**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное ГОСУДАРСТВЕННОЕ бюджетное ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информатики и вычислительной техники

Кафедра ИиСП

**Расчетно-графическая работа по дисциплине**

**«Теория автоматов и формальных языков»**

Выполнили:

студенты группы ПС-21

Винокуров Д.

Иванов Н.С.

Веселов Р.В.

Проверила:

Нехорошкова Л.Г.

г. Йошкар-Ола

2015

Постановка задачи

Для существующего языка программирования написать лексический распознаватель (в качестве ЯП выбран C++).

Программа запускается с помощью командной строки в формате  
lex.exe <input\_file> <output\_file>

Во входном файле содержится текст некоторой программы на ЯП, который выбрал студент.

В выходном файле программа выводит все лексемы в порядке их следования в программе, в формате  
<номер строки> <токен> <лексема>

Если была обнаружена лексема с токеном error, она выводится в output согласно формату, а выполнение программы завершается.

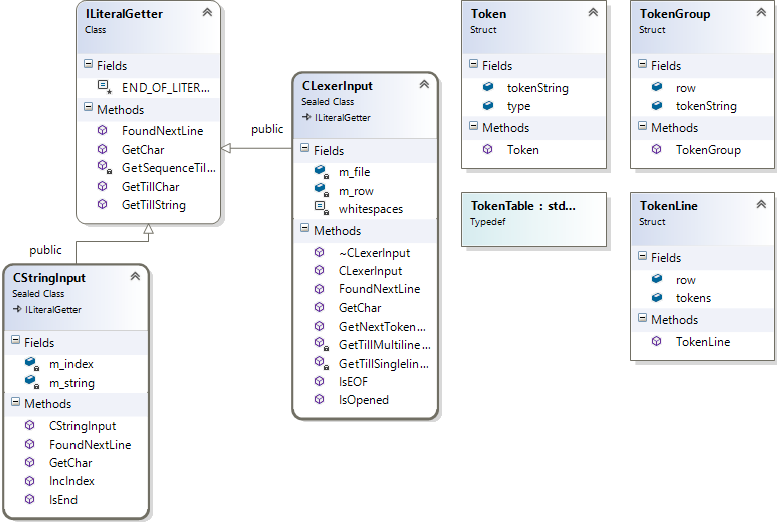
Алгоритм работы

Для достижения необходимого результата разбили работу на подзадачи:

1. Парсер входного файла
2. Создание списка токенов
3. Реализация автоматов для неконстантных токенов

Парсер:

Парсер считывает из входного файла все строки и в каждой строке ищет токены, пока не найдет ошибку. Токены ищутся по разделителям, вроде пробела, табуляции и т.п. Затем токены анализируются и получается список интерпретированных лексем.



Неконстантные токены:

std::map<TokenType,std::function<bool(std::string)>> tokenFunc =

{

{ TokenType::INTEGER\_DEC\_NUMBER, IsIntNumber },

/\*Все элементы – цифры

Валидный суффикс ("u" || "l" || "lu" || "ul" || "ll" || "ull" || "lul" || "llu“ )\*/

{ TokenType::FLOAT\_NUMBER, IsFloatNumber },

/\*Символы : цифры, ‘e’, ‘.’, суффикс ‘l’

Может не иметь целую часть (.222)

Может быть представлено через E (1.1e-1)

Может иметь суффикс (1.1L)\*/

{ TokenType::CONST\_CHAR, IsString },

/\*Первый и последний символ = “

Внутри может быть непрерывная пара “” \*/

{ TokenType::CHARACTER, IsChar },

/\*Первый и последний символ = ‘

Между ними либо 1 символ, либо управляющий символ \*/

{ TokenType::INTEGER\_OCT\_NUMBER, IsOCT },

/\*Начинается с 0

Далее – символы { '0', '1', '2', '3', '4', '5', '6', '7'}\*/

{ TokenType::INTEGER\_HEX\_NUMBER, IsHEX },

/\*Начинается с 0х

Далее – символы { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f' } \*/

{ TokenType::IDENTIFIER, IsIdentifier}

/\*Может состоять из букв, чисел (не в начале) и \_ \*/

};

Тесты

*Input 1:*

int main ()

{

cout << "Hello world!!!\n";

\_getch();

return 0;

}

*Output 1:*

1 INT int

1 IDENTIFIER main

1 PARENTHESIS\_L (

1 PARENTHESIS\_R )

2 BRACE\_L {

3 IDENTIFIER cout

3 LEFT\_OP <<

3 STRING "Hello world!!!\n"

3 SEMICOLON ;

4 IDENTIFIER \_getch

4 PARENTHESIS\_L (

4 PARENTHESIS\_R )

4 SEMICOLON ;

5 RETURN return

5 CONST\_INTEGER 0

5 SEMICOLON ;

6 BRACE\_R }

*Input 2:*

auto a = 2.9l;

a = 1.;

a = 1e-1;

a = 2.8e1;

a = 1e50;

*Output 2:*

1 AUTO auto

1 IDENTIFIER a

1 ASSIGN\_OP =

1 CONST\_FLOAT 2.9l

1 SEMICOLON ;

2 IDENTIFIER a

2 ASSIGN\_OP =

2 CONST\_FLOAT 1.

2 SEMICOLON ;

3 IDENTIFIER a

3 ASSIGN\_OP =

3 CONST\_FLOAT 1e-1

3 SEMICOLON ;

4 IDENTIFIER a

4 ASSIGN\_OP =

4 CONST\_FLOAT 2.8e1

4 SEMICOLON ;

5 IDENTIFIER a

5 ASSIGN\_OP =

5 CONST\_FLOAT 1e50

5 SEMICOLON ;

*Input 3:*

double GetTickCount(int time)

{

return (time \* 1000.0 / g\_freq);

}

*Output 3:*

1 DOUBLE double

1 IDENTIFIER GetTickCount

1 PARENTHESIS\_L (

1 INT int

1 IDENTIFIER time

1 PARENTHESIS\_R )

2 BRACE\_L {

3 RETURN return

3 PARENTHESIS\_L (

3 IDENTIFIER time

3 STAR \*

3 CONST\_FLOAT 1000.0

3 SLASH /

3 IDENTIFIER g\_freq

3 PARENTHESIS\_R )

3 SEMICOLON ;

4 BRACE\_R }

*Input 4:*

if(a>==>b)

{

}

*Output 4:*

1 IF if

1 PARENTHESIS\_L (

1 IDENTIFIER a

1 GE\_OP >=

1 ASSIGN\_OP =

1 CHEVRON\_R >

1 IDENTIFIER b

1 PARENTHESIS\_R )

2 BRACE\_L {

3 BRACE\_R }