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

Лабораторна робота № 1

з дисципліни
“ІПЗ. Основи проектування трансляторів”

Тема: “Розробка лексичного аналізатора (ЛА)”

Виконав:
Студент групи КВ-82
Іваненко Олександр
Варіант 10

Київ 2021

Постановка задачі

Розробити програму лексичного аналізатора (ЛА) для підмножини мови програмування SIGNAL.

Програма має забезпечувати наступне (якщо це передбачається граматикою варіанту):

- згортання ідентифікаторів;
- згортання ключових слів;
- згортання цілих десяткових констант;
- згортання дійсних десяткових констант;
- згортання строкових констант, якщо вони визначені в заданій мові;

Також у всіх варіантах необхідно забезпечити:

- видалення коментарів, заданих у вигляді (*<текст коментарю>*)

Для кодування лексем необхідно використовувати числові діапазони.

Входом ЛА має бути наступне:

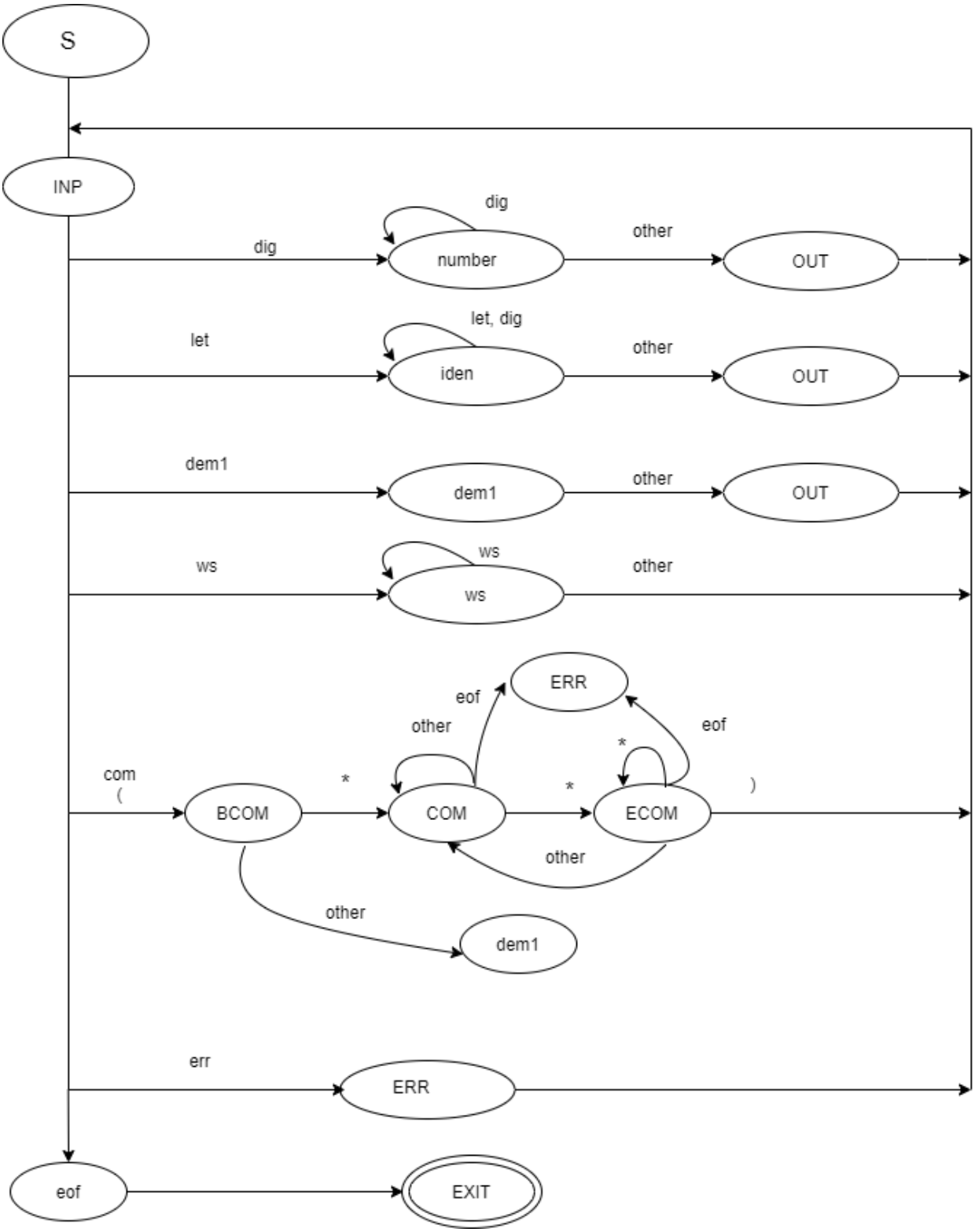
- вихідна програма, написана підмножиною мови SIGNAL відповідно до варіанту;
- таблиця кодів ASCII з атрибутами для визначення токенів;
- таблиця багатосимвольних роздільників;
- таблиця ідентифікаторів, в яку попередньо занесені ключові слова з атрибутом ключового слова;

