

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

# Кафедра системного програмування і спеціалізованих комп’ютерних систем

**Лабораторна робота №1**

з дисципліни

**«Основи проектування трансляторів»**

Тема: ”Розробка лексичного аналізатора”

Виконав: студент ІII курсу

ФПМ групи КВ-93

Мазенко М. О.

Перевірив:

Київ – 2021

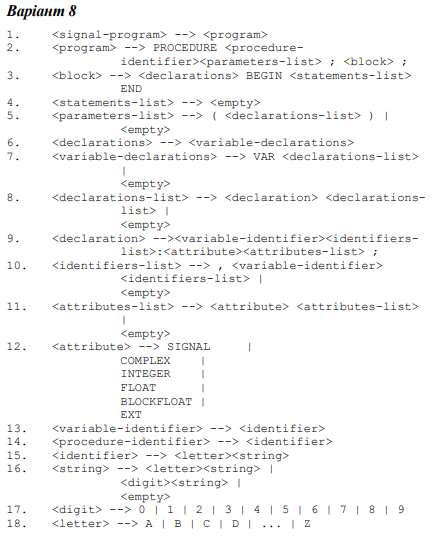
**Мета лабораторної роботи**

Метою лабораторної роботи «Розробка лексичного аналізатора» є засвоєння теоретичного матеріалу та набуття практичного досвіду і практичних навичок розробки лексичних аналізаторів (сканерів).

**Постановка задачі**

1. Розробити програму лексичного аналізатора (ЛА) для підмножини мови програмування SIGNAL.
2. Лексичний аналізатор має забезпечувати наступні дії:

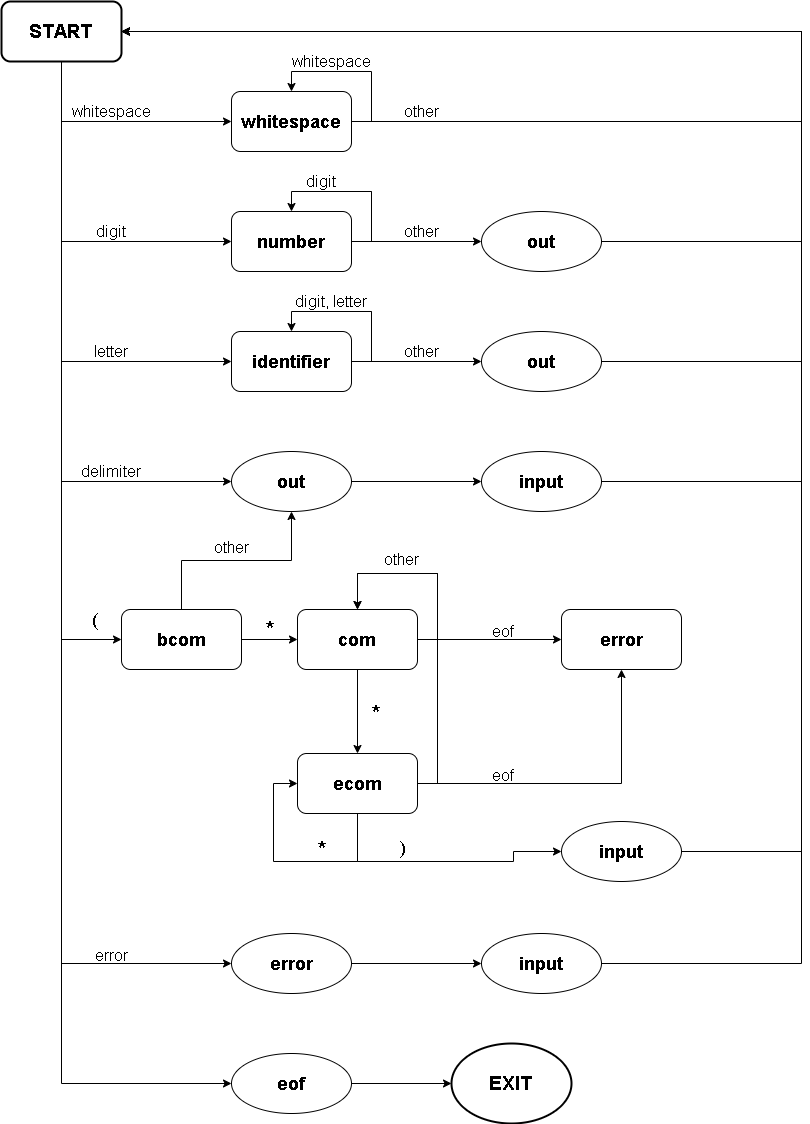
* видалення (пропускання) пробільних символів: пробіл (код ASCII 32), повернення каретки (код ASCII 13); перехід на новий рядок (код ASCII 10), горизонтальна та вертикальна табуляція (коди ASCII 9 та 11), перехід на нову сторінку (код ASCII 12);
* згортання ключових слів;
* згортання багато-символьних роздільників (якщо передбачаються граматикою варіанту);
* згортання констант із занесенням до таблиці значення та типу константи (якщо передбачаються граматикою варіанту);
* згортання ідентифікаторів;
* видалення коментарів, заданих у вигляді (\*<текст коментаря>\*);
* формування рядка лексем зінформацією про позиції лексем;
* заповнення таблиць ідентифікаторів та констант інформацією, отриманою підчас згортки лексем;
* виведення повідомлень про помилки.



