Белорусский государственный университет информатики и радиоэлектроники

Кафедра информатики

Лабораторная работа № 1

**Лексический анализатор**

Выполнил:

Губский М.Д.

Проверил:

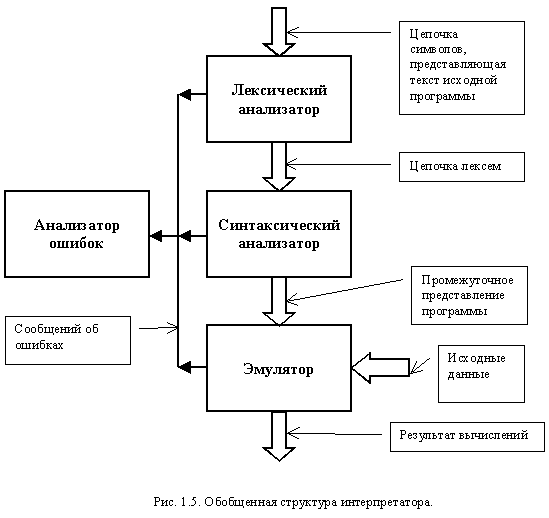
Шиманский В.В.

Минск, 2016

1. ***Постановка задачи:***

В данной работе ставится задача по исследованию области лексических анализаторов, рассмотрению аналогов и написанию (по возможности) своего собственного анализатора лексем выбранного подмножества языка программирования. Построить список встречающихся лексем и вывести их типы в виде (WHILE, 'while', line\_number, position).

1. ***Теория:***

 Лексический анализатор представляет собой первую фазу компиляции, его основная задача состоит в чтении входных символов исходной программы, их группировании в лексемы и вывод последовательностей токенов для всех лексем исходной программы.

Поскольку лексический анализатор является частью компилятора, которая читает исходный текст, он может заодно выполнять и некоторые другие действия, помимо идентификации лексем. Одной из таких задач является отбрасывание пробельных символов и табуляции из-за незначимости в контексте выбранного языка.

1. ***Используемые технологии:***

В качестве препарируемого языка был выбран язык JavaScript. В качестве языка для анализа языка Java был выбран Python из-за его привлекательных возможностей. Python достаточно прост в освоении и имеет большое количество дополнительных библиотек. Для лексического разбора существует библиотека PLY (**P**ython **L**ex-**Y**acc.), из которой мною был использован Lex, он разбивает текст на базовые элементы языка и работает он с регулярными выражениями. Также Lex требователен к коду, он требует обязательного присутствия массива с именем tokens. Для каждого элемента этого массива мы обязаны иметь в коде регулярное выражение или функцию с именем t\_ЭЛЕМЕНТ.

1. ***Программа и комментарии:***

Список известных лексическому анализатору лексем можно увидеть ниже:

tokens = (

'DELIMITER',

'RESERVED',

'LOGICAL\_OPERATOR',

'BIN\_OPERATOR',

'MATH\_OPERATOR',

'ASSIGNMENT\_OPERATOR',

'KEYWORD',

'BOOLEAN',

'STRING',

'ID', 'NUMBER',

'METHOD\_OF\_CLASS\_CONSOLE',

)

Лексемы задаются следующими регулярными выражениями:

t\_DELIMITER = r';|\,'

t\_RESERVED = r'\(|\)|{|}'

t\_MATH\_OPERATOR = '\+\+?|\-\-?|\\*|\/|%'

t\_LOGICAL\_OPERATOR = r'\=\=\=?|<=?|>=?|!=|&&|\|\|'

t\_ASSIGNMENT\_OPERATOR = r'\='

t\_BIN\_OPERATOR = r'&|\||\^|\~'

t\_METHOD\_OF\_CLASS\_CONSOLE = r'console.log'

t\_KEYWORD = r'if|else|function|while|for|in|var|return'

t\_BOOLEAN = r'true|false'

t\_STRING = r'\"([^\"]+)\"|\'([^\']+)\''

t\_NUMBER = r'[0-9]+(\.[0-9]{0,15})?|\.[0-9]{1,15}'

t\_ID = r'[A-Za-z][A-Za-z0-9\_]\*'

Номер строки отлавливается с помощью следующей процедуры:

def t\_newline(t):

r'\n+'

t.lexer.lineno += 1

Код игнорирования табуляции и комментариев:

t\_ignore = ' \t'

t\_ignore\_COMMENT = r'(\/\/[^\n]\*)|(\/\\*.\*(\\*\/))'

Код обработки лексических ошибок:

def t\_error(t):

print(colorama.Fore.RED + '\n!ERROR!\nIllegal character {0} at line {1}\n'.format(t.value[0], t.lexer.lineno))

print(colorama.Fore.WHITE)

t.lexer.skip(1)

