ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

**БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ**

**ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

(НИУ «БелГУ»)

ИНСТИТУТ ИНЖЕНЕРНЫХ И ЦИФРОВЫХ ТЕХНОЛОГИЙ

КАФЕДРА МАТЕМАТИЧЕСКОГО

И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

**Отчет по лабораторной работе 4**

**по дисциплине: «Теория формальных языков»**

студента очного отделения

4 курса 12001801 группы

Капустина Виктора Сергеевича

Проверил(а):

доц. Великая Яна Геннадьевна

Белгород 2022

**Вариант 4**

**Цель работы:** изучить и научиться применять методы разработки и реализации лексических анализаторов.

**Задания**

1. Построить конечные распознаватели лексем заданного языка (см. варианты заданий).

2. Определить действия, выполняемые лексическим анализатором.

3. Программно реализовать лексический анализатор заданным методом в виде подпрограммы. При обнаружении лексической ошибки лексический анализатор должен выдавать сообщение о типе ошибки и её расположении (номер строки) в исходном тексте.

4. Написать программу, преобразующую исходный текст программы на заданном языке в таблицу лексем.

5. Обработать исходные тексты, содержащие и несодержащие лексические ошибки.

**Вариант 4**

**Лексика языка:**

1. Идентификаторы целочисленных переменных представляют собой букву, за которой могут следовать цифры. Все идентификаторы имеют один и тот же тип лексемы, значение лексемы – идентификатор.

2. Числовые константы представляют собой целые числа. Все числовые константы имеют один и тот же тип лексемы, значение лексемы – значение константы.

3. Строковая константа представляет собой последовательность символов, заключённую в апострофы. Если строковая константа должна содержать в себе апостроф, то в последовательности символов такой апостроф дублируется (записывается дважды). Все строковые константы имеют один и тот же тип лексемы, значение лексемы – последоветельность символов, заключённая в апострофы.

4. Ключевые слова: BEGIN, END, READ, WRITE, FOR, TO, DO. Каждое слово имеет свой тип лексемы. Значение лексема не имеет.

5. Аддитивные операции: + и - . Операции имеют один и тот же тип лексемы. Значение лексемы – номер операции.

6. Мультипликативные операции: \* и / . Операции имеют один и тот же тип лексемы. Значение лексемы – номер операции.

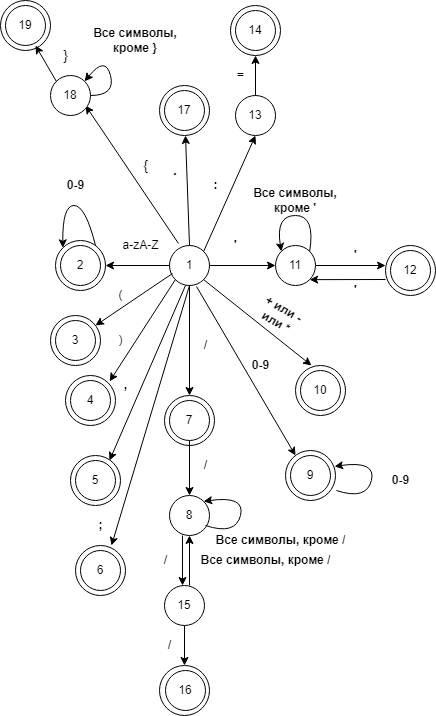
7. Знак оператора присваивания: := . Значения лексема не имеет.

8. Специальные знаки: (, ), , , ; . Каждый знак имеет свой тип лексемы. Значение лексема не имеет.

9. Разделителями являются пробел, конец строки, специальные знаки, знаки арифметических операций, знак оператора присваивания.

10. Комментарии начинаются и заканчиваются символами //.

