МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

НГТУ
НЭТИ | Факультет прикладной
математики и информатики

Кафедра теоретической и прикладной информатики

Лабораторная работа № 2

по дисциплине «Языки программирования и методы трансляции»

РАЗРАБОТКА И РЕАЛИЗАЦИЯ БЛОКА ЛЕКСИЧЕСКОГО АНАЛИЗА

Бригада ГОЛУБЬ АНДРЕЙ

БУДАНЦЕВ ДМИТРИЙ

Группа ПМ-13

Вариант 8

Преподаватель ДВОРЕЦКАЯ ВИКТОРИЯ КОНСТАНТИНОВНА

Новосибирск,2024

1. Цель работы

Изучить методы лексического анализа. Получить представление о методах обработки лексических ошибок. Научиться проектировать сканер на основе детерминированных конечных автоматов.

2. Задачи

Подмножество языка C++ включает:

- данные типа **int, float, struct** из элементов указанных типов**;**
- инструкции описания переменных;
- операторы присваивания в любой последовательности;
- операции +, – , <, >, побитовые операции <<, >>, &, | .

В соответствии с выбранным вариантом задания к лабораторным работам разработать и реализовать лексический анализатор на основе детерминированных конечных автоматов. Исходными данными для сканера является программа на языке C++ и постоянные таблицы, реализованные в лабораторной работе №1. Результатом работы сканера является создание файла токенов, переменных таблиц (таблицы символов и таблицы констант) и файла сообщений об ошибках.

3. Входные и выходные данные

Постоянные таблицы заполняются с помощью текстовых файлов, хранящие ключевые слова, допустимые символы, числа, операции, разделители. На файлы не накладываются дополнительные ограничения, кроме аппаратных ограничений компьютера.

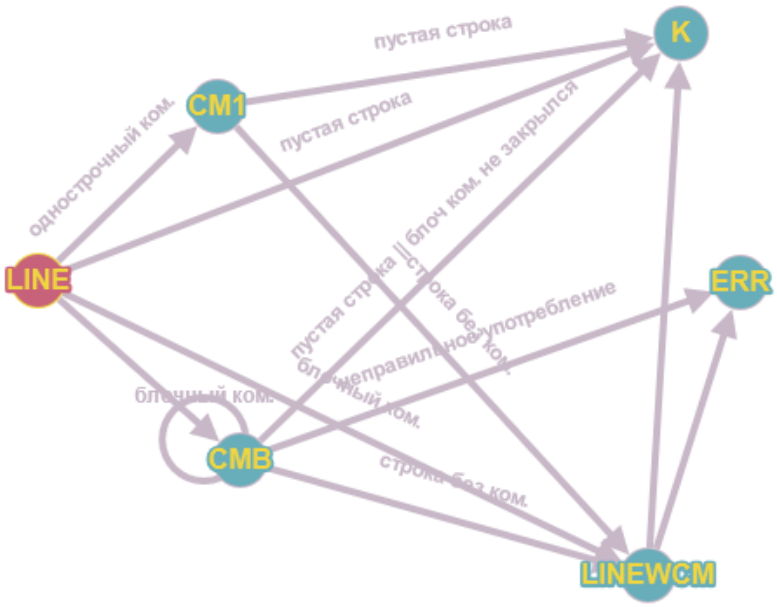
Переменные таблицы заполняются в ходе работы программы, используя в качестве ключа идентификатор элемента или значение константы. Есть возможность изменять данные в процессе работы.

На количество элементов в таблицах накладываются только аппаратные ограничения.

