|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |
| Институт Информационных Технологий | | |
| Кафедра Вычислительной техники | | |

|  |  |
| --- | --- |
| **ОТЧЕТ О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ №3**  «Лексический анализатор на базе конечного автомата на python» | |
|  | |
| **по дисциплине** | |
| **«**Теория формальных языков**»** | |
|  | |
| Выполнил студент группы ИКБО-04-22 | *Основин А.И.* |
| Принял старший преподаватель | *Боронников А.С.* |

|  |  |  |
| --- | --- | --- |
| Практическая работа выполнена | «\_\_»\_\_\_\_\_\_\_2023 г. |  |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_2023 г. |  |

СОДЕРЖАНИЕ

[1 ПОСТАНОВКА ЗАДАЧИ 3](#_Toc153848325)

[1.1 Условия задачи 3](#_Toc153848326)

[1.2 Описание задания 3](#_Toc153848327)

[2 РЕАЛИЗАЦИЯ ПРОГРАММЫ 5](#_Toc153848328)

[3 ТЕСТИРОВАНИЕ 7](#_Toc153848329)

[3.1 Содержимое тестового файла 7](#_Toc153848330)

[3.2 Ожидаемый вывод программы 7](#_Toc153848331)

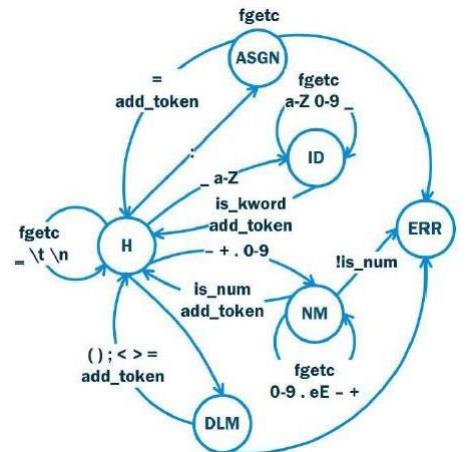
[3.3 Результаты тестирования 7](#_Toc153848332)

[4 ВЫВОД 9](#_Toc153848333)

# ПОСТАНОВКА ЗАДАЧИ

## Условия задачи

Написать на любом языке программирования лексический анализатор на базе конечного автомата входного языка, описанного диаграммой состояний, представленной на Рисунке 1.



**Рисунок 1 – Диаграмма состояний конечного автомата входного языка**

## Описание задания

Входной язык, содержит операторы цикла for (…; …; …) do …, разделённые символом «;». Операторы цикла содержат идентификаторы, знаки сравнения, =, десятичные числа с плавающей точкой (в обычной и экспоненциальной форме), знак присваивания (:=).

Описанный выше входной язык может быть задан с помощью КС грамматики:

G ({for, do, ‘:=’, ‘’, ‘=’, ‘-’, ‘+’, ‘(‘, ‘)’, ‘;’, ‘.’, ‘\_’, ‘a’, ‘b’, ‘c’, ..., ‘x’, ‘y’, ‘z’, ‘A’, ‘B’, ‘C’, ..., ‘X’, ‘Y’, ‘Z’, ‘0’, ‘1’, ‘2’, ’3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’}, {S, A, B, C, I, L, N, Z}, P, S) с правилами Р (правила представлены в расширенной форме Бэкуса-Наура):

S → for (A; A; A;) do A; | for (A; A; A;) do S; | for (A; A; A;) do A; S A → I := B B → C > C | C < C | C = C C → I | N I → (\_ | L) {\_ | L | Z | 0} N → [- | +] ({0 | Z} . {0 | Z} | {0 | Z}. | {0 | Z}) [(e | E) [- | +]{0 | Z}][f | l | F | L] L → a | b | c | … | x | y | z | A | B | C | … | X | Y | Z Z → 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Целевым символом грамматики является символ S.

Лексемы входного языка разделим на несколько классов:

* ключевые слова языка (for, do);
* разделители и знаки операций (‘(‘, ‘)’, ‘;’, '<', ‘>’, ‘=’);
* знак операции присваивания (‘:=’);
* идентификаторы;
* десятичные числа с плавающей точкой (в обычной и экспоненциальной форме).

