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»

Студенты ИСАКИН ДАНИИЛ

ВОСТРЕЦОВА ЕКАТЕРИНА

Группа ПМ-13

Бригада 10

Преподаватель ДВОРЕЦКАЯ ВИКТОРИЯ КОНСТАНТИНОВНА

Новосибирск, 2024

1 Цель

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

2 Задание

1) Подмножество языка C++ включает:
   - данные типа **int**;
   - инструкции описания переменных;
   - операторы присваивания в любой последовательности;
   - полный набор арифметических, логических операций и операций сравнения.

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 Структура входных и выходных файлов

Входные данные:

source.txt – файл с анализируемым кодом

table_letters.txt – файл с допустимыми буквами

table_numbers.txt – файл с допустимыми цифрами

table_operations.txt – файл с допустимыми операторами

table_keywords.txt - файл с служебными словами

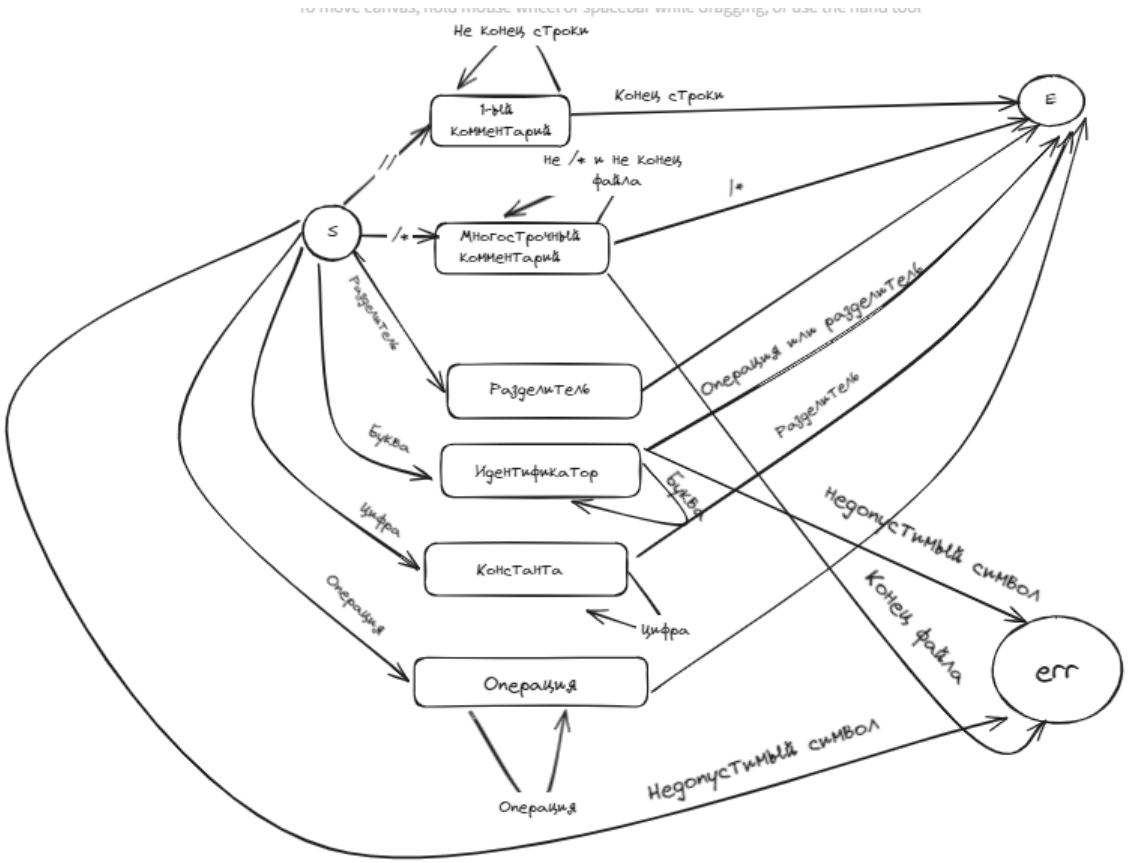
table_separators.txt – файл с разрешимыми операциями

Выходные данные:

errors.txtм- файл с ошибками

token.txt – файл с токенами

4 Детерменированный конечный автомат



5 Алгоритм разбора

1. Считать строку. Если конец файла – перейти к шагу 9.

