1   9  1000 qw
   1  11    59 ;
   2   1   404 procedure
   2  11  1001 check1
   2  17    40 (
   2  18    41 )
   2  19    59 ;
   3   1   404 procedure
   3  11  1002 check2
   3  17    59 ;
   7   1   402 begin
   8   1   403 end
   8   5    59 ;


All ids:
        qw 1000
    check1 1001
    check2 1002


All separators:
    ; 59
    ) 41
```

Приклади з помилками

```
program qw;
procedure check1(fie)(**)
begin
end ;
(**)
```

```
<signal-program>
....<program>
........ 401 program
........<procedure-identifier>
............<identifier> 1000 qw
........ 59 ;
........<block>
............<declarations>
..............<procedure-declarations>
..................<procedure>
.......................... 404 procedure
.......................<procedure-identifier>
...........................<identifier> 1001 check1
.......................<parameters-list>
........................... 40 (
...........................<declarations-list>
.............................<declaration>
...................................<variable-identifier>
.....................................<identifier> 1002 fie
...................................<identifiers-list>
.....................................<empty>
```

```
All ids:
   1   1   401 program
   1   9  1000 qw
   1  11    59 ;
   2   1   404 procedure
   2  11  1001 check1
   2  17    40 (
   2  18  1002 fie
   2  21    41 )
   3   1   402 begin
   4   1   403 end
   4   5    59 ;
```

```
All ids:
        qw 1000
    check1 1001
       fie 1002
```

```
All separators:
    ; 59
    ) 41


All errors:
Error: (parser) absent ':' in declaration [col:21][row:2]
```

```
program qw;
procedure check1(fie : float;)(**)
begin
end ; ?
(**)

All ids:
   1   1   401 program
   1   9  1000 qw
   1  11    59 ;
   2   1   404 procedure
   2  11  1001 check1
   2  17    40 (
   2  18  1002 fie
   2  22    58 :
   2  24   407 float
   2  29    59 ;
   2  30    41 )
   3   1   402 begin
   4   1   403 end
   4   5    59 ;


All ids:
        qw 1000
    check1 1001
       fie 1002


All separators:
    ; 59
    : 58
    ) 41


All errors:
Error: (lexer) Unaccepted token '?' [col:7][row:4]
```

```
program qw;
procedure check1(fie : float;);(**)
begin
 ;
(**)

<signal-program>
....<program>
........ 401 program
........<procedure-identifier>
............<identifier> 1000 qw
........ 59 ;
........<block>
............<declarations>
..............<procedure-declarations>
....................<procedure>
.......................... 404 procedure
.......................<procedure-identifier>
.........................<identifier> 1001 check1
.......................<parameters-list>
.......................... 40 (
.........................<declarations-list>
.............................<declaration>
...................................<variable-identifier>
.....................................<identifier> 1002 fie
...................................<identifiers-list>
.....................................<empty>
.................................... 58 :
...................................<attribute>
..................................... 407 float
...................................<attributes-list>
.....................................<empty>
.................................... 59 ;
.............................<declarations-list>
...................................<empty>
.......................... 41 )
......................... 59 ;
...................<procedure-declarations>
.......................<empty>
............ 402 begin
............<statement-list>
..............<empty>


All ids:
   1   1   401 program
   1   9  1000 qw
   1  11    59 ;
   2   1   404 procedure
```

```
   2  11  1001 check1
   2  17    40 (
   2  18  1002 fie
   2  22    58 :
   2  24   407 float
   2  29    59 ;
   2  30    41 )
   2  31    59 ;
   3   1   402 begin
   4   2    59 ;


All ids:
        qw 1000
    check1 1001
       fie 1002


All separators:
    ; 59
    : 58
    ) 41


All errors:
Error: (parser) absent 'end' [col:2][row:4]
```