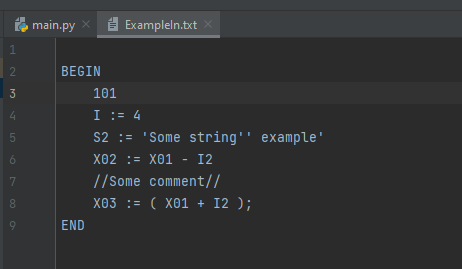
**Выполнение работы:**

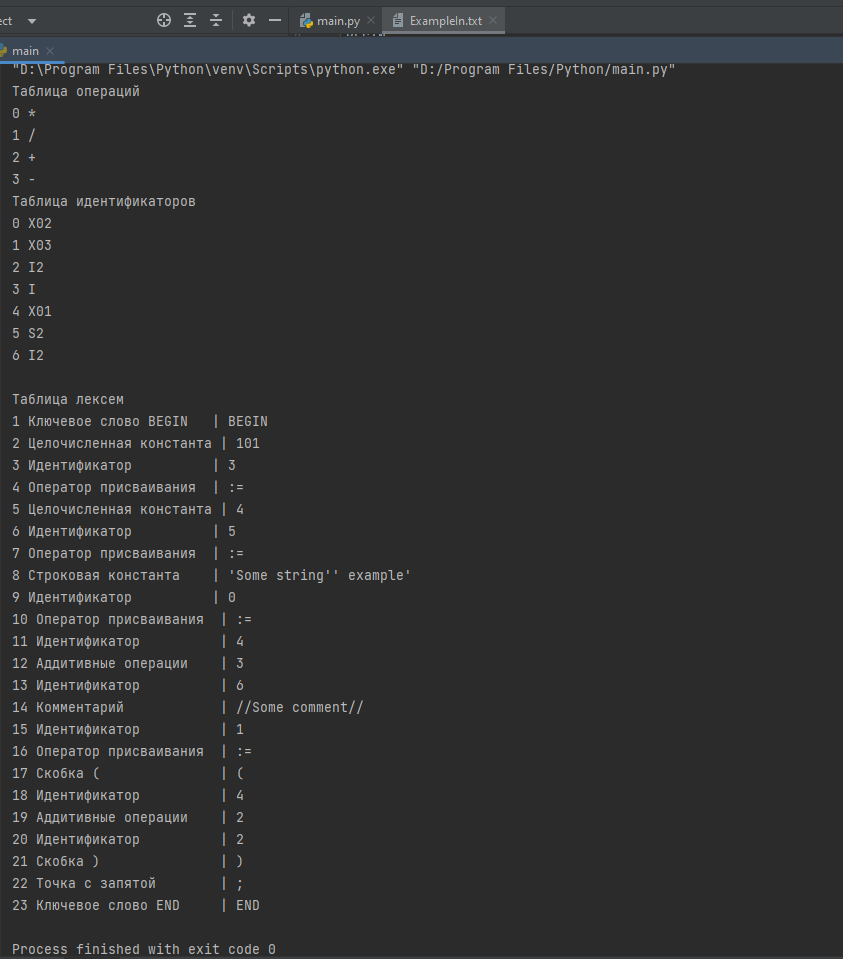
  
Рис. 1 Распознаватель лексем

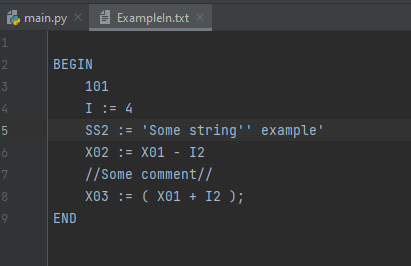
Программа будет работать следующим образом: из исходного файла читается строка, строка разбивается на слова и каждое слово будет проходить посимвольную проверку переходя по расположенному выше распознавателю. После этого следует цикл с посимвольным анализом. Переменная получает строку и разбивает ее на слова. После чего программа попадает в первое состояние, в котором проверяется первый символ слова, после чего, при совпадении, отправляет в соответствующее состояние.

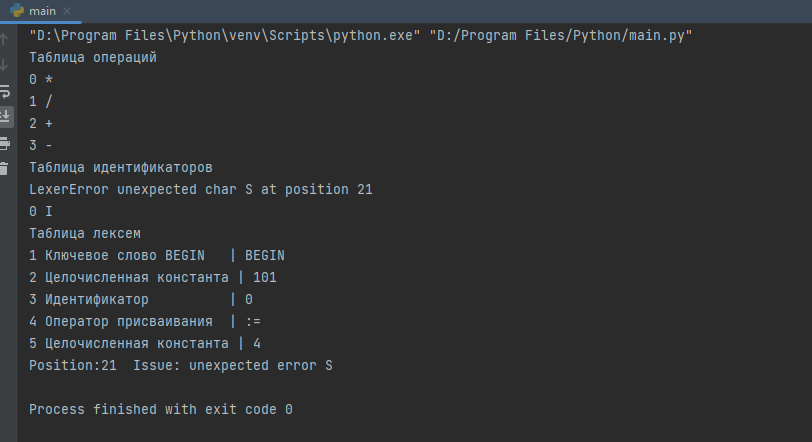
Проведем тестирование приложения. Для этого распишем несколько тестовых строк с Идентификаторами, специальными символами, аддитивными символами, знаками операторами присваивания, разделителями, комментариями, численными и строковыми константами, а также ключевыми словами. Для начала введем позитивный результат. Во второй раз намеренно допустим ошибки и проверим выдающие исключения.