2. Очистить строку от комментариев.

3. Анализировать первые два символа. Если первый символ – разделитель, перейти к шагу 7; если первый символ или первые два символа – операция, перейти к шагу 6; если первый символ – буква, перейти к шагу 4; если первый символ – цифра, перейти к шагу 5; если строка пуста – к шагу 1; иначе перейти к шагу 8.

4. Выделить идентификатор путем добавления к первому символу всех последующих букв и цифр. Если идентификатор – ключевое слово, сформировать и вывести соответствующий токен, иначе добавить идентификатор в таблицу идентификаторов и вывести соответствующий токен. За строку считать строку после идентификатора и перейти к шагу 3.

5. Выделить константу путем добавления к первому символу всех последующих цифр и/или одной точки. Если после константы нет разделителя или знака операции, перейти к шагу 8; иначе сформировать и вывести соответствующий токен для константы. За строку считать строку после константы и перейти к шагу 3.

6. Выделить одно- или двухсимвольную операцию, сформировать и вывести соответствующий токен. За строку считать строку после операции и перейти к шагу 3.

7. Выделить разделитель, сформировать и вывести соответствующий токен. За строку считать строку после разделителя и перейти к шагу 3.

8. Ошибка, вывести соответствующее сообщение и прекратить разбор.

9. Конец.

6 Тесты

6.1 Не закрытый комментарий

Исходный код

```
int main()
{
/*
   return 0;
}
```

Файл токенов	Файл с ошибками	Содержание таблиц
3 1 -1 3 2 -1 4 0 -1 4 1 -1 4 5 -1	Error: incorrect coment Error in string 3: /*	Identificators: Consts:

6.2 Недопустимый символ

Исходный код

```
@int main()
{
/*
   return 0;
}*/
```

Файл токенов	Файл с ошибками	Содержание таблиц
	Error: can`t determine symbol "@" Error in string 1: @int main()	Identificators: Consts:

6.3 Недопустимый символ в названии переменной

Исходный код

```
int main()
{
   a@343=5;
   return 0;
}
```

Файл токенов	Файл с ошибками	Содержание таблиц
3 1 -1 3 2 -1 4 0 -1 4 1 -1	Error: can`t determine symbol "@" Error in string 4: a@343=5;	Identificators: 97: [a Type = notype Dim=1 Init=[0]] Consts:

4 5 -1 5 97 0		

6.4 Некорректное число

Исходный код

```
int main()
{
   a=5a343;
   return 0;
}
```

Файл токенов	Файл с ошибками	Содержание таблиц
3 1 -1 3 2 -1 4 0 -1 4 1 -1 4 5 -1 5 97 0 4 11 -1	Error: incorrect constant Error in string 4: a=5a343;	Identificators: 97: [a Type = notype Dim=1 Init=[0]] Consts:

6 5 Комментарии обоих видов

Исходный код

```
int main()
{
   a=5343;
/*
abcdifgh
*/
   return 0;
//qwerty
}
```

Файл токенов	Файл с ошибками	Содержание таблиц
3 1 -1 3 2 -1 4 0 -1 4 1 -1 4 5 -1 5 97 0 4 11 -1 6 7 -1 4 2 -1		Identificators: 97: [a Type = notype Dim=1 Init=[0]] Consts: 7: [5343 Type = no-type Dim=1 Init=[0]] 48: [0 Type = notype Dim=1 Init=[0]]

3 3 -1 6 48 -1 4 2 -1		

6.6 С возможными ситуациями

Исходный код

```
int main()
{
   a=5343;
/*
abcdifgh
*/
   return 0;
//qwerty
   b++;
}
```

Файл токенов	Файл с ошибками	Содержание таблиц
3 1 -1 3 2 -1 4 0 -1 4 1 -1 4 5 -1 5 97 0 4 11 -1 6 7 -1 4 2 -1 3 3 -1 6 48 -1 4 2 -1 5 98 0 4 4 -1 4 2 -1		Identificators: 97: [a Type = notype Dim=1 Init=[0]] 98: [b Type = notype Dim=1 Init=[0]] Consts: 7: [5343 Type = no-type Dim=1 Init=[0]] 48: [0 Type = notype Dim=1 Init=[0]]

