Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**Отчёт**

**по лабораторной работе №1**

**Дисциплина: Методы разработки трансляторов**

**Тема: «Построение лексического анализатора»**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_К. А. Корнилов

Направление подготовки 02.03.02 Фундаментальная информатика и

информационные технологии

Направленность (профиль) Математическое и программное обеспечение

компьютерных технологий

Преподаватель

д-р техн. наук, проф. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю. М. Вишняков

**Содержание**

[1 Вариант задания 3](#_Toc97893076)

[2 Основные понятия лексического анализа 3](#_Toc97893077)

[3 Функции и таблицы лексического анализа 4](#_Toc97893078)

[4 Диаграмма состояний лексического процессора 9](#_Toc97893079)

[5 Результаты экспериментов 15](#_Toc97893080)

[Приложение А Описание правил записи элементов заданного входного языка 20](#_Toc97893081)

[Приложение Б Листинг программы и комментарии к нему 24](#_Toc97893082)

**1 Вариант задания**

Вариант задания представляет собой пару: входной язык и выходной язык (таблица 1).

Таблица 1 – Вариант задания

|  |  |  |
| --- | --- | --- |
| № варианта | Входной язык | Выходной язык |
| 37 | PHP | R |

Разработать лексический анализатор, корректно распознающий все лексемы и формирующего таблицы лексем и внутреннее представление проанализированного текста.

На вход программы подается файл, содержащий текст на входном языке программирования. Результатом работы программы должен быть файл, содержащий последовательность кодов лексем входной программы, а также один или несколько файлов, содержащих все таблицы лексем.

Отчет по работе должен содержать описание правил записи элементов заданного входного языка, стандартные таблицы лексем, диаграмма состояний сканера, листинг программы и комментарии к нему, пример лексического разбора.

**2 Основные понятия лексического анализа**

Лексический анализ всегда предшествует синтаксическому анализу и заключается в предварительной обработке программы и ее перекодировании к виду, удобному для синтаксического анализа. Лексический анализ принято отделять от синтаксического анализа для сокращения времени компиляции, поскольку синтаксис (морфология) слов всегда проще, чем синтаксис программ.

Лексический анализ сводится к разбиению текста программы на лексические единицы (слова, лексемы) – конструкции, которые выступают в качестве терминальных символов для синтаксического анализа. Лексемы еще иногда называют символами или атомами.

Большинство лексем в языках программирования можно сгруппировать в следующие классы:

1. класс идентификаторов;
2. класс служебных слов;
3. класс констант (числовых или символьных);
4. класс операций (одно-, дву- или многолитерных);
5. класс разделителей (однолитерных или двулитерных).

В лексическом анализе лексема представляется в виде пары (класс, значение).

**3 Функции и таблицы лексического анализа**

Основная функция лексического анализатора (сканера) состоит в выделении лексем из исходной программы и передаче их синтаксическому анализатору в некотором внутреннем представлении. При этом лексический анализатор использует таблицы, соответствующие классам лексем. Таблицы служебных слов, разделителей и операций определяются входным языком и должны быть сформированы при построении сканера, они являются постоянными и неизменяемыми. Таблицы идентификаторов и констант являются временными и создаются непосредственно в процессе лексического разбора исходной программы.

Комментарии в процессе лексического анализа игнорируются, так как они специфицируют исходную программу только для программиста и не влияют на семантику программы.

Для рассматриваемого языка сформированные таблицы служебных слов (таблица 2), таблица операций (таблица 3) и таблица разделителей (таблица 4) имеют вид:

Таблица 2 – Таблица служебных слов

|  |  |
| --- | --- |
| Служебное слово | Код |
| If | 1 |
| else | 2 |
| else if | 3 |
| while | 4 |
| break | 5 |
| continue | 6 |
| function | 7 |
| return | 8 |
| echo | 9 |
| true | 10 |
| false | 11 |
| array | 12 |
| null | 13 |
| do | 14 |