Границами лексем будут служить пробелы, знаки табуляции, знаки перевода строки и возврата каретки, круглые скобки, точка с запятой и знак двоеточия. При этом круглые скобки и точка с запятой сами являются лексемами, а знак двоеточия, являясь границей лексемы, в то же время является и началом другой лексемы – операции присваивания.

# РЕАЛИЗАЦИЯ ПРОГРАММЫ

В Листинге 1 представлена реализация программы лексический анализатор цикла.

*Листинг 1 – Код программы на языке программирования Python*

from enum import Enum

from os import path, SEEK\_SET

import re

def tokenize(filepath: str) -> dict:

buffer: str

states = Enum('state', ['h', 'id', 'nm', 'dlm', 'asgn', 'err'])

tokens = {

'KEYWORDS': [],

'DELIMITERS': [],

'ASSIGN': [],

'IDs': [],

'NUMBERS': [],

'UNKNOWN': []

}

with open(filepath, 'r') as file:

current\_state = states['h']

while symbol := file.read(1):

match current\_state.name:

case 'h':

while symbol in [' ', '\t', '\n']:

symbol = file.read(1)

if symbol == ':':

current\_state = states['asgn']

elif symbol.isalpha() or symbol == '\_':

current\_state = states['id']

elif symbol.isdigit() or symbol in ['-', '+', '.']:

current\_state = states['nm']

elif symbol in ['(', ')', ';', '<', '>', '=']:

current\_state = states['dlm']

buffer = symbol

case 'asgn':

buffer += symbol

if symbol == '=':

tokens['ASSIGN'].append(buffer)

else:

tokens['UNKNOWN'].append(buffer)

buffer = ''

current\_state = states['h']

case 'id':

while symbol.isalnum() or symbol == '\_':

buffer += symbol

symbol = file.read(1)

if buffer in ['for', 'do']:

tokens['KEYWORDS'].append(buffer)

else:

tokens['IDs'].append(buffer)

file.seek(file.tell() - 1, SEEK\_SET)

current\_state = states['h']

case 'nm':

while symbol.isdigit() or symbol in ['.', 'e', 'E', '-', '+']:

buffer += symbol

symbol = file.read(1)

if re.match('^(?!^\s\*$)\d\*(\.\d+)?((e|E)(-|\+)?\d+)?$', buffer):

tokens['NUMBERS'].append(buffer)

else:

tokens['UNKNOWN'].append(buffer)

file.seek(file.tell() - 1, SEEK\_SET)

current\_state = states['h']

case 'dlm':

tokens['DELIMITERS'].append(buffer)

file.seek(file.tell() - 1, SEEK\_SET)

current\_state = states['h']

return tokens

def main():

filepath = path.abspath(input('Enter filepath: ')) # files/test.txt

if path.exists(filepath):

tokens = tokenize(filepath)

else:

exit(f"Path {filepath} doesn't exists.")

for (key, value) in tokens.items():

print(key, ":", value)