7 Текст программы

Lexeme.cpp

```
#include "Lexeme.h"

// Конструктор по умолчанию
Lexeme::Lexeme() {}

// Конструктор с заданием имени идентификатора или значения константы
```

```
Lexeme::Lexeme(string new_name)
{
   Name = new_name;
   Type = 0;
   IsInit.push_back(false);
   Dimension = 1;
}

// Деструктор
Lexeme::~Lexeme()
{
   IsInit.clear();
}
```

main.cpp

```
#include <iostream>
#include <stdio.h>
#include "TableConst.h"
#include "TableVar.h"
#include "translator.h"

using namespace std;

int main()
{
   /*TableConst<string> a;
   a.ReadFile("reserved_words.txt");
   cout << "a.Contains(\"int\") = " << a.Contains("int") << endl;
   cout << "a.Contains(\"double\") = " << a.Contains("double") << endl;

   int num;
   a.GetNum("return", num);
   cout << "a.GetNum(\"return\", num): num = " << num << endl;

   string str;
   a.GetVal(num, str);
   cout << "a.GetVal(num, str): str = " << str << endl;

   TableVar b;
   b.Add("avriable");
   b.Add("vairable");
   b.Add("vairalbe");
   b.Add("variable");
   int hash, Chain;
   b.GetLocation("variable", hash, Chain);
   cout << "b.GetLocation(\"variable\", hash, Chain): hash = " << hash << "  Chain = " << Chain << endl;

   b.SetType("variable", 2);
   b.SetDimension("variable", 3);
   b.SetIsInit("variable", true);
   b.SetIsInit("variable", false, 1);
   b.SetIsInit("variable", true, 2);
   lexeme c;
   b.GetLexeme("variable", c);

   cout << "c.Name = " << c.Name << endl;
   cout << "c.Type = " << c.Type << endl;
   cout << "c.IsInit[0] = " << c.IsInit[0] << endl;
   cout << "c.IsInit[1] = " << c.IsInit[1] << endl;
   cout << "c.IsInit[2] = " << c.IsInit[2] << endl;*/

   Translator a;
   cout << a.AnalyzeLexical("files/source.txt","files/tokens.txt","files/errors.txt");
   a.DebugPrint(cout);
   return 0;
```

```
}
```