Таблица 3 – Таблица операций

|  |  |
| --- | --- |
| Операция | Код |
| + | 1 |
| - | 2 |
| \* | 3 |
| / | 4 |
| % | 5 |
| \*\* | 6 |
| = | 7 |
| == | 8 |
| > | 9 |
| >= | 10 |
| < | 11 |
| <= | 12 |
| <> | 13 |
| & | 14 |
| | | 15 |
| ^ | 16 |
| ~ | 17 |
| << | 18 |
| >> | 19 |
| ! | 20 |
| And | 21 |
| Xor | 22 |
| or | 23 |
| pow | 24 |
| sqrt | 25 |
| sin | 26 |
| cos | 27 |
| tan | 28 |
| abs | 29 |
| log | 30 |
| log10 | 31 |
| max | 32 |
| min | 33 |
| . | 34 |

Таблица 4 – Таблица разделителей

|  |  |
| --- | --- |
| Разделитель | Код |
| ; | 1 |
| ( | 2 |
| ) | 3 |
| Пробел | 4 |
| \n | 5 |
| \t | 6 |
| ‘ | 7 |
| “ | 8 |
| <? | 9 |
| ?> | 10 |
| $ | 11 |
| } | 12 |
| { | 13 |
| # | 14 |

Каждая лексема, обработанная сканером, преобразуется к виду:

<буква><код>,

где: <буква> – это признак класса лексемы (W, I, O, R, N или C),   
<код> – номер лексемы в соответствующей таблице.

При лексическом разборе входной цепочки пробелы не кодируются как разделители и в выходную цепочку не попадают. Это связано с тем, что в исходном тексте программы пробелы служат только для структурирования текста программы и не несут дополнительной смысловой нагрузки.

**4 Диаграмма состояний лексического процессора**

Лексический анализатор представляет собой процессор или, иначе говоря, сканер (конечный автомат) для разбора и классификации лексем, связанный с некоторыми семантическими процедурами. На вход такого процессора последовательно подаются символы исходной программы, причем каждый входной символ вызывает изменение состояния сканера.

Если анализируемый символ означает конец разбираемой лексемы, то с переходом связывается некоторая семантическая процедура, позволяющая либо определить код лексемы по таблице (в случае служебных слов, разделителей, операций), либо пополнить таблицы (для констант и идентификаторов), а затем в выходную строку выдать очередной код лексемы во внутреннем представлении. Для эффективного разбора и оптимизации сканера, а также для его программной реализации в алфавите автомата выделяют несколько подмножеств символов, по которым сканер выполняет одинаковые переходы и действия. В нашем случае такими подмножествами являются:

1. буквы {A, …, Z, a, …, z};
2. цифры {0, …, 9};
3. разделители
4. операции
5. конец файла.

С учетом вышесказанного диаграмма состояний сканера для рассматриваемого языка имеет вид, представленный на рисунке 1.