Початкові значення в таблицях лексем:

|  |  |
| --- | --- |
| Тип | Лексеми |
| Роздільник | ;  (  )  ,  : |
| Константа | PI  EU |
| Ідентифікатор | SIGNAL  COMPLEX  INTEGER  FLOAT  BLOCKFLOAT  EXT |
| Ключове слово | PROCEDURE  BEGIN  END  VAR |

Граф автомату, що визначає алгоритм лексичного аналізатора:



**Тести програми**

|  |
| --- |
| True тест |
| PROCEDURE TrueTest(VAR newVar:FLOAT;);  VAR newVar1:EXT;  VAR newVar2:SIGNAL;  BEGIN  (\* Test comment with symbold 142#@$%!(~$8)\*)  (\*\*\*) (\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)  (\*\*) (\*(()\*())\*)  (\* bafsssd  \*)  END; |
|  |

|  |
| --- |
| False тест |
| PROCE?DURE TrueTest(VAR newVar:FLOAT;);  VAR newVar1:EXT;  VAR #newVar2:SIGNAL;  BEGIN.  (\* Test comment with symbold 142#@$%!(~$8)\*)  (\*\*\*) (\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)  (\*\*  END; |
|  |

**Лістинг коду програми**

**main.cpp**

#include "view.h"

void LexerTrueTest() {

translator translator\_object = translator();

view::ShowFile("txt/lexer\_true\_test.txt");

translator\_object.Lexer("txt/lexer\_true\_test.txt");

view::PrintLexemTable(translator\_object.GetLexemTable());

view::PrintInformTable(translator\_object.GetIdentifiers());

view::PrintInformTable(translator\_object.GetConstants());

view::PrintInformTable(translator\_object.GetDelimiters());

view::PrintInformTable(translator\_object.GetKeywords());

}

void LexerFalseTest() {

translator translator\_object = translator();

view::ShowFile("txt/lexer\_false\_test.txt");

translator\_object.Lexer("txt/lexer\_false\_test.txt");

view::PrintLexemTable(translator\_object.GetLexemTable());

view::PrintInformTable(translator\_object.GetIdentifiers());

view::PrintInformTable(translator\_object.GetConstants());

view::PrintInformTable(translator\_object.GetDelimiters());

view::PrintInformTable(translator\_object.GetKeywords());

}

int main() {

//PrintInformTable(translator\_object.GetKeywords());

//PrintInformTable(translator\_object.GetDelimiters());

//PrintAttributeTable(translator\_object.GetAttributes());

//LexerTrueTest();

LexerFalseTest();

return 0;

}

**fileReader.h**

#pragma once

#include <fstream>

#include <iostream>

using namespace std;

class fileReader

{

private:

pair<int, int> coordinates;

fstream file;

public:

fileReader() : fileReader("txt/file\_1.txt") {};

fileReader(string path) {

coordinates.first = 0;

coordinates.second = 1;

file = fstream(path, ios\_base::in);

}

~fileReader() {

file.close();

}

bool FileIsEof() {

return file.eof();

}

bool FileIsSuitable() {

return file.is\_open() && !FileIsEof() ? true : false;

}

char ReadSymbol();

pair<int, int> GetCoordinates() { return coordinates; }

};

**fileReader.cpp**

#include "fileReader.h"

char fileReader::ReadSymbol() {

char symbol;

file.get(symbol);

if ((int)symbol == 10) {

coordinates.second++;

coordinates.first = 0;

}

else

coordinates.first++;

return symbol;

}

**translator.h**

#pragma once

#include <string>

#include <vector>

#include <list>

#include "fileReader.h"

#define TableSetBasicInformation(name, position) SetBasicInformation(#name, &name, position)

struct lexem {

pair<int, int> coordinates;

int code;

string word;

};

struct inform\_table {

string word;

int code;

