Министерство образования Российской Федерации

Федеральное агентство по образованию

КАЗАНСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. А.Н. Туполева - КАИ

Кафедра АСОИУ

Лабораторные работы №1-8

по дисциплине

«ТЕОРИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ» и «Методы трансляции»

Выполнил:

Студент группы 4309

Галиуллин А.Л.

Проверила: Бикмуллина И. И.

Казань 2023

**Лабораторная работа № 1. Разработка транслитератора**

**Текст задания:**

1. Спроектировать и отладить экранную форму для ввода исходных данных, вывода сообщений программы и управления программой.
2. Разработать и отладить транслитератор **void GetSymbol()**, пример имеется в модуле **uLexicalAnalizer** из папки «Программы».
3. Для отладки транслитератора временно включить в обработчик нажатия кнопки цикл чтения с помощью функции **GetSymbol()** символов исходного текста и вывода результатов анализа в поле диагностических сообщений.

Form1.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp45

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

tbFSource.AppendText("01ab" + "\r\n");

tbFSource.AppendText("1 a");

int n = tbFSource.Lines.Length;

}

private void btnFStart\_Click(object sender, EventArgs e)

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFSource.Lines;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.GetSymbol(); // Выводятся литеры и классификация

Lex.NextToken();

String s = "";

String s1 = "";

switch (Lex.enumFSelectionCharType)

{

case TCharType.Letter: { s1 = "Letter"; break; }

case TCharType.Digit: { s1 = "Digit"; break; }

case TCharType.Space: { s1 = "Space"; break; }

case TCharType.ReservedSymbol: { s1 = "Reserved"; break; }

case TCharType.EndRow: { s = "KC"; s1 = "EndRow"; break; }

case TCharType.EndText: { s = "KT"; s1 = "EndText"; break; }

}

String m = $"({s},{s1})"; //литера и ее тип

tbFMessage.Text += m; //добавляется в строку сообщение

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFMessage.Select();

tbFMessage.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Lex.intPSourceRowSelection; i++) n += tbFMessage.Lines[i].Length + 2;

n += Lex.intPSourceColSelection;

tbFMessage.SelectionLength = n;

}

}

}

}

**uLex**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp45

{

public enum TState { Start, Continue, Finish }; //тип состояния

public enum TCharType { Letter, Digit, EndRow, EndText, Space, ReservedSymbol }; // тип символа

public enum TToken { lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma };

public class CLex //класс лексический анализатор

{

private String[] strFSource; // указатель на массив строк

private String[] strFMessage; // указатель на массив строк

public TCharType enumFSelectionCharType;

public char chrFSelection;

private TState enumFState;

private int intFSourceRowSelection;

private int intFSourceColSelection;

private String strFLexicalUnit;

private TToken enumFToken;

public String[] strPSource { set { strFSource = value; } get { return strFSource; } }

public String[] strPMessage { set { strFMessage = value; } get { return strFMessage; } }

public TState enumPState { set { enumFState = value; } get { return enumFState; } }

public String strPLexicalUnit { set { strFLexicalUnit = value; } get { return strFLexicalUnit; } }

public TToken enumPToken { set { enumFToken = value; } get { return enumFToken; } }

public int intPSourceRowSelection { get { return intFSourceRowSelection; } set { intFSourceRowSelection = value; } }

public int intPSourceColSelection { get { return intFSourceColSelection; } set { intFSourceColSelection = value; } }

public CLex()

{

}

public void GetSymbol() //метод класса лексический анализатор

{

if (intFSourceColSelection > strFSource[intFSourceRowSelection].Length - 1)

{

intFSourceRowSelection++;

if (intFSourceRowSelection <= strFSource.Length - 1)

{

intFSourceColSelection = -1;

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndRow;

enumFState = TState.Continue;

}

else

{

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndText;

enumFState = TState.Finish;

}

}

else

{

char[] reservedSymbols = new char[] { '(', ')', '[', ']', '.', ',', '!', '?', '$', '&', '-', '+' };

chrFSelection = strFSource[intFSourceRowSelection][intFSourceColSelection]; //классификация прочитанной литеры

if (chrFSelection == ' ') enumFSelectionCharType = TCharType.Space;

else if (chrFSelection >= 'a' && chrFSelection <= 'd') enumFSelectionCharType = TCharType.Letter;

else if (chrFSelection == '0' || chrFSelection == '1') enumFSelectionCharType = TCharType.Digit;

else if (chrFSelection == '/') enumFSelectionCharType = TCharType.ReservedSymbol;

else if (chrFSelection == '\*') enumFSelectionCharType = TCharType.ReservedSymbol;

else if (reservedSymbols.Contains(chrFSelection)) enumFSelectionCharType = TCharType.ReservedSymbol;

else throw new System.Exception("Cимвол вне алфавита");

enumFState = TState.Continue;

}

intFSourceColSelection++; // продвигаем номер колонки

}

private void TakeSymbol()

{

char[] c = { chrFSelection };

String s = new string(c);

strFLexicalUnit += s;

GetSymbol();

}

public void NextToken()

{

strFLexicalUnit = "";

if (enumFState == TState.Start)

{

intFSourceColSelection = -1;

GetSymbol();

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

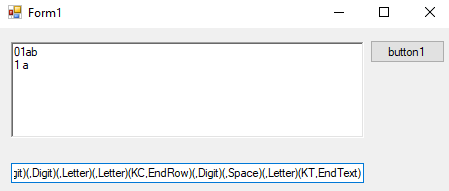
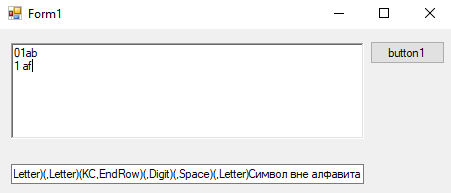
GetSymbol();

}

}

}

}

**Лабораторная работа № 2. Разработка лексического анализатора**

**Текст задания:**

1. Спроектировать и отладить экранную форму для ввода исходных данных, вывода сообщений программы и управления программой.
2. Включить из лабораторной работы № 1 транслитератор **void GetSymbol().**
3. Составить регулярную грамматику для каждого вида слов.
4. Построить конечные автоматы для каждого вида слов, как правило, они будут недетерминированными.
5. Построить детерминированные конечные автоматы для каждого вида слов.
6. Составить объединенный конечный автомат.
7. Написать и отладить модуль лексического анализатора по алгоритму объединенного конечного автомата. Для чтения исходного текста использовать транслитератор. Предусмотреть обработчик лексических ошибок исходного текста, используется конструкция **try … catch**.
8. Для отладки лексического анализатора временно включить в обработчик нажатия кнопки цикл чтения слов исходного текста и вывода результатов лексического анализа.

|  |  |  |
| --- | --- | --- |
| (000)\*100(010)\* | (a|b|c|d)+ | Первые два символа всегда ac |

**Первое слово:**

1

(000)\*100(010)\*

0

A → 0B | 1D

0

0

B → 0C

C → 0A

0

D → 0E

0

E → FFin

FFin → 0 | 0G

0

0

G → 1H

H → 0FFin

**Недетерминированная матрица:**

10

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| A | B | D |
| B | C |  |
| C | A |  |
| D | B |  |
| E | F,Fin |  |
| F | G |  |
| G |  | H |
| H | F,Fin |  |
| Fin |  |  |

**Граф:**

**Детерминированная матрица:**

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| A | B | D |
| B | C |  |
| C | A |  |
| D | E |  |
| E | Fin |  |
| Fin | G |  |
| G |  | H |
| H | Fin |  |

**Второе слово:**

(a|b|c|d)+

Вторые 2 символа всегда ab

A → aB

B → c | cC

D → a | b | c | d | aFin | bFin | cFin | dFin

**Недетерминированная матрица:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | a | b | c | d |
| A | B |  |  |  |
| B |  |  | Fin |  |
| Fin | Fin | Fin | Fin | Fin |

**Детерминированная матрица:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | a | b | c | d |
| A | B |  |  |  |
| B |  |  | CFin |  |
| CFin | CFin | CFin | CFin | CFin |

abcd

c

a

Код программы:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp45

{

public enum TState { Start, Continue, Finish }; //тип состояния

public enum TCharType { Letter, Digit, EndRow, EndText, Space, ReservedSymbol }; // тип символа

public enum TToken { lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma, lxmDollar, lxmMinus, lxmPlus, lxmExclamation, lxmQuestion,

lxmAnd, lxmLeftParenthSqr, lxmRightParenthSqr };

public class CLex //класс лексический анализатор

{

private String[] strFSource; // указатель на массив строк

private String[] strFMessage; // указатель на массив строк

public TCharType enumFSelectionCharType;

public char chrFSelection;

private TState enumFState;

private int intFSourceRowSelection;

private int intFSourceColSelection;

private String strFLexicalUnit;

private TToken enumFToken;

public String[] strPSource { set { strFSource = value; } get { return strFSource; } }

public String[] strPMessage { set { strFMessage = value; } get { return strFMessage; } }

public TState enumPState { set { enumFState = value; } get { return enumFState; } }

public String strPLexicalUnit { set { strFLexicalUnit = value; } get { return strFLexicalUnit; } }

public TToken enumPToken { set { enumFToken = value; } get { return enumFToken; } }

public int intPSourceRowSelection { get { return intFSourceRowSelection; } set { intFSourceRowSelection = value; } }

public int intPSourceColSelection { get { return intFSourceColSelection; } set { intFSourceColSelection = value; } }

public CLex()

{

}

public void GetSymbol() //метод класса лексический анализатор

{

if (intFSourceColSelection > strFSource[intFSourceRowSelection].Length - 1)

{

intFSourceRowSelection++;

if (intFSourceRowSelection <= strFSource.Length - 1)

{

intFSourceColSelection = -1;

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndRow;

enumFState = TState.Continue;

}

else

{

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndText;

enumFState = TState.Finish;

}

}

else

{

char[] reservedSymbols = new char[] { '(', ')', '[', ']', '.', ',', '!', '?', '$', '&', '-', '+' };

chrFSelection = strFSource[intFSourceRowSelection][intFSourceColSelection]; //классификация прочитанной литеры

if (chrFSelection == ' ') enumFSelectionCharType = TCharType.Space;

else if (chrFSelection >= 'a' && chrFSelection <= 'd') enumFSelectionCharType = TCharType.Letter;

else if (chrFSelection == '0' || chrFSelection == '1') enumFSelectionCharType = TCharType.Digit;

else if (chrFSelection == '/') enumFSelectionCharType = TCharType.ReservedSymbol;

else if (chrFSelection == '\*') enumFSelectionCharType = TCharType.ReservedSymbol;

else if (reservedSymbols.Contains(chrFSelection)) enumFSelectionCharType = TCharType.ReservedSymbol;

else throw new System.Exception("Cимвол вне алфавита");

enumFState = TState.Continue;

}

intFSourceColSelection++; // продвигаем номер колонки

}

private void TakeSymbol()

{

char[] c = { chrFSelection };

String s = new string(c);

strPLexicalUnit += s;

GetSymbol();

}

public void NextToken()