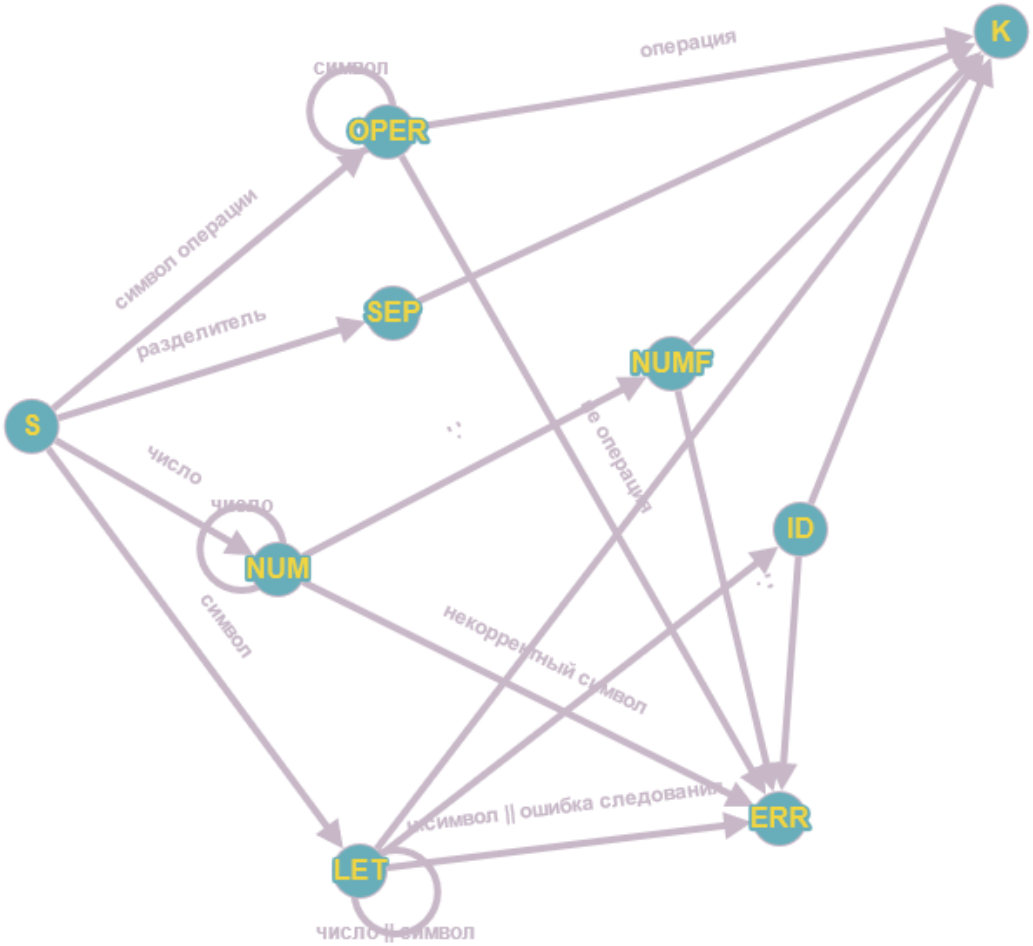
Выходными файлами являются файлы: файл токенов, файл ошибок, файл идентификаторов, файл констант, файл данных хранящихся в постоянных таблицах.

4. Детерминированный конечный автомат

Обработка комментариев



Разбор строчки без комментариев



Описание состояний

LINE – очередная строчка

CM1 – строчка с однострочным комментарием

CMB – строчка с блочными комментариями

LINEWCM – строчка без комментариев

K – конечное состояние
ERR – ошибка
S – первый символ строчки без комментариев
NUM – целое число
LET – ключевое слово
OPER – операция
SEP – разделитель
ID – идентификатор
NUMF – число с плавающей запятой