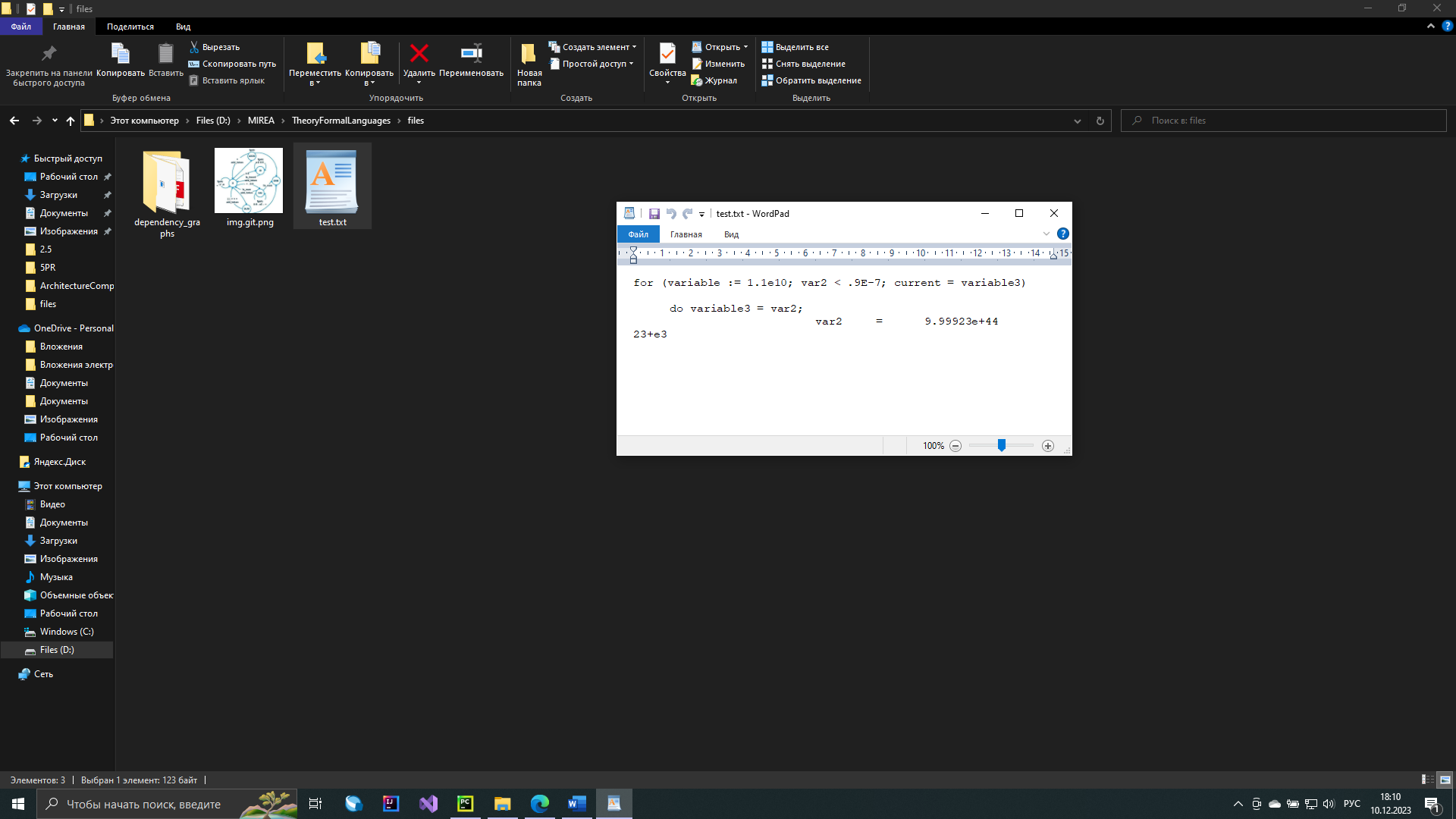
if \_\_name\_\_ == "\_\_main\_\_":

main()

# ТЕСТИРОВАНИЕ

## Содержимое тестового файла

На Рисунке 2 представлено содержимое текстового тестового файла, который подаётся на вход программе, реализующей лексический анализатор.



