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

Кандаев К.И.

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.Text;

using System.Windows.Forms;

namespace Lab1

{

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 = tbFMessage.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 = "ReservedSymbol"; 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;

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;

}

}

}

}

**uLex**

using System;

using System.Collections.Generic;

using System.Text;

namespace Lab1

{

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

{

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 (chrFSelection == '(' || chrFSelection == ')' || chrFSelection == ':' || chrFSelection == '-' || chrFSelection == ',' || chrFSelection == '.' || chrFSelection == '!' || chrFSelection == ';' || chrFSelection == '[' || chrFSelection == ']' || 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)

{

intFSourceRowSelection = 0;

intFSourceColSelection = -1;

GetSymbol();

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

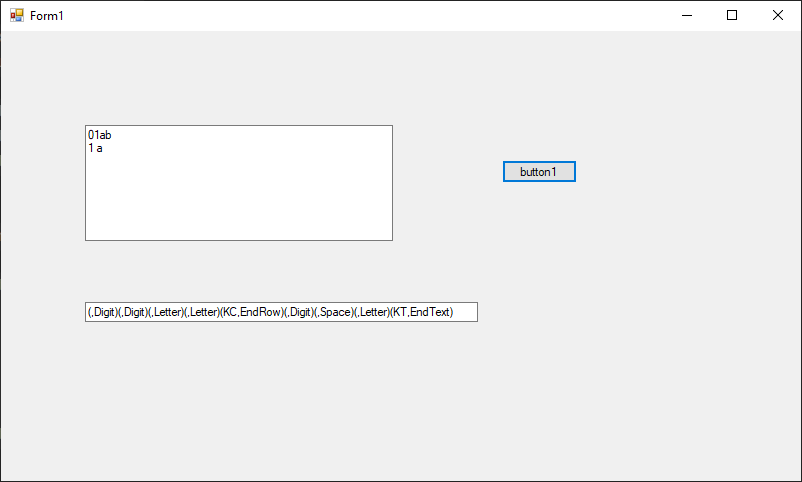
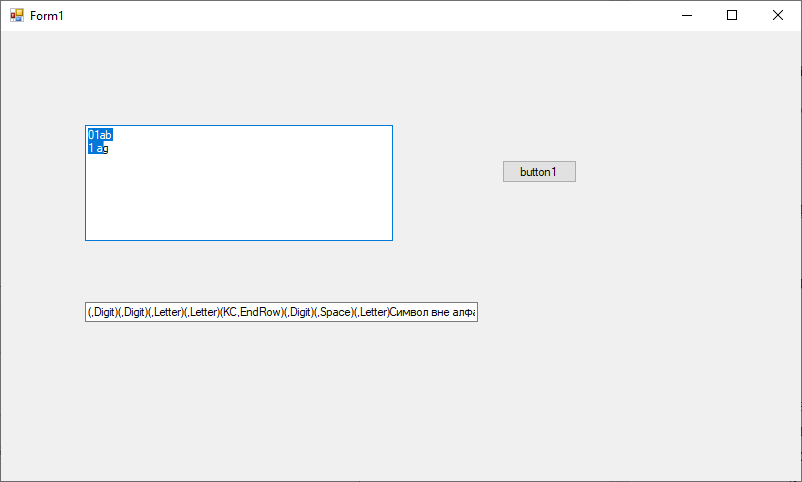
GetSymbol();

}

}

}

}

**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.

|  |  |  |
| --- | --- | --- |
| (001)\*011(000)\* | (a|b|c|d)+ | Первые два символа всегда bd |

**Первое слово:**

0

(001)\*011(000)\*

1

A → 0B | 0D

1

0

B → 0C

1

C → 1A

0

D → 1E

0

1

E → 1F| 1

00

F → 0G

G → 0H

0

H → 0F|0

00

**Недетерминированная матрица:**

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

**Граф:**

**Детерминированная матрица:**

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| A | BD |  |
| BD | C | E |
| C |  | A |
| E |  | FFin |
| FFin | Fin |  |
| G | H |  |
| H | Fin |  |