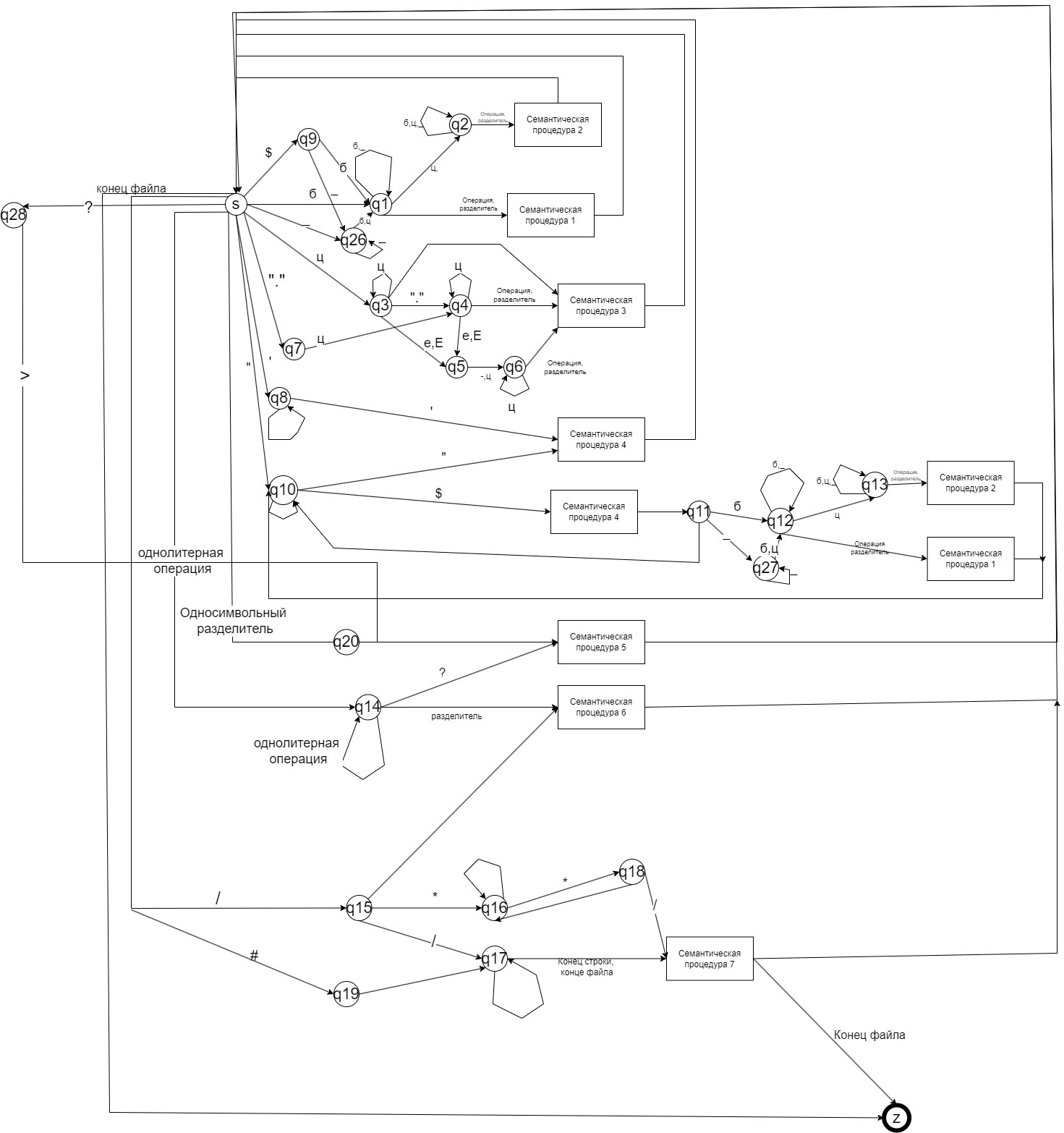


Рисунок 1 – Диаграмма состояний сканера

На данной диаграмме состояния сканера представлены вершинами, а переходы между состояниями – дугами (направленными линиями). Каждый переход связан с чтением очередного символа их текста входной программы. Поэтому дуга взвешена (помечена) символом или множеством символов, которые вызывают данный переход. Если в диаграмме состояний есть невзвешенная дуга, ведущая из какого-либо состояния, то считается, что она взвешена любыми символами кроме тех, которыми взвешены другие дуги, исходящие из данного состояния. Это позволяет не перегружать диаграмму лишними символами. Скругленный прямоугольник в разрыве дуги указывает на семантическую процедуру, выполняемую при данном переходе. Если переход не сопровождается семантической процедурой, то текущий символ добавляется к буферу, в котором формируется лексема. Работа каждой семантической процедуры завершается очищением буфера.

Сканер всегда содержит три стандартных состояния:

1. S – начальное состояние сканера. Лексический разбор всегда начинается из этого состояния.
2. Z – заключительное состояние сканера. Если в процессе разбора достигнуто данное состояние, это означает, что разбор успешно завершен.
3. F – состояние ошибки. Если встретился символ, не входящий во входной алфавит, то сканер переходит в состояние ошибки и прекращает разбор. Кроме того, если в каком-либо состоянии на входе появился символ, по которому не предусмотрен переход из этого состояния, сканер также переходит в состояние ошибки (на диаграмме эти переходы не изображаются, чтобы избежать лишнего загромождения рисунка).

За семантическими процедурами процессора закреплены следующие функции:

Семантическая процедура 1: Провести поиск сформированного слова в таблицах служебных слов, операций. Если такое есть служебное слово в таблице служебных слов или операция в таблице операций, то выполнить сформировать и выдать в выходную последовательность лексему служебного слова или операции во внутреннем представлении. Иначе выполнить семантическую процедуру 2

Семантическая процедура 2: Провести поиск сформированного слова в таблице идентификаторов. Если такое слово в таблице идентификаторов не найдено, то занести сформированное слово в таблицу идентификаторов. Сформировать и выдать в выходную последовательность лексему идентификатора во внутреннем представлении.