**Рисунок 2 – Содержимое текстового тестового файла**

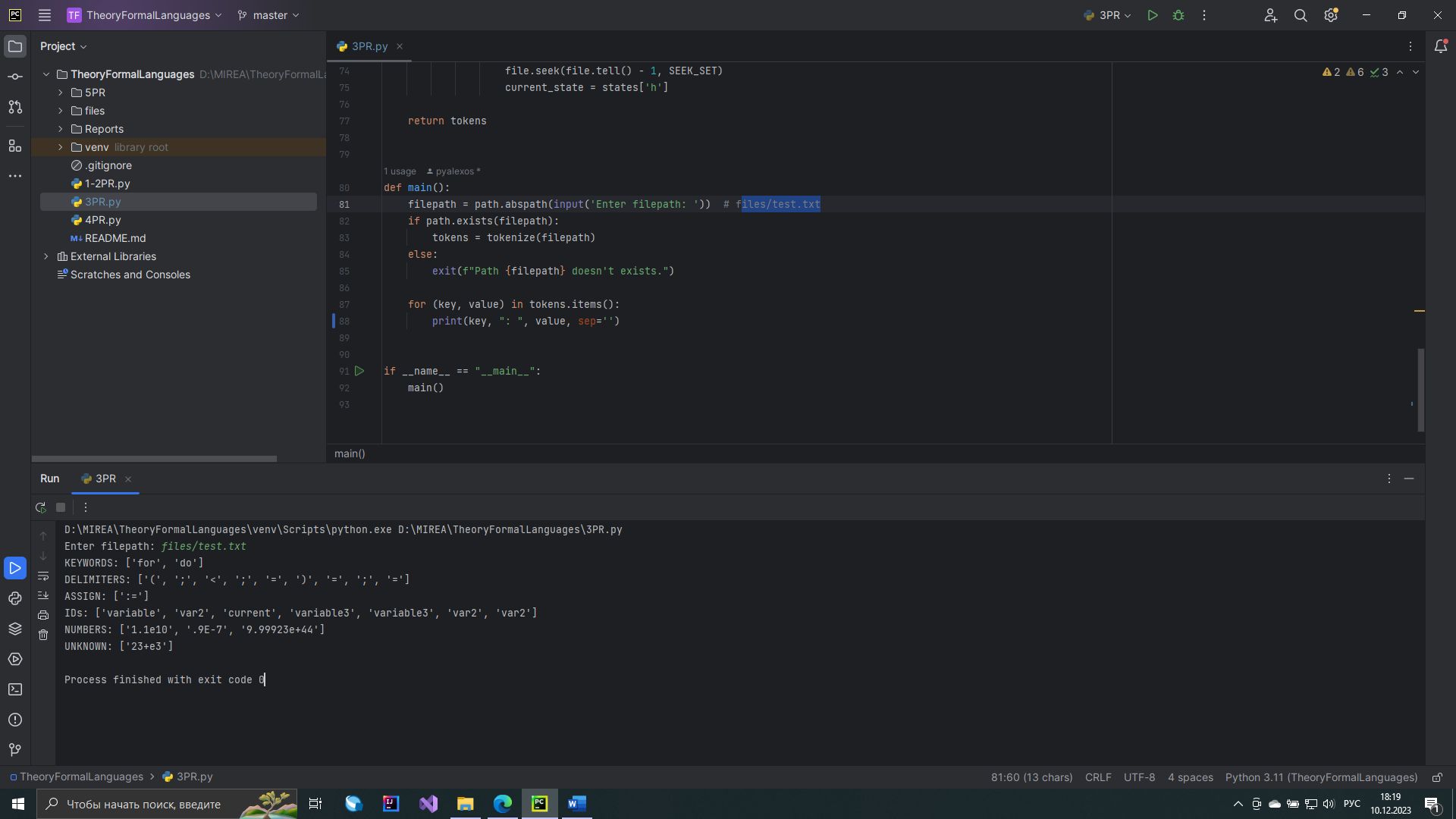
## Ожидаемый вывод программы

При данном содержимом тестового файла вывод программы должен быть следующий (лексемы, разбитые по классам):

* ключевые слова языка: ['for', 'do'];
* разделители и знаки операций: ['(', ';', '<', ';', '=', ')', '=', ';', '='];
* знак операции присваивания: [':='];
* идентификаторы: ['variable', 'var2', 'current', 'variable3', 'variable3', 'var2', 'var2'];
* десятичные числа с плавающей точкой (в обычной и экспоненциальной форме): ['1.1e10', '.9E-7', '9.99923e+44'];
* нераспознанные лексемы: ['23+e3'].

## Результаты тестирования

На Рисунке 3 представлен вывод программы. Как видно из Рисунка, ожидаемый результат совпадает с выводом программы, следовательно, разработанный лексический анализатор цикла работает корректно.



**Рисунок 3 – Результат работы программы**

# ВЫВОД

В ходе выполнения данной практической работы был разработан лексический анализатор цикла на базе конечного автомата входного языка. Программа успешного прошла тестирование, следовательно, реализация корректна.