**Второе слово:**

(a|b|c|d)+

Вторые 2 символа всегда bd

A → bB

B → d | dC

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

d

b

Код программы:

**Ulex:**

using System;

using System.Collections.Generic;

using System.Text;

namespace Lab1

{

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, lxmText };

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 void GetSymbol() //метод класса лексический анализатор

{

intFSourceColSelection++;

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

{

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 (chrFSelection == '(' || chrFSelection == ')' || chrFSelection == ':' || chrFSelection == '-' || chrFSelection == ',' || chrFSelection == '.' || chrFSelection == '!' || chrFSelection == ';' || chrFSelection == '[' || chrFSelection == ']' || 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)

{

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 b c d

// A | | B | | |

// B | | | |CFin|

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

A:

{

if (chrFSelection == 'b')

{

TakeSymbol();

goto B;

}

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

}

B:

{

if (chrFSelection == 'd')

{

TakeSymbol();

goto CFin;

}

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

}

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 | C |

// B | D | |

// C | E | |

// D | A | |

// E | |FFin |

// FFin | | G |

// G | H | |

// 0 1

// A | BD | |

// BD | C | E |

// C | | A |

// E | |FFin |

// FFin | G | |

// G | H | |

// H |FFin | |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto BD;

}

else throw new Exception("Ожидался 0");

BD:

if (chrFSelection == '0')

{

TakeSymbol();

goto C;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto E;

}

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

C:

if (chrFSelection == '1')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидался 1");

E:

if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидалась 1");

FFin:

if (chrFSelection == '0')

{

TakeSymbol();

goto G;

}

else if (enumFSelectionCharType != TCharType.Digit) { enumFToken = TToken.lxmNumber; return; }

else throw new Exception("Ожидалась 0");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 0");

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;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

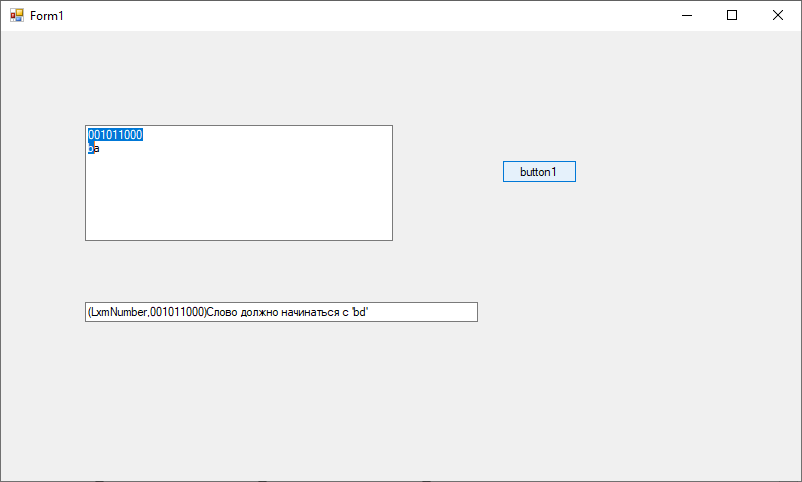
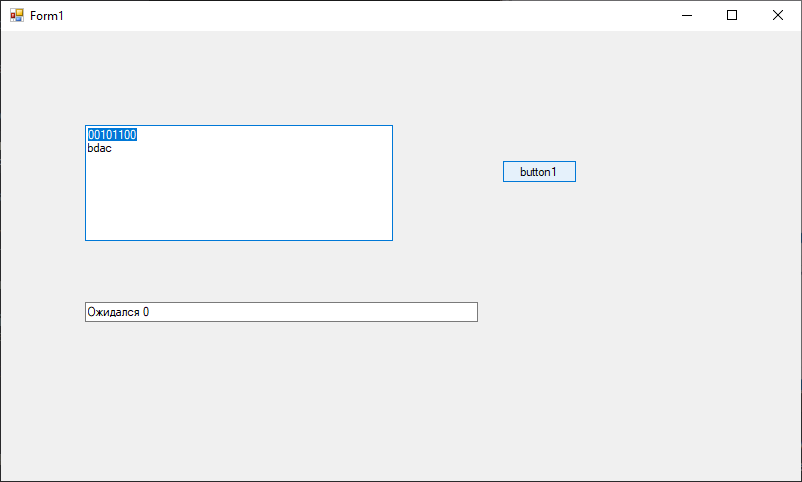
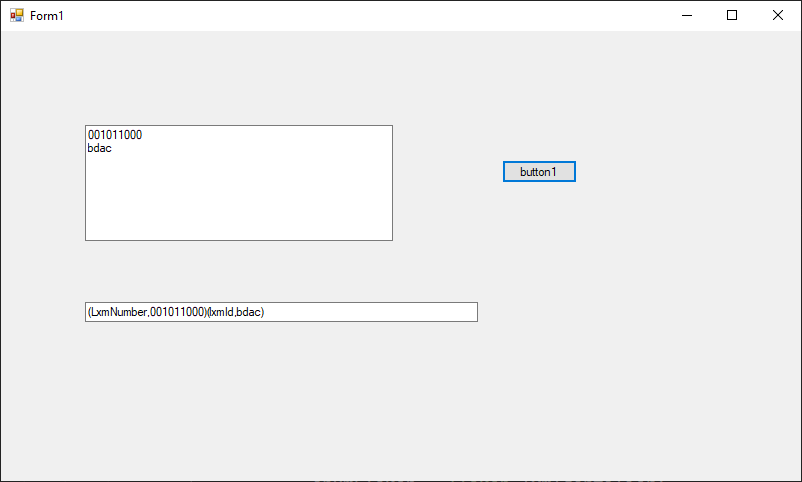
break;