Семантическая процедура 3: Занести сформированное слово в таблицу числовых констант. Сформировать и выдать в выходную последовательность лексему числовой константы во внутреннем представлении.

Семантическая процедура 4: Занести сформированное слово в таблицу символьных констант. Сформировать и выдать в выходную последовательность лексему символьной константы во внутреннем представлении.

Семантическая процедура 5: Провести поиск сформированного слова в таблице разделителей. Сформировать и выдать в выходную последовательность лексему разделителя во внутреннем представлении.

Семантическая процедура 6: Провести поиск текущего слова в таблице операций. Сформировать и выдать в выходную последовательность лексему операции во внутреннем представлении.

**5 Результаты экспериментов**

Пример работы программы для программы для тестирования: на рисунке 8 представлен скриншот файла, содержащего текст на входном языке программирования, на рисунке 9 – скриншот файла, содержащего последовательность кодов лексем входной программы, на рисунках   
10-15 – скриншоты файлов, содержащие все лексемы.

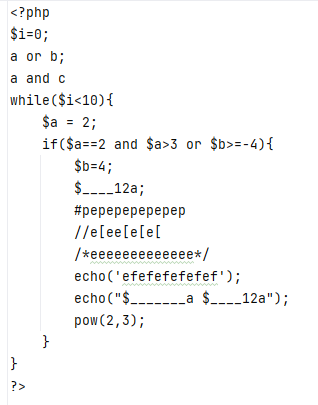


Рисунок 2 – Скриншот файла, содержащего текст на входном языке программирования

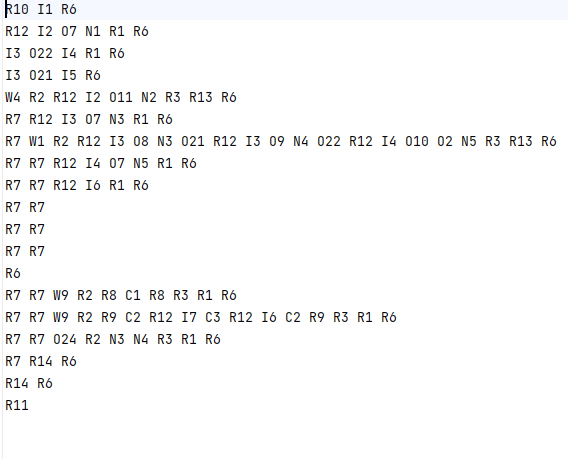


Рисунок 3 – Скриншот файла, содержащего последовательность кодов лексем входной программы

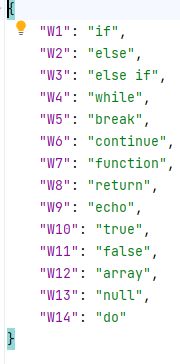


Рисунок 4 – Скриншот файла, содержащего лексемы служебных слов

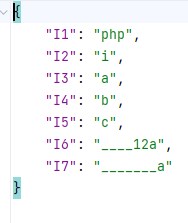


Рисунок 5 – Скриншот файла, содержащего лексемы идентификаторов

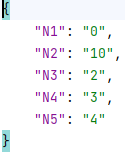


Рисунок 6 – Скриншот файла, содержащего лексемы числовых констант

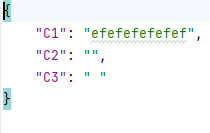


Рисунок 7 – Скриншот файла, содержащего лексемы символьных констант

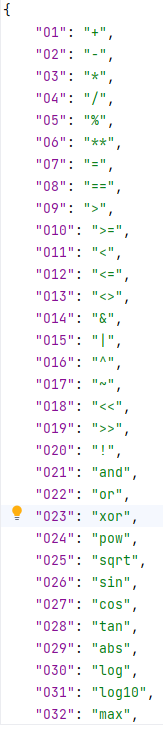


Рисунок 8 – Скриншот файла, содержащего лексемы операций

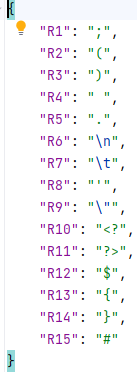


Рисунок 9 – Скриншот файла, содержащего лексемы разделителей

**Приложение А   
Описание правил записи элементов заданного входного языка**

Идентификаторы

Идентификаторы начинаются с символа «$», после которого идёт произвольная последовательность букв, цифр, символов подчёркивания, начинающаяся с буквы или символа подчёркивания. Регистр имеет значение.