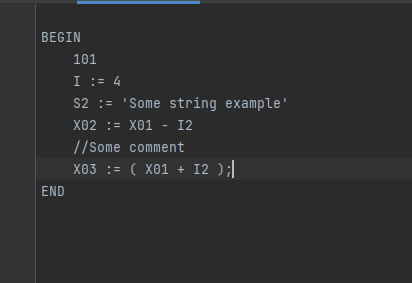
**Тестирование программы**

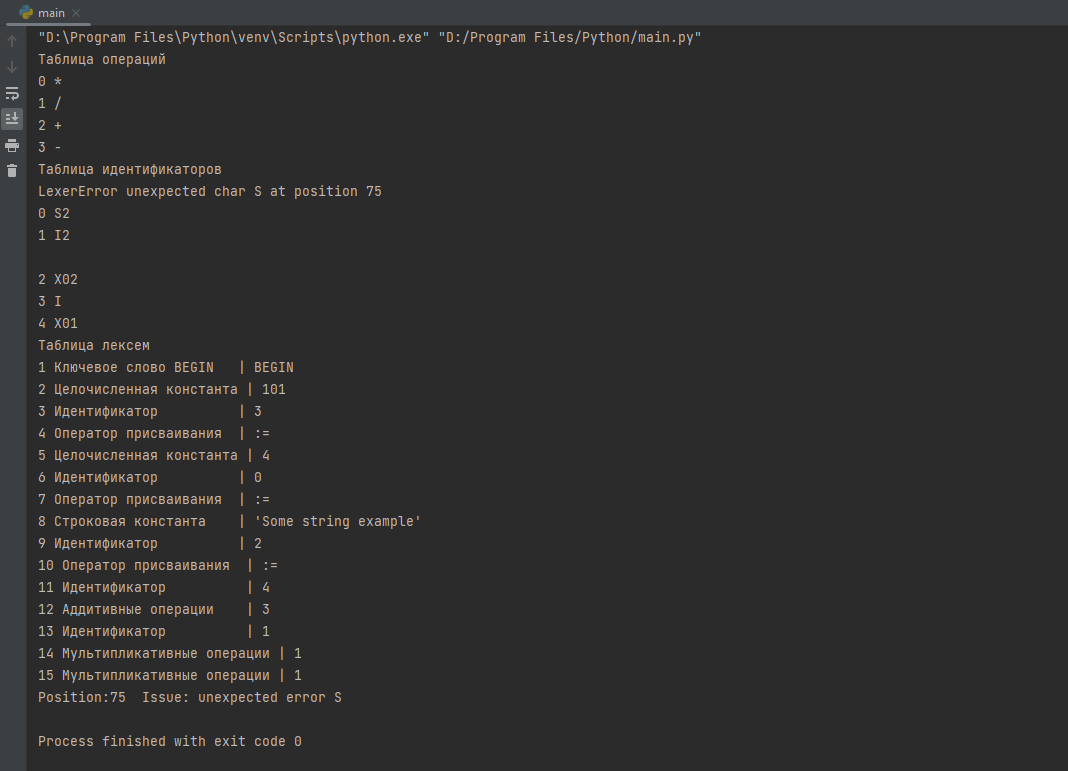
  
Рис. 2 Введенные позитивные данные

  
Рис. 3 Результат работы программы. Позитивный исход

  
Рис 4. Введенные негативные данные (Нарушен идентификатор)

  
Рис. 5 Результат работы программы. Негативный исход

  
Рис 6. Введенные негативные данные (Нарушен комментарий)