}

}

}

}

}

**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.

**11 Вариант**

**S** 🡪 **A:B**

**A** 🡪 **<2>**

**B** 🡪 **B;C**

**B** 🡪 **C**

**C** 🡪 **C , D**

**C** 🡪 **D**

**D** 🡪 **<1>**

**D** 🡪 **<2>**

Избавление от левой рекурсии:

**S** 🡪 **A:B**

**A** 🡪 **<2>**

**B** 🡪 **C|CE**

**E** 🡪 **;C |;CE**

**C** 🡪 **D|DF**

**F** 🡪 **,D|,DF**

**D** 🡪 **<1>**

**D** 🡪 **<2>**

1: S 🡪 A:B 🡪 <2>:C 🡪 <2>:D 🡪 <2>:<1>

2: S 🡪 A:B 🡪 <2>:C 🡪 <2>:DF 🡪 <2>:<1>,D 🡪 <2>:<1>,<1>

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

**Код программы**

**uSyntAnalyzer.cs**

using System;

using System.Collections.Generic;

using System.Text;

using nsLex;

namespace nsLex

{

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 void S()

{

A();

if (Lex.enumPToken == TToken.lxmddt)

{

B();

}

else throw new Exception("Ожидалось :");

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

}

public void A()

{

if (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

}

else throw new Exception("Ожидался идентификатор");

}

public void B()

{

C();

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmdtcomma)

{

E();

}

}

public void E()

{

if (Lex.enumPToken == TToken.lxmdtcomma)

{

C();

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmdtcomma)

{

E();

}

}

}

public void C()

{

D();

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmComma)

{

F();

}

}

public void F()

{

if (Lex.enumPToken == TToken.lxmComma)

{

D();

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmComma)

{

F();

}

}

}

public void D()

{

Lex.NextToken();

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

{

Lex.NextToken();

}

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

}

}

}

**Form1.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

namespace nsLex

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

tbFSource.AppendText("bdac:001011000" + "\r\n");

int n = tbFSource.Lines.Length;

}

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

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

Lex.intPSourceColSelection = -1;

Lex.intPSourceRowSelection = 0;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