Числовые константы целого типа

Произвольная последовательность цифр без знака.

Числовые константы вещественного типа, представленные с фиксированной точкой

Последовательность цифр, включающая одну десятичную точку вида 123.45

Числовые константы вещественного типа, представленные с плавающей точкой

Последовательность, включающая цифры, десятичную точку (необязательную), символ «е» или «E», а также знак «+» или «-» (необязательный) вида

1.23е-25

1.23е+25

1.23е2

1E-78

1е67

Символьные (строковые) константы

В PHP существует два способа задания строк в двойных, либо одинарных кавычках (“acb 12\_& ?tu”, ‘Bob’). Разница между ними заключается в том, что если внутри строки первого типа написать имя переменной, то интерпретатор PHP подставит значение этой переменной при результирующем выводе. Во втором случае (одинарные кавычки) подстановки не происходит.

Переменные с индексами (массивы и элементы массивов)

Идентификатор, после которого в квадратных скобках стоит выражение-индекс, вида:

$Abc[12]

$b[1+$i]

Комментарии (строчные и блочные)

Блочные – последовательность символов, начинающаяся с «/\*» и оканчивающаяся «\*/», возможно содержащая несколько строк:

/\* Это комментарий,

Который содержит 2 строки\*/

Строчные – от символов «//» до конца строки или от символа «#» до конца строки:

$i=$i+1; // увеличение i на единицу

$j=$j+1; #увеличение j на единицу

Обращения к функциям пользователя

Идентификатор, после которого в круглых скобках следует последовательность выражений-аргументов, разделённых запятыми. Отсутствие аргументов не допускается:

F(12, 4, $i)

f($av-6)

Арифметические операции

Сложение +

Вычитание -

Умножение \*

Деление /

Остаток от деления на

Операции сравнения

|  |  |
| --- | --- |
| Меньше | < |
| Больше | > |
| Равно | == |
| Не равно | != |
| Меньше или равно | <= |
| Больше или равно | >= |

**Оператор присваивания**

Имеет вид «=». Слева стоит идентификатор или элемент вектора, а справа – выражение. Пример:

$A=$b+$c;

$a[1]=$x+1;

Операторы блока

{ – начало блока

операторы

} - конец блока

Структура программы

***Программа на PHP должна быть обрамлена в теги «<?php» и «?>». Программа представляет собой последовательность операторов, в том числе операторов описания функций. Операторы отделяются друг от друга точкой с запятой. Пример:***

***<?php  
 $index = 10;  
 print($index);  
?>***

Операторы описания данных (идентификаторов и массивов)

***Объявления не являются обязательными. Переменная начинает существовать с момента присвоения ей значения или с момента первого использования. Если использование начинается раньше присвоения, то переменная будет содержать значение по умолчанию. Тип переменной определяется динамически присвоенным значением и текущей операцией.***

Операторы описания функций

Функции имеют заголовок вида

function <идентификатор> (<список формальных параметров, разделённых запятыми>)

Тело функции представляет список операторов, заключённый в операторы блока

{ …; return (<значение>);}

Пример:

***function foo($arg\_1, $arg\_2)***

***{***

***echo "Пример функции.\n";***

***return $arg\_1+$arg\_2;***

***}***

Оператор условного перехода

Начинается с ключевого слова «if», имеет полный и неполный формат:

if (*логическое выражение*)

{

*операторы*

}

if (*логическое выражение*)

{

*Операторы\_1*

}

else

{

*Операторы\_2*

}

**Цикл с предусловием**

Цикл с предусловием имеет следующий вид: while (<условие>) {<операторы>}. Пример:

while($i <= 3) {  
 echo “$i”;