{

strPLexicalUnit = "";

if (enumFState == TState.Start)

{

intFSourceRowSelection = 0;

intFSourceColSelection = -1;

GetSymbol();

}

while (enumFSelectionCharType == TCharType.Space || enumFSelectionCharType == TCharType.EndRow)

{

GetSymbol();

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// A | B | | | |

// B | | | CFin | |

// CFin | CFin | CFin | CFin | CFin |

A:

{

if (chrFSelection == 'a')

{

TakeSymbol();

goto B;

}

else throw new Exception("Слово должно начинаться с 'ac'");

}

B:

{

if (chrFSelection == 'c')

{

TakeSymbol();

goto CFin;

}

else throw new Exception("Слово должно начинаться с 'ac'");

}

CFin:

{

if (chrFSelection == 'a' || chrFSelection == 'b' || chrFSelection == 'c' || chrFSelection == 'd')

{

TakeSymbol();

goto CFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

case TCharType.Digit:

{

// 0 1

// A | B | D |

// B | C | |

// C | A | |

// D | E | |

// E |FFin | |

// FFin | G | |

// G | | H |

// H |FFin | |

A:

{

if (chrFSelection == '0')

{

TakeSymbol();

goto B;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

else throw new Exception("Ожидался 0 или 1");

}

B:

{

if (chrFSelection == '0')

{

TakeSymbol();

goto C;

}

else throw new Exception("Ожидался 0");

}

C:

{

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидался 0");

}

D:

{

if (chrFSelection == '0')

{

TakeSymbol();

goto E;

}

else throw new Exception("Ожидался 0");

}

E:

{

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0");

}

FFin:

{

if (chrFSelection == '0')

{

TakeSymbol();

goto G;

}

else if (enumFSelectionCharType != TCharType.Digit)

{

enumFToken = TToken.lxmNumber;

return;

}

else throw new Exception("Ожидался 0");

}

G:

{

if (chrFSelection == '1')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидалась 1");

}

H:

{

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0");

}

}

case TCharType.ReservedSymbol:

{

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

{

while (enumFSelectionCharType != TCharType.EndRow)

GetSymbol();

}

GetSymbol();

}

if (chrFSelection == '(')

{

enumFToken = TToken.lxmLeftParenth;

GetSymbol();

return;

}

if (chrFSelection == ')')

{

enumFToken = TToken.lxmRightParenth;

GetSymbol();

return;

}

if (chrFSelection == '.')

{

enumFToken = TToken.lxmDot;

GetSymbol();

return;

}

if (chrFSelection == ',')

{

enumFToken = TToken.lxmComma;

GetSymbol();

return;

}

if (chrFSelection == '[')

{

enumFToken = TToken.lxmLeftParenthSqr;

GetSymbol();

return;

}

if (chrFSelection == ']')

{

enumFToken = TToken.lxmRightParenthSqr;

GetSymbol();

return;

}

if (chrFSelection == '!')

{

enumFToken = TToken.lxmExclamation;

GetSymbol();

return;

}

if (chrFSelection == '?')

{

enumFToken = TToken.lxmQuestion;

GetSymbol();

return;

}

if (chrFSelection == '&')

{

enumFToken = TToken.lxmAnd;

GetSymbol();

return;

}

if (chrFSelection == '$')

{

enumFToken = TToken.lxmDollar;

GetSymbol();

return;

}

if (chrFSelection == '+')

{

enumFToken = TToken.lxmPlus;

GetSymbol();

return;

}

if (chrFSelection == '-')

{

enumFToken = TToken.lxmMinus;

GetSymbol();

return;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

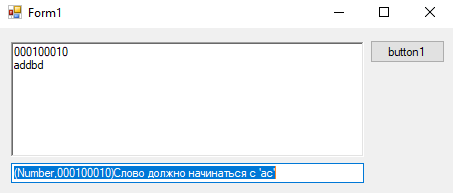
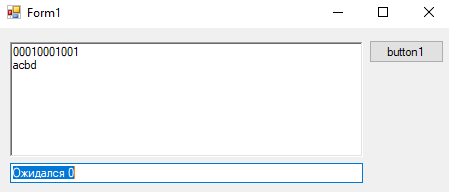
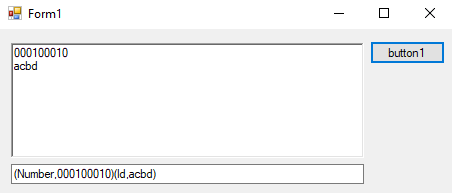
}

}

}

}

}

****

**Лабораторная работа № 3. Разработка транслитератора**

**Текст задания:**

Для предложенного преподавателем варианта КС-грамматики разработать методом рекурсивного спуска синтаксический анализатор.

Примечание: здесь и далее через <1> и <2> обозначены слова из лабораторной работы №1.

Указания:

1. Лексический анализатор из лабораторной работы №1 должен быть расширен обработкой появившихся в КС-грамматике новых слов и включен в виде подпрограммы, поля класса или метода класса в синтаксический анализатор.

2. Оформить синтаксический анализатор в виде процедуры, функции или класса, которые при обращении обрабатывают весь исходный текст.

3. Если грамматика леворекурсивная, то устранить левую рекурсию.

4. При обнаружении лексической ошибки целесообразно возбуждать исключительную ситуацию, которая будет обрабатываться в главной форме программы.

**5 Вариант**

**S** 🡪 **( A )**

**A** 🡪 **( <2> B )**

**A** 🡪 **<1>**

**B** 🡪 **( C )**

**C** 🡪 **C , <1>**

**C** 🡪 **<1>**

Избавление от левой рекурсии:

**S** 🡪 **( A )**

**A** 🡪 **<1> | ( <2> B )**

**B** 🡪 **( C )**

**C** 🡪 **<1> | <1> D**

**D** 🡪 **, <1> | , <1> D**

1: S 🡪 (A) 🡪 (<1>)

2: S 🡪 (A) 🡪 ((<2> B)) 🡪 ((<2> (C))) 🡪 ((<2> (<1>)))

3: S 🡪 (A) 🡪 ((<2> B)) 🡪 ((<2> (C))) 🡪 ((<2> (<1> D))) 🡪 ((<2> (<1>, <1>)))

**Код программы**

**uSyntAnalyzer.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp45

{

public class uSyntAnalyzer

{

private String[] strFSource;

private String[] strFMessage;

public String[] strPSource { set { strFSource = value; } get { return strFSource; } }

public String[] strPMessage { set { strFMessage = value; } get { return strFMessage; } }

public CLex Lex = new CLex();

public TToken firstToken;

public void S()

{

if (Lex.enumPToken == TToken.lxmLeftParenth) //(A)

{

A();

if (Lex.enumPToken == TToken.lxmRightParenth)

{

throw new Exception("Конец слова, текст верный.");

}

else throw new Exception("Ожидалось )");

}

else throw new Exception("Ожидалось (");

}

public void A()

{

if (Lex.enumPToken == TToken.lxmLeftParenth) //(<2>B)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier || Lex.enumPToken == TToken.lxmNumber)

{

firstToken = (TToken)(1 - (int)Lex.enumPToken);

B();

if (Lex.enumPToken == TToken.lxmRightParenth)

{

Lex.NextToken();

}

else throw new Exception("Ожидалось )");

}

else throw new Exception("Ожидался идентификатор или число");

}

else //<1>

{

if(Lex.enumPToken == TToken.lxmIdentifier || Lex.enumPToken == TToken.lxmNumber)

{

firstToken = Lex.enumPToken;

}

else throw new Exception("Ожидался идентификатор или число или (");

}

}

public void B()

{

if (Lex.enumPToken == TToken.lxmLeftParenth) //(C)

{

Lex.NextToken();

C();

if (Lex.enumPToken == TToken.lxmRightParenth)

{

Lex.NextToken();

}

else throw new Exception("Ожидалось )");

}

else throw new Exception("Ожидалось (");

}

public void C()

{

if(Lex.enumPToken == firstToken)

{

Lex.NextToken();

D();

}

else throw new Exception("Ожидался идентификатор или число");

}

public void D()

{

if (Lex.enumPToken == TToken.lxmComma)

{

Lex.NextToken();

if (Lex.enumPToken == firstToken)

{

D();

}

}

}

}

}

**Form1.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp45

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void btnFStart\_Click(object sender, EventArgs e)

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFSource.Lines;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.S();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Synt.Lex.intPSourceRowSelection; i++) n += tbFSource.Lines[i].Length + 2;

n += Synt.Lex.intPSourceColSelection;

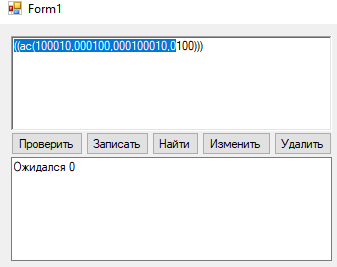
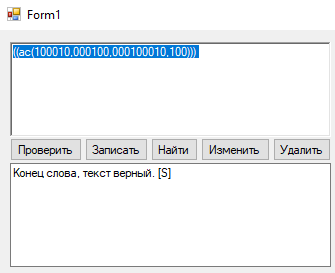
tbFSource.SelectionLength = n;

}

}

}

}

****

**Лабораторная работа № 4. Введение табличного способа хранения слов**

**Задание:**

1. Подключить класс «Массив хеш-таблиц» к программе.
2. Завести три таблицы для хранения слов первого типа, слов второго типа и служебных слов (многосимвольных)
3. Отладить программу до рабочего состояния.