Вихід ЛА має бути наступним:

- закодований рядок лексем;
- таблиці ідентифікаторів, числових, символьних та рядкових констант, сформовані для конкретної програми;

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

Main.cpp

```
#include "lexer.h"

#include <iostream>

int main(int argc, char* argv[]){
    string test_folder = argv[1];
    Lexer lex;
    lex.parse_file(test_folder);
    lex.print_all_lexemes();
    lex.fill_generated(test_folder);
    return 0;
}
```

Lexer.cpp

```
#include "lexer.h"



pair<int, int> get_category(char s){
    int ascii_num = (int)s;
    //ws
    if ((ascii_num == 32) || (ascii_num == 8) || (ascii_num == 9)
|| (ascii_num == 10) || (ascii_num == 13)|| (ascii_num == 0))
        return make_pair(ascii_num, 0);
    //num
    else if ((ascii_num >= 48) && (ascii_num <= 57))
        return make_pair(ascii_num, 1);
    //let
    else if (((ascii_num >= 65) && (ascii_num <= 90)) ||
((ascii_num >= 97) && (ascii_num <= 122)))
        return make_pair(ascii_num, 2);
    // : ; = , .
    else if ( (ascii_num == 58) || (ascii_num == 59) || (ascii_num
== 61) || (ascii_num == 44) || (ascii_num == 46) || (ascii_num ==
41))
        return make_pair(ascii_num, 3);
    // (***)
    else if (ascii_num == 40)
        return make_pair(ascii_num, 5);
    // unexpected
    else
        return make_pair(ascii_num, 6);
}
```

```
void next_smb(ifstream &file, char &s, int &column){
    s = file.get();
    column++;
}

void Lexer::parse_file(string folder_name) {
    ifstream file(folder_name + "input.sig", ifstream::in);
    ofstream out;
    out.open(folder_name + "generated.txt");
    char s = file.get();
    if (!file.is_open()) {
        cout << "Not found file\n";
        return;
    }
    int curr_column = 0;
    int curr_row = 0;
    if (out.is_open()) {
        while (!file.eof()) {
            pair<int, int> curr_pair = get_category(s);
            string tmp_string = "";
            if (curr_pair.second == whitespace) {
                while (!file.eof()) {
                    if (s == '\n') {
                        curr_column = 0;
                        curr_row++;
                    }
                    if (curr_pair.second == whitespace) {
                        next_smb(file, s, curr_column);
                        curr_pair = get_category(s);
                    } else {
                        break;
                    }
                }
            }
            if (curr_pair.second == number) {
                tmp_string = "";
                while (!file.eof()) {
                    tmp_string.push_back(s);
                    next_smb(file, s, curr_column);
                    curr_pair = get_category(s);
                    if (curr_pair.second != number) {
                        if (curr_pair.second == letter) {
                            out << "Error: letter after digit.Col:"
<< curr_column << "Row:" << curr_row << endl;
                            printf("Error on line %d column %d.
Letter after digit!\n", curr_row, curr_column);
                        } else {
                            bool flag = true;
```

```
                            int lex_code;
                            for (auto it = all_numbers.begin(); it
!= all_numbers.end(); it++) {
                                if ((*it).first == tmp_string) {
                                    flag = false;
                                    lex_code = (*it).second;
                                }
                            }
                            if (flag) {
                                lex_code = 500 +
all_numbers.size();

all_numbers.push_back(make_pair(tmp_string, lex_code));
                            }
                            all_lexemes.push_back(
                                    Lexeme(lex_code, curr_row,
curr_column - tmp_string.size(), tmp_string));
                            break;
                        }

                    }
                }
            }

            if (curr_pair.second == letter) {
                int lex_code = -1;
                tmp_string = "";
                while (!file.eof()) {
                    tmp_string.push_back(s);
                    next_smb(file, s, curr_column);
                    curr_pair = get_category(s);
                    bool iskeyword = false;
                    if ((curr_pair.second != number) &&
(curr_pair.second != letter)) {
                        std::map<string, int>::iterator it;
                        it = keywords.find(tmp_string);
                        if (it != keywords.end()) {
                            iskeyword = true;
                            all_lexemes.emplace_back(
                                    Lexeme((*it).second, curr_row,