Код формирования таблицы имен:

lexer = lex.lex(reflags=re.UNICODE | re.DOTALL)

def parse(lines):

lexer.input(lines)

while True:

tok = lexer.token()

if not tok:

break

print(tok)

Код считывания из файла:

d = open('task1.txt', 'r')

lines = d.read()

d.close()

Код запуска программы на выполнение:

parse(lines)

Код, из которого выделяются лексемы:

var i=10;

var f=function(n){

if (n<=1){

return 1;

}

else{

return f(n-1)+f(n-2);

}

}

while (i>0 || i===10) {

console.log(f(i));

i--;

}//concole.log(x);

Результат выполнения

LexToken(KEYWORD,'var',1,0)

LexToken(ID,'i',1,4)

LexToken(ASSIGNMENT\_OPERATOR,'=',1,5)

LexToken(NUMBER,'10',1,6)

LexToken(DELIMITER,';',1,8)

LexToken(KEYWORD,'var',2,10)

LexToken(ID,'f',2,14)

LexToken(ASSIGNMENT\_OPERATOR,'=',2,15)

LexToken(KEYWORD,'function',2,16)

LexToken(RESERVED,'(',2,24)

LexToken(ID,'n',2,25)

LexToken(RESERVED,')',2,26)

LexToken(RESERVED,'{',2,27)

LexToken(KEYWORD,'if',3,30)

LexToken(RESERVED,'(',3,33)

LexToken(ID,'n',3,34)

LexToken(LOGICAL\_OPERATOR,'<=',3,35)

LexToken(NUMBER,'1',3,37)

LexToken(RESERVED,')',3,38)

LexToken(RESERVED,'{',3,39)

LexToken(KEYWORD,'return',4,43)

LexToken(NUMBER,'1',4,50)

LexToken(DELIMITER,';',4,51)

LexToken(RESERVED,'}',5,54)

LexToken(KEYWORD,'else',6,57)

LexToken(RESERVED,'{',6,61)

LexToken(KEYWORD,'return',7,65)

LexToken(ID,'f',7,72)

LexToken(RESERVED,'(',7,73)

LexToken(ID,'n',7,74)

LexToken(MATH\_OPERATOR,'-',7,75)

LexToken(NUMBER,'1',7,76)

LexToken(RESERVED,')',7,77)

LexToken(MATH\_OPERATOR,'+',7,78)

LexToken(ID,'f',7,79)

LexToken(RESERVED,'(',7,80)

LexToken(ID,'n',7,81)

LexToken(MATH\_OPERATOR,'-',7,82)

LexToken(NUMBER,'2',7,83)

LexToken(RESERVED,')',7,84)

LexToken(DELIMITER,';',7,85)

LexToken(RESERVED,'}',8,88)

LexToken(RESERVED,'}',9,90)

LexToken(KEYWORD,'while',10,92)

LexToken(RESERVED,'(',10,98)

LexToken(ID,'i',10,99)

LexToken(LOGICAL\_OPERATOR,'>',10,100)

LexToken(NUMBER,'0',10,101)

LexToken(LOGICAL\_OPERATOR,'||',10,103)

LexToken(ID,'i',10,106)

LexToken(LOGICAL\_OPERATOR,'===',10,107)

LexToken(NUMBER,'10',10,110)

LexToken(RESERVED,')',10,112)

LexToken(RESERVED,'{',10,114)

LexToken(ID,'console',11,117)

LexToken(NUMBER,'.',11,124)

LexToken(ID,'log',11,125)

LexToken(RESERVED,'(',11,128)

LexToken(ID,'f',11,129)

LexToken(RESERVED,'(',11,130)

LexToken(ID,'i',11,131)

LexToken(RESERVED,')',11,132)

LexToken(RESERVED,')',11,133)

LexToken(DELIMITER,';',11,134)

LexToken(ID,'i',12,137)

LexToken(MATH\_OPERATOR,'--',12,138)

LexToken(DELIMITER,';',12,140)

LexToken(RESERVED,'}',13,142)

Приведем пример по обработке лексических ошибок в исходном коде программы:

var i=10;

var f=function(n){\

if (n<=1){

return 1;

}

else{

return f(n-1)+f(n-2);