};

class translator

{

private:

vector<inform\_table> keyword\_table;

vector<inform\_table> identifier\_table;

vector<inform\_table> constant\_table;

vector<inform\_table> delimiter\_table;

vector<lexem> lexem\_table;

static vector<int> attributes;

static vector<int> SetAttributes();

bool FindAttribute(int, vector<int>);

void SetBasicInformation(const string, vector<inform\_table>\*, int);

char ReadWithAdding(char, vector<int>, list<vector<inform\_table>\*>, fileReader\*);

char ReadWithoutAdding(char, bool (\*)(int), fileReader\*);

char ReadComment(char, fileReader\*);

static bool WhitespaceChecker(int);

static bool CommentChecker(int);

void FindInTable(string, list<vector<inform\_table>\*>, pair<int, int>);

public:

translator();

void Lexer(string path);

void Parser();

vector<inform\_table> GetKeywords() { return keyword\_table; };

vector<inform\_table> GetIdentifiers() { return identifier\_table; };

vector<inform\_table> GetConstants() { return constant\_table; };

vector<inform\_table> GetDelimiters() { return delimiter\_table; };

vector<lexem> GetLexemTable() { return lexem\_table; };

vector<int> GetAttributes() { return attributes; };

};

**translator.cpp**

#include "translator.h"

vector<int> translator::attributes = translator::SetAttributes();

vector<int> translator::SetAttributes() {

vector<int> local\_list;

local\_list.assign(127, 6);

string current\_field;

int line\_iteration = 0, current\_number = 0;

fstream file("txt/attributes.txt", ios\_base::in);

if (file.is\_open()) {

while (file >> current\_field) {

if (current\_field == ".")

line\_iteration++;

else if (current\_field == "-") {

file >> current\_field;

for (int iteration = current\_number + 1; iteration <= stoi(current\_field); iteration++)

local\_list[iteration] = line\_iteration;

}

else {

current\_number = stoi(current\_field);

local\_list[current\_number] = line\_iteration;

}

}

}

return local\_list;

}

translator::translator() {

TableSetBasicInformation(keyword\_table, 300);

TableSetBasicInformation(identifier\_table, 400);

TableSetBasicInformation(constant\_table, 500);

TableSetBasicInformation(delimiter\_table, 55);

}

void translator::SetBasicInformation(const string name, vector<inform\_table>\* lst, int position) {

vector<inform\_table> inform\_table\_list;

inform\_table local\_inform\_table;

fstream file("txt/" + name + ".txt", ios\_base::in);

if (file.is\_open()) {

while (getline(file, local\_inform\_table.word)) {

local\_inform\_table.code = ++position;

inform\_table\_list.push\_back(local\_inform\_table);

}

\*lst = inform\_table\_list;

}

}

void translator::Lexer(string path) {

fileReader file\_object(path);

char current\_symbol;

bool suppress\_output = false;

vector<int> acceptable\_attributes;

list<vector<inform\_table>\*> acceptable\_tables;

int current\_value, current\_attribute;

if (file\_object.FileIsSuitable())

current\_symbol = file\_object.ReadSymbol();

else {

cout << "Lexer: invalid file name or content.\n";

return;

}

do {

current\_value = (int)current\_symbol;

current\_attribute = attributes[current\_value];

switch (current\_attribute) {

case 0:

current\_symbol = ReadWithoutAdding(file\_object.ReadSymbol(), &WhitespaceChecker, &file\_object);

break;

case 1:

current\_symbol = ReadWithAdding(current\_symbol, { 1 }, { &constant\_table }, &file\_object);

break;

case 2:

current\_symbol = ReadWithAdding(current\_symbol, { 1, 2 }, { &keyword\_table, &constant\_table, &identifier\_table }, &file\_object);

break;

case 3:

FindInTable(string(1, current\_symbol), { &delimiter\_table }, file\_object.GetCoordinates());

current\_symbol = file\_object.ReadSymbol();

break;

case 4:

current\_symbol = ReadComment(current\_symbol, &file\_object);

break;

default:

cout << "Lexer: wrong symbol \"" << current\_symbol << "\"\n";

current\_symbol = file\_object.ReadSymbol();

break;

}

} while (!file\_object.FileIsEof());

}

char translator::ReadComment(char start\_symbol, fileReader\* file\_object) {

char current\_symbol;

pair<int, int> start\_coordinates = file\_object->GetCoordinates();

current\_symbol = file\_object->ReadSymbol();

if (attributes[(int)current\_symbol] == 5) {

do {

ReadWithoutAdding(current\_symbol, &CommentChecker, file\_object);

current\_symbol = file\_object->ReadSymbol();

if (file\_object->FileIsEof()) {

cout << "Lexer: end of comment is missing\n";

return 0;