**Код программы:**

**Form1.cs:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp45

{

public partial class Form1 : Form

{

public CHashTableList htl = new CHashTableList(3);

public Form1()

{

InitializeComponent();

tbFSource.AppendText("((ac(100010,000100,000100010,100)))" + "\r\n");

//tbFSource.AppendText("((100(ac, acb, ac, acddc, ac)))" + "\r\n");

int n = tbFSource.Lines.Length;

}

public void TablesToMemo(object sender, System.EventArgs e)

{

List<string> listTable = new List<string>();

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

htl.TableToStringList(0, listTable);

for (int i = 0; i < listTable.Count; i++)

listBox1.Items.Add(listTable[i]);

listTable.Clear();

htl.TableToStringList(1, listTable);

for (int i = 0; i < listTable.Count; i++)

listBox2.Items.Add(listTable[i]);

listTable.Clear();

htl.TableToStringList(2, listTable);

for (int i = 0; i < listTable.Count; i++)

listBox3.Items.Add(listTable[i]);

listTable.Clear();

}

private void btnFStart\_Click(object sender, EventArgs e)

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.Lines;

Synt.Lex.intPSourceColSelection = -1;

Synt.Lex.intPSourceRowSelection = 0;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.S();

//throw new Exception("Текст верный");

}

catch (Exception exc)

{

if (exc.Message == "" || exc.Message == null)

{

tbFMessage.Text += "Неизвестная ошибка";

}

else

{

tbFMessage.Text += exc.Message;

}

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Synt.Lex.intPSourceRowSelection; i++)

{

n += tbFSource.Lines[i].Length + 2;

}

n += Synt.Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

private void btnFRecord\_Click(object sender, EventArgs e)

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

s1 = "id " + Lex.strPLexicalUnit; int b = 0;

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 0, ref b))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

s1 = "num " + Lex.strPLexicalUnit; int b = 0;

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 1, ref b))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmRightParenth:

{

Reserved(ref s1, Lex, e);

break;

}

case TToken.lxmLeftParenth:

{

Reserved(ref s1, Lex, e);

break;

}

case TToken.lxmComma:

{

Reserved(ref s1, Lex, e);

break;

}

}

String m = "(" + s + "" + s1 + ")";

tbFMessage.Text += m;

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Lex.intPSourceRowSelection; i++) n += tbFSource.Lines[i].Length + 2;

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

void Reserved(ref string s1, CLex Lex, EventArgs e)

{

s1 = "res '" + Lex.strPLexicalUnit + "'";

int b = 0;

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b))

{

TablesToMemo(this, e);

}

}

private void btnDelete\_Click(object sender, EventArgs e)

{

if (listBox2.SelectedIndex >= 0)

{

if (htl.DeleteLexicalUnit(listBox2.Items[listBox2.SelectedIndex].ToString(), 1))

{

listBox2.Items.RemoveAt(listBox2.SelectedIndex);

listBox2.SelectedIndex = -1;

}

}

}

private void findBtn\_Click(object sender, EventArgs e)

{

if (listBox2.SelectedIndex >= 0)

{

int d = 0;

if (htl.SearchLexicalUnit(listBox2.Items[listBox2.SelectedIndex].ToString(), 1, ref d))

{

tbFMessage.Text = d.ToString();

}

else

{

tbFMessage.Text = "Не удалось найти индекс элемента";

}

}

}

private void ChngBtn\_Click(object sender, EventArgs e)

{

if (listBox2.SelectedIndex >= 0)

{

htl.ChangeLexicalUnit(listBox2.Items[listBox2.SelectedIndex].ToString(), 1, tbFMessage.Text);

List<string> listTable = new List<string>();

listBox2.Items.Clear();

htl.TableToStringList(1, listTable);

for (int i = 0; i < listTable.Count; i++)

listBox2.Items.Add(listTable[i]);

listTable.Clear();

}

}

}

}

**CHashTableList.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows.Forms;

using System.IO;

namespace WindowsFormsApp45

{

public class CHashTableList

{

private List<THashTable> arrFHashTableList = new List<THashTable>();

private byte byteFTablesSize;

public CHashTableList(byte byteATableCount)

{

this.byteFTablesSize = byteATableCount;

for (int i = 0; i < byteATableCount; i++)

{

arrFHashTableList.Add(new THashTable());

}

}

public bool SearchLexicalUnit(string strALexicalUnit, byte byteATable, ref int intALexicalCode)

{

return arrFHashTableList[byteATable].SearchLexicalUnit(strALexicalUnit, ref intALexicalCode);

}

public bool AddLexicalUnit(string strALexicalUnit, byte byteATable, ref int intALexicalCode)

{

return arrFHashTableList[byteATable].AddLexicalUnit(strALexicalUnit, ref intALexicalCode);

}

public bool ChangeLexicalUnit(string strALexicalUnit, byte byteATable, string newLexUnit)

{

int d = 0;

arrFHashTableList[byteATable].DeleteLexicalUnit(strALexicalUnit);

return arrFHashTableList[byteATable].AddLexicalUnit(newLexUnit, ref d);

}

public bool DeleteLexicalUnit(string strALexicalUnit, byte byteATable)

{

return arrFHashTableList[byteATable].DeleteLexicalUnit(strALexicalUnit);

}

public void TableToStringList(byte byteATable, List<string> sList)

{

arrFHashTableList[byteATable].GetLexicalUnitList(ref sList);

}

public int GetHashIndex(byte Table)

{

return arrFHashTableList[Table].intFHashIndex;

}

}

}

**THashTable.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Windows.Forms;

namespace WindowsFormsApp45

{

public class TableItem

{

public int value;

public string lex;

public TableItem next;

public TableItem(int v, string s, TableItem t)

{

value = v;

lex = s;

next = t;

}

}

public class THashTable

{

public List<TableItem> arrFHashTable = new List<TableItem>();

public int intFHashIndex;

static int tableSize = 20;

public THashTable()

{

Init(tableSize);

}

public void Init(int count)

{

arrFHashTable.Clear();

Resize(arrFHashTable, count);

}

static void Resize(List<TableItem> list, int size)

{

if (size > list.Count)

while (size > list.Count)

list.Add(new TableItem(0, "", null));

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

int HashFunction(string strALexicalUnit)

{

int h = 0;

for (int i = 0, l = strALexicalUnit.Length; i < l; i++)

{

h += strALexicalUnit[i];

}

return h % tableSize;

}

public void HashIndex(string strALexicalUnit)

{

int h = HashFunction(strALexicalUnit);

intFHashIndex = h;

}

public bool SearchLexicalUnit(string strAlexicalUnit, ref int intALexicalCode)

{

HashIndex(strAlexicalUnit);

if (arrFHashTable[intFHashIndex].value == 0) return false;

else

{

intALexicalCode = arrFHashTable[intFHashIndex].value;

return true;

}

}

public bool ChangeLexicalUnit(string strPrevUnit, string strNewUnit)

{

HashIndex(strPrevUnit);

TableItem item = arrFHashTable[intFHashIndex];

while(item.next != null && item.lex != strPrevUnit)

{

item = item.next;

}

if(item.lex == strPrevUnit)

{

item.lex = strNewUnit;

}

return false;

}

public bool AddLexicalUnit(string strALexicalUnit, ref int intALexicalCode)

{

HashIndex(strALexicalUnit);

intALexicalCode = intFHashIndex;

TableItem item = arrFHashTable[intFHashIndex];

TableItem prev = arrFHashTable[intFHashIndex];

bool exist = false;

if (prev.lex == strALexicalUnit)

{

exist = true;

}

while (prev.next != null)

{

prev = prev.next;

if (prev.lex == strALexicalUnit)

{

exist = true;

}

}

if (!exist)

{

if(item.lex == prev.lex && item.lex == "")

{

item.value = intALexicalCode;

item.lex = strALexicalUnit;

}

else

{

TableItem newItem = new TableItem(intALexicalCode, strALexicalUnit, null);

prev.next = newItem;

}

//MessageBox.Show($"{item.lex}, {item.value}");

return true;

}

return false;

}

public bool DeleteLexicalUnit(string strALexicalUnit)

{

HashIndex(strALexicalUnit);

int indx = intFHashIndex;

if (arrFHashTable[indx] != null)

{

TableItem item = arrFHashTable[indx];

while (item.next != null && item.lex != strALexicalUnit)

{

item = item.next;

}

if (item.lex == strALexicalUnit)

{

if (item.next == null)

{

TableItem prev = arrFHashTable[indx];

while (prev.next != null && prev.next != item)

{

prev = prev.next;

}

prev.next = null;

item.value = 0;

item.lex = "";

}

else

{

TableItem prev = arrFHashTable[indx];

while (prev.next != null && prev.next != item)

{

prev = prev.next;

}

item.value = 0;

item.lex = "";

prev.next = item.next;

}

return true;

}

}

return false;

}

public void GetLexicalUnitList(ref List<string> sList)

{

for (int i = 0; i < tableSize; i++)

{

TableItem item = arrFHashTable[i];

while (item != null)

{

if (item.lex != "")

{

sList.Add($"{item.lex}");

}

item = item.next;

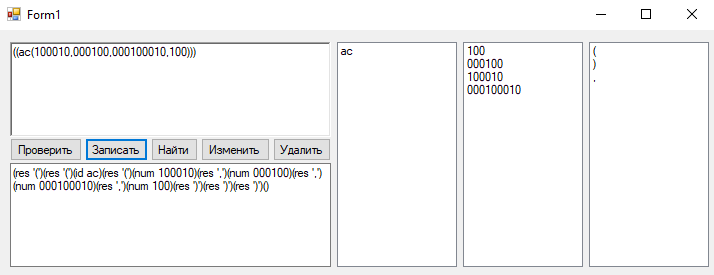
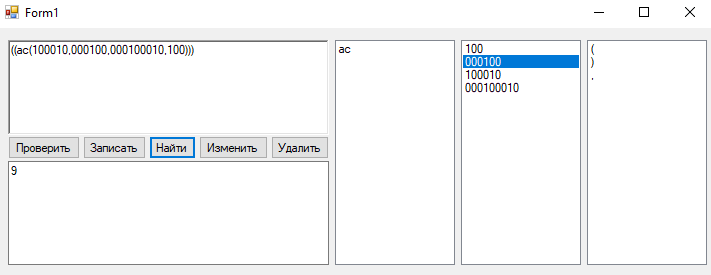
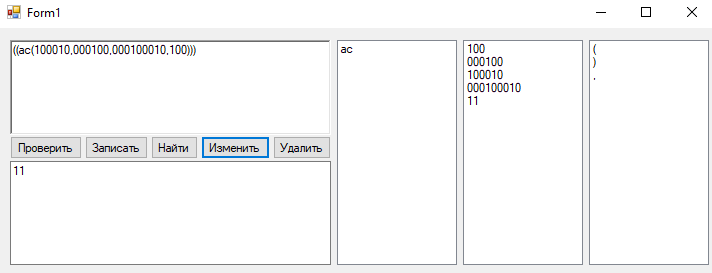
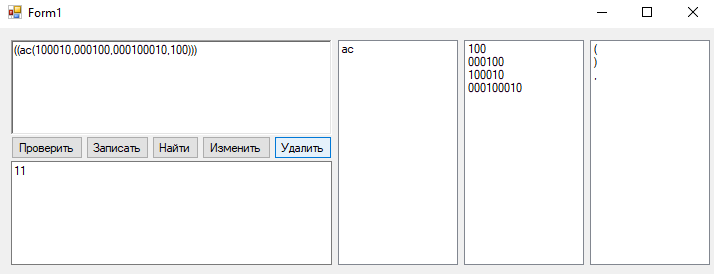
}

}

}

}

}

**Лабораторная работа № 5. Построение синтаксического дерева**

**Задание:**   
включить в синтаксический анализатор из лабораторной работы №.3 построение синтаксического дерева. Использовать атрибутный метод Кнута, т.е. преобразовать КС – грамматику из лабораторной работы № 3 в атрибутную грамматику добавлением атрибутов и правил построения синтаксического дерева. Расширить программу синтаксического анализатора из лабораторной работы № 3 введением действий по построению синтаксического дерева.