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

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

String s = "";

String s1 = "";

switch (Lex.enumPToken)

{

case TToken.lxmNumber: { s = "LxmNumber"; s1 = Lex.strPLexicalUnit; break; }

case TToken.lxmIdentifier: { s = "lxmId"; s1 = Lex.strPLexicalUnit; break; }

}

String m = "(" + s + "," + s1 + ")"; //литера и ее тип

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

}

}

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;

}

}

private void button2\_Click(object sender, EventArgs e)

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.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;

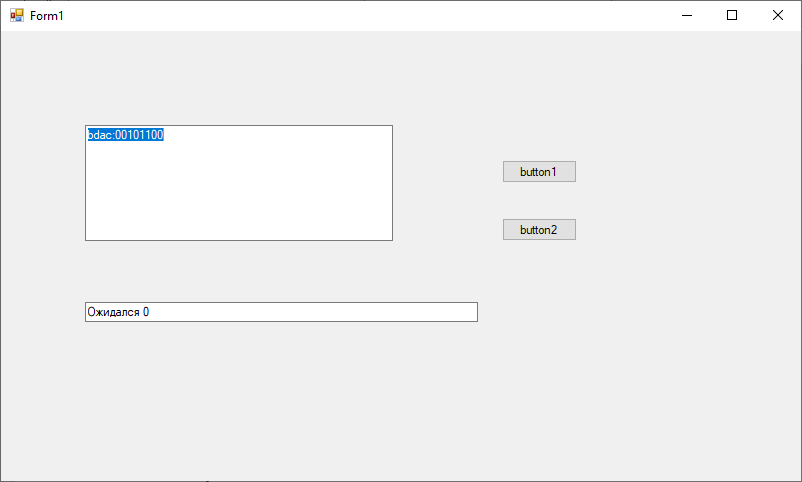
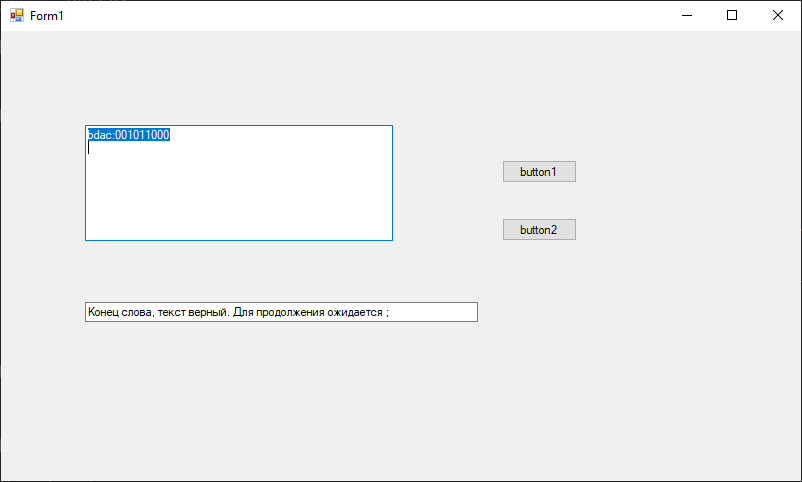
tbFSource.SelectionLength = n;

}

}

}

}



**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.Drawing;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

public partial class Form1 : Form

{

Dictionary<int, List<string>> hashTableIdentifier = new Dictionary<int, List<string>>();

Dictionary<int, List<string>> hashTableDigital = new Dictionary<int, List<string>>();

Dictionary<int, List<string>> hashTableRezerv = new Dictionary<int, List<string>>();

public MyHashFunction hashFunction = new MyHashFunction();

public CHashTableList htl = new CHashTableList(2);

public Form1()

{

InitializeComponent();

tbFSource.AppendText("bdac:bd;001011000,011" + "\r\n");

int n = tbFSource.Lines.Length;

}

private void Form1\_Load(object sender, EventArgs e)

{

}

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

{

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

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

//htl.

htl.TableToStringList(0, listTable);

foreach (var entry in hashTableIdentifier)