curr_column - tmp_string.size(), tmp_string));
                        } else {
                            bool flag = true;
                            for (auto it = all_ids.begin(); it !=
all_ids.end(); it++) {
                                if ((*it).first == tmp_string) {
                                    flag = false;
                                    lex_code = (*it).second;
                                }
```

```
                            }
                            if (flag) {
                                lex_code = 1000 + all_ids.size();
                                all_ids.emplace_back(tmp_string,
lex_code);
                            }
                        }
                        if (!iskeyword) {
                            all_lexemes.emplace_back(lex_code,
curr_row, curr_column - tmp_string.size(), tmp_string);
                        }
                        break;
                    }

                }
            }

            if (curr_pair.second == delimiter) {
                tmp_string = "";
                while (!file.eof()) {
                    tmp_string.push_back(s);
                    next_smb(file, s, curr_column);
                    curr_pair = get_category(s);
                    if (curr_pair.second != delimiter) {
                        for (auto it = delimiters.begin(); it !=
delimiters.end(); it++) {
                            if ((*it).first == tmp_string) {

all_lexemes.emplace_back((*it).second, curr_row, curr_column -
tmp_string.size(),

tmp_string);
                                break;
                            }
                        }
                        break;
                    } else {
                        out << "Error: Undefined delimiter.Col:" <<
curr_column << "Row:" << curr_row << endl;
                        printf("Undefined delimiter on %d column %d
row\n", curr_column, curr_row);
                        break;
                    }
                }
            }

            if (curr_pair.second == comment) {
                tmp_string = "";
                while (!file.eof()) {
```

```
                    tmp_string.push_back(s);
                    next_smb(file, s, curr_column);
                    if (s == '*') {
                        bool comment = true;
                        string com_str = "*";
                        //start comment
                        while (!file.eof()) {
                            next_smb(file, s, curr_column);
                            if (s == '\n') {
                                curr_column = 0;
                                curr_row++;
                                continue;
                            }
                            com_str.push_back(s);
                            if (com_str.substr(com_str.length() -
2, 2) == "*)") {
                                next_smb(file, s, curr_column);
                                comment = false;
                                break;
                            }
                        }
                        if (comment) {
                            out << "Comment not ended" << endl;
                            cout << "Comment not ended\n";
                        }
                    } else {
                        if (tmp_string == "(") {
                            all_lexemes.emplace_back(40, curr_row,
curr_column - tmp_string.size(), tmp_string);
                        }
                        if (tmp_string == ")") {
                            all_lexemes.emplace_back(41, curr_row,
curr_column - tmp_string.size(), tmp_string);
                        }
                    }
                    break;
                }

            }
            if ((curr_pair.second == unacceptable) &&
(!file.eof())) {
                out << "Error: Unaccepted token.Col:" <<
curr_column << "Row:" << curr_row << endl;
                printf("Unaccepted token on column %d row %d\n",
curr_column, curr_row);
                while (!file.eof()) {
                    next_smb(file, s, curr_column);
                    curr_pair = get_category(s);
                    if (curr_pair.second != unacceptable) break;
```

```
                }
                next_smb(file, s, curr_column);
            }
        }
    }
}


void Lexer::print_all_lexemes() {
    for(auto i : all_lexemes){
        i.print_lexeme();
    }
}

void Lexer::fill_generated(string test_folder) {
    ofstream out;
    out.open(test_folder + "generated.txt", ios::app);
    if (out.is_open()){
        for(auto lex : all_lexemes){
            out << setw(4) << lex.get_row() << setw(4) <<
lex.get_column() << setw(6) << lex.get_code() << ' ' <<
lex.get_name() << endl;
        }
    }
    out.close();
}
```