**Код программы:**

**Form1.cs:**

using System;

...

using System.Windows.Forms;

namespace WindowsFormsApp45

{

public partial class Form1 : Form

{

public CHashTableList htl = new CHashTableList(3);

public Form1()

{

InitializeComponent();

tbFSource.AppendText("((ac(100010,000100,000100010,100)))" + "\r\n");

//tbFSource.AppendText("((100(ac, acb, ac, acddc, ac)))" + "\r\n");

int n = tbFSource.Lines.Length;

}

public void TablesToMemo(object sender, System.EventArgs e)

{

List<string> listTable = new List<string>();

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

htl.TableToStringList(0, listTable);

for (int i = 0; i < listTable.Count; i++)

listBox1.Items.Add(listTable[i]);

listTable.Clear();

htl.TableToStringList(1, listTable);

for (int i = 0; i < listTable.Count; i++)

listBox2.Items.Add(listTable[i]);

listTable.Clear();

htl.TableToStringList(2, listTable);

for (int i = 0; i < listTable.Count; i++)

listBox3.Items.Add(listTable[i]);

listTable.Clear();

}

private void btnFStart\_Click(object sender, EventArgs e)

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

Synt.tree = treeView1;

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.Lines;

Synt.Lex.intPSourceColSelection = -1;

Synt.Lex.intPSourceRowSelection = 0;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.S();

//throw new Exception("Текст верный");

}

catch (Exception exc)

{

if (exc.Message == "" || exc.Message == null)

{

tbFMessage.Text += "Неизвестная ошибка";

}

else

{

tbFMessage.Text += exc.Message;

}

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Synt.Lex.intPSourceRowSelection; i++)

{

n += tbFSource.Lines[i].Length + 2;

}

n += Synt.Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

private void btnFRecord\_Click(object sender, EventArgs e)

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

s1 = "id " + Lex.strPLexicalUnit; int b = 0;

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 0, ref b))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

s1 = "num " + Lex.strPLexicalUnit; int b = 0;

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 1, ref b))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmRightParenth:

{

Reserved(ref s1, Lex, e);

break;

}

case TToken.lxmLeftParenth:

{

Reserved(ref s1, Lex, e);

break;

}

case TToken.lxmComma:

{

Reserved(ref s1, Lex, e);

break;

}

}

String m = "(" + s + "" + s1 + ")";

tbFMessage.Text += m;

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Lex.intPSourceRowSelection; i++) n += tbFSource.Lines[i].Length + 2;

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

void Reserved(ref string s1, CLex Lex, EventArgs e)

{

s1 = "res '" + Lex.strPLexicalUnit + "'";

int b = 0;

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b))

{

TablesToMemo(this, e);

}

}

private void btnDelete\_Click(object sender, EventArgs e)

{

if (listBox1.SelectedIndex >= 0)

{

if (htl.DeleteLexicalUnit(listBox1.Items[listBox1.SelectedIndex].ToString(), 0))

{

listBox1.Items.RemoveAt(listBox1.SelectedIndex);

listBox1.SelectedIndex = -1;

}

}

if (listBox2.SelectedIndex >= 0)

{

if (htl.DeleteLexicalUnit(listBox2.Items[listBox2.SelectedIndex].ToString(), 1))

{

listBox2.Items.RemoveAt(listBox2.SelectedIndex);

listBox2.SelectedIndex = -1;

}

}

if (listBox3.SelectedIndex >= 0)

{

if (htl.DeleteLexicalUnit(listBox3.Items[listBox3.SelectedIndex].ToString(), 2))

{

listBox3.Items.RemoveAt(listBox3.SelectedIndex);

listBox3.SelectedIndex = -1;

}

}

}

private void findBtn\_Click(object sender, EventArgs e)

{

if (listBox1.SelectedIndex >= 0)

{

int d = 0;

if (htl.SearchLexicalUnit(listBox1.Items[listBox1.SelectedIndex].ToString(), 0, ref d))

{

tbFMessage.Text = d.ToString();

}

else

{

tbFMessage.Text = "Не удалось найти индекс элемента";

}

}

if (listBox2.SelectedIndex >= 0)

{

int d = 0;

if (htl.SearchLexicalUnit(listBox2.Items[listBox2.SelectedIndex].ToString(), 1, ref d))

{

tbFMessage.Text = d.ToString();

}

else

{

tbFMessage.Text = "Не удалось найти индекс элемента";

}

}

if (listBox3.SelectedIndex >= 0)

{

int d = 0;

if (htl.SearchLexicalUnit(listBox3.Items[listBox3.SelectedIndex].ToString(), 2, ref d))

{

tbFMessage.Text = d.ToString();

}

else

{

tbFMessage.Text = "Не удалось найти индекс элемента";

}

}

}

private void ChngBtn\_Click(object sender, EventArgs e)

{

if (listBox1.SelectedIndex >= 0)

{

htl.ChangeLexicalUnit(listBox1.Items[listBox1.SelectedIndex].ToString(), 0, tbFMessage.Text);

List<string> listTable = new List<string>();

listBox1.Items.Clear();

htl.TableToStringList(0, listTable);

for (int i = 0; i < listTable.Count; i++)

listBox1.Items.Add(listTable[i]);

listTable.Clear();

}

if (listBox2.SelectedIndex >= 0)

{

htl.ChangeLexicalUnit(listBox2.Items[listBox2.SelectedIndex].ToString(), 1, tbFMessage.Text);

List<string> listTable = new List<string>();

listBox2.Items.Clear();

htl.TableToStringList(1, listTable);

for (int i = 0; i < listTable.Count; i++)

listBox2.Items.Add(listTable[i]);

listTable.Clear();

}

if (listBox3.SelectedIndex >= 0)

{

htl.ChangeLexicalUnit(listBox3.Items[listBox3.SelectedIndex].ToString(), 2, tbFMessage.Text);

List<string> listTable = new List<string>();

listBox3.Items.Clear();

htl.TableToStringList(2, listTable);

for (int i = 0; i < listTable.Count; i++)

listBox3.Items.Add(listTable[i]);

listTable.Clear();

}

}

private void listBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

listBox2.ClearSelected();

listBox3.ClearSelected();

}

private void listBox2\_SelectedIndexChanged(object sender, EventArgs e)

{

listBox1.ClearSelected();

listBox3.ClearSelected();

}

private void listBox3\_SelectedIndexChanged(object sender, EventArgs e)

{

listBox1.ClearSelected();

listBox2.ClearSelected();

}

}

}

uSyntAnalyzer.cs

using System;

...

using System.Windows.Forms;

namespace WindowsFormsApp45

{

public class uSyntAnalyzer

{

private String[] strFSource;

private String[] strFMessage;

public String[] strPSource { set { strFSource = value; } get { return strFSource; } }

public String[] strPMessage { set { strFMessage = value; } get { return strFMessage; } }

public CLex Lex = new CLex();

public TToken firstToken;

public TreeView tree;

public void S()

{

TreeNode parent = new TreeNode("S");

tree.Nodes.Add(parent);

if (Lex.enumPToken == TToken.lxmLeftParenth) //(A)

{

parent.Nodes.Add(new TreeNode(Lex.strPLexicalUnit));

A(parent);

if (Lex.enumPToken == TToken.lxmRightParenth)

{

parent.Nodes.Add(new TreeNode(Lex.strPLexicalUnit));

throw new Exception("Конец слова, текст верный. [S]");

}

else throw new Exception("Ожидалось ) [S]");

}

else throw new Exception("Ожидалось ( [S]");

}

public void A(TreeNode highParent)

{

TreeNode parent = new TreeNode("A");

highParent.Nodes.Add(parent);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmLeftParenth) //(<2>B)

{

parent.Nodes.Add(new TreeNode(Lex.strPLexicalUnit));

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier || Lex.enumPToken == TToken.lxmNumber)

{

parent.Nodes.Add(new TreeNode(Lex.strPLexicalUnit));

firstToken = (TToken)(1 - (int)Lex.enumPToken);

B(parent);

if (Lex.enumPToken == TToken.lxmRightParenth)

{

parent.Nodes.Add(new TreeNode(Lex.strPLexicalUnit));

Lex.NextToken();

}

else throw new Exception("Ожидалось ) [A]");

}

else throw new Exception("Ожидался идентификатор или число [A]");

}

else //<1>

{

if(Lex.enumPToken == TToken.lxmIdentifier || Lex.enumPToken == TToken.lxmNumber)

{

firstToken = Lex.enumPToken;

Lex.NextToken();

}

else throw new Exception("Ожидался идентификатор или число или ( [A]");

}

}

public void B(TreeNode highParent)

{

TreeNode parent = new TreeNode("B");

highParent.Nodes.Add(parent);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmLeftParenth) //(C)

{

parent.Nodes.Add(new TreeNode(Lex.strPLexicalUnit));

Lex.NextToken();

C(parent);

if (Lex.enumPToken == TToken.lxmRightParenth)

{

parent.Nodes.Add(new TreeNode(Lex.strPLexicalUnit));

Lex.NextToken();

}

else throw new Exception("Ожидалось ) [B]");

}

else throw new Exception("Ожидалось ( [B]");

}

public void C(TreeNode highParent)

{

TreeNode parent = new TreeNode("C");

highParent.Nodes.Add(parent);

if (Lex.enumPToken == firstToken)

{

parent.Nodes.Add(new TreeNode(Lex.strPLexicalUnit));

D(parent);

}

else throw new Exception("Ожидался идентификатор или число [C]");

}

public void D(TreeNode highParent)

{

TreeNode parent = new TreeNode("D");

highParent.Nodes.Add(parent);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmComma)

{

parent.Nodes.Add(new TreeNode(Lex.strPLexicalUnit));

Lex.NextToken();

if (Lex.enumPToken == firstToken)

{

parent.Nodes.Add(new TreeNode(Lex.strPLexicalUnit));

D(parent);

}

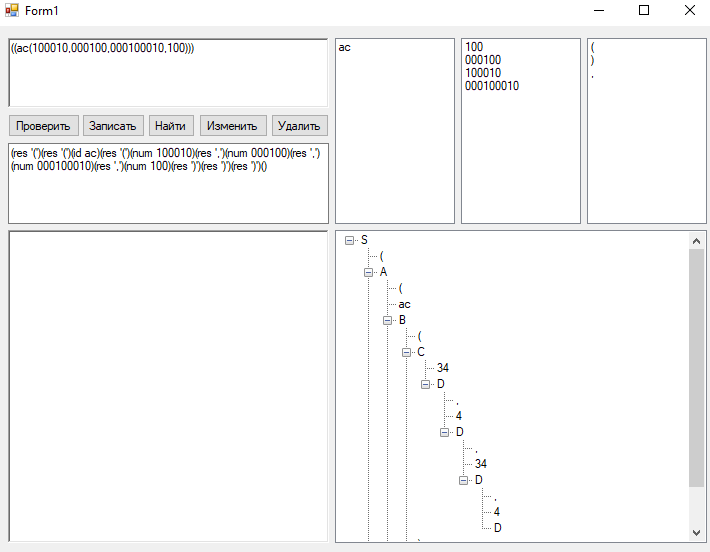
else throw new Exception("Ожидался идентификатор или число [D]");

}

}

}

}



**Лабораторная работа № 6. Разработка генератора (2 часа)**

**Задание:**

1. Перевести все числа в десятичное представление.

2. Выполнить вывод исходного текста в структурированном виде.

