|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| 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.

1. **Задание**
2. Подмножество языка С++ 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 С++ 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

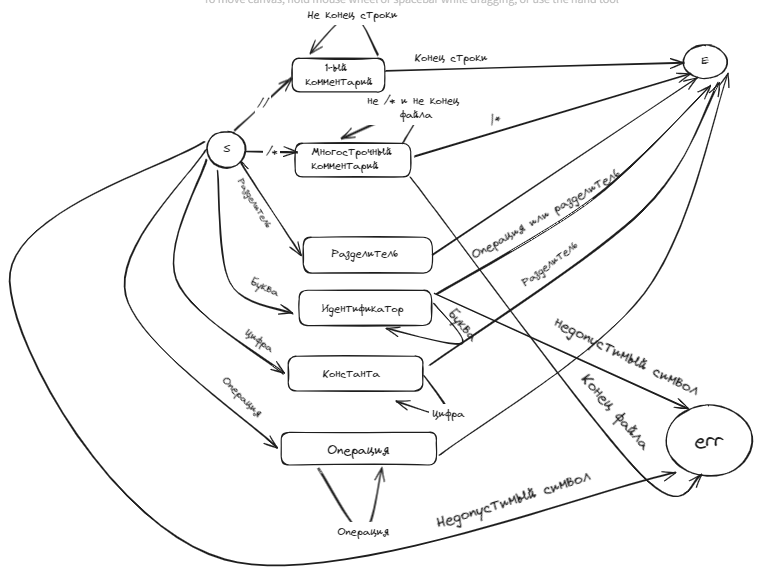
table\_separators.txt – файл с разрешимыми операциями

**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  4 5 -1  5 97 0 | Error: can`t determine symbol "@"  Error in string 4: a@343=5; | Identificators:  97: [ a Type = notype Dim=1 Init=[0] ]  Consts: |

**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  3 3 -1  6 48 -1  4 2 -1 |  | Identificators:  97: [ a Type = notype Dim=1 Init=[0] ]  Consts:  7: [ 5343 Type = notype Dim=1 Init=[0] ]  48: [ 0 Type = notype Dim=1 Init=[0] ] |

**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 = notype 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