5. Алгоритм разбора

1. Считать строку. Если это конец файла то переходим на шаг 8.
2. Очистить строку от комментариев.
3. Если мы находимся в конце строки, то переходим на шаг 1. Считать следующий символ строчки ch. Если это разделитель и один из {“ ”, “;”} и записываемая очередь не пустая и то, что хранится в очереди, является тем, что предполагалось до этого, тогда формируем токен этого, и переходим на шаг 3. Иначе формируем токен разделителя и переходим на шаг 3.
4. Если символ ch число и очередь пустая то, мы кладём ch в очередь, предполагая, что далее записываем число и переходим на шаг 3. Если до этого в очереди мы предполагали, что записываем операцию, то переходим на шаг 7. Иначе записываем ch в очередь и переходим на шаг 3.
5. Если символ ch из множества допустимых символов и очередь пустая то, записываем символ, в очередь, предполагая, что записываем ключевое слово или идентификатор. Если очередь не пуста и предполагалось, что записывалась операция или число, то переходим на шаг 7. Если предполагалось, что записывалось ключевое слово или идентификатор, то записываем символ в очередь и переходим на шаг 3.
6. Если очередь пуста, то записываем символ, в очередь, предполагая, что записываем символ операции и переходим на шаг 3. Иначе если предполагаемо, в очереди операция добавляем в очередь символ и если слово из очереди является операцией, то записываем токен операции, очищаем очередь и переходим на шаг 3. Иначе переходим на шаг 7.
7. Ошибка, вывести соответствующее сообщение и прекратить разбор.
8. Конец.