$i=$i+1;  
 }

**Приложение Б   
Листинг программы и комментарии к нему**

from PyQt6.QtCore import QRunnable, QThread  
from PyQt6.QtGui import QTextCursor  
import json  
  
class LecAnalysis():  
 def createTokensCod(self,token\_class,token\_value):  
 if not(token\_value in self.tokens[token\_class]):  
 token\_code = str(len(self.tokens[token\_class])+1)  
 self.tokens[token\_class][token\_value] = token\_class + token\_code  
 def \_\_init\_\_(self):  
 self.OPERATIONS = ['+','-','\*','/','%','\*\*','=','==','>','>=','<','<=','<>','&','|','^','&','|','|','^','~','<<','>>','!','and','or','xor','pow','sqrt','sin','cos','tan','abs','log','log10','max','min']  
 self.SERVICE\_WORDS = ['if','else','else if','while','break','continue','function','return','echo','true','false','array','null','do']  
 self.SEPARATORS = [';','(',')',' ','.','\n','\t',"'",'"','<?','?>','$','{','}','#']  
 self.tokens = {'W':{},'I':{},'O':{},'R':{},'N':{},'C':{}}  
 def process(self):  
 for service\_word in self.SERVICE\_WORDS:  
 self.createTokensCod('W',service\_word)  
 for operation in self.OPERATIONS:  
 self.createTokensCod('O',operation)  
 for separator in self.SEPARATORS:  
 self.createTokensCod('R', separator)  
 f = open('./files/Input.txt','r')  
 input\_sequence = f.read()  
 f.close()  
  