TableVar.cpp

```
#include "TableVar.h"
// Размер хэш-таблицы по умолчанию
#define DefaultHashnum 100

// Подсчет хэша
int TableVar::GetHash(string Name)
{
   int hash = 0;
   for(int i = 0; i < static_cast<int>(Name.size()); i++)
      hash += Name[static_cast<size_t>(i)];
   return hash % HashNum;
}

// Подсчет номера в цепочке
int TableVar::GetChain(string Name)
{
   for(int i = 0, h = GetHash(Name); i < static_cast<int>(Table[h].size()); i++)
      if(Name == Table[static_cast<size_t>(h)][static_cast<size_t>(i)].Name) return i;
   return -1;
}

// Конструктор с размером таблицы по умолчанию
TableVar::TableVar()
{
   HashNum=DefaultHashnum;
   Table = new vector<Lexeme> [HashNum];
}

// Конструктор с пользовательским размером таблицы
TableVar::TableVar(int new_hashnum)
{
   HashNum=new_hashnum;
   Table = new vector<Lexeme> [HashNum];
}

// Деструктор
TableVar::~TableVar()
{
   for(int i = 0; i < HashNum; i++)
      Table[i].clear();
   delete [] Table;
}

// Проверка есть ли элемент в таблице
inline bool TableVar::Contains(string Name)
{
   if(GetChain(Name) != -1) return true;
   return false;
}

// Добавление нового имени идентификатора или значения константы
bool TableVar::Add(string Name)
{
   if(Contains(Name)) return false;
   int h = GetHash(Name);
   Table[h].push_back(Lexeme(Name));
   return true;
}

// Задание типа по хэшу и номеру в цепочке
bool TableVar::SetType(int hash, int Chain, int Type)
```

```
{
    if(Chain == -1) return false;
    Table[static_cast<size_t>(hash)][static_cast<size_t>(Chain)].Type = Type;
    return true;
}

// Задание типа по имени идентификатора или значению константы
bool TableVar::SetType(string Name, int Type)
{
    int hash = GetHash(Name), Chain = GetChain(Name);
    return SetType(hash, Chain, Type);
}

// Задание размерности по хэшу и номеру в цепочке
bool TableVar::SetDimension(int hash, int Chain, int Dimension)
{
    if(Chain == -1) return false;
    Table[static_cast<size_t>(hash)][static_cast<size_t>(Chain)].Dimension = Dimension;
    Table[static_cast<size_t>(hash)][static_cast<size_t>(Chain)].IsInit.resize(static_cast<size_t>(Dimension));
    for(int i = 0; i < Dimension; i++)
        Table[static_cast<size_t>(hash)][static_cast<size_t>(Chain)].IsInit[static_cast<size_t>(i)] = false;
    return true;
}

// Задание размерности по имени идентификатора или значению константы
bool TableVar::SetDimension(string Name, int Dimension)
{
    int hash = GetHash(Name), Chain = GetChain(Name);
    return SetDimension(hash, Chain, Dimension);
}

// Задание флага инициализации для массивов по хэшу и номеру в цепочке
bool TableVar::SetIsInit(int hash, int Chain, bool is_init, int init_index)
{
    if(Chain == -1) return false;
    Table[static_cast<size_t>(hash)][static_cast<size_t>(Chain)].IsInit[static_cast<size_t>(init_index)] = is_init;
    return true;
}

// Задание флага инициализации для массивов по имени идентификатора или значению константы
bool TableVar::SetIsInit(string Name, bool is_init, int init_index)
{
    int hash = GetHash(Name), Chain = GetChain(Name);
    return SetIsInit(hash, Chain, is_init, init_index);
}

// Задание флага инициализации по хэшу и номеру в цепочке
bool TableVar::SetIsInit(int hash, int Chain, bool is_init)
{
    return SetIsInit(hash, Chain, is_init, 0);
}

// Задание флага инициализации по имени идентификатора или значению константы
bool TableVar::SetIsInit(string Name, bool is_init)
{
    return SetIsInit(Name, is_init, 0);
}

// Определение хэша и номера в цепочке
bool TableVar::GetLocation(string Name, int &hash, int &Chain)
{
    int h = GetHash(Name), c = GetChain(Name);
    if(Chain == -1) return false;
    hash = h;
    Chain = c;
    return true;
```

```
}

// Получение структуры lexeme по хэшу и номеру в цепочке
bool TableVar::GetLexeme(int hash, int Chain, Lexeme &lexeme)
{
   if(Chain == -1) return false;
   lexeme = Table[static_cast<size_t>(hash)][static_cast<size_t>(Chain)];
   return true;
}

// Получение структуры lexeme по имени идентификатора или значению константы
bool TableVar::GetLexeme(string Name, Lexeme &lexeme)
{
   int hash = GetHash(Name), Chain = GetChain(Name);
   return GetLexeme(hash, Chain, lexeme);
}

// Отладочная печать
void TableVar::DebugPrint(ostream& stream)
{
   for(int i = 0; i < HashNum; i++)
   {
      if(Table[i].size())
      {
         stream << i << ":\t[ ";
         for(int j = 0; j < static_cast<int>(Table[i].size()); j++)
         {
            stream << Table[static_cast<size_t>(i)][static_cast<size_t>(j)].Name;
            switch(Table[static_cast<size_t>(i)][static_cast<size_t>(j)].Type)
            {
            case 0:
               stream << "\tType = notype\tDim=";
               break;
            case 1:
               stream << "\tType = int\tDim=";
               break;
            }
            stream << Table[static_cast<size_t>(i)][static_cast<size_t>(j)].Dimension << "\tInit=[";
            for(int k = 0; k < Table[static_cast<size_t>(i)][static_cast<size_t>(j)].Dimension; k++)
            {
               stream << Table[static_cast<size_t>(i)][static_cast<size_t>(j)].IsInit[static_cast<size_t>(k)];
               if(k != Table[static_cast<size_t>(i)][static_cast<size_t>(j)].Dimension - 1) stream << " ";
            }
            if(j != static_cast<int>(Table[static_cast<size_t>(i)].size() - 1)) stream << "],\t";
            else stream << "]";
         }
         stream << " ]" << endl;
      }
   }
}

#undef  DefaultHashnum
```