6. Текст программ

Translator.h

```
#pragma once
#include "ConstTable.h"
#include "VariableHashTable.h"
#include "Token.h"
#include "Error.h"
#define KEYWORDFILENAME "input_files/Types.txt"
#define SEPARATORFILENAME "input_files/Separator.txt"
#define NUMFILENAME "input_files/Numbers.txt"
#define OPERATIONFILENAME "input_files/Operation.txt"
#define LETTERSFILENAME "input_files/Letters.txt"


#ifndef TRANSLATOR
```

```
class Translator {
private:
    ConstantTable<string> operation; // 0
    ConstantTable<char> numbers; // 1
    ConstantTable<string> keyword; // 2
    ConstantTable<char> separators; // 3
    ConstantTable<char> letters; // 4
    VariableHashTable identificators; // 5
    VariableHashTable constans; // 6
    queue<Errors> QErrors;
    queue<Token> QToken;

    int num_line = 0;

    void ltrim(string& str) {
        str.erase(0, str.find_first_not_of("\t\n\v\f\r "));
    }
    void rtrim(string& str) {
        str.erase(str.find_last_not_of("\t\n\v\f\r ") + 1);
    }
    void CheckingComments(string& line, bool& continuecheck, bool& nextcheck) {
        size_t ind3 = line.find("*/");
        if (continuecheck) {
            if (ind3 != -1) {
                line.erase(0, ind3 + 2);
                ind3 = line.find("*/");
            }
            else {
                line = "";
                continuecheck = true;
                return;
            }
        }
        size_t ind1 = line.find("//"); size_t ind2 = line.find("/*");
        while (ind1 != -1 || ind2 != -1 || ind3 != -1) {
            if (ind1 != -1 && (ind1 < ind2 || ind2 == -1) && (ind1 < ind3 || ind3
== -1)) {
                line.erase(ind1);
                rtrim(line);
                continuecheck = false;
                return;
            }
            else if (ind2 != -1) {
                if (ind3 == -1) {
                    line.erase(ind2);
                    rtrim(line);
                    nextcheck = true;
                    continuecheck = false;
                    return;
                }
                else if (ind2 < ind3) {
```

```
                    line.erase(ind2, ind3 - ind2 + 2);
                    ind1 = line.find("//"); ind2 = line.find("/*"); ind3 =
line.find("*/");
                    continue;
                }
                else {
                    QErrors.push(Errors(num_line, ERR_BLOCKCOMMENT));
                    line = "";
                    continuecheck = false;
                    return;
                }
            }
            else {
                QErrors.push(Errors(num_line, ERR_BLOCKCOMMENT));
                line = "";
                continuecheck = false;
                return;
            }
        }
        ltrim(line);
        rtrim(line);
        continuecheck = false;
        return;
    }

    string buffer_to_string(queue<char> buffer) {
        stringstream ss;
        while (!buffer.empty()) {
            ss << buffer.front();
            buffer.pop();
        }
        return ss.str();
    }
    void clear_buffer(queue<char>& buffer) {
        while (!buffer.empty()) buffer.pop();
    }
    Code_Errors ScanLineWithoutComments(string& line) {
        regex re0("^([+-\/*%=]=)$");

        ltrim(line);
        queue<char> buffer;
        queue<char> old_buffer;
        int old_mode = 0;
        int temp_mode = 0, num_dot = 0;
        string temp_str, dop_str;
        char temp_char;
        for (char ch : line) {
            if (separators.Contains(ch) || ch == ' ') {
                if (ch != ' ')
                    separators.Add(ch);
                switch (temp_mode)
```

```
                {
                case 0:
                    if (ch != ' ')
                        QToken.push(Token(3, separators.getIndex(ch)));
                    continue;
                case 1:
                    if (ch == ' ' || ch == ';') {
                        temp_str = buffer_to_string(buffer);
                        if (!constans.Contains(temp_str)) {
                            if (!constans.Add(temp_str, "const")) return
ERR_FAILED_ADD_TABLE;
                        }
                        QToken.push(Token(6, constans.getIndex(temp_str)));
                        if (ch == ';') QToken.push(Token(3,
separators.getIndex(ch)));
                        temp_mode = 0;
                        num_dot = 0;
                        old_mode = 1;
                        old_buffer.swap(buffer);
                        clear_buffer(buffer);
                        continue;
                    }
                    return ERR_INVALID_INT_NUM;
                case 2:
                    if (ch == ' ' || ch == ';') {
                        temp_str = buffer_to_string(buffer);
                        if (keyword.Contains(temp_str)) {
                            keyword.Add(temp_str);
                            if (old_mode == 2) return ERR_KEYWORD_IN_VAR_NAME;
                            QToken.push(Token(2, keyword.getIndex(temp_str)));
                            if (ch == ';') QToken.push(Token(3,
separators.getIndex(ch)));
                            old_mode = temp_mode;
                            temp_mode = 0;
                            old_buffer.swap(buffer);
                            clear_buffer(buffer);
                            continue;
                        }
                        else {
                            if (!identificators.Contains(temp_str)) {
                                if (old_mode == 2) {
                                    dop_str = buffer_to_string(old_buffer);
                                    if (!identificators.Add(temp_str, dop_str))
return ERR_FAILED_ADD_TABLE;
                                }
                                else return ERR_UNKNOWN_VARIABLE;
                            }
                            else if (old_mode == 2) return ERR_VAR_OVERRIDES;
                            QToken.push(Token(5,
identificators.getIndex(temp_str)));
```

```
                            if (ch == ';') QToken.push(Token(3,
separators.getIndex(ch)));
                            old_mode = 4;
                            temp_mode = 0;
                            old_buffer.swap(buffer);
                            clear_buffer(buffer);
                            continue;
                        }
                    }
                    return ERR_INVALID_NAME;
                case 3:
                    if (ch == ' ') {
                        temp_str = buffer_to_string(buffer);
                        if (operation.Contains(temp_str)) {
                            operation.Add(temp_str);
                            if (old_mode != 4 && old_mode != 1) return
ERR_INCORRECT_USE_OPERATION;
                            QToken.push(Token(0, operation.getIndex(temp_str)));
                            old_mode = 3;
                            old_buffer.swap(buffer);
                            clear_buffer(buffer);
                            temp_mode = 0;
                            continue;
                        }
                    }
                    return ERR_INVALID_OPERATION;
                default:
                    break;
                }
            }
            else if (numbers.Contains(ch)) {
                switch (temp_mode)
                {
                case 0:
                    buffer.push(ch);