 i=0  
 state = 'S'  
 output\_sequance = buffer = ''  
 while i!=len(input\_sequence):  
 symbol = input\_sequence[i]  
 if state=='S':  
 buffer=''  
 if symbol=='$':  
 state = 'q9'  
 output\_sequance+=self.tokens['R']['$']+' '  
 elif symbol=='\_':  
 state='q26'  
 buffer=symbol  
 elif symbol=='?':  
 buffer = symbol  
 state='q28'  
 elif symbol.isalpha():  
 state = 'q1'  
 buffer=symbol  
 elif symbol.isdigit():  
 buffer=symbol  
 state = 'q3'  
 elif symbol=='.':  
 buffer=symbol  
 state = 'q7'  
 elif symbol == "'":  
 state = 'q8'  
 output\_sequance += self.tokens['R'][symbol] + ' '  
 elif symbol =='"':  
 state='q10'  
 output\_sequance += self.tokens['R'][symbol] + ' '  
 elif symbol == '/':  
 state = 'q15'  
 buffer=symbol  
 elif symbol == '#':  
 state = 'q19'  
 elif i == len(input\_sequence) - 1:  
 state = 'Z'  
 elif symbol in self.tokens['O'].keys():  
 state = 'q14'  
 buffer=symbol  
 elif symbol in self.tokens['R'].keys():  
 state='q20'  
 buffer=symbol  
 elif state=='q9':  
 if symbol=='\_':  
 state='q26'  
 buffer=symbol  
 if symbol.isalpha():  
 state='q1'  
 buffer=buffer+symbol  
 elif state =='q1':  
 if symbol.isalpha() or symbol=='\_':  
 buffer = buffer + symbol  
 state='q1'  
 elif symbol.isdigit():  
 buffer = buffer + symbol  
 state='q2'  
 else:  
 if buffer in self.SERVICE\_WORDS:  
 output\_sequance+=self.tokens['W'][buffer]+ ' '  
 elif buffer in self.OPERATIONS:  
 output\_sequance+=self.tokens['O'][buffer]+' '  
 else:  
 self.createTokensCod('I',buffer)  
 output\_sequance+=self.tokens['I'][buffer]+' '  
 state='S'  
 i=i-1  
 elif state=='q2':  
 if symbol.isalpha() or symbol.isdigit() or symbol=='\_':  
 state='q2'  
 buffer = buffer + symbol  
 else:  
 if buffer in self.SERVICE\_WORDS:  
 output\_sequance+=self.tokens['W'][buffer]+ ' '  
 elif buffer in self.OPERATIONS:  
 output\_sequance+=self.tokens['O'][buffer]+' '  
 else:  
 self.createTokensCod('I',buffer)  
 output\_sequance+=self.tokens['I'][buffer]+' '  
 state='S'  
 i=i-1  
 elif state=='q3':  
 if symbol.isdigit():  
 state='q3'  
 buffer = buffer + symbol  
 elif symbol=='.':  
 state='q4'  
 buffer = buffer + symbol  
 elif symbol=='E' or symbol=='e':  
 state='q5'  
 buffer = buffer + symbol  
 else:  
 self.createTokensCod('N', buffer)  
 output\_sequance += self.tokens['N'][buffer] + ' '  
 state='S'  
 i=i-1  
 elif state=='q7':  
 if symbol.isdigit():  
 state='q4'  
 buffer = buffer + symbol  
 elif state=='q5':  
 if symbol.isdigit() or symbol=='-':  
 state='q6'  
 buffer = buffer + symbol  
 elif state=='q6':  
 if symbol.isdigit():  
 state='q6'  
 buffer = buffer + symbol  
 else:  
 self.createTokensCod('N',buffer)  
 output\_sequance+=self.tokens['N'][buffer]+' '  
 state = 'S'  
 i=i-1  
 elif state=='q4':  
 if symbol.isdigit():  
 state='q4'  
 buffer = buffer + symbol  
 else:  
 self.createTokensCod('N', buffer)  
 output\_sequance += self.tokens['N'][buffer] + ' '  
 state='S'  
 i=i-1  
 elif state=='q8':  
 if symbol!="'":  
 state='q8'  
 buffer=buffer+symbol  
 else:  
 self.createTokensCod('C',buffer)  
 output\_sequance+=self.tokens['C'][buffer] + ' ' + self.tokens['R'][symbol] + ' '  
 state = 'S'  
 elif state=='q10':  
 if symbol !='"' and symbol!="$":  
 state='q10'  
 buffer = buffer + symbol  
 elif symbol=='"':  
 self.createTokensCod('C', buffer)  
 output\_sequance += self.tokens['C'][buffer] + ' ' + self.tokens['R'][symbol]+ ' '  
 state = 'S'  
 else:  
 self.createTokensCod('C', buffer)  
 output\_sequance += self.tokens['C'][buffer] + ' ' + self.tokens['R']['$'] + ' '  
 state = 'q11'  
 buffer = ''  
 elif state=='q11':  
 buffer=''  
 if symbol.isalpha():  
 state='q12'  
 buffer = buffer + symbol  
 elif symbol=='\_':  
 state='q27'  
 buffer = buffer + symbol  
 elif state=='q12':  
 if symbol.isalpha() or symbol=='\_':  
 state='q12'  
 buffer = buffer + symbol  
 elif symbol.isdigit():  
 state='q13'  
 buffer = buffer + symbol  
 else:  
 if buffer in self.SERVICE\_WORDS:  
 output\_sequance+=self.tokens['W'][buffer]+ ' '  
 elif buffer in self.OPERATIONS:  
 output\_sequance+=self.tokens['O'][buffer]+' '  
 else:  
 self.createTokensCod('I',buffer)  
 output\_sequance+=self.tokens['I'][buffer]+' '  
 state='q10'  
 buffer=''  
 i=i-1  
 elif state=='q13':  
 if symbol.isalpha() or symbol.isdigit() or symbol=='\_':  
 state='q13'  
 buffer = buffer + symbol  
 else:  
 if buffer in self.SERVICE\_WORDS:  
 output\_sequance+=self.tokens['W'][buffer]+ ' '  
 elif buffer in self.OPERATIONS:  
 output\_sequance+=self.tokens['O'][buffer]+' '  
 else:  
 self.createTokensCod('I',buffer)  
 output\_sequance+=self.tokens['I'][buffer]+' '  
 state='q10'  
 buffer = ''  
 i = i - 1  
 elif state=='q14':  
 if symbol=='?':  
 buffer=buffer+symbol  
 self.createTokensCod('R',buffer)  
 state='S'  
 output\_sequance+=self.tokens['R'][buffer]+ ' '  
 elif symbol=='-':  
 state = 'S'  
 self.createTokensCod('O', buffer)  
 output\_sequance += self.tokens['O'][buffer] + ' '  
 i=i-1  
 elif symbol in self.tokens['O'].keys():  
 state='q14'  
 buffer=buffer+symbol  
 else:  
 state='S'  
 self.createTokensCod('O', buffer)  
 output\_sequance += self.tokens['O'][buffer]+ ' '  
 i=i-1  
 elif state=='q15':  
 if symbol=='\*':  
 state='q16'  
 buffer=buffer+symbol  
 elif symbol=='/':  
 state='q17'  
 buffer=buffer+symbol  
 else:  
 state = 'S'  
 self.createTokensCod('O', buffer)  
 output\_sequance += self.tokens['O'][buffer] + ' '  
 i=i-1  
 elif state=='q16':  
 if symbol=='\*':  
 state='q18'  
 buffer=buffer+symbol  
 else:  
 state='q16'  
 buffer=buffer+symbol  
 elif state=='q17':  
 if i==len(output\_sequance)-1:  
 output\_sequance+='\n'  
 state='Z'  
 buffer=''  
 elif symbol=='\n':  
 output\_sequance+='\n'  
 state='S'  
 elif state=='q18':  
 if symbol!='/':  
 state='q16'  
 buffer=buffer+symbol  
 else:  
 output\_sequance += '\n'  
 state = 'S'  
 elif state=='q19':  
 state='q17'  
 buffer=buffer+symbol  
 elif state=='q20':  
 self.createTokensCod('R',buffer)  
 state='S'  
 if buffer!=' ':  
 output\_sequance+=self.tokens['R'][buffer]+' '  
 if buffer=='\n':  
 output\_sequance += '\n'  
 i=i-1  
 elif state=='q26':  
 if symbol=='\_':  
 state='q26'  
 buffer=buffer+symbol  
 else:  
 state='q1'  
 buffer=buffer+symbol  
 elif state == 'q27':  
 if symbol == '\_':  
 state = 'q27'  
 buffer = buffer + symbol  
 else:  
 state = 'q12'  
 buffer = buffer + symbol  
 elif state=='q28':  
 if symbol=='>':  
 buffer=buffer+symbol  
 self.createTokensCod('R', buffer)  
 state = 'Z'  
 output\_sequance += self.tokens['R'][buffer]  
 elif state=='Z':  
 break  
 i=i+1  
 for token\_class in self.tokens.keys():  
 with open('./files/%s.json' % token\_class, 'w') as write\_file:  
 data = {val: key for key, val in self.tokens[token\_class].items()}  
 json.dump(data, write\_file, indent=4, ensure\_ascii=False)  
 return output\_sequance,input\_sequence

