Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

**МЕХАНІКО-ТЕХНОЛОГІЧНИЙ ФАКУЛЬТЕТ**

Кафедра кібербезпеки та програмного забезпечення

**Звіт**

з виконаної лабораторної роботи № 2

Дисципліна: Системне програмне забезпечення

Виконав :

студент академічної групи КІ-15

Аннаєв А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Перевірив :

Викладач

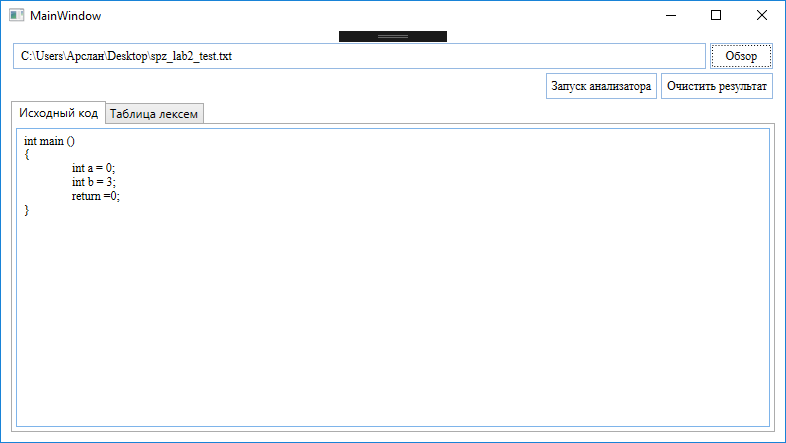
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

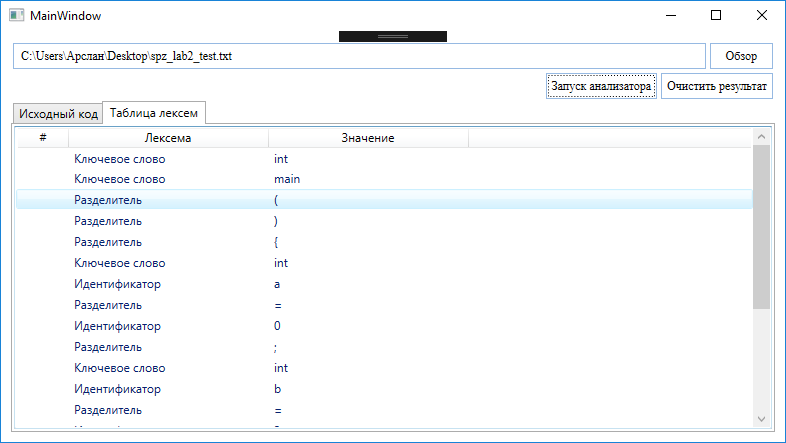
Кропивницкий- 2018

Лабораторная работа № 2

**Тема**: Проектирование лексического анализа.

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





using CompilerAPI.Enum;

namespace CompilerAPI.LexemAnalyzerModel

{

/// <summary>

/// Структурная единица языка

/// </summary>

public class Lexem

{

/// <summary>

/// Группа к которой относится лексема

/// </summary>

public LexemGroupType TokenType { get; set; }

/// <summary>

/// Значение хранимое лексемой

/// </summary>

public string Value { get; set; }

public Lexem(LexemGroupType tokenType)

{

TokenType = tokenType;

Value = string.Empty;

}

public Lexem(LexemGroupType tokenType, string value)

{

TokenType = tokenType;

Value = value;

}

public Lexem Clone()

{

return new Lexem(TokenType, Value);

}

}

}

using CompilerAPI.Enum;

using System.Linq;

using System.Text.RegularExpressions;

namespace CompilerAPI.LexemAnalyzerModel

{

/// <summary>

/// Определение синтаксической группы

/// </summary>

public class LexemGroup

{

public string Value { get; private set; }

private Regex \_regex;

private readonly LexemGroupType \_tokenType;

public LexemGroup(LexemGroupType tokenGroupType, string[] groupItems)

{

\_tokenType = tokenGroupType;

if (groupItems != null)

{

Value += @"";

foreach (string item in groupItems)

{

if(!string.IsNullOrEmpty(item))

{

if (item != groupItems.Last())

Value += item + "|";

else

Value += item;

}

}

\_regex = new Regex(Value, RegexOptions.IgnoreCase);

} else

{

\_regex = new Regex("?");

}

}

public Lexem Match(string inputString)

{

Match match = \_regex.Match(inputString);

if (match.Success)

{ return new Lexem(\_tokenType, match.Value); }

return null;

}

}

}

using CompilerAPI.Enum;

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using System.Text.RegularExpressions;

namespace CompilerAPI.LexemAnalyzerModel

{

public class LexemParser

{

#region Переменные

public const int MAX\_IDENTIER\_LENGTH = 32;

private List<LexemGroup> lexemGroup;

#endregion

#region Конструктор

public LexemParser()

{

lexemGroup = new List<LexemGroup>();

Initialize();

}

#endregion

#region Методы

public ReadOnlyCollection<Lexem> GetLexems(string sourceText)

{

List<Lexem> tokens = null;

if (lexemGroup == null || lexemGroup.Count <= 0)

{ throw new ArgumentException("Отсутствует грамматика исходного языка"); }

string formingPattern = string.Empty;

foreach (LexemGroup item in lexemGroup)

{

if (item != lexemGroup.Last())

formingPattern += item.Value + "|";

else

formingPattern += item.Value;

}

if (string.IsNullOrEmpty(formingPattern))

{ throw new ArgumentException("Не удалось сформировать регулярное

выражение"); }

MatchCollection collection = Regex.Matches(sourceText, formingPattern);

if (collection.Count > 0)

{

tokens = new List<Lexem>();

Lexem context = null;

foreach (Match item in collection)

{

if (string.IsNullOrEmpty(item.Value))

continue;

context = FindContext(item.Value);

if (context != null)

tokens.Add(context);

}

}

else

{

throw new System.Exception("Не корректные входные данные");

}

tokens.Add(new Lexem(LexemGroupType.END\_OF\_SEQUENCE, string.Empty));

return tokens.AsReadOnly();

}

private Lexem FindContext(string lqlText)

{

Lexem lexem = null;

foreach (var tokenDefinition in lexemGroup)

{

lexem = tokenDefinition.Match(lqlText);

if (lexem != null)

{ return lexem; }

}

return null;

}

private void Initialize()

{

string[] separators = new string[]

{

@"\{", @"\}", @"\(", @"\)", @"\[", @"\]", @"\;" , @"\:", @"\`", @"\'",

@"\." , @"\?", @"\+", @"\-", @"\\*", @"\/", @"\^",

@"\~", @"\=", @"\!", @"\>", @"\<"

};

string[] keywords = new string[]

{

"int", "double", "float", "char", "string","struct", "main", "static",

"if", "else", "switch",

"while", "break", "continue", "return"

};

string[] identifiers = new string[]

{

@"\_\_\w{1,32}\s?" , @"\_\w{1,32}\s?", @"\b\w+\s?"

};

lexemGroup.Add(new LexemGroup(LexemGroupType.KEYWORD, keywords));

lexemGroup.Add(new LexemGroup(LexemGroupType.IDENTIFIER, identifiers));

lexemGroup.Add(new LexemGroup(LexemGroupType.SEPARATOR, separators));

}

#endregion

}

}