{

listBox1.Items.Add(string.Join(", ", entry.Value));

}

listTable.Clear();

htl.TableToStringList(1, listTable);

foreach (var entry in hashTableDigital)

{

listBox2.Items.Add(string.Join(", ", entry.Value));

}

listTable.Clear();

foreach (var entry in hashTableRezerv)

{

listBox3.Items.Add(string.Join(", ", entry.Value));

}

listTable.Clear();

}

private void resetBtnsColor()

{

searchBtn.ResetBackColor();

searchBtn.UseVisualStyleBackColor = true;

addBtn.ResetBackColor();

addBtn.UseVisualStyleBackColor = true;

deleteBtn.ResetBackColor();

deleteBtn.UseVisualStyleBackColor = true;

changeBtn.ResetBackColor();

changeBtn.UseVisualStyleBackColor = true;

}

private void btnFStart\_Click\_1(object sender, EventArgs e)

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.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;

}

}

private void btnFRecord\_Click\_1(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:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmddt):

{

hashFunction.AddWord(hashTableRezerv, ":");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmdtcomma):

{

hashFunction.AddWord(hashTableRezerv, ";");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmComma):

{

hashFunction.AddWord(hashTableRezerv, ",");

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

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

{

TablesToMemo(this, 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;

}

}

private void searchBtn\_Click\_1(object sender, EventArgs e)

{

if (hashFunction.SearchWord(hashTableIdentifier, textBox1.Text.ToString()) == 1)

{

searchBtn.BackColor = Color.Green;

}

else

{

searchBtn.BackColor = Color.Red;

}

}

private void addBtn\_Click\_1(object sender, EventArgs e)

{

hashFunction.AddWord(hashTableIdentifier, textBox1.Text.ToString());

addBtn.BackColor = Color.Green;

}

private void deleteBtn\_Click\_1(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox1.SelectedItem.ToString()))

{

deleteBtn.BackColor = Color.Green;

}

else

{

deleteBtn.BackColor = Color.Red;

}

}

private void changeBtn\_Click\_1(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox1.SelectedItem.ToString()))

{

hashFunction.AddWord(hashTableIdentifier, textBox1.Text.ToString());

changeBtn.BackColor = Color.Green;

}

else

{

changeBtn.BackColor = Color.Red;

}

}

private void reloadBtn\_Click\_1(object sender, EventArgs e)

{

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

resetBtnsColor();

foreach (var entry in hashTableIdentifier)

{

listBox1.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableDigital)

{

listBox2.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableRezerv)

{

listBox3.Items.Add(string.Join(", ", entry.Value));

}

}

}

}

**HashTableList.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

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;

}

}

}

**Thash.Tables.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

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;

}

}

}

}

}

**MyHashFunction.cs**

using System;

using System.Collections.Generic;

namespace WindowsFormsApp4

{

public class MyHashFunction

{

public int HashFunction(string word)

{

int hashValue = 0;

foreach (char c in word)

{

hashValue += (int)c;

}

return hashValue;

}

public void AddWord(Dictionary<int, List<string>> hashTable, string word)

{

int hashValue = HashFunction(word);

if (!hashTable.ContainsKey(hashValue))

{

hashTable[hashValue] = new List<string>();

}

else

{

if (hashTable[hashValue].Contains(word))

{

Console.WriteLine("Слово уже существует: " + word);

return;

}

}

hashTable[hashValue].Add(word);

}

public int SearchWord(Dictionary<int, List<string>> hashTable, string word)

{

int hashValue = HashFunction(word);

if (hashTable.ContainsKey(hashValue))

{

return 1;

}

else

{

return 0;

}

}

public bool RemoveWord(Dictionary<int, List<string>> hashTable, string word)

{

int hashValue = HashFunction(word);

if (hashTable.ContainsKey(hashValue))

{

List<string> words = hashTable[hashValue];

if (words.Contains(word))

{

words.Remove(word);

if (words.Count == 0)

