Министерство образования и науки Российской Федерации

Новосибирский государственный технический университет

Кафедра прикладной математики

Языки программирования и методы трансляции

Лабораторная работа №2

Факультет прикладной математики и информатики

Группа ПМ-01

Студенты Александров М.Е.

Жигалов П.С.

Преподаватели Еланцева И.Л.

Полетаева И.А.

Вариант 7

Новосибирск

2013

1. Цель работы

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

2. Задание

Подмножество языка С++ включает:

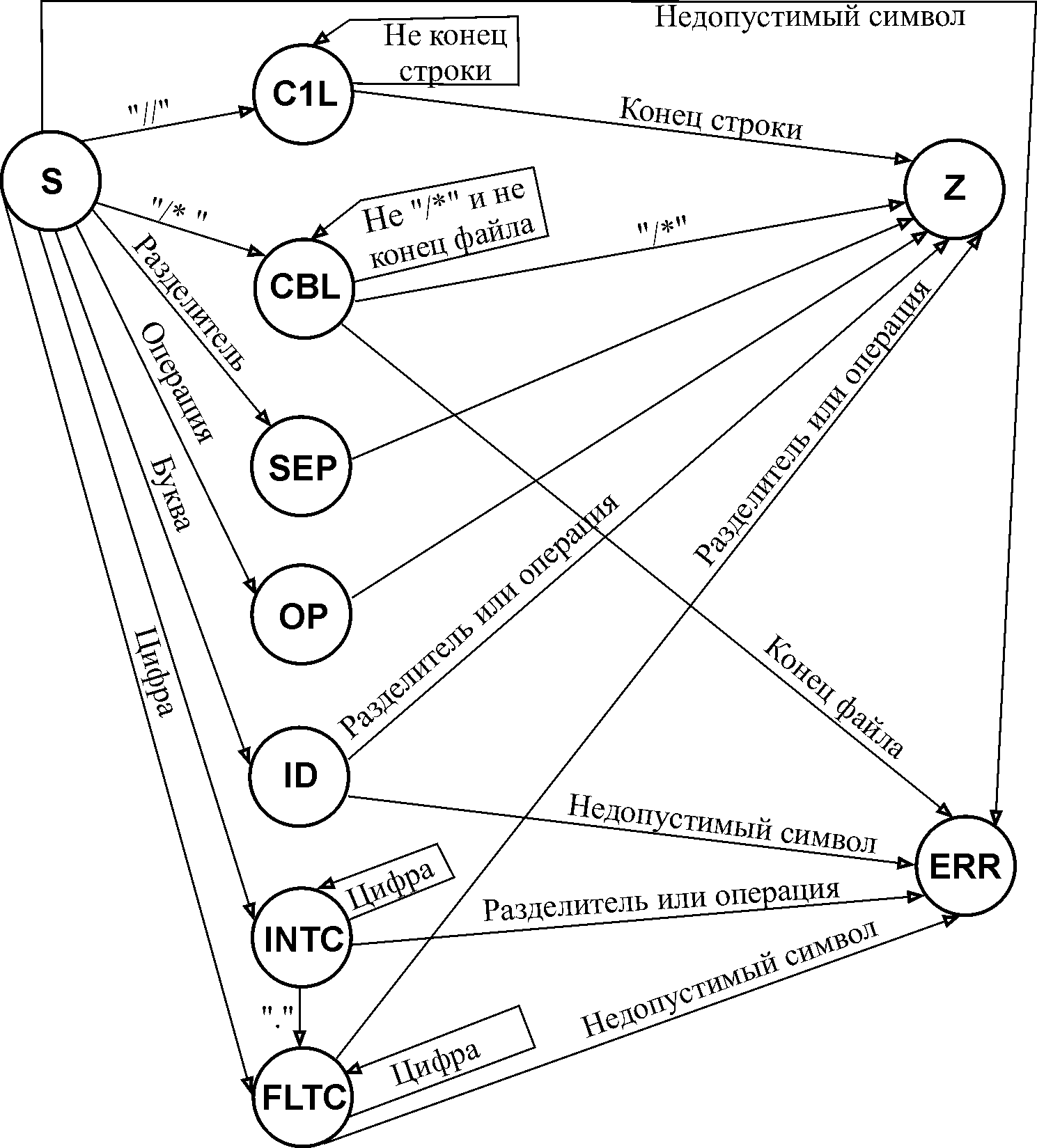
* данные типа **int, float, массивы** из элементов указанных типов;
* инструкции описания переменных;
* операторы присваивания в любой последовательности;
* операции **+, – , \*, = =, != , <, >** .