import time  
from PyQt6.QtGui import QTextCursor  
  
from models.LecAnalysis import LecAnalysis  
  
  
class MainWindowController():  
 def \_\_init\_\_(self,window):  
 super(MainWindowController,self).\_\_init\_\_()  
 self.\_window = window  
 self.\_model = LecAnalysis()  
 @property  
 def window(self):  
 return self.\_window  
 @property  
 def model(self):  
 return self.\_model  
 def start(self):  
 out,inp=self.\_model.process()  
 self.\_window.output.setText(out)  
 self.\_window.input.setText(inp)

from PyQt6.QtWidgets import QMainWindow  
from PyQt6 import uic  
import os  
import sys  
from PyQt6.QtCore import Qt  
  
from controllers.MainWindowController import MainWindowController  
  
  
class MainWindow(QMainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 def create(self):  
  
 print(os.path.join(os.path.dirname(\_\_file\_\_), '..\\ui\\main.ui'))  
 uic.loadUi(os.path.join(os.path.dirname(\_\_file\_\_), '..\\ui\\main.ui'), self)  
 self.controller = MainWindowController(self)  
 self.startButton.clicked.connect(lambda :self.controller.start())  
 return self  
 def show(self):  
 super().show()  
 return self  
  
 def closeEvent(self, QCloseEvent):  
 del self.controller  
 sys.exit()

from PyQt6.QtWidgets import QApplication  
  
from views.MainWindow import MainWindow  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = QApplication([])  
 mainWindow = MainWindow()  
 mainWindow.create()  
 mainWindow.show()  
 app.exec()