{

hashTable.Remove(hashValue);

}

return true;

}

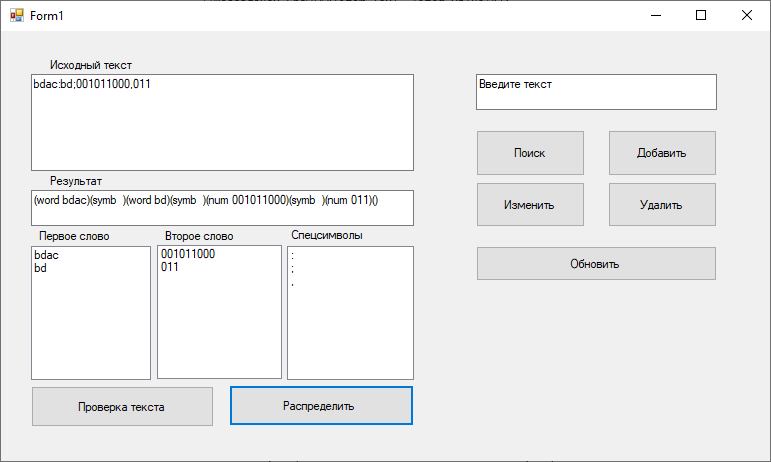
}

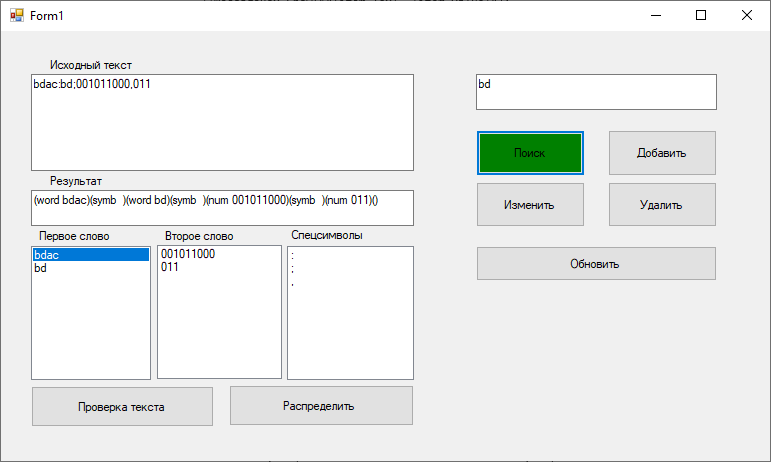
return false;

