



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
входного языка на Python»

по дисциплине

«Теория формальных языков»

Выполнил студент группы ИКБО-15-22

Оганнисян Г.А.

Принял старший преподаватель

Боронников А.С.

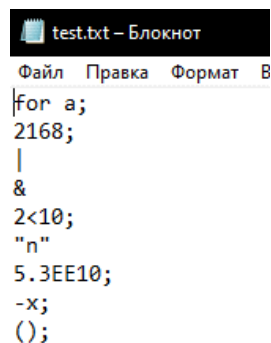
Практическая работа
выполнена

«__»_____2023 г.

«Зачтено»

«__»_____2023 г.

Москва 2021

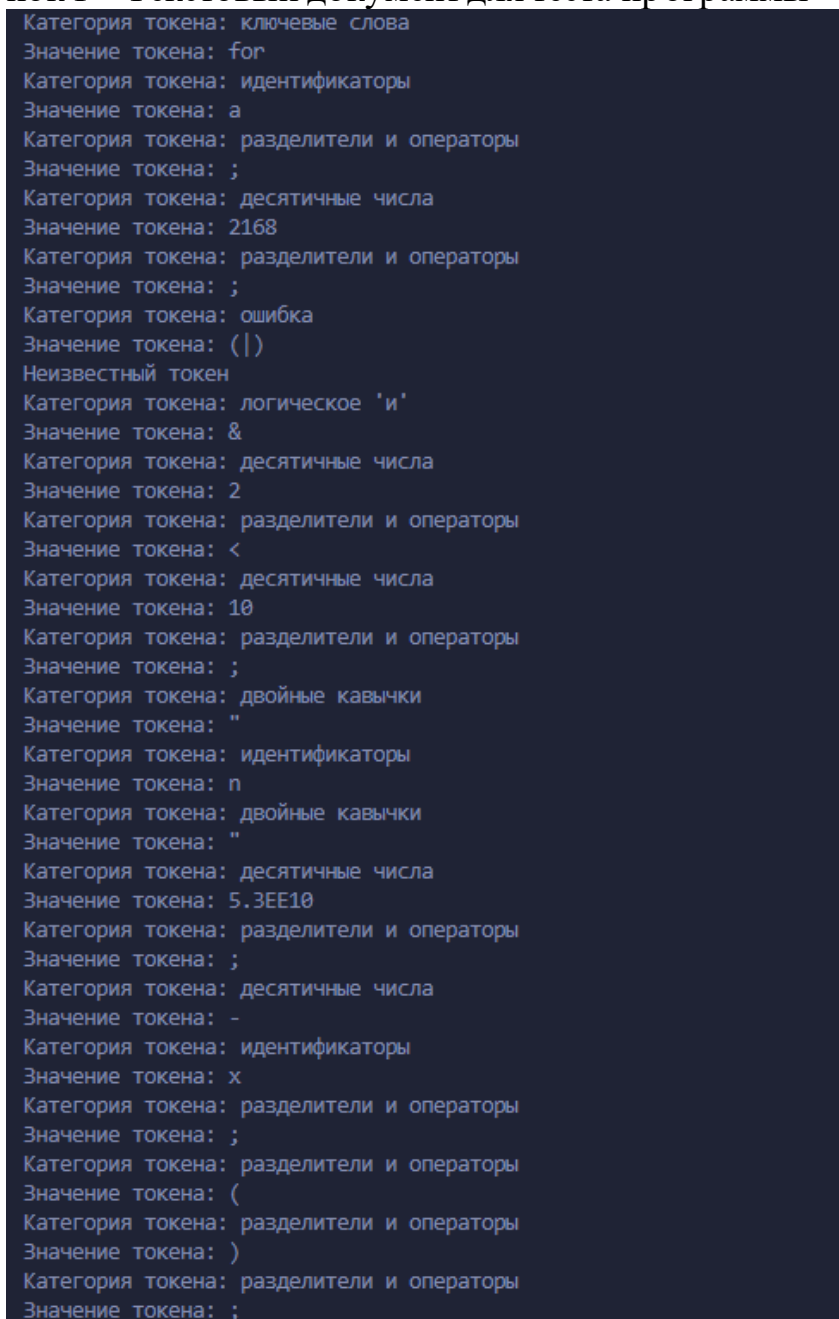


```

test.txt – Блокнот
Файл  Правка  Формат  В
for a;
2168;
|
&
2<10;
"n"
5.3EE10;
-x;
();