Код программы:

Form1.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp45

{

public partial class Form1 : Form

{

public CHashTableList htl = new CHashTableList(3);

public Form1()

{

InitializeComponent();

tbFSource.AppendText("((ac(100010,000100,000100010,100)))" + "\r\n");

//tbFSource.AppendText("((100(ac, acb, ac, acddc, ac)))" + "\r\n");

int n = tbFSource.Lines.Length;

}

public void TablesToMemo(object sender, System.EventArgs e)

{

List<string> listTable = new List<string>();

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

htl.TableToStringList(0, listTable);

for (int i = 0; i < listTable.Count; i++)

listBox1.Items.Add(listTable[i]);

listTable.Clear();

htl.TableToStringList(1, listTable);

for (int i = 0; i < listTable.Count; i++)

listBox2.Items.Add(listTable[i]);

listTable.Clear();

htl.TableToStringList(2, listTable);

for (int i = 0; i < listTable.Count; i++)

listBox3.Items.Add(listTable[i]);

listTable.Clear();

}

private void btnFStart\_Click(object sender, EventArgs e)

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

Synt.tree = syntTree;

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.Lines;

Synt.Lex.intPSourceColSelection = -1;

Synt.Lex.intPSourceRowSelection = 0;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.S();

//throw new Exception("Текст верный");

}

catch (Exception exc)

{

if (exc.Message == "" || exc.Message == null)

{

tbFMessage.Text += "Неизвестная ошибка";

}

else

{

tbFMessage.Text += exc.Message;

}

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Synt.Lex.intPSourceRowSelection; i++)

{

n += tbFSource.Lines[i].Length + 2;

}

n += Synt.Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

private void btnFRecord\_Click(object sender, EventArgs e)

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

Generator gen = new Generator();

gen.Restruct(syntTree, structTree);

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

s1 = "id " + Lex.strPLexicalUnit; int b = 0;

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 0, ref b))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

s1 = "num " + Lex.strPLexicalUnit; int b = 0;

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 1, ref b))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmRightParenth:

{

Reserved(ref s1, Lex, e);

break;

}

case TToken.lxmLeftParenth:

{

Reserved(ref s1, Lex, e);

break;

}

case TToken.lxmComma:

{

Reserved(ref s1, Lex, e);

break;

}

}

String m = "(" + s + "" + s1 + ")";

tbFMessage.Text += m;

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Lex.intPSourceRowSelection; i++) n += tbFSource.Lines[i].Length + 2;

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

void Reserved(ref string s1, CLex Lex, EventArgs e)

{

s1 = "res '" + Lex.strPLexicalUnit + "'";

int b = 0;

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b))

{

TablesToMemo(this, e);

}

}

private void btnDelete\_Click(object sender, EventArgs e)

{

if (listBox1.SelectedIndex >= 0)

{

if (htl.DeleteLexicalUnit(listBox1.Items[listBox1.SelectedIndex].ToString(), 0))

{

listBox1.Items.RemoveAt(listBox1.SelectedIndex);

listBox1.SelectedIndex = -1;

}

}

if (listBox2.SelectedIndex >= 0)

{

if (htl.DeleteLexicalUnit(listBox2.Items[listBox2.SelectedIndex].ToString(), 1))

{

listBox2.Items.RemoveAt(listBox2.SelectedIndex);

listBox2.SelectedIndex = -1;

}

}

if (listBox3.SelectedIndex >= 0)

{

if (htl.DeleteLexicalUnit(listBox3.Items[listBox3.SelectedIndex].ToString(), 2))

{

listBox3.Items.RemoveAt(listBox3.SelectedIndex);

listBox3.SelectedIndex = -1;

}

}

}

private void findBtn\_Click(object sender, EventArgs e)

{

if (listBox1.SelectedIndex >= 0)

{

int d = 0;

if (htl.SearchLexicalUnit(listBox1.Items[listBox1.SelectedIndex].ToString(), 0, ref d))

{

tbFMessage.Text = d.ToString();

}

else

{

tbFMessage.Text = "Не удалось найти индекс элемента";

}

}

if (listBox2.SelectedIndex >= 0)

{

int d = 0;

if (htl.SearchLexicalUnit(listBox2.Items[listBox2.SelectedIndex].ToString(), 1, ref d))

{

tbFMessage.Text = d.ToString();

}

else

{

tbFMessage.Text = "Не удалось найти индекс элемента";

}

}

if (listBox3.SelectedIndex >= 0)

{

int d = 0;

if (htl.SearchLexicalUnit(listBox3.Items[listBox3.SelectedIndex].ToString(), 2, ref d))

{

tbFMessage.Text = d.ToString();

}

else

{

tbFMessage.Text = "Не удалось найти индекс элемента";

}

}

}

private void ChngBtn\_Click(object sender, EventArgs e)

{

if (listBox1.SelectedIndex >= 0)

{

htl.ChangeLexicalUnit(listBox1.Items[listBox1.SelectedIndex].ToString(), 0, tbFMessage.Text);

List<string> listTable = new List<string>();

listBox1.Items.Clear();

htl.TableToStringList(0, listTable);

for (int i = 0; i < listTable.Count; i++)

listBox1.Items.Add(listTable[i]);

listTable.Clear();

}

if (listBox2.SelectedIndex >= 0)

{

htl.ChangeLexicalUnit(listBox2.Items[listBox2.SelectedIndex].ToString(), 1, tbFMessage.Text);

List<string> listTable = new List<string>();

listBox2.Items.Clear();

htl.TableToStringList(1, listTable);

for (int i = 0; i < listTable.Count; i++)

listBox2.Items.Add(listTable[i]);

listTable.Clear();

}

if (listBox3.SelectedIndex >= 0)

{

htl.ChangeLexicalUnit(listBox3.Items[listBox3.SelectedIndex].ToString(), 2, tbFMessage.Text);

List<string> listTable = new List<string>();

listBox3.Items.Clear();

htl.TableToStringList(2, listTable);

for (int i = 0; i < listTable.Count; i++)

listBox3.Items.Add(listTable[i]);

listTable.Clear();

}

}

private void listBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

listBox2.ClearSelected();

listBox3.ClearSelected();

}

private void listBox2\_SelectedIndexChanged(object sender, EventArgs e)

{

listBox1.ClearSelected();

listBox3.ClearSelected();

}

private void listBox3\_SelectedIndexChanged(object sender, EventArgs e)

{

listBox1.ClearSelected();

listBox2.ClearSelected();

}

}

}

Generator.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp45

{

public class Generator

{

public List<string> ViewTree(TreeView tree, bool edit = true)

{

List<string> treeContent = new List<string>();

foreach(TreeNode node in tree.Nodes)

{

treeContent.AddRange(ViewNode(node, edit));

}

return treeContent;

}

public List<string> ViewNode(TreeNode node, bool edit)

{

List<string> nodeContent = new List<string>();

foreach (TreeNode child in node.Nodes)

{

nodeContent.AddRange(ViewNode(child, edit));

}

if (edit)

{

int value = -1;

if (int.TryParse(node.Text, out value))

{

int newValue = ConvertToBase10(value);

node.Text = newValue.ToString();

}

}

nodeContent.Add(node.Text);

return nodeContent;

}

public int ConvertToBase10(int num)

{

int k = 0;

for (int i = 0; i < num.ToString().Length; i++)

{

int r = num.ToString().Length - i;

int v = num.ToString()[i] == '1' ? 1 : 0;

k += (int)Math.Pow(2, r - 1) \* v;

}

return k;

}

public TreeView currentTree;

public int depth = 0;

public void Restruct(TreeView box, TreeView tree)

{

currentTree = tree;

tree.Nodes.Clear();

TreeNode sNode = null;

foreach (TreeNode node in box.Nodes)

{

TreeNode newNode = new TreeNode(node.Text);

//MessageBox.Show(node.Text);

ReNode(null, node, ref newNode);

if(newNode.Text == "S")

{

sNode = newNode;

TreeNode nNode = new TreeNode("");

sNode.Nodes.Add(nNode);

sNode = nNode;

}

tree.Nodes.Add(newNode);

}

List<TreeNode> remove = new List<TreeNode>();

foreach (TreeNode node in tree.Nodes)

{

if(node.Text != "S" && sNode != null)

{

remove.Add(node);

}

}

TreeNode neNode = new TreeNode("");

sNode.Nodes.Add(neNode);

foreach(TreeNode node in remove)

{

if (node.Text != "A")

{

tree.Nodes.Remove(node);

sNode.Nodes.Add(node);

}

else

{

tree.Nodes.Remove(node);

neNode.Nodes.Add(node);

}

}

tree.ExpandAll();

}

public void ReNode(TreeNode parent, TreeNode node, ref TreeNode newNode)

{

bool skipToRoot = false;

foreach (TreeNode child in node.Nodes)

{

//MessageBox.Show(child.Text);

TreeNode newn = new TreeNode(child.Text);

int value = -1;

if (int.TryParse(newn.Text, out value))

{

int newValue = ConvertToBase10(value);

newn.Text = newValue.ToString();

}

if(child.FullPath.Contains('('))

{

skipToRoot = true;

}

ReNode(newNode, child, ref newn);

}

if(parent != null)

{

if(currentTree != null && skipToRoot)

{

currentTree.Nodes.Add(newNode);

}

else

{

parent.Nodes.Add(newNode);

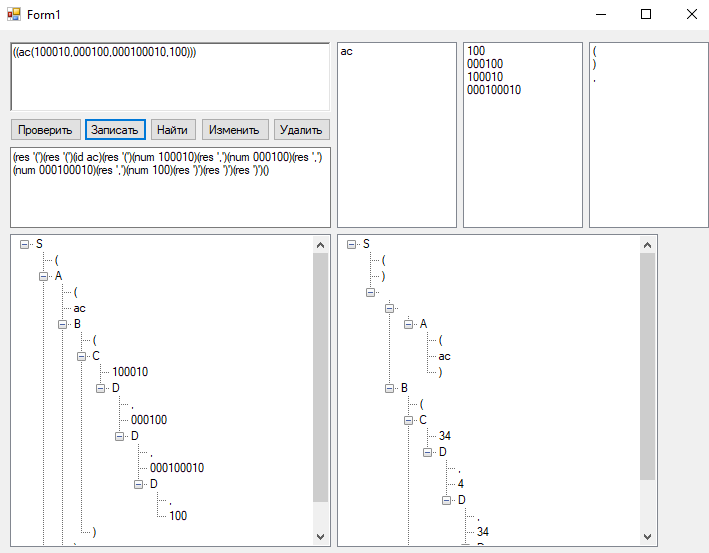
}

}

}

}

}



**Лабораторная работа № 7. Разработка контекстного анализатора**

**Задание:** для предложенного преподавателем варианта контекстного условия расширить атрибутную грамматику из лабораторной работы № 4 добавлением атрибутов, правил их вычисления, правил вычисления контекстных условий. Включить в программу синтаксического анализатора из лабораторной работы № 4 действия по вычислению атрибутов и проверки контекстных условий.

