

Министерство науки и высшего образования Российской Федерации
Муромский институт (филиал)
Федерального государственного бюджетного образовательного учреждения
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

Факультет _____ ИТР _____

Кафедра _____ ПИИ _____

ЛАБОРАТОРНАЯ РАБОТА №2

По Теория автоматов и формальных языков

Руководитель

Кульков Я.Ю.

(фамилия, инициалы)

(подпись)

(дата)

Студент _____ ПИИ - 121

(группа)

Ермилов М.В.

(фамилия, инициалы)

(подпись)

(дата)

Муром 2023

Лабораторная работа №2

Цель работы: ознакомление с назначением и принципами работы лексических анализаторов, получение практических навыков построения сканера на примере заданного входного языка.

Ход работы: задача: классификация и сохранение информации об обнаруженной лексеме.

Задание: дополнить лексический анализатор, подготовленный на первой лабораторной работе, выполнив классификацию выделенных лексем:

- Продумать архитектуру системы;
- Продумать организацию структур данных;
- Дополнить программу лексического анализа классификатором лексем в виде токенов;
- Выводить на форму лексемы с указанием типа каждой из них;
- Составить тестовые наборы данных и проверить на них работу программы.

Текст входной последовательности:

Dim a as integer

b = 1

do while (a < 10 or b > c)

b = b + a

loop

					МИ ВлГУ 09.03.04			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.						Лит.	Лист	Листов
Провер.	.						2	6
Реценз.						ПИН-121		
Н. Контр.								
Утверд.	.							

Код по заданию

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;

namespace work
{
    public class generation
    {
        public static string[] Split(string code)
        {
            code = code.Replace('\t', ' ');
            code = code.Replace('\n', ' ');

            string pattern = @"(";
            int i = 0;
            foreach (KeyValuePair<char, TokenType> entry in Token.SpecialSymbols)
            {
                pattern += "\\\"";
                pattern += entry.Key;
                if (Token.SpecialSymbols.Count > i + 1) pattern += "|";
                else pattern += @")";
                i++;
            }
            string[] myArray = code.Split(' ').SelectMany(
                word => Regex.Split(
                    word, pattern
                )
            ).ToArray();
            return myArray;
        }
        public static Token[] Tokens(string[] keys)
        {
            List<Token> tokens = new List<Token>();
            for(int i = 0; i < keys.Length; i++)
                if (!String.IsNullOrEmpty(keys[i]) && keys[i].Length != 0 &&
                    keys[i][0] != ' ')
                    tokens.Add(new Token(keys[i]));
            return tokens.ToArray<Token>();
        }
    }

    public enum TokenType
    {
        INTEGER, DOUBLE, STRING,
        DIM, AS,
        DO, WHILE, LOOP,
        PLUS, MINUS, MULTIPLY, DIVIDE,
        OR, AND,
        DEGREE,
        EQUAL, MORE, LESS,
        COMMA, DOT, COLON, SEMICOLON,
        LPAR, RPAR,
        UNDERSCORE,
        IDENTIFIER, LITERAL,
        TOKEN_ERROR
    }

    public class Token
    {
        static TokenType[] Delimiters = new TokenType[]
        {
            TokenType.PLUS, TokenType.MINUS, TokenType.MULTIPLY,
```

					МИ ВлГУ 09.03.04	Лист
						3
Изм.	Лист	№ докум.	Подпись	Дата		

```

        TokenType.DIVIDE, TokenType.OR, TokenType.AND, TokenType.DEGREE,
        TokenType.EQUAL, TokenType.MORE, TokenType.LESS,
        TokenType.COMMA, TokenType.DOT, TokenType.COLON,
        TokenType.SEMICOLON, TokenType.LPAR, TokenType.RPAR,
        TokenType.UNDERSCORE
    };

    public static Dictionary<string, TokenType> SpecialWords = new
Dictionary<string, TokenType>() {
        {"integer", TokenType.INTEGER},
        {"double", TokenType.DOUBLE},
        {"string", TokenType.STRING},
        {"Dim", TokenType.DIM},
        {"as", TokenType.AS},
        {"or", TokenType.OR},
        {"and", TokenType.AND},
        {"do", TokenType.DO},
        {"while", TokenType.WHILE},
        {"loop", TokenType.LOOP}
    };

    public static Dictionary<char, TokenType> SpecialSymbols = new Dictionary<char,
TokenType>()
    {
        {'+', TokenType.PLUS},
        {'-', TokenType.MINUS},
        {'*', TokenType.MULTIPLY},
        {'/', TokenType.DIVIDE},
        {'|', TokenType.OR},
        {'^', TokenType.DEGREE},
        {'=', TokenType.EQUAL},
        {'>', TokenType.MORE},
        {'<', TokenType.LESS},
        {',', TokenType.COMMA},
        {'.', TokenType.DOT},
        {':', TokenType.COLON},
        {';', TokenType.SEMICOLON},
        {'(', TokenType.LPAR},
        {')', TokenType.RPAR},
        //{'_', TokenType.UNDERSCORE}
    };

    public TokenType Type;
    public string Value;

    public Token(string word)
    {
        Value = word;
        if (int.TryParse(word, out int a)) Type = TokenType.LITERAL;
        else if (word.Length == 1 && IsSpecialSymbol(word[0])) Type =
SpecialSymbols[word[0]];
        else if (IsSpecialWord(word)) Type = SpecialWords[word];
        else if (isIdentifier(word)) Type = TokenType.IDENTIFIER;
        else Type = TokenType.TOKEN_ERROR;
    }

    public static bool isIdentifier(string word)
    {
        if (string.IsNullOrEmpty(word))
            return false;
        if (
            word[0] >= 65 && word[0] <= 90 || //A-Z
            word[0] >= 97 && word[0] <= 122) //a-z
        {
            for (int i = 1; i < word.Length; i++)
            {
                if (!(

```

```

        word[0] >= 65 && word[0] <= 90 || //A-Z
        word[0] >= 97 && word[0] <= 122 || //a-z
        word[0] >= 48 && word[0] <= 57)) //0-9
        return false;
    }
}
else
    return false;
return true;
}
public static bool IsSpecialWord(string word)
{
    if (string.IsNullOrEmpty(word))
        return false;
    return SpecialWords.ContainsKey(word);
}
public static bool IsDelimiter(Token token) => Delimiters.Contains(token.Type);
public static bool IsSpecialSymbol(char ch) => SpecialSymbols.ContainsKey(ch);
public override string ToString()
{
    if (Type == TokenType.TOKEN_ERROR)
        return Value + " - is an identifier with an error";
    return Type + " - " + Value;
}
}
}
using System.Windows.Forms;

namespace work
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void OpenFile(object sender, EventArgs e)
        {
            OpenFileDialog fileDialog = new OpenFileDialog();
            fileDialog.Filter = "Файлы txt (*.txt)|*.txt";
            if (fileDialog.ShowDialog() == DialogResult.OK)
            {
                StreamReader rdr = new StreamReader(fileDialog.FileName);
                string line = rdr.ReadToEnd();
                rdr.Close();
                richTextBox1.Text = line;
                code(line);
            }
        }

        private void Tokens(object sender, EventArgs e)
        {
            code(richTextBox1.Text);
        }
        void code(string code)
        {
            listBox1.Items.Clear();
            Token[] tokens = generation.Tokens(generation.Split(code));
            foreach (Token token in tokens)
            {
                listBox1.Items.Add(token);
            }
        }
    }
}

```