                    temp_mode = 1;
                    continue;
                case 1:
                    temp_char = buffer.front();
                    if (temp_char != '0') {
                        numbers.Add(ch);
                        buffer.push(ch);
                        continue;
                    }
                    else {
                        return ERR_INVALID_INT_NUM;
                    }
                case 2:
                    numbers.Add(ch);
                    buffer.push(ch);
                    continue;
```

```
            case 3:
                return ERR_INVALID_SYMBOL;
            default:
                break;
            }
        }
        else if (letters.Contains(ch)) {
            letters.Add(ch);
            switch (temp_mode)
            {
            case 0:
                buffer.push(ch);
                temp_mode = 2;
                continue;
            case 1:
                if (old_mode == 2) return ERR_INVALID_NAME;
                return ERR_INVALID_INT_NUM;
            case 2:
                buffer.push(ch);
                continue;
            case 3:
                return ERR_INVALID_OPERATION;
            default:
                break;
            }
        }
        else {
            if (temp_mode == 0) {
                if (ch == '=') {
                    if (old_mode != 4) return ERR_INVALID_ASSIGNMENT_OPERATION;
                    dop_str = buffer_to_string(old_buffer);
                    if (!identificators.SetValue(dop_str, true)) return
ERR_FAILED_ADD_TABLE;
                    old_mode = 3;
                    clear_buffer(old_buffer);
                    continue;
                }
                temp_mode = 3;
                buffer.push(ch);
                continue;
            }
            else {
                if (temp_mode != 3) {
                    if (ch == '.' && temp_mode == 1) {
                        if (num_dot == 0) {
                            buffer.push(ch);
                            num_dot++;
                            continue;
                        }
                        else {
                            return ERR_INVALID_FLOAT_NUM;
```

```
                        }
                    }
                    return ERR_INVALID_NAME;
                }
                else {
                    buffer.push(ch);
                    temp_str = buffer_to_string(buffer);
                    if (operation.Contains(temp_str)) {
                        operation.Add(temp_str);
                        QToken.push(Token(0, operation.getIndex(temp_str)));
                        clear_buffer(buffer);
                        temp_mode = 0;
                        continue;
                    }
                    else if (regex_match(temp_str, re0)) {
                        if (old_mode == 4) {
                            dop_str = buffer_to_string(old_buffer);
                            if (identificators.getElement(dop_str).value) {
                                old_mode = 3;
                                clear_buffer(old_buffer);
                                clear_buffer(buffer);
                                temp_mode = 0;
                                continue;
                            }
                            else return
ERR_PERFORMING_OPERATION_UNINITIALIZED_VAR;
                        }
                        else return ERR_INCORRECT_USE_OPERATION;
                    }
                    else return ERR_INVALID_OPERATION;
                }
            }
        }
    }
    return NO_ERROR;
}

public:
    Translator() {
        if (!operation.ReadKeysFile(OPERATIONFILENAME)) QErrors.push(Errors(0,
FATALERR_FAILLEDREADFILE));
        if (!numbers.ReadKeysFile(NUMFILENAME)) QErrors.push(Errors(0,
FATALERR_FAILLEDREADFILE));
        if (!keyword.ReadKeysFile(KEYWORDFILENAME)) QErrors.push(Errors(0,
FATALERR_FAILLEDREADFILE));
        if (!separators.ReadKeysFile(SEPARATORFILENAME)) QErrors.push(Errors(0,
FATALERR_FAILLEDREADFILE));
        if (!letters.ReadKeysFile(LETTERSFILENAME)) QErrors.push(Errors(0,
FATALERR_FAILLEDREADFILE));
    }
```

```
    void PrintToken() {
        cout << "TABLE" << "\t" << "NUM_TABLE" << endl;
        queue<Token> copy_token = QToken;
        Token temp_token;
        while (!copy_token.empty()) {
            temp_token = copy_token.front();
            copy_token.pop();
            cout << temp_token.NUM_TABLE << "\t" << temp_token.INDEX << endl;
        }
    }
    void PrintErrors() {
        cout << "NUM LINE" << "\t" << "CODE ERROR" << endl;
        queue<Errors> copy_Error = QErrors;
        Errors temp_error;
        while (!copy_Error.empty()) {
            temp_error = copy_Error.front();
            copy_Error.pop();
            cout << temp_error.NUM_LINE << "\t" << temp_error.CODE << endl;
        }
    }
    void ScanningFile(const string& NameFile) {
        if (operation.empty() || numbers.empty() || keyword.empty() ||
letters.empty() || separators.empty()) return;

        ifstream ScanFile(NameFile, ios::in);
        if (!ScanFile.is_open()) {
            QErrors.push(Errors(0, FATALERR_FAILLEDREADSCANFILE));
            return;
        }

        string line;
        bool check = false, nextcheck = false;
        Code_Errors code;
        while (getline(ScanFile, line)) {
            if (nextcheck) {
                check = true;
                nextcheck = false;
            }
            CheckingComments(line, check, nextcheck);
            if (check == true || line.empty()) {
                num_line++;
                continue;
            }
            if ((code = ScanLineWithoutComments(line)) != NO_ERROR)
QErrors.push(Errors(num_line, code));
            num_line++;
        }
        ScanFile.close();
        if (check || nextcheck) QErrors.push(Errors(num_line, ERR_BLOCKCOMMENT));
    }
```

```
    bool outFileToken(ofstream& FileOut) {
        if (!FileOut.is_open()) {
            cerr << "Error! File not opened!" << endl;
            return false;
        }
        queue<Token> CQtoken = QToken;
        Token t_token;
        while (!CQtoken.empty()) {
            t_token = CQtoken.front();
            CQtoken.pop();
            FileOut << t_token.NUM_TABLE << "\t" << t_token.INDEX << endl;
        }
        return true;
    }
    bool outFileErrors(ofstream& FileOut) {
        if (!FileOut.is_open()) {
            cerr << "Error! File not opened!" << endl;
            return false;
        }
        queue<Errors> CQErrors = QErrors;
        Errors t_error;
        while (!CQErrors.empty()) {
            t_error = CQErrors.front();
            CQErrors.pop();
            FileOut << t_error.info() << endl;
        }
        return true;
    }
    bool outFileIC(ofstream& FileOut) {
        return identificators.outFile(FileOut);
    }
    bool outFileC(ofstream& FileOut) {
        return constans.outFile(FileOut);
    }
    bool outConstTable(ofstream& FileOut) {
        if (!FileOut.is_open()) {
            cerr << "Error! File not opened!" << endl;
            return false;
        }
        letters.outfile(FileOut);
        keyword.outfile(FileOut);
        operation.outfile(FileOut);
        separators.outfile(FileOut);
        numbers.outfile(FileOut);
        return true;
    }
};

#endif // !TRANSLATOR
```