}

} while (current\_symbol != ')');

current\_symbol = file\_object->ReadSymbol();

}

else {

FindInTable(string(1, start\_symbol), { &delimiter\_table }, start\_coordinates);

}

return current\_symbol;

}

char translator::ReadWithoutAdding(char current\_symbol, bool (\*Condition)(int), fileReader\* file\_object) {

while (!file\_object->FileIsEof() && Condition(attributes[(int)current\_symbol])) {

current\_symbol = file\_object->ReadSymbol();

}

return current\_symbol;

}

void translator::FindInTable(string lexem\_word, list<vector<inform\_table>\*> acceptable\_tables, pair<int, int> coordinates) {

for (vector<inform\_table>\* current\_table\_type : acceptable\_tables) {

for (inform\_table current\_inform\_table : \*current\_table\_type) {

if (current\_inform\_table.word.compare(lexem\_word) == 0) {

lexem new\_lexem = {

coordinates,

current\_inform\_table.code,

lexem\_word };

lexem\_table.push\_back(new\_lexem);

return;

}

}

}

int new\_code = acceptable\_tables.back()->back().code + 1;

acceptable\_tables.back()->push\_back(inform\_table{ lexem\_word, new\_code });

lexem\_table.push\_back(lexem{ coordinates, new\_code, lexem\_word });

}

char translator::ReadWithAdding(char symbol, vector<int> acceptable\_attributes, list<vector<inform\_table>\*> acceptable\_tables, fileReader\* file\_object) {

string current\_lexem;

pair<int, int> start\_coordinates = file\_object->GetCoordinates();

char current\_symbol = symbol;

do {

current\_lexem += current\_symbol;

current\_symbol = file\_object->ReadSymbol();

} while (FindAttribute(attributes[(int)current\_symbol], acceptable\_attributes));

FindInTable(current\_lexem, acceptable\_tables, start\_coordinates);

return current\_symbol;

}

bool translator::WhitespaceChecker(int attribute) {

return attribute == 0 ? true : false;

}

bool translator::CommentChecker(int attribute) {

return attribute != 5 ? true : false;

}

bool translator::FindAttribute(int attribute, vector<int> attribute\_list) {

for (int current\_attribute : attribute\_list) {

if (attribute == current\_attribute) {

return true;

}

}

return false;

}

void translator::Parser() {}

**view.h**

#pragma once

#include "translator.h"

#include <iomanip>

static class view

{

public:

static void PrintInformTable(vector<inform\_table>);

static void PrintAttributeTable(vector<int>);

static void PrintLexemTable(vector<lexem>);

static void ShowFile(string);

};

**view.cpp**

#include "view.h"

#include <time.h>

void view::ShowFile(string path) {

fileReader file\_object(path);

char symbol = 10;

cout << "Input file:\n";

cout << "\_\_\_\_\_\_\_\_\_\_\_";

while (!file\_object.FileIsEof()) {

cout << symbol;

symbol = file\_object.ReadSymbol();

}

cout << "\n\_\_\_\_\_\_\_\_\_\_\_\n";

}

void view::PrintLexemTable(vector<lexem> lexem\_table) {

cout << "+=========================================+\n";

cout << "| Column| Row| Code| Lexem|\n";

cout << "|=========================================|\n";

for (lexem current\_lexem : lexem\_table) {

cout

<< "|"

<< setw(7) << current\_lexem.coordinates.first << "|"

<< setw(7) << current\_lexem.coordinates.second << "|"

<< setw(7) << current\_lexem.code << "|"

<< setw(17) << current\_lexem.word << "|\n";

}

cout << "+=========================================+\n\n";

}

void view::PrintInformTable(vector<inform\_table> lst) {

cout << "+==============================+\n";

cout << "|" << setw(20) << "Name|" << setw(10) << "Code" << "|\n";

cout << "|==============================|\n";

for (inform\_table local\_inform\_table : lst) {

cout << "|" << setw(20) << local\_inform\_table.word + "|" << setw(10) << local\_inform\_table.code << "|\n";

}

cout << "|==============================|\n\n";

}

void view::PrintAttributeTable(vector<int> attributes) {

cout << "+=======================================================================+\n";

for (int iteration = 32; iteration < attributes.size(); iteration++) {

cout << "| " << attributes[iteration] << " " << setw(5) << (char)iteration;

if (iteration % 8 == 7) {

cout << "|" << endl;

}

}

cout << "|\t|\n";

cout << "+=======================================================================+\n\n";

}