Lexer.h

```
#ifndef LAB1_LEXER_H
#define LAB1_LEXER_H

#include "lexeme.h"
#include <fstream>
#include <iostream>
#include <map>
#include <vector>
#include <iomanip>
#include <algorithm>

using namespace std;

class Lexer {
    private:
        map <string, int> keywords = {
                {"program", 401},
```

```
                {"begin", 402},
                {"end", 403},
                {"procedure", 404},
                {"signal", 405},
                {"integer", 406},
                {"float", 407},
                {"blockfloat", 408},
                {"ext", 409},
        };
        map <string, int> delimiters = { {";", 59},
                                         {",", 44},
                                         {":", 58},
                                         {"(", 40},
                                         {")", 41} };
        vector <pair<string, int>> all_numbers;
        vector <pair<string, int>> all_ids;
        vector <Lexeme> all_lexemes;
        enum categories { whitespace = 0, number = 1, letter = 2,
delimiter = 3, comment = 5, unacceptable = 6 };

    public:
        void parse_file(string folder_name);
        void print_all_lexemes();
        void fill_generated(string test_folder);
};



#endif //LAB1_LEXER_H
```

lexeme.cpp

```
#include "lexeme.h"

Lexeme::Lexeme(int code, int row_, int column_, string name_){
    lex_code = code;
    row = row_;
    column = column_;
    name = name_;
}

void Lexeme::set_code(int code){
    lex_code = code;
}

int Lexeme::get_code(){
    return lex_code;
```

```
}

void Lexeme::set_row(int row_){
    row = row_;
}

int Lexeme::get_row(){
    return row;
}

void Lexeme::set_column(int column_){
    column = column_;
}

int Lexeme::get_column(){
    return column;
}

void Lexeme::set_name(string name_){
    name = name_;
}

string Lexeme::get_name(){
    return name;
}

void Lexeme::print_lexeme(){
    printf("%4d %4d %6d   ", row, column, lex_code);
    cout << name << endl;
}
```

lexeme.h

```
#ifndef LAB1_LEXEME_H
#define LAB1_LEXEME_H

#include <iostream>
#include <string>

using namespace std;

class Lexeme {
private:
    int lex_code;
    int row;
    int column;
    string name;
```

```
public:
    Lexeme(int code, int row, int column, string name);
    void set_code(int code);
    int get_code();
    void set_row(int code);
    int get_row();
    void set_column(int code);
    int get_column();
    void set_name(string code);
    string get_name();

    void print_lexeme();
};

#endif //LAB1_LEXEME_H
```