|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатики и систем управления

КАФЕДРА Теоретической информатики и компьютерных технологий

**Лабораторная работа № 3**

**«Лексический анализатор на основе регулярных выражений»**

***по курсу «Конструирование компиляторов»***

Студент *Ионов Т.Р. 61Б*

Преподаватель *Коновалов А.B.*

*Москва, 2022 г.*

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc106224508)

[2 Индивидуальное задание 3](#_Toc106224509)

[3 Реализация 3](#_Toc106224510)

[4 Тестирование 4](#_Toc106224511)

[Вывод 4](#_Toc106224512)

# 1 Цель работы

Целью данной работы является приобретение навыка разработки простейших лексических анализаторов, работающих на основе поиска в тексте по образцу, заданному регулярным выражением.

# 2 Индивидуальное задание

Числа фибоначчиевой системы счисления: последовательности знаков “0”, “1”, причем две единицы не могут соседствовать друг с другом.

Идентификаторы: последовательности латинский букв, в которых гласные и согласные чередуются.

3 Реализация

Листинг 1 −скрипт lexer.py.

import re

import sys

class Token(object):

def \_\_init\_\_(self, type, val, col, row):

self.type = type

self.val = val

self.col = col

self.row = row

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

return f"{self.type}\t({self.row},{self.col}):\t{self.val}"

class LexerError(Exception):

def \_\_init\_\_(self, col, row):

self.col = col

self.row = row

def \_\_str\_\_(self) -> str:

return f"syntax error\t({self.row},{self.col})"

class Lexer(object):

def \_\_init\_\_(self, rules):

regex\_parts = []

for regex, type in rules:

regex\_parts.append('(?P<%s>%s)' % (type, regex))

self.regex = re.compile('|'.join(regex\_parts))

self.ws\_re = re.compile('\S')

def input(self, buf, row):

self.buf = buf

self.col = 0

self.row = row

def token(self):

if self.col >= len(self.buf):

return None

m = self.ws\_re.search(self.buf, self.col)

if m:

self.col = m.start()

else:

return None

m = self.regex.match(self.buf, self.col)

if m:

tok\_type = m.lastgroup

tok = Token(tok\_type, m.group(tok\_type), self.col, self.row)

self.col = m.end()

return tok

raise LexerError(self.col, self.row)

def tokens(self):

""" Returns an iterator to the tokens found in the buffer.

"""

while 1:

token = self.token()

if token is None: break

yield token

if \_\_name\_\_ == '\_\_main\_\_':

VOWELS = "[aeiou]"

CONS = "[b-df-hj-np-tv-z]"

IDENT = "\\b((0(10)\*1?)|(1(01)\*0?))\\b".\

replace("0", CONS).replace("1", VOWELS)

NUMBER = "\\b(1(0+1?)\*)|0\\b"

rules = [

(NUMBER, 'NUMBER'),

(IDENT, 'IDENT')

]

lexer = Lexer(rules)

path = sys.argv[1]

lines = open(path, 'r').read().split("\n")

for i, line in enumerate(lines):

lexer.input(line, i+1)

try:

for token in lexer.tokens():

print(token)

except LexerError as e:

print(e)

# 4 Тестирование

Вход:

a

b

bab

abb

101

01

0

1

Результат:

IDENT (1,2): a

IDENT (2,0): b

IDENT (3,0): bab

syntax error (4,0)

NUMBER (5,0): 101

syntax error (6,0)

NUMBER (7,0): 0

NUMBER (8,0): 1

# Вывод

В результате выполнения данной лабораторной работы были приобретены навыки разработки простейших лексических анализаторов, работающих на основе поиска в тексте по образцу, заданному регулярным выражением.