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 ПИн

*ЛАБОРАТОРНАЯ*

*РАБОТА №6*

# По Теория автоматов и формальных языков

Тема СИНТАКСИЧЕСКИЙ АНАЛИЗ. Программирование

Руководитель

Кульков Я.Ю.

(фамилия, инициалы)

(подпись) (дата)

Студент ПИН - 120

(группа)

Комаров Р.Д.

(фамилия, инициалы)

(подпись) (дата)

Муром 2022

**Лабораторная работа № 6**

**Тема:** СИНТАКСИЧЕСКИЙ АНАЛИЗ. Программирование

**Цели и задачи:** Получить навык в программировании синтаксических анализаторов

**Ход работы:**

**Реализация синтаксического анализа:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows.Forms;

namespace Laba1

{

public class llkGrammatiks

{

public static int number = 0;

public static string GetLexem(ListWithDuplicates keyValuePairs, List<string> keyWords, List<string> separators, List<string> literals, List<string> variables, int i)

{

string lexem = "";

if (keyValuePairs.ElementAt(i).Key == "1")

{

lexem = keyWords.ElementAt(Convert.ToInt32(keyValuePairs.ElementAt(i).Value));

}

else if (keyValuePairs.ElementAt(i).Key == "2")

{

lexem = separators.ElementAt(Convert.ToInt32(keyValuePairs.ElementAt(i).Value));

}

else if (keyValuePairs.ElementAt(i).Key == "3")

{

lexem = literals.ElementAt(Convert.ToInt32(keyValuePairs.ElementAt(i).Value));

}

else if (keyValuePairs.ElementAt(i).Key == "4")

{

lexem = variables.ElementAt(Convert.ToInt32(keyValuePairs.ElementAt(i).Value));

}

return lexem;

}

public static string lexeme = "";

public static void Next(ListWithDuplicates keyValuePairs, List<string> keyWords, List<string> separators, List<string> literals, List<string> variables)

{

number++;

if(number < keyValuePairs.Count)

{

lexeme = GetLexem(keyValuePairs, keyWords, separators, literals, variables, number);

}

}

public static void Back()

{

number--;

}

public static bool LoopOperator(ListWithDuplicates keyValuePairs, int k, List<string> keyWords, List<string> separators, List<string> literals, List<string> variables)

{

if (lexeme != "for")

{

Errors.errorFor();

return false;

}

Next(keyValuePairs, keyWords, separators, literals, variables);

if (!variables.Contains(lexeme))

{

Errors.errorVariables();

return false;

}

Next(keyValuePairs, keyWords, separators, literals, variables);

if (lexeme != "in")

{

Errors.errorIn();

return false;

}

Next(keyValuePairs, keyWords, separators, literals, variables);

if (lexeme != "[")

{

Errors.errorOpenScob();

return false;

}

Next(keyValuePairs, keyWords, separators, literals, variables);

if(!listOfVariables(keyValuePairs, number, keyWords, separators, literals, variables))

return false;

if (!ListOfAction(keyValuePairs, number, keyWords, separators, literals, variables))

return false;

if (lexeme != "end")

{

Errors.errorEnd();

return false;

}

if (true)

{

return true;

}

else

{

return false;

}

}

public static bool assignment(ListWithDuplicates keyValuePairs, int k, List<string> keyWords, List<string> separators, List<string> literals, List<string> variables)

{

if(number < keyValuePairs.Count)

{

if (!variables.Contains(lexeme))

{

Errors.errorVariables();

return false;

}

Next(keyValuePairs, keyWords, separators, literals, variables);

if (number < keyValuePairs.Count)

{

if (lexeme != "=")

{

Errors.errorRavno();

return false ;

}

Next(keyValuePairs, keyWords, separators, literals, variables);

expr(keyValuePairs, number, keyWords, separators, literals, variables);

if(lexeme != ";")

{

Errors.errorPunct();

return false;

}

}

else

{

Errors.errorSintax();

return false;

}

}

else

{

Errors.errorSintax();

return false;

}

if (true)

{

return true;

}

else

{

return false;

}

}

public static void expr(ListWithDuplicates keyValuePairs, int k, List<string> keyWords, List<string> separators, List<string> literals, List<string> variables)

{

if (number < keyValuePairs.Count)

{

if (variables.Contains(lexeme) || literals.Contains(lexeme))

{

Next(keyValuePairs, keyWords, separators, literals, variables);

if (number < keyValuePairs.Count)

{

if (lexeme == "+" || lexeme == "-" || lexeme == "/" || lexeme == "\*")