}\_

}#

while (i>0 || i===10) {

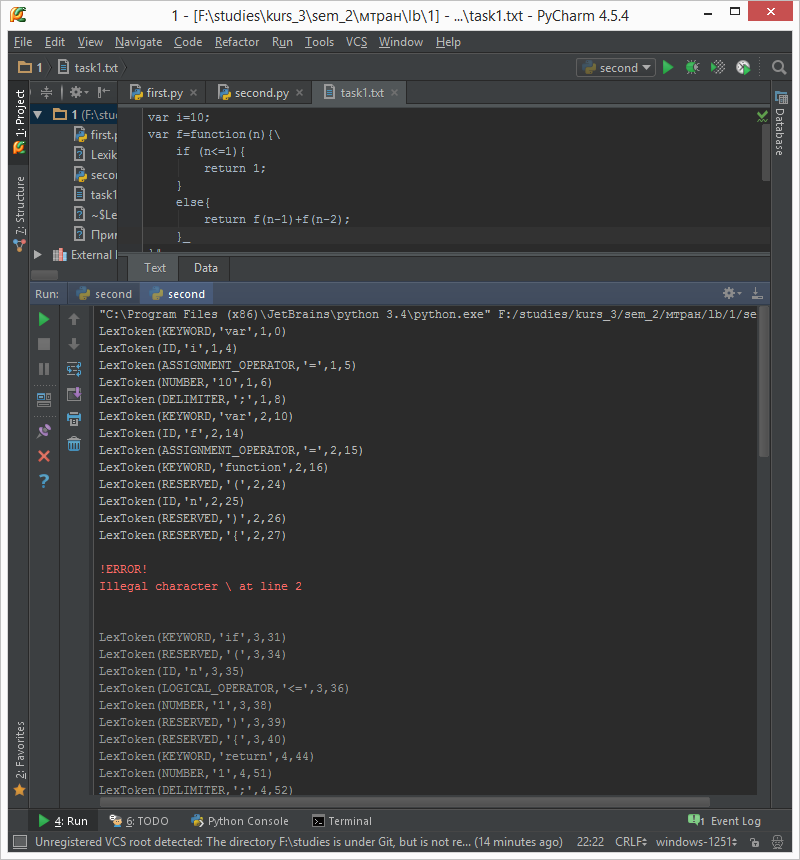
console.log(f(i));

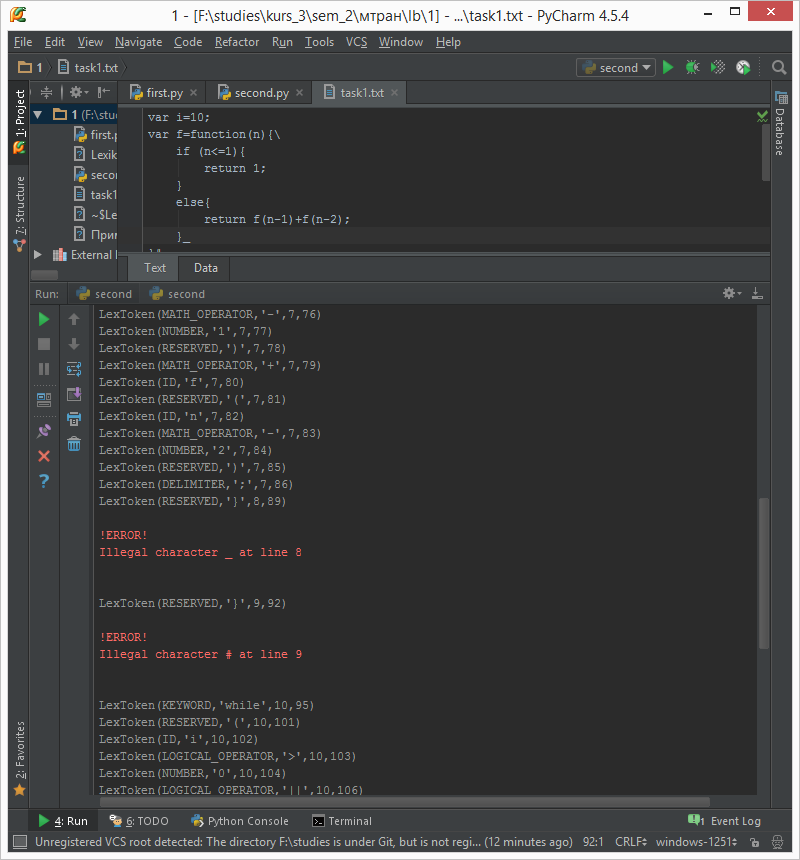
i--;