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 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 данных

Входные данные представляют собой имена файлов: содержащего исходный код, файла токенов и файла ошибок, а также файлы, содержащие допустимые буквы, числа, операции, ключевые слова и разделители. Результатом работы программы являются два файла – файл токенов и файл ошибок.

4. Детерминированный конечный автомат



Описание состояний

**S** – начальное состояние

**C1L** – однострочный комментарий

**CBL** – блочный комментарий

**SEP** – разделитель

**OP** – операция

**ID** – идентификатор или ключевое слово

**INTC** – целочисленная константа

**FLTC** – вещественная константа

**Z** – конечное состояние

**ERR** – ошибка

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, успешный рабор.

6. Тесты

6.1 Верный исходный код с разнообразным форматированием и стилем комментариев:

Исходный код

/\*\*

\* Doxygen-style

\* commentary

\*/

int main()

{

//34d;

int a = 0; //!< it is a

float b[2];

b[ 0 ] = 2.0 ;

b[1]=1.5;

a--;

/\* multi

string

comment

\*/

;

/\* \*/ /\* \*/ a += b[1] + b[0] /\* this is b1+b0 \*/ \* 2;

a++;

/\* // \*/

/\* //

Файл токенов

3 2 -1

3 3 -1

4 0 -1

4 1 -1

4 5 -1

3 2 -1

5 97 0

4 11 -1

6 48 -1

4 2 -1

3 1 -1

5 98 0

4 3 -1

6 48 -1

4 4 -1

4 2 -1

5 98 0

4 3 -1

6 48 -1

4 4 -1

4 11 -1

6 44 -1

4 2 -1

//

// \*/

return 0;

}

Содержимое таблиц

ID`s:

97: [ a notype dim=1 init={0} ]

98: [ b notype dim=1 init={0} ]

CONST`s:

44: [ 2.0 notype dim=1 init={0} ]

48: [ 0 notype dim=1 init={0},

1.5 notype dim=1 init={0} ]

49: [ 1 notype dim=1 init={0} ]

50: [ 2 notype dim=1 init={0} ]

5 98 0

4 3 -1

6 48 -1

4 4 -1

4 11 -1

6 44 -1

4 2 -1

5 97 0

4 7 -1

4 2 -1

4 2 -1

5 97 0

4 5 -1

5 98 0

4 3 -1

6 49 -1

4 4 -1

4 3 -1

5 98 0

4 3 -1

6 48 -1

4 4 -1

4 1 -1

6 50 -1

4 2 -1

5 97 0

4 4 -1

4 2 -1

3 4 -1

6 48 -1

4 2 -1

4 6 -1

6.2 Незакрытый комментарий

Исходный код

int main()

{

/\*

return 0;

}

Файл токенов

3 2 -1

3 3 -1

4 0 -1

4 1 -1

4 5 -1

Файл ошибок

Error: incorrect coment

Error in string 3: /\*

Содержимое таблиц

ID`s:

CONST`s:

6.3 Недопустимые символы в коде

Исходный код

int main()

{

@#@#@#@#

return 0;

}

Файл токенов

3 2 -1

3 3 -1

4 0 -1

4 1 -1

4 5 -1

Файл ошибок

Error: can`t determine symbol "@"

Error in string 3: @#@#@#@#

Содержимое таблиц

ID`s:

CONST`s:

6.4 Некорректный идентификатор

Исходный код

int main()

{

abc@d = 0;

return 0;