**Вариант 5:** все числа должны быть разными.

Код программы:  
uSyntAnalyzer.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp45

{

public class uSyntAnalyzer

{

private String[] strFSource;

private String[] strFMessage;

public String[] strPSource { set { strFSource = value; } get { return strFSource; } }

public String[] strPMessage { set { strFMessage = value; } get { return strFMessage; } }

public CLex Lex = new CLex();

public TToken firstToken;

public List<string> numTokenList = new List<string>();

public bool noDuplications = true;

public TreeView tree;

public void AddTokenToTree(TToken token, string input, TreeNode parent)

{

if(token == TToken.lxmNumber)

{

for (int i = 0; i < numTokenList.Count; i++)

{

if(input == numTokenList[i])

{

noDuplications = false;

}

}

numTokenList.Add(input);

}

parent.Nodes.Add(new TreeNode(input));

}

public void S()

{

numTokenList.Clear();

noDuplications = true;

tree.Nodes.Clear();

TreeNode parent = new TreeNode("S");

tree.Nodes.Add(parent);

if (Lex.enumPToken == TToken.lxmLeftParenth) //(A)

{

AddTokenToTree(Lex.enumPToken, Lex.strPLexicalUnit, parent);

A(parent);

if (Lex.enumPToken == TToken.lxmRightParenth)

{

AddTokenToTree(Lex.enumPToken, Lex.strPLexicalUnit, parent);

tree.ExpandAll();

if (!noDuplications)

{

throw new Exception("Найдены повторяющиеся числа!");

}

throw new Exception("Конец слова, текст верный. [S]");

}

else throw new Exception("Ожидалось ) [S]");

}

else throw new Exception("Ожидалось ( [S]");

}

public void A(TreeNode highParent)

{

TreeNode parent = new TreeNode("A");

highParent.Nodes.Add(parent);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmLeftParenth) //(<2>B)

{

AddTokenToTree(Lex.enumPToken, Lex.strPLexicalUnit, parent);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier || Lex.enumPToken == TToken.lxmNumber)

{

AddTokenToTree(Lex.enumPToken, Lex.strPLexicalUnit, parent);

firstToken = (TToken)(1 - (int)Lex.enumPToken);

B(parent);

if (Lex.enumPToken == TToken.lxmRightParenth)

{

AddTokenToTree(Lex.enumPToken, Lex.strPLexicalUnit, parent);

Lex.NextToken();

}

else throw new Exception("Ожидалось ) [A]");

}

else throw new Exception("Ожидался идентификатор или число [A]");

}

else //<1>

{

if(Lex.enumPToken == TToken.lxmIdentifier || Lex.enumPToken == TToken.lxmNumber)

{

firstToken = Lex.enumPToken;

Lex.NextToken();

}

else throw new Exception("Ожидался идентификатор или число или ( [A]");

}

}

public void B(TreeNode highParent)

{

TreeNode parent = new TreeNode("B");

highParent.Nodes.Add(parent);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmLeftParenth) //(C)

{

AddTokenToTree(Lex.enumPToken, Lex.strPLexicalUnit, parent);

Lex.NextToken();

C(parent);

if (Lex.enumPToken == TToken.lxmRightParenth)

{

AddTokenToTree(Lex.enumPToken, Lex.strPLexicalUnit, parent);

Lex.NextToken();

}

else throw new Exception("Ожидалось ) [B]");

}

else throw new Exception("Ожидалось ( [B]");

}

public void C(TreeNode highParent)

{

TreeNode parent = new TreeNode("C");

highParent.Nodes.Add(parent);

if (Lex.enumPToken == firstToken)

{

AddTokenToTree(Lex.enumPToken, Lex.strPLexicalUnit, parent);

D(parent);

}

else throw new Exception("Ожидался идентификатор или число [C]");

}

public void D(TreeNode highParent)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmComma)

{

TreeNode parent = new TreeNode("D");

highParent.Nodes.Add(parent);

AddTokenToTree(Lex.enumPToken, Lex.strPLexicalUnit, parent);

Lex.NextToken();

if (Lex.enumPToken == firstToken)

{

AddTokenToTree(Lex.enumPToken, Lex.strPLexicalUnit, parent);

D(parent);

}

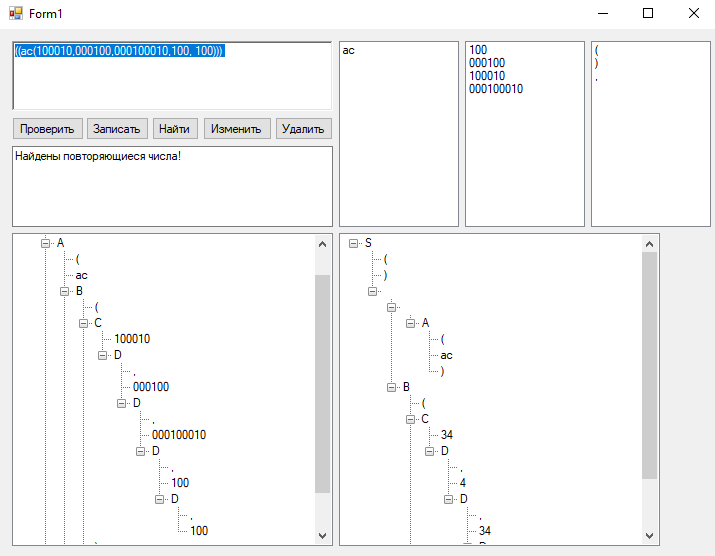
else throw new Exception("Ожидался идентификатор или число [D]");

}

}

}

}



**Лабораторная работа №8. Разработка семантического анализатора.**

**Задание:** Разработать семантический анализатор. Выполнить проверку внеконтекстной грамматики.

**Вариант:** При соответствии количества блоков записи <1>, <1> количеству согласных в слове <2>, предложение должно содержать слово «100»

Код программы:

Form1.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp45

{

public partial class Form1 : Form

{

public CHashTableList htl = new CHashTableList(3);

public Semantic sem;

public Form1()

{

InitializeComponent();

tbFSource.AppendText("((acbd(100010,000100,000100010)))" + "\r\n");

//tbFSource.AppendText("((100(ac, acb, ac, acddc, ac)))" + "\r\n");

int n = tbFSource.Lines.Length;

}

public void TablesToMemo(object sender, System.EventArgs e)

{

List<string> listTable = new List<string>();

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

htl.TableToStringList(0, listTable);

for (int i = 0; i < listTable.Count; i++)

listBox1.Items.Add(listTable[i]);

listTable.Clear();

htl.TableToStringList(1, listTable);

for (int i = 0; i < listTable.Count; i++)

listBox2.Items.Add(listTable[i]);

listTable.Clear();

htl.TableToStringList(2, listTable);

for (int i = 0; i < listTable.Count; i++)

listBox3.Items.Add(listTable[i]);

listTable.Clear();

}

private void btnFStart\_Click(object sender, EventArgs e)

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

Synt.tree = syntTree;

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.Lines;

Synt.Lex.intPSourceColSelection = -1;

Synt.Lex.intPSourceRowSelection = 0;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.S();

}

catch (Exception exc)

{

if (exc.Message == "" || exc.Message == null)

{

tbFMessage.Text += "Неизвестная ошибка";

}

else

{

tbFMessage.Text += exc.Message;

}

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Synt.Lex.intPSourceRowSelection; i++)

{

n += tbFSource.Lines[i].Length + 2;

}

n += Synt.Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

private void btnFRecord\_Click(object sender, EventArgs e)

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

Generator gen = new Generator();

gen.Restruct(syntTree, structTree);

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

s1 = "id " + Lex.strPLexicalUnit; int b = 0;

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 0, ref b))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

s1 = "num " + Lex.strPLexicalUnit; int b = 0;

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 1, ref b))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmRightParenth:

{

Reserved(ref s1, Lex, e);

break;

}

case TToken.lxmLeftParenth:

{

Reserved(ref s1, Lex, e);

break;

}

case TToken.lxmComma:

{

Reserved(ref s1, Lex, e);

break;

}

}

String m = "(" + s + "" + s1 + ")";

tbFMessage.Text += m;

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Lex.intPSourceRowSelection; i++) n += tbFSource.Lines[i].Length + 2;

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

void Reserved(ref string s1, CLex Lex, EventArgs e)

{

s1 = "res '" + Lex.strPLexicalUnit + "'";

int b = 0;

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b))

{

TablesToMemo(this, e);

}

}

private void btnDelete\_Click(object sender, EventArgs e)

{

if (listBox1.SelectedIndex >= 0)

{

if (htl.DeleteLexicalUnit(listBox1.Items[listBox1.SelectedIndex].ToString(), 0))

{

listBox1.Items.RemoveAt(listBox1.SelectedIndex);

listBox1.SelectedIndex = -1;

}

}

if (listBox2.SelectedIndex >= 0)

{

if (htl.DeleteLexicalUnit(listBox2.Items[listBox2.SelectedIndex].ToString(), 1))

{

listBox2.Items.RemoveAt(listBox2.SelectedIndex);

listBox2.SelectedIndex = -1;

}

}

if (listBox3.SelectedIndex >= 0)

{

if (htl.DeleteLexicalUnit(listBox3.Items[listBox3.SelectedIndex].ToString(), 2))

{

listBox3.Items.RemoveAt(listBox3.SelectedIndex);

listBox3.SelectedIndex = -1;

}

}

}

private void findBtn\_Click(object sender, EventArgs e)

{

if (listBox1.SelectedIndex >= 0)

{

int d = 0;

if (htl.SearchLexicalUnit(listBox1.Items[listBox1.SelectedIndex].ToString(), 0, ref d))

{

tbFMessage.Text = d.ToString();

}

else

{

tbFMessage.Text = "Не удалось найти индекс элемента";

}

}

if (listBox2.SelectedIndex >= 0)

{

int d = 0;

if (htl.SearchLexicalUnit(listBox2.Items[listBox2.SelectedIndex].ToString(), 1, ref d))

{

tbFMessage.Text = d.ToString();

}

else

{

tbFMessage.Text = "Не удалось найти индекс элемента";

}

}

if (listBox3.SelectedIndex >= 0)

{

int d = 0;

if (htl.SearchLexicalUnit(listBox3.Items[listBox3.SelectedIndex].ToString(), 2, ref d))

{

tbFMessage.Text = d.ToString();

}

else

{

tbFMessage.Text = "Не удалось найти индекс элемента";

}

}

}

private void ChngBtn\_Click(object sender, EventArgs e)

{

if (listBox1.SelectedIndex >= 0)

{

htl.ChangeLexicalUnit(listBox1.Items[listBox1.SelectedIndex].ToString(), 0, tbFMessage.Text);

List<string> listTable = new List<string>();

listBox1.Items.Clear();

htl.TableToStringList(0, listTable);

for (int i = 0; i < listTable.Count; i++)

listBox1.Items.Add(listTable[i]);

listTable.Clear();

}

if (listBox2.SelectedIndex >= 0)

{

htl.ChangeLexicalUnit(listBox2.Items[listBox2.SelectedIndex].ToString(), 1, tbFMessage.Text);

List<string> listTable = new List<string>();

listBox2.Items.Clear();

htl.TableToStringList(1, listTable);

for (int i = 0; i < listTable.Count; i++)

listBox2.Items.Add(listTable[i]);

listTable.Clear();