  
Рис. 7 Результат работы программы. Негативный исход

**Вывод:** в ходе лабораторной работы изучены и применены методы разработки и реализации лексических анализаторов.

**Приложение**

**Листинг приложения 1**

import re

class Lexing(object):

def \_\_init\_\_(self, rules, skip\_whitespace=True):

self.rules = []

for Regexr, TYPE in rules:

self.rules.append((re.compile(Regexr), TYPE))

self.skip\_whitespace = skip\_whitespace

self.ServiceWord = re.compile('[^\r\n\t\f\v ]')

def Buffer(self, buffer):

self.buffer = buffer

self.position = 0

def Token(self):

if self.position >= len(self.buffer):

return None

if self.skip\_whitespace:

FlagPos = self.ServiceWord.search(self.buffer, self.position)

if FlagPos:

self.position = FlagPos.start()

else:

return None

for regex, type in self.rules:

FlagPos = regex.match(self.buffer, self.position)

if FlagPos:

tok = Token(type, FlagPos.group(), self.position)

self.position = FlagPos.end()

return tok

raise LexerError(self.position, self.buffer[self.position])

def tokens(self):

while 1:

tok = self.Token()

if tok is None: break

yield tok

class Token(object):

def \_\_init\_\_(self, TYPE, value, position):

self.TYPE = TYPE

self.value = value

self.position = position

def \_\_str\_\_(self):

return f'{self.TYPE:<22} | {self.value:<22}'

class LexerError(Exception):

def \_\_init\_\_(self, position, char):

self.position = position

self.char = char

BEGIN = 'Ключевое слово BEGIN'

END = 'Ключевое слово END'

READ = 'Ключевое слово READ'

WRITE = 'Ключевое слово WRITE'

FOR = 'Ключевое слово FOR'

TO = 'Ключевое слово TO'

DO = 'Ключевое слово DO'

ASSIGN = "Оператор присваивания"

STRING = "Строковая константа"

ADDITIV = "Аддитивные операции"

COMP = "Операция сравнения"

MULTI = "Мультипликативные операции"

INT = 'Целочисленная константа'

ID = 'Идентификатор'

COMMENT = "Комментарий"

SPEC1 = "Запятая"

SPEC2 = "Скобка ("

SPEC3 = "Скобка )"

SPEC4 = "Точка с запятой"

Operations = ["\*", "/", "+", "-"]

id\_table = []

counter = 0

Patterns = [

(r'BEGIN', BEGIN),

(r'END', END),

(r'READ', READ),

(r'WRITE', WRITE),

(r'FOR', FOR),

(r'TO', TO),

(r'DO', DO),

(r'([/]{2}.\*[/]{2})', COMMENT),

(r"([']{1}[']{0,2}.\*[']{1})", STRING),

(r'\:=', ASSIGN),

(r'\+', ADDITIV),

(r'-', ADDITIV),

(r'\\*', MULTI),

(r'\/', MULTI),

(r'[0-9]+', INT),

(r'\,', SPEC1),

(r'\(', SPEC2),

(r'\)', SPEC3),

(r'\;', SPEC4),

(r'[a-zA-Z]{1}\d{0,}\W', ID),

]

print('Таблица операций')

for operate in Operations:

print(counter, operate)

counter += 1

print('Таблица идентификаторов')

Lex1 = Lexing(Patterns, skip\_whitespace=True)

with open("ExampleIn.txt", "r") as myfile:

Examples = myfile.read()

Lex1.Buffer(Examples)

try:

for tok in Lex1.tokens():

if tok is not None:

if tok.TYPE == ID:

id\_table.append(tok.value)

except LexerError as err:

print(f'LexerError unexpected char {err.char} at position {err.position}')

unique = set(id\_table)

id\_list = list(unique)

count1 = 0

for operate in id\_list:

print(count1, operate)

count1 += 1

print("Таблица лексем")

lx = Lexing(Patterns, skip\_whitespace=True)

with open("ExampleIn.txt", "r") as myfile:

data = myfile.read()

lx.Buffer(data)

count = 0

try:

for tok in lx.tokens():

if tok is not None:

if tok.TYPE == ID:

if tok.value in unique:

id\_list = list(unique)

tok.value = id\_list.index(tok.value)

if tok.TYPE == ADDITIV:

if tok.value == '+':

tok.value = Operations.index('+')

if tok.value == '-':

tok.value = Operations.index('-')

if tok.TYPE == MULTI:

if tok.value == '\*':

tok.value = Operations.index('\*')

if tok.value == '/':

tok.value = Operations.index('/')

count += 1

print(count, tok)

except LexerError as err:

print(f'Position:{err.position} Issue: unexpected error {err.char}')

**Конец листинга 1**