|  |
| --- |
|  |
| 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

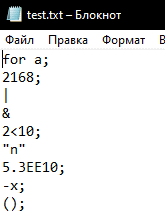


Рисунок 1 – Текстовый документ для теста программы

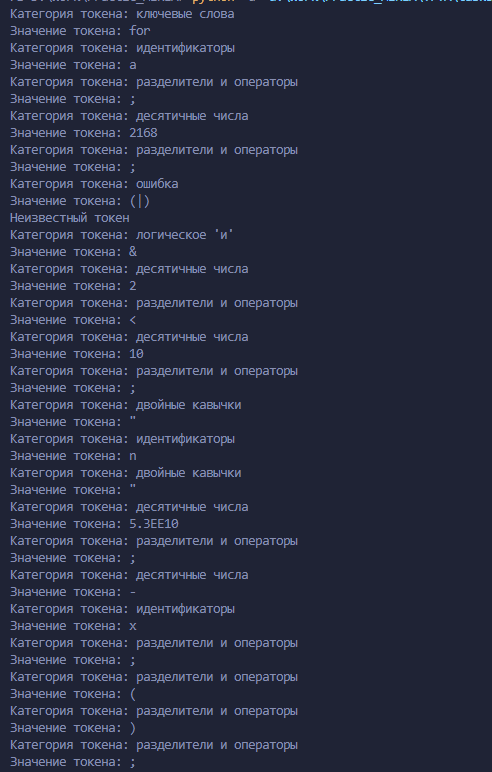


Рисунок 2 – Вывод программы

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 проходится 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()