}

Файл токенов

3 2 -1

3 3 -1

4 0 -1

4 1 -1

4 5 -1

5 94 0

Файл ошибок

Error: can`t determine symbol "@"

Error in string 3: abc@d = 0;

Содержимое таблиц

ID`s:

94: [ abc notype dim=1 init={0} ]

CONST`s:

6.5 Некорректная константа

Исходный код

int main()

{

a = 12.0asbd;

return 0;

}

Файл токенов

3 2 -1

3 3 -1

4 0 -1

4 1 -1

4 5 -1

5 97 0

4 11 -1

Файл ошибок

Error: incorrect constant

Error in string 3: a = 12.0asbd;

Содержимое таблиц

ID`s:

97: [ a notype dim=1 init={0} ]

CONST`s:

6.6 Две точки в константе

Исходный код

int main()

{

a = 12.0.4;

return 0;

}

Файл токенов

3 2 -1

3 3 -1

4 0 -1

4 1 -1

4 5 -1

5 97 0

4 11 -1

Файл ошибок

Error: incorrect constant

Error in string 3: a = 12.0.4;

Содержимое таблиц

ID`s:

97: [ a notype dim=1 init={0} ]

CONST`s:

Текст программы

translator.h

#ifndef TRANSLATOR\_H\_INCLUDED

#define TRANSLATOR\_H\_INCLUDED

#include <iostream>

#include <stdio.h>

#include <sstream>

#include <fstream>

#include <string>

#include "table\_const.h"

#include "table\_var.h"

#include "lexeme.h"

#include "token.h"

using namespace std;

class translator

{

private:

// Постоянные таблицы

table\_const<char> letters; // 0

table\_const<char> numbers; // 1

table\_const<string> operations; // 2

table\_const<string> keywords; // 3

table\_const<char> separators; // 4

// Переменные таблицы

table\_var identifiers; // 5

table\_var constants; // 6

// Файловые потоки

ifstream in\_source;

ofstream out\_token;

ofstream out\_error;

// Анализ строки

bool analyze\_lexical\_string(string str);

// Удаление комментариев

bool analyze\_lexical\_decomment(string& str, bool is\_changed);

// Счетчики для подробных сообщений об ошибке

int analyze\_lexical\_strnum, analyze\_lexical\_strinc;

// Удаление пробелов

static inline void ltrim(string& out\_)

{

int notwhite = out\_.find\_first\_not\_of(" \t\n");

out\_.erase(0, notwhite);

}

static inline void rtrim(string& out\_)

{

int notwhite = out\_.find\_last\_not\_of(" \t\n");

out\_.erase(notwhite + 1);

}

static inline void trim(string& out\_)

{

ltrim(out\_);

rtrim(out\_);

}

public:

// Конструктор со вводом постоянных таблиц

translator();

// Лексический анализ

bool analyze\_lexical(string file\_source, string file\_tokens, string file\_error);

// Отладочный вывод таблиц

void debug\_print(ostream& stream);

};

#endif // TRANSLATOR\_H\_INCLUDED

translator.cpp

#include "translator.h"

// Конструктор со вводом постоянных таблиц

translator::translator()

{

letters.read\_file("files/table\_letters.txt");

numbers.read\_file("files/table\_numbers.txt");

operations.read\_file("files/table\_operations.txt");

keywords.read\_file("files/table\_keywords.txt");

separators.read\_file("files/table\_separators.txt");

}

// Лексический анализ

bool translator::analyze\_lexical(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;

analyze\_lexical\_strnum = 1;

while(!in\_source.eof() && !flag\_error)

{

getline(in\_source, str);

if(!in\_source.eof())

{

analyze\_lexical\_strinc = 0;

string strold = str;

if(!analyze\_lexical\_decomment(str, true))

{

out\_error << "Error in string " << analyze\_lexical\_strnum << ": " << strold << endl;

cout << "Error in string " << analyze\_lexical\_strnum << ": " << strold << endl;

return false;

}

analyze\_lexical\_strnum += analyze\_lexical\_strinc;

flag\_error = !analyze\_lexical\_string(str);

if(flag\_error)