Token.h

```
#pragma once
#ifndef TOKEN

class Token {

public:
    int NUM_TABLE = 0;
    int INDEX = 0;

    Token() {}
    Token(int num_table, int index) {
        NUM_TABLE = num_table;
        INDEX = index;
    }
};


#endif // !TOKEN
```

Error.h

```
#pragma once
#include "Base.h"
#define ERROR0 "No Error"
#define ERROR1 "FATAL ERROR! File is not opened!"
#define ERROR2 "FATAL ERROR! Scanning file is not opened!"
#define ERROR3 "ERROR! Comment block used incorrectly in line "
#define ERROR4 "ERROR! The number is used incorrectly in line "
#define ERROR5 "ERROR! The incorrect number of dots is used in writing a real
number in line "
#define ERROR6 "ERROR! Invalid operation in line "
#define ERROR7 "ERROR! Invalid name of variable in line "
#define ERROR8 "ERROR! Failed to add to table in line "
#define ERROR9 "ERROR! Invalid type of variable in line "
#define ERROR10 "ERROR! Unknown name in line "
#define ERROR11 "ERROR! Incorrect use of operation in line "
#define ERROR12 "ERROR! Incorrect variable assignment in line "
#define ERROR13 "ERROR! Performing operations on an uninitialized variable in line
"
#define ERROR14 "ERROR! Overwriting a Variable in line "
#define ERROR15 "ERROR! Using a reserved keyword to indicate a variable name in
line "
#define ERROR16 "ERROR! Invalid symbol in line "

#ifndef ERROR

enum Code_Errors {
    NO_ERROR = 0,
    FATALERR_FAILLEDREADFILE,
    FATALERR_FAILLEDREADSCANFILE,
    ERR_BLOCKCOMMENT,
    ERR_INVALID_INT_NUM,
    ERR_INVALID_FLOAT_NUM,
```

```
    ERR_INVALID_OPERATION,
    ERR_INVALID_NAME,
    ERR_FAILED_ADD_TABLE,
    ERR_INVALID_TYPE_VAR,
    ERR_UNKNOWN_VARIABLE,
    ERR_INCORRECT_USE_OPERATION,
    ERR_INVALID_ASSIGNMENT_OPERATION,
    ERR_PERFORMING_OPERATION_UNINITIALIZED_VAR,
    ERR_VAR_OVERRIDES,
    ERR_KEYWORD_IN_VAR_NAME,
    ERR_INVALID_SYMBOL
};

class Errors {

public:
    int NUM_LINE = 0;
    Code_Errors CODE = NO_ERROR;
    Errors() {}
    Errors(int num_line, Code_Errors error) {
        NUM_LINE = num_line;
        CODE = error;
    }
    inline string info() {
        switch (CODE)
        {
        case NO_ERROR:
            return ERROR0;
        case FATALERR_FAILLEDREADFILE:
            return ERROR1;
        case FATALERR_FAILLEDREADSCANFILE:
            return ERROR2;
        case ERR_BLOCKCOMMENT:
            return ERROR3 + to_string(NUM_LINE) + ".";
        case ERR_INVALID_INT_NUM:
            return ERROR4 + to_string(NUM_LINE) + ".";
        case ERR_INVALID_FLOAT_NUM:
            return ERROR5 + to_string(NUM_LINE) + ".";
        case ERR_INVALID_OPERATION:
            return ERROR6 + to_string(NUM_LINE) + ".";
        case ERR_INVALID_NAME:
            return ERROR7 + to_string(NUM_LINE) + ".";
        case ERR_FAILED_ADD_TABLE:
            return ERROR8 + to_string(NUM_LINE) + ".";
        case ERR_INVALID_TYPE_VAR:
            return ERROR9 + to_string(NUM_LINE) + ".";
        case ERR_UNKNOWN_VARIABLE:
            return ERROR10 + to_string(NUM_LINE) + ".";
        case ERR_INCORRECT_USE_OPERATION:
            return ERROR11 + to_string(NUM_LINE) + ".";
        case ERR_INVALID_ASSIGNMENT_OPERATION:
```

```
            return ERROR12 + to_string(NUM_LINE) + ".";
        case ERR_PERFORMING_OPERATION_UNINITIALIZED_VAR:
            return ERROR13 + to_string(NUM_LINE) + ".";
        case ERR_VAR_OVERRIDES:
            return ERROR14 + to_string(NUM_LINE) + ".";
        case ERR_KEYWORD_IN_VAR_NAME:
            return ERROR15 + to_string(NUM_LINE) + ".";
        case ERR_INVALID_SYMBOL:
            return ERROR16 + to_string(NUM_LINE) + ".";
        default:
            break;
        }
    }

};

#endif // !
```

