Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №2

По дисциплине: «Языковые процессы интеллектуальных систем»

# Тема: «Проектирование лексического анализатора»

Выполнил:

Студент 4 курса

Группы ИИ-21

Литвинюк Т. В.

Проверил:

Монтик Н.С.

Брест 2024

**Цель работы:** изучение основных понятий теории регулярных грамматик, ознакомление с назначением и принципами работы лексических анализаторов (сканеров), получение практических навыков построения сканера на примере заданного простейшего входного языка.

11. Входной язык содержит последовательность описаний записей (record) в соответствии со спецификацией языка Паскаль, разделенных символом ;(точка с запятой). Считать, что записи могут содержать только поля скалярных типов integer, real, byte, word, char и строки string с возможным указанием длины строки в квадратных скобках. import re

KEYWORDS = {'record', 'integer', 'real', 'byte', 'word', 'char', 'string'}

OPERATORS = {';', '=', '[', ']', '(', ')', ':', ',', '.'}

WHITESPACE = {' ', '\t', '\n', '\r'}

COMMENT\_START = '{'

COMMENT\_END = '}'

MAX\_ID\_LENGTH = 32  # Максимальная длина идентификатора и строковых констант

TOKEN\_TYPES = {

    'KEYWORD': 'Ключевое слово',

    'IDENTIFIER': 'Идентификатор',

    'OPERATOR': 'Оператор',

    'NUMBER': 'Число',

    'STRING': 'Строковая константа',

    'COMMENT': 'Комментарий',

    'ERROR': 'Ошибка'

}

class Token:

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

        self.type = type\_

        self.value = value

        self.position = position

    def \_\_repr\_\_(self):

        return f"{TOKEN\_TYPES[self.type]}, '{self.value}', at position {self.position}"

class Lexer:

    def \_\_init\_\_(self, input\_text):

        self.text = input\_text

        self.position = 0

        self.tokens = []

    def get\_char(self):

        """Возвращает текущий символ и переходит к следующему."""

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

            return None

        char = self.text[self.position]

        self.position += 1

        return char

    def peek\_char(self):

        """Просматривает текущий символ, но не изменяет позицию."""

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

            return None

        return self.text[self.position]

    def tokenize(self):

        """Основной метод, который разбивает текст на токены."""

        while self.position < len(self.text):

            char = self.get\_char()

            if char in WHITESPACE:

                continue

            if char == COMMENT\_START:

                self.handle\_comment()

                continue

            if char.isalpha() or char == '\_':

                self.handle\_identifier\_or\_keyword(char)

                continue

            if char.isdigit():

                self.handle\_number(char)

                continue

            if char == '"':

                self.handle\_string()

                continue

            if char in OPERATORS:

                self.tokens.append(Token('OPERATOR', char, self.position))

                continue

            self.tokens.append(Token('ERROR', char, self.position))

        return self.tokens

    def handle\_comment(self):

        """Обрабатывает комментарии, пропуская текст до закрывающей скобки."""

        comment\_content = ''

        while (char := self.get\_char()) != COMMENT\_END:

            if char is None:  # Ошибка: незакрытый комментарий

                self.tokens.append(

                    Token('ERROR', 'Unclosed comment', self.position))

                return

            comment\_content += char

        self.tokens.append(Token('COMMENT', comment\_content, self.position))

    def handle\_identifier\_or\_keyword(self, char):

        """Обрабатывает идентификаторы или ключевые слова."""

        identifier = char

        while (char := self.peek\_char()) and (char.isalnum() or char == '\_'):

            identifier += self.get\_char()

        if len(identifier) > MAX\_ID\_LENGTH:

            self.tokens.append(

                Token('ERROR', 'Identifier too long', self.position))

        elif identifier in KEYWORDS:

            self.tokens.append(Token('KEYWORD', identifier, self.position))

        else:

            self.tokens.append(Token('IDENTIFIER', identifier, self.position))

    def handle\_number(self, char):

        """Обрабатывает числовые литералы."""

        number = char

        while (char := self.peek\_char()) and char.isdigit():

            number += self.get\_char()

        if char == '.':  # Это вещественное число

            number += self.get\_char()

            while (char := self.peek\_char()) and char.isdigit():

                number += self.get\_char()

        self.tokens.append(Token('NUMBER', number, self.position))

    def handle\_string(self):

        """Обрабатывает строковые константы."""

        string\_literal = ''

        while (char := self.get\_char()) != '"':

            if char is None:  # Ошибка: незакрытая строка

                self.tokens.append(

                    Token('ERROR', 'Unclosed string literal', self.position))

                return

            string\_literal += char

        if len(string\_literal) > MAX\_ID\_LENGTH:

            self.tokens.append(

                Token('ERROR', 'String too long', self.position))

        else:

            self.tokens.append(Token('STRING', string\_literal, self.position))

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

    input\_text = """

    record MyRecord;

    integer a;

    real b;

    string[30] str;

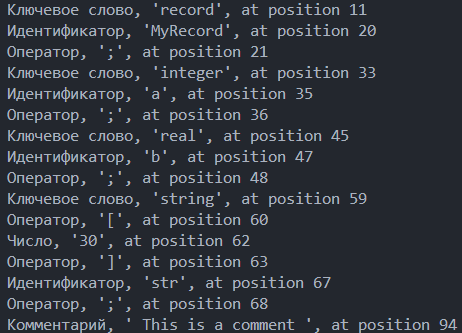
    { This is a comment }

    """

    lexer = Lexer(input\_text)

    tokens = lexer.tokenize()

    print(\*tokens, sep="\n")

****

**Вывод:** в ходе выполнения лабораторной работы изучил основные понятия теории регулярных грамматик, ознакомился с назначением и принципами работы лексических анализаторов (сканеров), получил практические навыки построения сканера на примере заданного простейшего входного языка.