{

out\_error << "Error in string " << analyze\_lexical\_strnum << ": " << str << endl;

cout << "Error in string " << analyze\_lexical\_strnum << ": " << str << endl;

}

analyze\_lexical\_strnum ++;

}

}

in\_source.close();

out\_token.close();

out\_error.close();

return !flag\_error;

}

// Очистка от комментариев

bool translator::analyze\_lexical\_decomment(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);

analyze\_lexical\_strinc++;

}

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);

analyze\_lexical\_strinc++;

}

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 analyze\_lexical\_decomment(str, change);

}

return true;

}

// Анализ строки

bool translator::analyze\_lexical\_string(string str)

{

trim(str);

bool flag\_error = false;

if(str.size())

{

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 0: // Идентификатор

{

// Получим полное название идентификатора

string idname = str;

int i;

bool finded = false;

for(i = 1; i < idname.size() && !finded; i++)

finded = !(letters.contains(str[i]) || numbers.contains(str[i]));

if(finded)

{

idname.erase(i - 1);

str.erase(0, i - 1);

}

else

str.erase(0);

trim(idname);

trim(str);

if(keywords.contains(idname)) // Если ключевое слово

{

if(keywords.get\_num(idname, i))

out\_token << token(3, i, -1);

}

else // Иначе в таблицу идентификаторов

{

identifiers.add(idname);

int table, chain;

identifiers.get\_location(idname, table, chain);

out\_token << token(5, table, chain);

}

return analyze\_lexical\_string(str);

}

break;

case 1: // Константа

{

string constval = str;

int i;

bool finded = false;

for(i = 1; i < constval.size() && !finded; i++)

finded = !(numbers.contains(str[i]) || str[i] == '.' || str[i] == ' ');

string str\_t1, str\_t2;

stringstream str\_stream\_t;

str\_stream\_t << str[i - 1];

str\_t1 = str\_stream\_t.str();

str\_stream\_t << str[i];

str\_t2 = str\_stream\_t.str();

if(!operations.contains(str\_t1) && !operations.contains(str\_t2) && !separators.contains(str[i - 1]))

{

out\_error << "Error: incorrect constant" << endl;

cout << "Error: incorrect constant" << endl;

return false;

}

if(finded)

{

constval.erase(i - 1);

str.erase(0, 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.get\_location(constval, table, chain);

out\_token << token(6, table, chain);

}

return analyze\_lexical\_string(str);

}

break;

case 2: // Операция

{

int table;

if(operations.contains(str\_2)) // Двухсимвольная

{

operations.get\_num(str\_2, table);

out\_token << token(4, table, -1);

str.erase(0, 2);

trim(str);

return analyze\_lexical\_string(str);

}

if(operations.contains(str\_1)) // Односимвольная

{

operations.get\_num(str\_1, table);

out\_token << token(4, table, -1);

str.erase(0, 1);

trim(str);

return analyze\_lexical\_string(str);

}

}

break;

case 3: // Разделитель

{

int table;

separators.get\_num((const char)str[0], table);

out\_token << token(4, table, -1);

str.erase(0, 1);

trim(str);

return analyze\_lexical\_string(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::debug\_print(ostream& stream)

{

stream << "ID`s:" << endl;

identifiers.debug\_print(stream);

stream << "CONST`s:" << endl;

constants.debug\_print(stream);

}

token.h

#ifndef TOKEN\_H\_INCLUDED

#define TOKEN\_H\_INCLUDED

#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

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& token\_)

{

istream\_ >> token\_.table >> token\_.place >> token\_.chain;

return istream\_;

}

ostream& operator << (ostream& ostream\_, const token::token& token\_)

{

ostream\_ << token\_.table << " " << token\_.place << " " << token\_.chain << endl;

return ostream\_;

}

main.cpp

#include <iostream>

#include <stdio.h>

#include "translator.h"

using namespace std;

int main()

{

translator a;

a.analyze\_lexical("files/source.txt","files/tokens.txt","files/errors.txt");

a.debug\_print(cout);

return 0;

}