Token.cpp

```
#include "Token.h"

Token::Token() { }

Token::Token(int table_, int place_, int chain_)
{
   Table = table_;
   Place = place_;
```

```
    Chain = chain_;
}

istream& operator >> (istream& istream_, Token& token_)
{
    istream_ >> token_.Table >> token_.Place >> token_.Chain;
    return istream_;
}

ostream& operator << (ostream& ostream_, const Token& token_)
{
    ostream_ << token_.Table << " " << token_.Place << " " << token_.Chain << endl;
    return ostream_;
}
```

Translator.cpp

```
#include "Translator.h"

// Конструктор со вводом постоянных таблиц
Translator::Translator()
{
    Letters.ReadFile("files/table_letters.txt");
    Numbers.ReadFile("files/table_numbers.txt");
    Operations.ReadFile("files/table_operations.txt");
    KeyWords.ReadFile("files/table_keywords.txt");
    Separators.ReadFile("files/table_separators.txt");
}

// Лексический анализ
bool Translator::AnalyzeLexical(string file_source, string file_tokens, string file_error)
{
    in_source.open(file_source.c_str(), ios::in);
    out_token.open(file_tokens.c_str(), ios::out);
    out_error.open(file_error.c_str(), ios::out);
    bool flag_error = false;
    // bool flag_coment = false;
    string str;
    AnalyzeLexicalStrnum = 1;
    while(!in_source.eof() && !flag_error)
    {
        getline(in_source, str);
        if(!in_source.eof())
        {
            AnalyzeLexicalStrinc = 0;
            string strold = str;
            if(!AnalyzeLexicalDecomment(str, true))
            {
                out_error << "Error in string " << AnalyzeLexicalStrnum << ": " << strold << endl;
                cout << "Error in string " << AnalyzeLexicalStrnum << ": " << strold << endl;
                return false;
            }
            AnalyzeLexicalStrnum += AnalyzeLexicalStrinc;
            flag_error = !AnalyzeLexicalString(str);
            if(flag_error)
            {
                out_error << "Error in string " << AnalyzeLexicalStrnum << ": " << str << endl;
                cout << "Error in string " << AnalyzeLexicalStrnum << ": " << str << endl;
            }
            AnalyzeLexicalStrnum ++;
        }
    }
    in_source.close();
    out_token.close();
    out_error.close();
```

```
    return !flag_error;
}

// Очистка от комментариев
bool Translator::AnalyzeLexicalDecomment(string& str, bool is_changed)
{
    if(str.size())
    {
        bool change = false;
        size_t index_c = str.find("//"), index_c1 = str.find("/*"), index_c2;
        if (index_c != string::npos && index_c < index_c1)
        {
            str.erase(index_c);
            change = true;
        }
        index_c1 = str.find("/*");
        index_c2 = str.find("*/");
        if(index_c2 < index_c1)
        {
            out_error << "Error: incorrect coment" << endl;
            cout << "Error: incorrect coment" << endl;
            return false;
        }
        while(index_c1 != string::npos && index_c2 != string::npos)
        {
            string tmpstr = str;
            str.erase(index_c1);
            tmpstr.erase(0, index_c2 + 2);
            str += tmpstr;
            index_c1 = str.find("/*");
            index_c2 = str.find("*/");
            change = true;
        }
        index_c1 = str.find("/*");
        index_c2 = str.find("*/");
        if(index_c1 != string::npos && index_c2 == string::npos)
        {
            str.erase(index_c1);
            string tmpstr;
            if(!in_source.eof())
            {
                getline(in_source, tmpstr);
                AnalyzeLexicalStrinc++;
            }
            else
            {
                out_error << "Error: incorrect coment" << endl;
                cout << "Error: incorrect coment" << endl;
                return false;
            }
            while(tmpstr.find("*/") == string::npos)
            {
                if(!in_source.eof())
                {
                    getline(in_source, tmpstr);
                    AnalyzeLexicalStrinc++;
                }
                else
                {
                    out_error << "Error: incorrect coment" << endl;
                    cout << "Error: incorrect coment" << endl;
                    return false;
                }
            }
            index_c2 = tmpstr.find("*/");
            tmpstr.erase(0, index_c2 + 2);
```

```
            str += " " + tmpstr;
            change = true;
        }
        index_c1 = str.find("/*");
        index_c2 = str.find("*/");
        if((index_c1 != string::npos && index_c2 == string::npos) ||
            (index_c1 == string::npos && index_c2 != string::npos))
        {
            out_error << "Error: incorrect coment" << endl;
            cout << "Error: incorrect coment" << endl;
            return false;
        }
        if(is_changed)
            return AnalyzeLexicalDecomment(str, change);
    }
    return true;
}

// Анализ строки
bool Translator::AnalyzeLexicalString(string str)
{
    Trim(str);
    bool flag_error = false;
    if(str.size())
    {
        cout << "AnalyzeLexicalString started with str = "<< str << endl;
        char sym_1 = str[0], sym_2 = str[1];
        // Проверка первого символа
        string str_1, str_2;
        stringstream str_stream;