}

if (listBox3.SelectedIndex >= 0)

{

htl.ChangeLexicalUnit(listBox3.Items[listBox3.SelectedIndex].ToString(), 2, tbFMessage.Text);

List<string> listTable = new List<string>();

listBox3.Items.Clear();

htl.TableToStringList(2, listTable);

for (int i = 0; i < listTable.Count; i++)

listBox3.Items.Add(listTable[i]);

listTable.Clear();

}

}

private void listBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

listBox2.ClearSelected();

listBox3.ClearSelected();

}

private void listBox2\_SelectedIndexChanged(object sender, EventArgs e)

{

listBox1.ClearSelected();

listBox3.ClearSelected();

}

private void listBox3\_SelectedIndexChanged(object sender, EventArgs e)

{

listBox1.ClearSelected();

listBox2.ClearSelected();

}

}

}

CHashTableList.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows.Forms;

using System.IO;

namespace WindowsFormsApp45

{

public class CHashTableList

{

private List<THashTable> arrFHashTableList = new List<THashTable>();

private byte byteFTablesSize;

public CHashTableList(byte byteATableCount)

{

this.byteFTablesSize = byteATableCount;

for (int i = 0; i < byteATableCount; i++)

{

arrFHashTableList.Add(new THashTable());

}

}

public bool SearchLexicalUnit(string strALexicalUnit, byte byteATable, ref int intALexicalCode)

{

return arrFHashTableList[byteATable].SearchLexicalUnit(strALexicalUnit, ref intALexicalCode);

}

public bool AddLexicalUnit(string strALexicalUnit, byte byteATable, ref int intALexicalCode)

{

return arrFHashTableList[byteATable].AddLexicalUnit(strALexicalUnit, ref intALexicalCode);

}

public bool ChangeLexicalUnit(string strALexicalUnit, byte byteATable, string newLexUnit)

{

int d = 0;

arrFHashTableList[byteATable].DeleteLexicalUnit(strALexicalUnit);

return arrFHashTableList[byteATable].AddLexicalUnit(newLexUnit, ref d);

}

public bool DeleteLexicalUnit(string strALexicalUnit, byte byteATable)

{

return arrFHashTableList[byteATable].DeleteLexicalUnit(strALexicalUnit);

}

public void TableToStringList(byte byteATable, List<string> sList)

{

arrFHashTableList[byteATable].GetLexicalUnitList(ref sList);

}

public int GetHashIndex(byte Table)

{

return arrFHashTableList[Table].intFHashIndex;

}

}

}

Generator.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp45

{

public class Generator

{

public List<string> ViewTree(TreeView tree, bool edit = true)

{

List<string> treeContent = new List<string>();

foreach(TreeNode node in tree.Nodes)

{

treeContent.AddRange(ViewNode(node, edit));

}

return treeContent;

}

public List<string> ViewNode(TreeNode node, bool edit)

{

List<string> nodeContent = new List<string>();

foreach (TreeNode child in node.Nodes)

{

nodeContent.AddRange(ViewNode(child, edit));

}

if (edit)

{

int value = -1;

if (int.TryParse(node.Text, out value))

{

int newValue = ConvertToBase10(value);

node.Text = newValue.ToString();

}

}

nodeContent.Add(node.Text);

return nodeContent;

}

public int ConvertToBase10(int num)

{

int k = 0;

for (int i = 0; i < num.ToString().Length; i++)

{

int r = num.ToString().Length - i;

int v = num.ToString()[i] == '1' ? 1 : 0;

k += (int)Math.Pow(2, r - 1) \* v;

}

return k;

}

public TreeView currentTree;

public int depth = 0;

public void Restruct(TreeView box, TreeView tree)

{

currentTree = tree;

tree.Nodes.Clear();

TreeNode sNode = null;

foreach (TreeNode node in box.Nodes)

{

TreeNode newNode = new TreeNode(node.Text);

//MessageBox.Show(node.Text);

ReNode(null, node, ref newNode);

if(newNode.Text == "S")

{

sNode = newNode;

TreeNode nNode = new TreeNode("");

sNode.Nodes.Add(nNode);

sNode = nNode;

}

tree.Nodes.Add(newNode);

}

List<TreeNode> remove = new List<TreeNode>();

foreach (TreeNode node in tree.Nodes)

{

if(node.Text != "S" && sNode != null)

{

remove.Add(node);

}

}

TreeNode neNode = new TreeNode("");

sNode.Nodes.Add(neNode);

foreach(TreeNode node in remove)

{

if (node.Text != "A")

{

tree.Nodes.Remove(node);

sNode.Nodes.Add(node);

}

else

{

tree.Nodes.Remove(node);

neNode.Nodes.Add(node);

}

}

tree.ExpandAll();

}

public void ReNode(TreeNode parent, TreeNode node, ref TreeNode newNode)

{

bool skipToRoot = false;

foreach (TreeNode child in node.Nodes)

{

//MessageBox.Show(child.Text);

TreeNode newn = new TreeNode(child.Text);

int value = -1;

if (int.TryParse(newn.Text, out value))

{

int newValue = ConvertToBase10(value);

newn.Text = newValue.ToString();

}

if(child.FullPath.Contains('('))

{

skipToRoot = true;

}

ReNode(newNode, child, ref newn);

}

if(parent != null)

{

if(currentTree != null && skipToRoot)

{

currentTree.Nodes.Add(newNode);

}

else

{

parent.Nodes.Add(newNode);

}

}

}

}

}

Semantic.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Drawing;

using System.Windows.Forms;

namespace WindowsFormsApp45

{

public class Semantic

{

public int countLetters = 0;

public int countDigits = 0;

public string strIndentifier;

public string strDigital;

private TreeView tree;

public bool contains100 = false;

public Semantic()

{

}

public Semantic(TreeView treeView)

{

tree = treeView;

TreeController(tree);

}

public void TreeController(TreeView tree)

{

foreach (TreeNode node in tree.Nodes)

{

TreeController(node);

}

Check();

}

public void TreeController(TreeNode node)

{

if (node.Text == "A")

{

if (node.Nodes.Count > 1)

{

strIndentifier = node.Nodes[1].Text.ToString();

CheckLetters(strIndentifier);

}

}

if (node.Text == "C")

{

//Блок С содержит искомый блок <2>

if (node.Nodes.Count > 0)

{

strDigital = node.Nodes[0].Text.ToString();

isNumber100(strDigital);

countDigits++;

}

}

if (node.Text == "D")

{

//Блок D содержит искомый блок , <2>

if (node.Nodes.Count > 1)

{

strDigital = node.Nodes[1].Text.ToString();

isNumber100(strDigital);

countDigits++;

}

}

foreach (TreeNode childNode in node.Nodes)

{

TreeController(childNode);

}

}

//Проверка на наличие элемента "100", в случае равенства количества согласных в конце идентификатора с количеством блоков двоичной записи в конструкциях вида <2>, <2>, ...

private void Check()

{

if (countDigits != countLetters)

{

throw new Exception("Конец слова, текст верный.");

}

else

{

if (contains100)

{

throw new Exception("Конец слова, текст верный.");

}

else

{

throw new Exception("Число согласных соответствует числу элементов, но среди элементов нет 100");

}

}

}

private void isNumber100(string input)

{

if(input == "100")

{

contains100 = true;

}

}

private void CheckLetters(string input)

{

for (int i = input.Length - 1; i >= 0; --i)

{

//Подсчет числа согласных

if (input[i] == 'd' || input[i] == 'c' || input[i] == 'b')

{

countLetters++;

}

}

}

}

}

THashTable.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Windows.Forms;

namespace WindowsFormsApp45

{

public class TableItem

{

public int value;

public string lex;

public TableItem next;

public TableItem(int v, string s, TableItem t)

{

value = v;

lex = s;

next = t;

}

}

public class THashTable

{

public List<TableItem> arrFHashTable = new List<TableItem>();

public int intFHashIndex;

static int tableSize = 20;

public THashTable()

{

Init(tableSize);

}

public void Init(int count)

{

arrFHashTable.Clear();

Resize(arrFHashTable, count);

}

static void Resize(List<TableItem> list, int size)

{

if (size > list.Count)

while (size > list.Count)

list.Add(new TableItem(0, "", null));

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

int HashFunction(string strALexicalUnit)

{

int h = 0;

for (int i = 0, l = strALexicalUnit.Length; i < l; i++)

{

h += strALexicalUnit[i];

}

return h % tableSize;

}

public void HashIndex(string strALexicalUnit)

{

int h = HashFunction(strALexicalUnit);

intFHashIndex = h;

}

public bool SearchLexicalUnit(string strAlexicalUnit, ref int intALexicalCode)

{

HashIndex(strAlexicalUnit);

if (arrFHashTable[intFHashIndex].lex == "") return false;

else

{

intALexicalCode = arrFHashTable[intFHashIndex].value;

return true;

}

}

public bool ChangeLexicalUnit(string strPrevUnit, string strNewUnit)

{

HashIndex(strPrevUnit);

TableItem item = arrFHashTable[intFHashIndex];

while(item.next != null && item.lex != strPrevUnit)

{

item = item.next;

}

if(item.lex == strPrevUnit)

{

item.lex = strNewUnit;

}

return false;

}

public bool AddLexicalUnit(string strALexicalUnit, ref int intALexicalCode)

{

HashIndex(strALexicalUnit);

intALexicalCode = intFHashIndex;

TableItem item = arrFHashTable[intFHashIndex];

TableItem prev = arrFHashTable[intFHashIndex];

bool exist = false;

if (prev.lex == strALexicalUnit)

{

exist = true;

}

while (prev.next != null)

{

prev = prev.next;

if (prev.lex == strALexicalUnit)

{

exist = true;

}

}

if (!exist)

{

if(item.lex == prev.lex && item.lex == "")

{

item.value = intALexicalCode;

item.lex = strALexicalUnit;

}

else

{

TableItem newItem = new TableItem(intALexicalCode, strALexicalUnit, null);

prev.next = newItem;

}

//MessageBox.Show($"{item.lex}, {item.value}");

return true;

}

return false;

}

public bool DeleteLexicalUnit(string strALexicalUnit)

{

HashIndex(strALexicalUnit);

int indx = intFHashIndex;

if (arrFHashTable[indx] != null)

{

TableItem item = arrFHashTable[indx];

while (item.next != null && item.lex != strALexicalUnit)

{

item = item.next;

}

if (item.lex == strALexicalUnit)

{

if (item.next == null)

{

TableItem prev = arrFHashTable[indx];

while (prev.next != null && prev.next != item)

{

prev = prev.next;

}

prev.next = null;

item.value = 0;

item.lex = "";

}

else

{

TableItem prev = arrFHashTable[indx];

while (prev.next != null && prev.next != item)

{

prev = prev.next;

}

item.value = 0;

item.lex = "";

prev.next = item.next;

}

return true;

}

}

return false;

}

public void GetLexicalUnitList(ref List<string> sList)

{

for (int i = 0; i < tableSize; i++)

{

TableItem item = arrFHashTable[i];

while (item != null)

{

if (item.lex != "")