{

Next(keyValuePairs, keyWords, separators, literals, variables);

expr(keyValuePairs, number, keyWords, separators, literals, variables);

}

else if (lexeme == ";")

{

}

}

else

{

return;

}

}

}

else

{

Errors.errorSintax();

return;

}

}

public static bool action(ListWithDuplicates keyValuePairs, int k, List<string> keyWords, List<string> separators, List<string> literals, List<string> variables)

{

if (lexeme == "for")

{

if(!LoopOperator(keyValuePairs, number, keyWords, separators, literals, variables))

{

return false;

}

}

else if (variables.Contains(lexeme))

{

if(!assignment(keyValuePairs, number, keyWords, separators, literals, variables))

{

return false;

}

}

else

{

Errors.errorForAndVariables();

return false;

}

if (true)

{

return true;

}

else

{

return false;

}

}

public static bool operand(ListWithDuplicates keyValuePairs, int k, List<string> keyWords, List<string> separators, List<string> literals, List<string> variables)

{

if (!variables.Contains(lexeme) && !literals.Contains(lexeme))

{

Errors.errorVariablesAndLiterals();

return false;

}

Next(keyValuePairs, keyWords, separators, literals, variables);

if (true)

{

return true;

}

else

{

return false;

}

}

public static bool fixingReferences(ListWithDuplicates keyValuePairs, int k, List<string> keyWords, List<string> separators, List<string> literals, List<string> variables)

{

if (lexeme != ",")

{

Errors.errorZap();

return false;

}

Next(keyValuePairs, keyWords, separators, literals, variables);

if(!operand(keyValuePairs, number, keyWords, separators, literals, variables))

{

return false;

}

if (!UstrLevFactorSpisPermen(keyValuePairs, number, keyWords, separators, literals, variables))

{

return false;

}

if (true)

{

return true;

}

else

{

return false;

}

}

public static bool listOfVariables(ListWithDuplicates keyValuePairs, int k, List<string> keyWords, List<string> separators, List<string> literals, List<string> variables)

{

if (!variables.Contains(lexeme) && !literals.Contains(lexeme))

{

Errors.errorVariablesAndLiterals();

return false;

}

if (!operand(keyValuePairs, number, keyWords, separators,literals, variables))

return false;

if (!UstrLevFactorSpisPermen(keyValuePairs, number, keyWords, separators, literals, variables))

return false;

if (true)

{

return true;

}

else

{

return false;

}

}

public static bool UstrLevFactorSpisPermen(ListWithDuplicates keyValuePairs, int k, List<string> keyWords, List<string> separators, List<string> literals, List<string> variables)

{

if (lexeme == "]" || lexeme == ",")

{

if (lexeme == ",")

{

if (!fixingReferences(keyValuePairs, number, keyWords, separators, literals, variables))

return false;

}

else if (lexeme == "]")

{

Next(keyValuePairs, keyWords, separators, literals, variables);

}

else

{

Errors.errorZapOROutputScob();

return false;

}

}

else

{

Errors.errorOutputScob();

return false;

}

if (true)

{

return true;

}

else

{

return false;

}

}

public static bool UstrLevFactorSpisAction(ListWithDuplicates keyValuePairs, int k, List<string> keyWords, List<string> separators, List<string> literals, List<string> variables)

{

if (lexeme == "end")

{

Next(keyValuePairs, keyWords, separators, literals, variables);

if (number > keyValuePairs.Count - 1)

{

return true;

}

else if(lexeme == "for" || variables.Contains(lexeme))

{

if (!UstrRecDeistv(keyValuePairs, number, keyWords, separators, literals, variables))

return false;

}

}

else if (lexeme == ";")

{

Next(keyValuePairs, keyWords, separators, literals, variables);

if(lexeme == "for" || variables.Contains(lexeme))

{

if (!UstrRecDeistv(keyValuePairs, number, keyWords, separators, literals, variables))

return false;

}

else if(number > keyValuePairs.Count - 1)

{

return true;

}

else if (lexeme == ";")

{

Errors.errorMinZap();

return false;

}

}

if (true)

{

return true;

}

else

{

return false;

}

}

public static bool UstrRecDeistv(ListWithDuplicates keyValuePairs, int k, List<string> keyWords, List<string> separators, List<string> literals, List<string> variables)

{

if (!action(keyValuePairs, number, keyWords, separators, literals, variables))

return false;

if(number < keyValuePairs.Count)

{

if (!UstrLevFactorSpisAction(keyValuePairs, number, keyWords, separators, literals, variables))

return false;

else

return true;