        str_stream << sym_1;
        str_1 = str_stream.str();
        str_stream << sym_2;
        str_2 = str_stream.str();
        int first_sym_type = -1;
        if(Letters.Contains(sym_1))
            first_sym_type = 0;
        if(Numbers.Contains(sym_1) || sym_1 == '-')
            first_sym_type = 1;
        if(Operations.Contains(str_1) || Operations.Contains(str_2))
            first_sym_type = 2;
        if(Separators.Contains(sym_1))
            first_sym_type = 3;

        switch(first_sym_type)
        {
        /*case -1: // Недопустимый символ
        {
            out_error << "Error: unresolved first symbol" << endl;
            cout << "Error: unresolved first symbol" << endl;
            return false;
        }
        break;*/
        case 0: // Идентификатор
        {
            // Получим полное название идентификатора
            string idname = str;
            int i;
            bool finded = false;
            for(i = 1; i < static_cast<int>(idname.size()) && !finded; i++)
                finded = !(Letters.Contains(str[static_cast<size_t>(i)]) || Numbers.Contains(str[static_cast<size_t>(i)]));
            if(finded)
            {
                idname.erase(static_cast<size_t>( i - 1 ));
                str.erase(0, static_cast<size_t>( i - 1 ));
            }
```

```
            else
                str.erase(0);
            Trim(idname);
            Trim(str);
            if(KeyWords.Contains(idname)) // Если ключевое слово
            {
                if(KeyWords.GetNum(idname, i))
                    out_token << Token(3, i, -1);
            }
            else // Иначе в таблицу идентификаторов
            {
                Identifiers.Add(idname);
                int Table, Chain;
                Identifiers.GetLocation(idname, Table, Chain);
                out_token << Token(5, Table, Chain);
            }
            return AnalyzeLexicalString(str);
        }
        break;
        case 1:  // Константа
        {
            string constval = str;
            int i;
            bool finded = false;
            for(i = 1; i < static_cast<int>(constval.size()) && !finded; i++)
                finded = !(Numbers.Contains(str[static_cast<size_t>(i)]) || str[static_cast<size_t>(i)] == '.' ||
str[static_cast<size_t>(i)] == ' ');
            string str_t1, str_t2;
            stringstream str_stream_t;
            str_stream_t << str[static_cast<size_t>(i-1)];
            str_t1 = str_stream_t.str();
            str_stream_t << str[static_cast<size_t>(i)];
            str_t2 = str_stream_t.str();
            if(!Operations.Contains(str_t1) && !Operations.Contains(str_t2) && !Separators.Contains(str[static_cast<size_t>(i-
1)]))
            {
                out_error << "Error: incorrect constant" << endl;
                cout << "Error: incorrect constant" << endl;
                return false;
            }
            if(finded)
            {
                constval.erase(static_cast<size_t>(i-1));
                str.erase(0, static_cast<size_t>(i-1));
            }
            else
                str.erase(0);
            Trim(constval);
            Trim(str);
            if(constval.find_last_of('.') - constval.find_first_of('.') != 0)
            {
                out_error << "Error: incorrect constant" << endl;
                cout << "Error: incorrect constant" << endl;
                return false;
            }
            else
            {
                Constants.Add(constval);
                int Table, Chain;
                Identifiers.GetLocation(constval, Table, Chain);
                out_token << Token(6, Table, Chain);
            }
            return AnalyzeLexicalString(str);
        }
        break;
        case 2: // Операция
```

```
        {
            int Table;
            if(Operations.Contains(str_2)) // Двухсимвольная
            {
                Operations.GetNum(str_2, Table);
                out_token << Token(4, Table, -1);
                str.erase(0, 2);
                Trim(str);
                return AnalyzeLexicalString(str);
            }
            if(Operations.Contains(str_1)) // Односимвольная
            {
                Operations.GetNum(str_1, Table);
                out_token << Token(4, Table, -1);
                str.erase(0, 1);
                Trim(str);
                return AnalyzeLexicalString(str);
            }
        }
        break;
        case 3: // Разделитель
        {
            int Table;
            Separators.GetNum(str[0], Table);
            out_token << Token(4, Table, -1);
            str.erase(0, 1);
            Trim(str);
            return AnalyzeLexicalString(str);
        }
        break;
        default: // Непонятно что
        {
            out_error << "Error: can`t determine symbol \"" << str_1 << "\"" << endl;
            cout << "Error: can`t determine symbol \"" << str_1 << "\"" << endl;
            return false;
        }
        break;
        }
    }
    return !flag_error;
}

// Отладочный вывод таблиц
void Translator::DebugPrint(ostream& stream)
{
    stream << "ID`s:" << endl;
    Identifiers.DebugPrint(stream);
    stream << "CONST`s:" << endl;
    Constants.DebugPrint(stream);
}
```