{

sList.Add($"{item.lex}");

}

item = item.next;

}

}

}

}

}

uLex.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp45

{

public enum TState { Start, Continue, Finish }; //тип состояния

public enum TCharType { Letter, Digit, EndRow, EndText, Space, ReservedSymbol }; // тип символа

public enum TToken { lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma, lxmDollar, lxmMinus, lxmPlus, lxmExclamation, lxmQuestion,

lxmAnd, lxmLeftParenthSqr, lxmRightParenthSqr, lxmtz, lxmDD };

public class CLex //класс лексический анализатор

{

private String[] strFSource; // указатель на массив строк

private String[] strFMessage; // указатель на массив строк

public TCharType enumFSelectionCharType;

public char chrFSelection;

private TState enumFState;

private int intFSourceRowSelection;

private int intFSourceColSelection;

private String strFLexicalUnit;

private TToken enumFToken;

public String[] strPSource { set { strFSource = value; } get { return strFSource; } }

public String[] strPMessage { set { strFMessage = value; } get { return strFMessage; } }

public TState enumPState { set { enumFState = value; } get { return enumFState; } }

public String strPLexicalUnit { set { strFLexicalUnit = value; } get { return strFLexicalUnit; } }

public TToken enumPToken { set { enumFToken = value; } get { return enumFToken; } }

public int intPSourceRowSelection { get { return intFSourceRowSelection; } set { intFSourceRowSelection = value; } }

public int intPSourceColSelection { get { return intFSourceColSelection; } set { intFSourceColSelection = value; } }

public CLex()

{

}

public void GetSymbol() //метод класса лексический анализатор

{

intFSourceColSelection++; // продвигаем номер колонки

if (intFSourceColSelection > strFSource[intFSourceRowSelection].Length - 1)

{

intFSourceRowSelection++;

if (intFSourceRowSelection <= strFSource.Length - 1)

{

intFSourceColSelection = -1;

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndRow;

enumFState = TState.Continue;

}

else

{

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndText;

enumFState = TState.Finish;

}

}

else

{

char[] reservedSymbols = new char[] { '(', ')', '[', ']', '.', ',', '!', '?', '$', '&', '-', '+', ';', ':', '/', '\*' };

chrFSelection = strFSource[intFSourceRowSelection][intFSourceColSelection]; //классификация прочитанной литеры

if (chrFSelection == ' ') enumFSelectionCharType = TCharType.Space;

else if (chrFSelection >= 'a' && chrFSelection <= 'd') enumFSelectionCharType = TCharType.Letter;

else if (chrFSelection == '0' || chrFSelection == '1') enumFSelectionCharType = TCharType.Digit;

else if (reservedSymbols.Contains(chrFSelection)) enumFSelectionCharType = TCharType.ReservedSymbol;

else throw new System.Exception("Cимвол вне алфавита");

enumFState = TState.Continue;

}

}

private void TakeSymbol()

{

char[] c = { chrFSelection };

String s = new string(c);

strPLexicalUnit += s;

GetSymbol();

}

public void NextToken()

{

strPLexicalUnit = "";

if (enumFState == TState.Start)

{

intFSourceRowSelection = 0;

intFSourceColSelection = -1;

GetSymbol();

}

while (enumFSelectionCharType == TCharType.Space || enumFSelectionCharType == TCharType.EndRow)

{

GetSymbol();

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// A | B | | | |

// B | | | CFin | |

// CFin | CFin | CFin | CFin | CFin |

A:

{

if (chrFSelection == 'a')

{

TakeSymbol();

goto B;

}

else throw new Exception("Слово должно начинаться с 'ac'");

}

B:

{

if (chrFSelection == 'c')

{

TakeSymbol();

goto CFin;

}

else throw new Exception("Слово должно начинаться с 'ac'");

}

CFin:

{

if (chrFSelection == 'a' || chrFSelection == 'b' || chrFSelection == 'c' || chrFSelection == 'd')

{

TakeSymbol();

goto CFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

case TCharType.Digit:

{

// 0 1

// A | B | D |

// B | C | |

// C | A | |

// D | E | |

// E |FFin | |

// FFin | G | |

// G | | H |

// H |FFin | |

A:

{

if (chrFSelection == '0')

{

TakeSymbol();

goto B;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

else throw new Exception("Ожидался 0 или 1");

}

B:

{

if (chrFSelection == '0')

{

TakeSymbol();

goto C;

}

else throw new Exception("Ожидался 0");

}

C:

{

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидался 0");

}

D:

{

if (chrFSelection == '0')

{

TakeSymbol();

goto E;

}

else throw new Exception("Ожидался 0");

}

E:

{

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0");

}

FFin:

{

if (chrFSelection == '0')

{

TakeSymbol();

goto G;

}

else if (enumFSelectionCharType != TCharType.Digit)

{

enumFToken = TToken.lxmNumber;

return;

}

else throw new Exception("Ожидался 0");

}

G:

{

if (chrFSelection == '1')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидалась 1");

}

H:

{

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0");

}

}

case TCharType.ReservedSymbol:

{

if (chrFSelection == '/')

{

TakeSymbol();

if (chrFSelection == '/')

{

while (enumFSelectionCharType != TCharType.EndRow)

TakeSymbol();

}

TakeSymbol();

}

if (chrFSelection == '(')

{

enumFToken = TToken.lxmLeftParenth;

TakeSymbol();

return;

}

if (chrFSelection == ')')

{

enumFToken = TToken.lxmRightParenth;

TakeSymbol();

return;

}

if (chrFSelection == '.')

{

enumFToken = TToken.lxmDot;

TakeSymbol();

return;

}

if (chrFSelection == ',')

{

enumFToken = TToken.lxmComma;

TakeSymbol();

return;

}

if (chrFSelection == '[')

{

enumFToken = TToken.lxmLeftParenthSqr;

TakeSymbol();

return;

}

if (chrFSelection == ']')

{

enumFToken = TToken.lxmRightParenthSqr;

TakeSymbol();

return;

}

if (chrFSelection == '!')

{

enumFToken = TToken.lxmExclamation;

TakeSymbol();

return;

}

if (chrFSelection == '?')

{

enumFToken = TToken.lxmQuestion;

TakeSymbol();

return;

}

if (chrFSelection == '&')

{

enumFToken = TToken.lxmAnd;

TakeSymbol();

return;

}

if (chrFSelection == '$')

{

enumFToken = TToken.lxmDollar;

TakeSymbol();

return;

}

if (chrFSelection == '+')

{

enumFToken = TToken.lxmPlus;

TakeSymbol();

return;

}

if (chrFSelection == '-')

{

enumFToken = TToken.lxmMinus;

TakeSymbol();

return;

}

if (chrFSelection == ';')

{

enumFToken = TToken.lxmtz;

TakeSymbol();

return;

}

if (chrFSelection == ':')

{

enumFToken = TToken.lxmDD;

TakeSymbol();

return;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

}

}

}

}

uSyntAnalyzer.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp45

{

public class uSyntAnalyzer

{

private String[] strFSource;

private String[] strFMessage;

public String[] strPSource { set { strFSource = value; } get { return strFSource; } }

public String[] strPMessage { set { strFMessage = value; } get { return strFMessage; } }

public CLex Lex = new CLex();

public TToken firstToken;

public List<string> numTokenList = new List<string>();

public bool noDuplications = true;

public TreeView tree;

public void AddTokenToTree(TToken token, string input, TreeNode parent)

{

if(token == TToken.lxmNumber)

{

for (int i = 0; i < numTokenList.Count; i++)

{

if(input == numTokenList[i])

{

noDuplications = false;

//FindAndSelectNode(parent, "");

}

}

numTokenList.Add(input);

}

TreeNode newNode = new TreeNode(input);

parent.Nodes.Add(newNode);

}

public void S()

{

numTokenList.Clear();

noDuplications = true;

tree.Nodes.Clear();

TreeNode parent = new TreeNode("S");

tree.Nodes.Add(parent);

if (Lex.enumPToken == TToken.lxmLeftParenth) //(A)

{

AddTokenToTree(Lex.enumPToken, Lex.strPLexicalUnit, parent);

A(parent);

if (Lex.enumPToken == TToken.lxmRightParenth)

{

AddTokenToTree(Lex.enumPToken, Lex.strPLexicalUnit, parent);

tree.ExpandAll();

if (!noDuplications)

{

throw new Exception("Найдены повторяющиеся числа!");

}

else

{

Semantic sem = new Semantic(tree);

}

}

else throw new Exception("Ожидалось ) [S]");

}

else throw new Exception("Ожидалось ( [S]");

}

public void A(TreeNode highParent)

{

TreeNode parent = new TreeNode("A");

highParent.Nodes.Add(parent);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmLeftParenth) //(<2>B)

{

AddTokenToTree(Lex.enumPToken, Lex.strPLexicalUnit, parent);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier || Lex.enumPToken == TToken.lxmNumber)

{

AddTokenToTree(Lex.enumPToken, Lex.strPLexicalUnit, parent);

firstToken = (TToken)(1 - (int)Lex.enumPToken);

B(parent);

if (Lex.enumPToken == TToken.lxmRightParenth)

{

AddTokenToTree(Lex.enumPToken, Lex.strPLexicalUnit, parent);

Lex.NextToken();

}

else throw new Exception("Ожидалось ) [A]");

}

else throw new Exception("Ожидался идентификатор или число [A]");

}

else //<1>

{

if(Lex.enumPToken == TToken.lxmIdentifier || Lex.enumPToken == TToken.lxmNumber)

{

firstToken = Lex.enumPToken;

Lex.NextToken();

}

else throw new Exception("Ожидался идентификатор или число или ( [A]");

}

}

public void B(TreeNode highParent)

{

TreeNode parent = new TreeNode("B");

highParent.Nodes.Add(parent);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmLeftParenth) //(C)

{

AddTokenToTree(Lex.enumPToken, Lex.strPLexicalUnit, parent);

Lex.NextToken();

C(parent);

if (Lex.enumPToken == TToken.lxmRightParenth)

{

AddTokenToTree(Lex.enumPToken, Lex.strPLexicalUnit, parent);

Lex.NextToken();

}

else throw new Exception("Ожидалось ) [B]");

}

else throw new Exception("Ожидалось ( [B]");

}

public void C(TreeNode highParent)

{

TreeNode parent = new TreeNode("C");

highParent.Nodes.Add(parent);

if (Lex.enumPToken == firstToken)

{

AddTokenToTree(Lex.enumPToken, Lex.strPLexicalUnit, parent);

D(parent);

}

else throw new Exception("Ожидался идентификатор или число [C]");

}

public void D(TreeNode highParent)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmComma)

{

TreeNode parent = new TreeNode("D");

highParent.Nodes.Add(parent);

AddTokenToTree(Lex.enumPToken, Lex.strPLexicalUnit, parent);

Lex.NextToken();

if (Lex.enumPToken == firstToken)

{

AddTokenToTree(Lex.enumPToken, Lex.strPLexicalUnit, parent);

D(parent);

}

else throw new Exception("Ожидался идентификатор или число [D]");

}

}

}

}