```

Рисунок 1 – Текстовый документ для теста программы



```

Категория токена: ключевые слова
Значение токена: for
Категория токена: идентификаторы
Значение токена: a
Категория токена: разделители и операторы
Значение токена: ;
Категория токена: десятичные числа
Значение токена: 2168
Категория токена: разделители и операторы
Значение токена: ;
Категория токена: ошибка
Значение токена: (|)
Неизвестный токен
Категория токена: логическое 'и'
Значение токена: &
Категория токена: десятичные числа
Значение токена: 2
Категория токена: разделители и операторы
Значение токена: <
Категория токена: десятичные числа
Значение токена: 10
Категория токена: разделители и операторы
Значение токена: ;
Категория токена: двойные кавычки
Значение токена: "
Категория токена: идентификаторы
Значение токена: n
Категория токена: двойные кавычки
Значение токена: "
Категория токена: десятичные числа
Значение токена: 5.3EE10
Категория токена: разделители и операторы
Значение токена: ;
Категория токена: десятичные числа
Значение токена: -
Категория токена: идентификаторы
Значение токена: x
Категория токена: разделители и операторы
Значение токена: ;
Категория токена: разделители и операторы
Значение токена: (
Категория токена: разделители и операторы
Значение токена: )
Категория токена: разделители и операторы
Значение токена: ;

```

Рисунок 2 – Вывод программы

Этот код представляет собой простой лексический анализатор, который

разбирает входной текст из файла "test.txt" на лексемы. Лексемы - это минимальные значимые элементы языка программирования. Код читает текст из файла, а затем анализирует его, распознавая и классифицируя различные лексемы.

Прежде всего, определены две функции: **get_next_char** возвращает следующий символ из входного текста, а **consume_char** удаляет обработанный символ из входного текста.

Затем созданы структуры данных для хранения лексем: класс **Token**, представляющий отдельную лексему, и класс **LexemeTable** для хранения всех лексем.

Далее код инициализирует лексемную таблицу **lt** и открывает файл "test.txt" для чтения. После этого следует основная функция **lex_analyzer**, которая проходит по входному тексту и распознает различные лексемы.

В цикле **while** проверяется текущий символ, и в зависимости от его типа выполняются различные действия:

- Пропускаются пробелы и пробельные символы.
- Распознаются и обрабатываются операторы присваивания ":=".
- Распознаются и обрабатываются различные операторы, скобки и другие символы.
- Распознаются и обрабатываются строки в двойных кавычках.
- Распознаются и обрабатываются логический оператор "&".
- Распознаются идентификаторы и ключевые слова.
- Распознаются и обрабатываются числа, включая десятичные.

Каждая распознанная лексема добавляется в лексемную таблицу с помощью метода **add_token**, который выводит информацию о категории и значении лексемы.

После завершения анализа вызывается функция **lex_analyzer**, которая выводит результаты на экран.

В случае обнаружения неизвестного символа выводится сообщение об ошибке.

Листинг Кода

```
# Функция, возвращающая следующий символ из входного текста
def get_next_char():
    global input_text
    if input_text:
        return input_text[0]
    return None

# Функция, потребляющая обработанный символ из входного текста
def consume_char():
    global input_text
    if input_text:
        input_text = input_text[1:]

# Открываем текстовый файл для чтения
with open("TFYA\\task3\\test.txt", "r") as file:
    input_text = file.read()

# Определение структур данных для хранения лексем
class Token:
    def __init__(self, token_category, token_value):
        self.token_category = token_category
        self.token_value = token_value

class LexemeTable:
    def __init__(self):
        self.lexemes = []

    def add_token(self, token):
        self.lexemes.append(token)
        print(f"Категория токена: {token.token_category}")
        print(f"Значение токена: {token.token_value}")

# Инициализация таблицы лексем
lt = LexemeTable()

# Функция для проверки, является ли идентификатор ключевым словом
def is_keyword(identifier):
    keywords = ["for", "do"]
    return identifier in keywords

# Функция для обработки лексем и добавления их в таблицу
def add_token(token):
    lt.add_token(token)

# Функция лексического анализа
def lex_analyzer():
    while input_text:
        current_char = get_next_char()
```

```

if current_char.isspace(): # Пропускаем пробелы
    while current_char.isspace():
        consume_char()
        current_char = get_next_char()
    continue # Пропускаем текущую итерацию

if current_char == ':':
    consume_char()
    if get_next_char() == '=':
        consume_char()
        token = Token("операторы присваивания", ":=")
        add_token(token)
elif current_char in '()<=>':
    consume_char()
    token = Token("разделители и операторы", current_char)
    add_token(token)
elif current_char == '"':
    consume_char()
    token = Token("двойные кавычки", '"')
    add_token(token)
elif current_char == '&':
    consume_char()
    token = Token("логическое 'и'", '&')
    add_token(token)
elif current_char.isalpha():
    identifier = current_char
    consume_char()
    while get_next_char().isalnum() or get_next_char() == '_':
        identifier += get_next_char()
        consume_char()
    if is_keyword(identifier):
        token = Token("ключевые слова", identifier)
    else:
        token = Token("идентификаторы", identifier)
    add_token(token)
elif current_char.isdigit() or current_char == '-':
    number = current_char
    consume_char()
    while get_next_char().isdigit() or get_next_char() in '.eE+-':
        number += get_next_char()
        consume_char()
    token = Token("десятичные числа", number)
    add_token(token)
else:
    consume_char()
    print("Категория токена: ошибка") # Нераспознанный символ
    print(f"Значение токена: ({current_char})")
    print("Неизвестный токен")

```

```

# Вызываем лексический анализатор
lex_analyzer()

```