Lexeme.h

```
#ifndef LEXEME_H_
#define LEXEME_H_

#include <string>
#include <vector>

using namespace std;

// Класс для хранения идентификаторов и констант
class Lexeme
{
public:
```

```
    // Имя идентификатора или значение константы
    string Name;

    // Тип, 0 - не определен, 1 - int
    int Type;

    // Массив флагов "илициализировано ли" размерности Dimension
    vector<bool> IsInit;

    // Размерность массива, для переменных и констант - 1.
    int Dimension;

    // Конструктор по умолчанию
    Lexeme();

    // Конструктор с заданием имени идентификатора или значения константы
    Lexeme(string new_name);

    // Оператор присваивания
    Lexeme &operator = (const Lexeme &other)
    {
        if(this != &other)
        {
            Name = other.Name;
            Type = other.Type;
            Dimension = other.Dimension;
            IsInit = other.IsInit;
        }
        return *this;
    }

    // Деструктор
    ~Lexeme();

};

#endif
```

TableConst.h

```
#ifndef TABLE_CONST_H_
#define TABLE_CONST_H_

#include <fstream>
#include <string>
#include <set>

using namespace std;

// Класс постоянных таблиц
template <typename Type> class TableConst
{
private:
    set<Type> Table;
public:
    // Конструктор по умолчанию
    TableConst() {}


    // Добавление элемента в таблицу
    inline void Add(Type elem)
    {
        Table.insert(elem);
```

```
    }

    // Чтение таблицы из файла
    bool ReadFile(string Name)
    {
        ifstream fs(Name.c_str(), ios::in);
        if(!fs.is_open()) return false;
        Type elem;
        while (!fs.eof())
        {
            fs >> elem;
            Add(elem);
        }
        return true;
    }
    // Проверка есть ли элемент в таблице
    bool Contains(Type elem)
    {
        typename set<Type>::iterator it = Table.find(elem);
        if(it == Table.end()) return false;
        return true;
    }
    // Поиск номера по значению
    bool GetNum(Type elem, int &num)
    {
        if(!Contains(elem)) return false;
        num = static_cast<int>(distance(Table.begin(), Table.find(elem)));
        return true;
    }

    // Поиск значения по номеру
    bool GetVal(int num, Type &elem)
    {
        if(num < 0 || num >= Table.size()) return false;
        typename set<Type>::iterator it = Table.begin();
        for(int i = 0; i < num; i++)
            it++;
        elem = *it;
        return true;
    }

    // Деструктор
    ~TableConst()
    {
        Table.clear();
    }
};

#endif
```