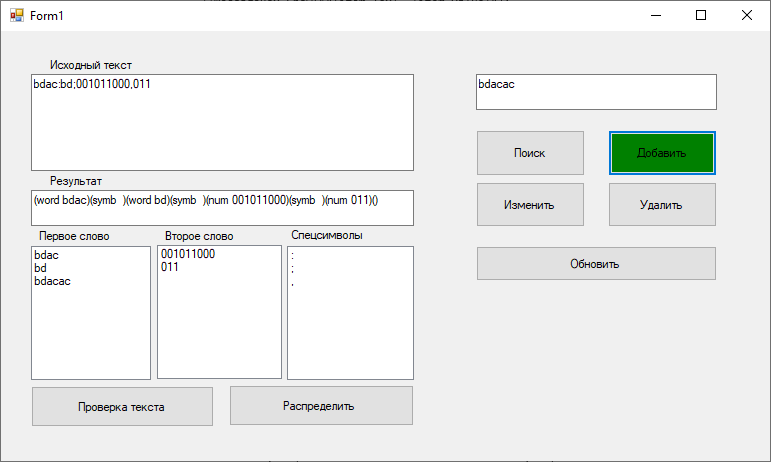
}

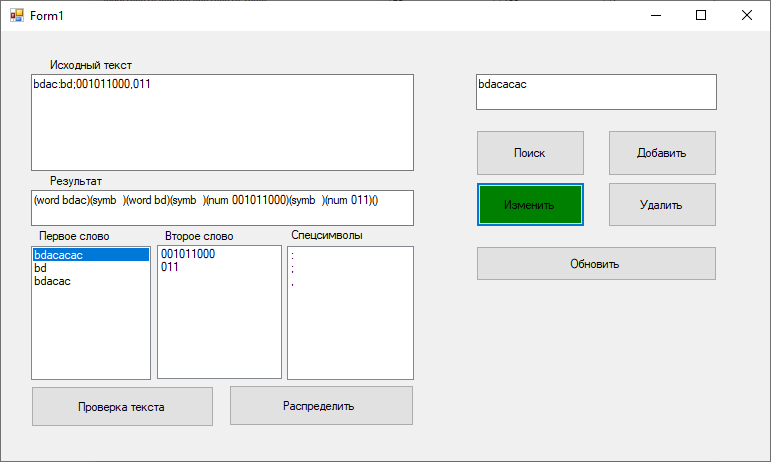
}

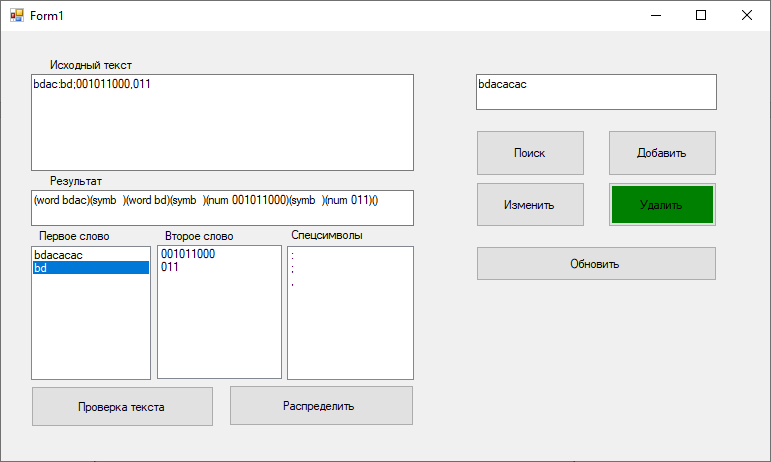
}











**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.Collections.Generic;

using System.Drawing;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

public partial class Form1 : Form

{

Dictionary<int, List<string>> hashTableIdentifier = new Dictionary<int, List<string>>();

Dictionary<int, List<string>> hashTableDigital = new Dictionary<int, List<string>>();

Dictionary<int, List<string>> hashTableRezerv = new Dictionary<int, List<string>>();

public MyHashFunction hashFunction = new MyHashFunction();

public CHashTableList htl = new CHashTableList(2);

public Form1()

{

InitializeComponent();

tbFSource.AppendText("bdac:bd;001011000,011" + "\r\n");

int n = tbFSource.Lines.Length;

}

private void Form1\_Load(object sender, EventArgs e)

{

}

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

{

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

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

//htl.

htl.TableToStringList(0, listTable);

foreach (var entry in hashTableIdentifier)

{

listBox1.Items.Add(string.Join(", ", entry.Value));

}

listTable.Clear();

htl.TableToStringList(1, listTable);

foreach (var entry in hashTableDigital)

{

listBox2.Items.Add(string.Join(", ", entry.Value));

}

listTable.Clear();

foreach (var entry in hashTableRezerv)

{

listBox3.Items.Add(string.Join(", ", entry.Value));

}

listTable.Clear();

}

private void resetBtnsColor()

{

searchBtn.ResetBackColor();

searchBtn.UseVisualStyleBackColor = true;

addBtn.ResetBackColor();

addBtn.UseVisualStyleBackColor = true;

deleteBtn.ResetBackColor();

deleteBtn.UseVisualStyleBackColor = true;

changeBtn.ResetBackColor();

changeBtn.UseVisualStyleBackColor = true;

}

private void btnFStart\_Click\_1(object sender, EventArgs e)

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

treeView1.Nodes.Clear();

