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

Кирилович А. А.

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.

Вариант 17. Входной язык содержит многоуровневые арифметические выражения, разделенные символом; (точка с запятой). Выражения могут включать идентификаторы, числовые константы, скобки для задания приоритетов, а также функции (например, sin, cos). Операции могут включать сложение, вычитание, умножение и деление.

import 'dart:io';

class TokenType {

  static const String IDENTIFIER = 'IDENTIFIER';    // Идентификаторы

  static const String NUMBER = 'NUMBER';            // Числовые константы

  static const String OPERATOR = 'OPERATOR';        // Операторы (+, -, \*, /)

  static const String LPAREN = 'LPAREN';            // Левая скобка (

  static const String RPAREN = 'RPAREN';            // Правая скобка )

  static const String SEMICOLON = 'SEMICOLON';      // Точка с запятой ;

  static const String FUNCTION = 'FUNCTION';        // Функции (sin, cos)

  static const String COMMENT = 'COMMENT';          // Комментарии

  static const String CONSTANT = 'CONSTANT';        // Константы (e, pi)

}

class Token {

  String type;

  String value;

  Token(this.type, this.value);

  @override

  String toString() {

    return 'Token(type: $type, value: "$value")';

  }

}

class Lexer {

  String input;

  int position = 0;

  List<Token> tokens = [];

  Lexer(this.input);

  String get currentChar => input[position];

  bool isAtEnd() => position >= input.length;

  void skipWhitespace() {

    while (!isAtEnd() && \_isWhitespace(currentChar)) {

      position++;

    }

  }

  bool \_isWhitespace(String char) {

    return char == ' ' || char == '\t' || char == '\n' || char == '\r';

  }

  bool \_isDigit(String char) {

    return char.codeUnitAt(0) >= 48 && char.codeUnitAt(0) <= 57; // '0'-'9'

  }

  bool \_isLetter(String char) {

    return (char.codeUnitAt(0) >= 65 && char.codeUnitAt(0) <= 90) || // 'A'-'Z'

           (char.codeUnitAt(0) >= 97 && char.codeUnitAt(0) <= 122);  // 'a'-'z'

  }

  bool \_isAlphanumeric(String char) {

    return \_isLetter(char) || \_isDigit(char);

  }

  Token number() {

    int start = position;

    while (!isAtEnd() && \_isDigit(currentChar)) {

      position++;

    }

    if (!isAtEnd() && currentChar == '.') {

      position++;

      while (!isAtEnd() && \_isDigit(currentChar)) {

        position++;

      }

    }

    return Token(TokenType.NUMBER, input.substring(start, position));

  }

  Token identifierOrFunction() {

    int start = position;

    while (!isAtEnd() && (\_isAlphanumeric(currentChar) || currentChar == '\_')) {

      position++;

    }

    String value = input.substring(start, position);

    if (value.length > 32) {

      throw FormatException("Ошибка: Идентификатор '$value' превышает 32 символа.");

    }

    if (value == 'sin' || value == 'cos') {

      return Token(TokenType.FUNCTION, value);

    }

    if (value == 'e' || value == 'pi') {

      return Token(TokenType.CONSTANT, value);

    }

    return Token(TokenType.IDENTIFIER, value);

  }

  Token operator() {

    String op = currentChar;

    position++;

    return Token(TokenType.OPERATOR, op);

  }

  void skipComment() {

    while (!isAtEnd() && currentChar != '\n') {

      position++;

    }

  }

  List<Token> tokenize() {

    while (!isAtEnd()) {

      skipWhitespace();

      if (isAtEnd()) break;

      String char = currentChar;

      if (\_isDigit(char)) {

        tokens.add(number());

      } else if (char == '/' && input[position + 1] == '/' ) {

        skipComment();

      } else if (\_isLetter(char) || char == '\_') {

        tokens.add(identifierOrFunction());

      } else if (char == '+' || char == '-' || char == '\*' || char == '/' || char == '=') {

        tokens.add(operator());

      } else if (char == '(') {

        tokens.add(Token(TokenType.LPAREN, char));

        position++;

      } else if (char == ')') {

        tokens.add(Token(TokenType.RPAREN, char));

        position++;

      } else if (char == ';') {

        tokens.add(Token(TokenType.SEMICOLON, char));

        position++;

      } else {

        throw FormatException('Неожиданный символ: $char на позиции $position');

      }

    }

    return tokens;

  }

}

void analyzeFile(String filename) {

  File file = File(filename);

  String inputText = file.readAsStringSync();

  Lexer lexer = Lexer(inputText);

  List<Token> tokens = lexer.tokenize();

  print('${'Тип токена'.padRight(20)}${'Значение'.padRight(40)}');

  print('-' \* 60);

  for (var token in tokens) {

    print('${token.type.padRight(20)}${token.value.padRight(40)}');

  }

}

void main() {

  String filename = 'input.txt';

  try {

    analyzeFile(filename);

  } catch (e) {

    print('Лексическая ошибка: $e');

  }

}

**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**

**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.