TableVar.h

```
#ifndef TABLE_VAR_H_
#define TABLE_VAR_H_

#include <fstream>
#include <string>
#include <vector>
#include "Lexeme.h"

using namespace std;

// Класс переменных таблиц
```

```
class TableVar
{
private:
    // Размер таблицы
    int HashNum;

    // Указатель на массив цепочек
    vector<Lexeme> *Table;

    // Подсчет хэша
    int GetHash(string Name);

    // Подсчет номера в цепочке
    int GetChain(string Name);
public:
    // Конструктор с размером таблицы по умолчанию
    TableVar();

    // Конструктор с пользовательским размером таблицы
    TableVar(int _hashnum);

    // Определение хэша и номера в цепочке
    bool GetLocation(string Name, int &hash, int &Chain);

    // Проверка есть ли элемент в таблице
    inline bool Contains(string Name);

    // Добавление нового имени идентификатора или значения константы
    bool Add(string Name);

    // Задание типа по хэшу и номеру в цепочке
    bool SetType(int hash, int Chain, int Type);

    // Задание типа по имени идентификатора или значению константы
    bool SetType(string Name, int Type);

    // Задание размерности по хэшу и номеру в цепочке
    bool SetDimension(int hash, int Chain, int Dimension);

    // Задание размерности по имени идентификатора или значению константы
    bool SetDimension(string Name, int Dimension);

    // Задание флага инициализации по хэшу и номеру в цепочке
    bool SetIsInit(int hash, int Chain, bool IsInit);

    // Задание флага инициализации по имени идентификатора или значению константы
    bool SetIsInit(string Name, bool IsInit);

    // Задание флага инициализации для массивов по хэшу и номеру в цепочке
    bool SetIsInit(int hash, int Chain, bool IsInit, int init_index);

    // Задание флага инициализации для массивов по имени идентификатора или значению константы
    bool SetIsInit(string Name, bool IsInit, int init_index);

    // Получение структуры lexeme по хэшу и номеру в цепочке
    bool GetLexeme(int hash, int Chain, Lexeme &lexeme);

    // Получение структуры lexeme по имени идентификатора или значению константы
    bool GetLexeme(string Name, Lexeme &lexeme);

    // Отладочная печать
    void DebugPrint(ostream& stream);

    // Деструктор
    ~TableVar();
};
```

```
#endif
```

Token.h

```
#ifndef TOKEN_H_
#define TOKEN_H_
#include <iostream>
#include <fstream>

using namespace std;

// Класс токенов
class Token
{
public:
   int Table; // Номер таблицы
   int Place; // Положение в таблице
   int Chain; // Положение в цепочке
   // Конструкторы
   Token();
   Token(int table_, int place_, int chain_);
   // Ввод-вывод токенов
   friend istream& operator >> (istream& istream_, Token& token_);
   friend ostream& operator << (ostream& ostream_, const Token& token_);
};

#endif // TOKEN_H_INCLUDED
```

Translator.h

```
#ifndef TRANSLATOR_H_
#define TRANSLATOR_H_

#include <iostream>
#include <stdio.h>
#include <sstream>
#include <fstream>
#include <string>
#include "TableConst.h"
#include "TableVar.h"
#include "Lexeme.h"
#include "Token.h"

using namespace std;

class Translator
{
private:
   // Постоянные таблицы
   TableConst<char> Letters;      // 0
   TableConst<char> Numbers;      // 1
   TableConst<string> Operations; // 2
   TableConst<string> KeyWords;   // 3
   TableConst<char> Separators;   // 4

   // Переменные таблицы
   TableVar Identifiers;          // 5
   TableVar Constants;            // 6
```

```
    // Файловые потоки
    ifstream in_source;
    ofstream out_token;
    ofstream out_error;

    // Счетчики для подробных сообщений об ошибке
    int AnalyzeLexicalStrnum, AnalyzeLexicalStrinc;

    // Анализ строки
    bool AnalyzeLexicalString(string str);

    // Удаление комментариев
    bool AnalyzeLexicalDecomment(string& str, bool is_changed);


    // Удаление пробелов
    static inline void Ltrim(string& out_)
    {
        int notwhite = static_cast<int>(out_.find_first_not_of(" \t\n"));
        out_.erase(0, static_cast<size_t>(notwhite));
    }

    static inline void Rtrim(string& out_)
    {
        int notwhite = static_cast<int>(out_.find_last_not_of(" \t\n"));
        out_.erase(static_cast<size_t>(notwhite + 1));
    }

    static inline void Trim(string& out_)
    {
        Ltrim(out_);
        Rtrim(out_);
    }

public:
    // Конструктор со вводом постоянных таблиц
    Translator();

    // Лексический анализ
    bool AnalyzeLexical(string file_source, string file_tokens, string file_error);

    // Отладочный вывод таблиц
    void DebugPrint(ostream& stream);

    ~Translator() = default;
};

#endif
```