}

else

{

Errors.errorZap();

return false;

}

}

public static bool ListOfAction(ListWithDuplicates keyValuePairs, int k, List<string> keyWords, List<string> separators, List<string> literals, List<string> variables)

{

if (number < keyValuePairs.Count)

{

if (lexeme != "for" && !variables.Contains(lexeme))

{

Errors.errorForAndVariables();

return false;

}

if (!action(keyValuePairs, number, keyWords, separators, literals, variables))

{

return false;

}

if (!UstrLevFactorSpisAction(keyValuePairs, number, keyWords, separators, literals, variables))

{

return false;

}

}

else

{

Errors.errorPunct();

return false;

}

if (true)

return true;

else

return false;

}

public static void Program(ListWithDuplicates keyValuePairs, List<string> keyWords, List<string> separators, List<string> literals, List<string> variables)

{

lexeme = GetLexem(keyValuePairs, keyWords, separators, literals, variables, number);

if (lexeme != "for" && !variables.Contains(lexeme))

{

Errors.errorForAndVariables();

return;

}

if (!ListOfAction(keyValuePairs, number, keyWords, separators, literals, variables))

{

return;

}

else

{

Complete(keyValuePairs);

}

keyValuePairs.Clear();

}

public static void Complete(ListWithDuplicates keyValuePairs)

{

if(number == keyValuePairs.Count)

{

MessageBox.Show("Синтаксический анализ выполнен успешно!!!", "Лексический анализ", MessageBoxButtons.OKCancel, MessageBoxIcon.Information);

return;

}

if (number > keyValuePairs.Count)

{

Errors.errorEnd();

return;

}

if(number < keyValuePairs.Count)

{

Errors.errorMinusLexem();

return;

}

}

}

}

Работа программы:

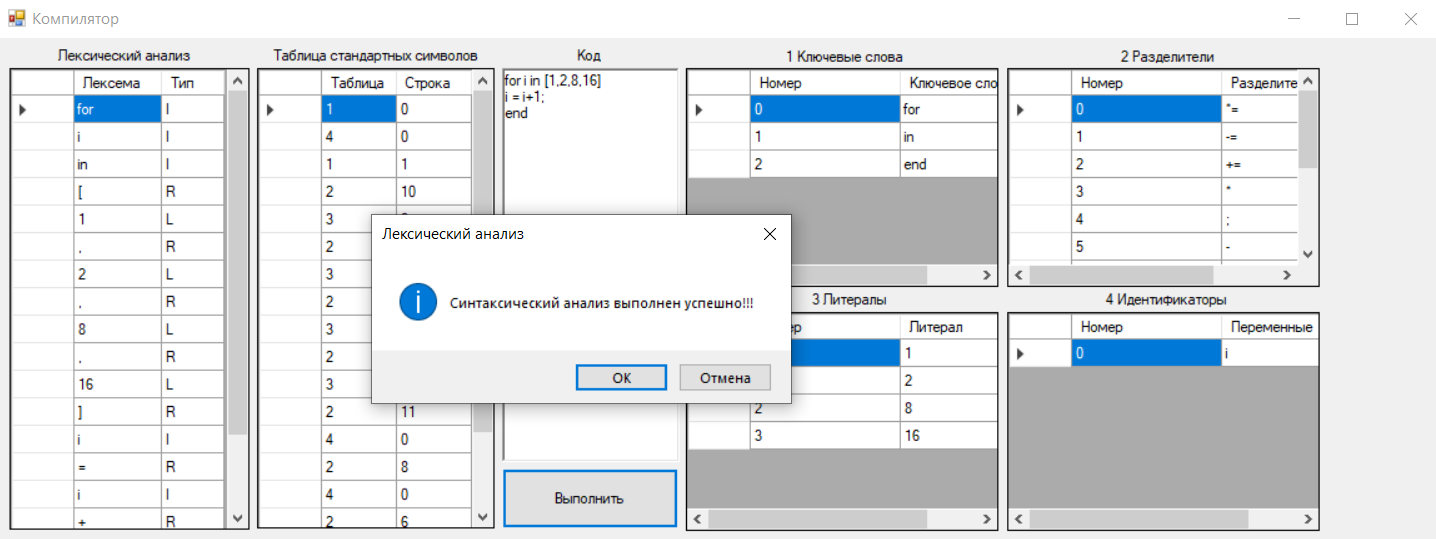


Рисунок 1 – работа программы

Вывод: в ходе лабораторной работы, были получены навыки в программировании синтаксических анализаторов