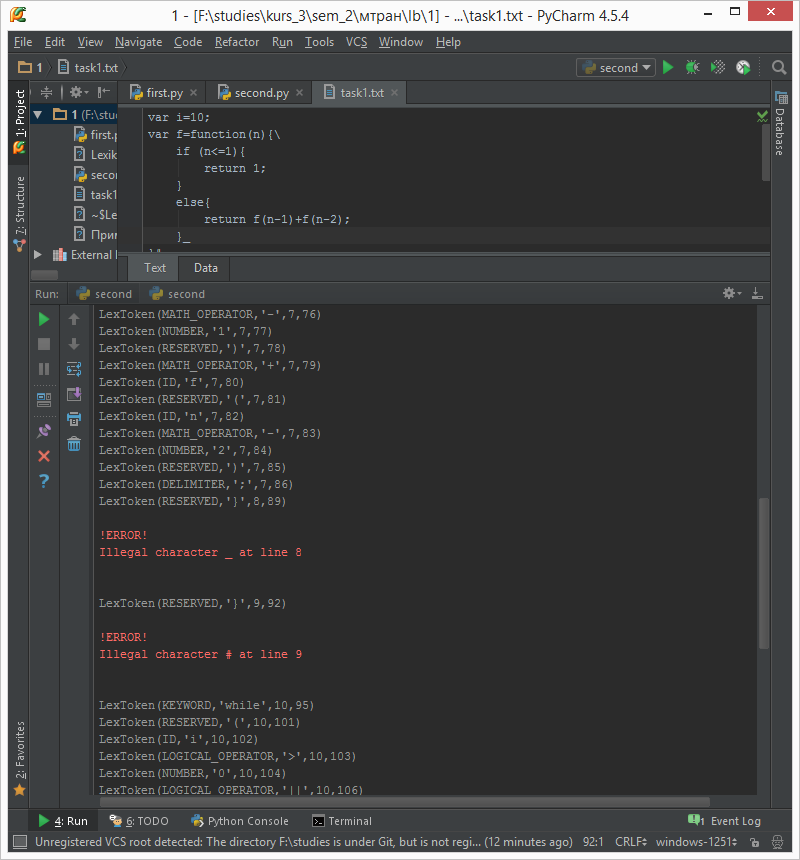
}//concole.log(x);

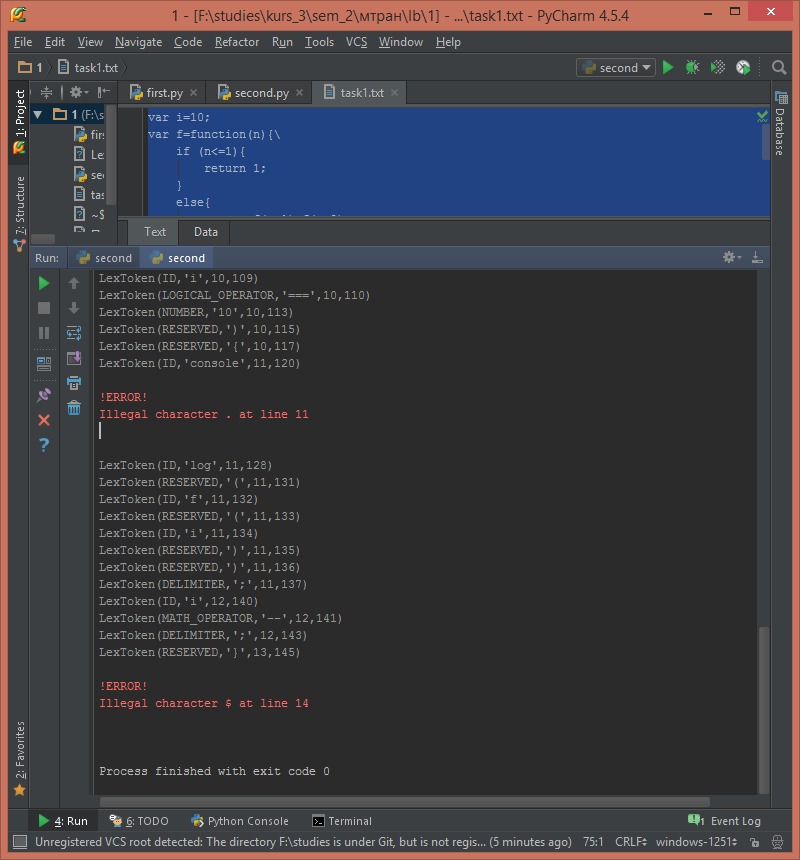
$

Получим уведомления об ошибках:









**Заключение**

В данной лабораторной работе был проведен анализ на языке программирования Python фрагмента кода на языке программирования JavaScript. В программе происходило чтение входных символов исходной программы, их группирование в лексемы и вывод списка  токенов для лексем исходной программы. При определении неверной последовательности символов в исходном файле программа выдает уведомление об ошибке с указанием неопознанных символов и номера строки, где ошибка была допущена. Примеры работы программы, а также примеры реакции на ошибки приведены.