MT1.cpp

```
#include "Translator.h"
#define NAMEFILEERRORS "out_files/error.txt"
#define NAMEFILETOKEN "out_files/token.txt"
#define NAMEFILEIDENTIFICATORS "out_files/identificators.txt"
#define NAMEFILECONSTANS "out_files/constans.txt"
#define NAMEFILECONSTTABLE "out_files/consttables.txt"
int main() {
    setlocale(LC_ALL, "Russian");
    Translator _TL;
    _TL.ScanningFile("source.txt");
    ofstream FileToken(NAMEFILETOKEN);
    if (!_TL.outFileToken(FileToken)) {
        cerr << NAMEFILETOKEN << " file not opened!";
        return 15;
    }
    FileToken.close();
    ofstream FileErrors(NAMEFILEERRORS);
    if (!_TL.outFileErrors(FileErrors)) {
        cerr << NAMEFILEERRORS << " file not opened!";
        return 16;
    }
    FileErrors.close();
    ofstream FileIC(NAMEFILEIDENTIFICATORS);
    if (!_TL.outFileIC(FileIC)) {
        cerr << NAMEFILEIDENTIFICATORS << " file not opened!";
        return 17;
    }
    FileIC.close();
    ofstream FileC(NAMEFILECONSTANS);
    if (!_TL.outFileC(FileC)) {
        cerr << NAMEFILECONSTANS << " file not opened!";
        return 18;
    }
```

```
    FileC.close();
    ofstream FileCon(NAMEFILECONSTTABLE);
    if (!_TL.outConstTable(FileCon)) {
        cerr << NAMEFILECONSTTABLE << " file not opened!";
        return 19;
    }
    FileCon.close();
    return 0;
}
```

7. Тестовые примеры

1. Верный исходный код с комментариями.

Входной код:

```
/*
(-(-_(-_-)_-)-)
*/
int help = 184; // Код помощи
float angle_alpha = help & 10;
struct Name;
// Конец кода /*
```

Файл токенов:

```
2       1
5       5
6       7
3       2
2       0
5       22
5       5
0       0
6       8
3       2
2       2
5       25
3       2
```

Файл ошибок пуст.

2. Неверное употребление блочного комментария.

Входной код:

```
*/
/*
int temp = 54;
```

Файл токенов пустой.

Файл ошибок:

```
ERROR! Comment block used incorrectly in line 0.
ERROR! Comment block used incorrectly in line 3.
```

3. Не правильно употребление целых чисел и чисел с плавающей запятой

Входной код:

```
int gl = 43t;
float csa = 12.98;
float dim = .1;
int f = 09;
float d = 12.2.4;
```

Файл токенов:

```
2       1
5       1
```

2 0
5 11
6 18
3 2
2 0
5 14
2 1
5 12
2 0
5 10

Файл ошибок:

ERROR! The number is used incorrectly in line 0.
ERROR! Invalid symbol in line 2.
ERROR! The number is used incorrectly in line 3.
ERROR! The incorrect number of dots is used in writing a real number in line 4.

4. Использование неинициализированных переменных. Использование необъявленных переменных.

Входной код:

```
temp = 89;
int dim;
dim += 34;
```

Файл токенов:

2 1
5 14
3 2
5 14

Файл ошибок:

ERROR! Unknown name in line 0.
ERROR! Performing operations on an uninitialized variable in line 2.

5. Использование неизвестных ключевых слов. Использование неизвестных операций.

Входной файл:

```
double glow = 0.4;
size_t elem = 3;
int t = 34 % 2;
```

Файл токенов:

2 1
5 26
6 13

Файл ошибок:

ERROR! Unknown name in line 0.
ERROR! Unknown name in line 1.
ERROR! Invalid operation in line 2.

6. Использование некорректный идентификатор

Входной файл:

```
int float = 45;
float 5oper = 12.3;
int git-lab = 89;
float dA021;
```

Файл токенов:

2 1
2 0
2 1
2 0
5 12

3 2

Файл ошибок:

ERROR! Using a reserved keyword to indicate a variable name in line 0.
ERROR! Invalid name of variable in line 1.
ERROR! Invalid name of variable in line 2.

7. Более сложные варианты комментариев.

Входной код:

```
/* Hello */  /* my name */
/*
 //
 //
*/
/* global */ int global = 2;
```

Файл токенов:

2 1
5 25
6 20
3 2

Файл ошибок пуст.