Synt.Tree = treeView1;

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.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;

}

}

private void btnFRecord\_Click\_1(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:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmddt):

{

hashFunction.AddWord(hashTableRezerv, ":");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmdtcomma):

{

hashFunction.AddWord(hashTableRezerv, ";");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmComma):

{

hashFunction.AddWord(hashTableRezerv, ",");

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

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

{

TablesToMemo(this, 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;

}

}

private void searchBtn\_Click\_1(object sender, EventArgs e)

{

if (hashFunction.SearchWord(hashTableIdentifier, textBox1.Text.ToString()) == 1)

{

searchBtn.BackColor = Color.Green;

}

else

{

searchBtn.BackColor = Color.Red;

}

}

private void addBtn\_Click\_1(object sender, EventArgs e)

{

hashFunction.AddWord(hashTableIdentifier, textBox1.Text.ToString());

addBtn.BackColor = Color.Green;

}

private void deleteBtn\_Click\_1(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox1.SelectedItem.ToString()))

{

deleteBtn.BackColor = Color.Green;

}

else

{

deleteBtn.BackColor = Color.Red;

}

}

private void changeBtn\_Click\_1(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox1.SelectedItem.ToString()))

{

hashFunction.AddWord(hashTableIdentifier, textBox1.Text.ToString());

changeBtn.BackColor = Color.Green;

}

else

{

changeBtn.BackColor = Color.Red;

}

}

private void reloadBtn\_Click\_1(object sender, EventArgs e)

{

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

resetBtnsColor();

foreach (var entry in hashTableIdentifier)

{

listBox1.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableDigital)

{

listBox2.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableRezerv)

{

listBox3.Items.Add(string.Join(", ", entry.Value));

}

}

}

}

**uSyntAnalyzer.cs**

using System;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

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 TreeView Tree;

public void S()

{

TreeNode Temp = new TreeNode("S");

Tree.Nodes.Add(Temp);

A(Temp);

if (Lex.enumPToken == TToken.lxmddt)

{

Temp.Nodes.Add(new TreeNode(":"));

B(Temp);

}

else throw new Exception("Ожидалось :");

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

}

public void A(TreeNode Par)

{

TreeNode Temp = new TreeNode("A");

Par.Nodes.Add(Temp);

if (Lex.enumPToken == TToken.lxmIdentifier)

{

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

Lex.NextToken();

}

else throw new Exception("Ожидался идентификатор");

}

public void B(TreeNode Par)

{

TreeNode Temp = new TreeNode("B");

Par.Nodes.Add(Temp);

C(Temp);

//Lex.NextToken();

if (Lex.enumPToken == TToken.lxmdtcomma)

{

//Temp.Nodes.Add(new TreeNode(";"));

E(Temp);

}

}

public void E(TreeNode Par)

{

TreeNode Temp = new TreeNode("E");

Par.Nodes.Add(Temp);

if (Lex.enumPToken == TToken.lxmdtcomma)

{

Temp.Nodes.Add(new TreeNode(";"));

C(Temp);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmdtcomma)

{

//Temp.Nodes.Add(new TreeNode(";"));

E(Temp);

}

}

else throw new Exception("Ожидался ;");

}

public void C(TreeNode Par)

{

TreeNode Temp = new TreeNode("C");

Par.Nodes.Add(Temp);

D(Temp);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmComma)

{

//Temp.Nodes.Add(new TreeNode(","));

F(Temp);

}

}

public void F(TreeNode Par)

{

TreeNode Temp = new TreeNode("F");

Par.Nodes.Add(Temp);

if (Lex.enumPToken == TToken.lxmComma)

{

Temp.Nodes.Add(new TreeNode(","));

D(Temp);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmComma)

{

//Temp.Nodes.Add(new TreeNode(","));

F(Temp);

}

}

}

public void D(TreeNode Par)

{

TreeNode Temp = new TreeNode("D");

Par.Nodes.Add(Temp);

Lex.NextToken();

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

{

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

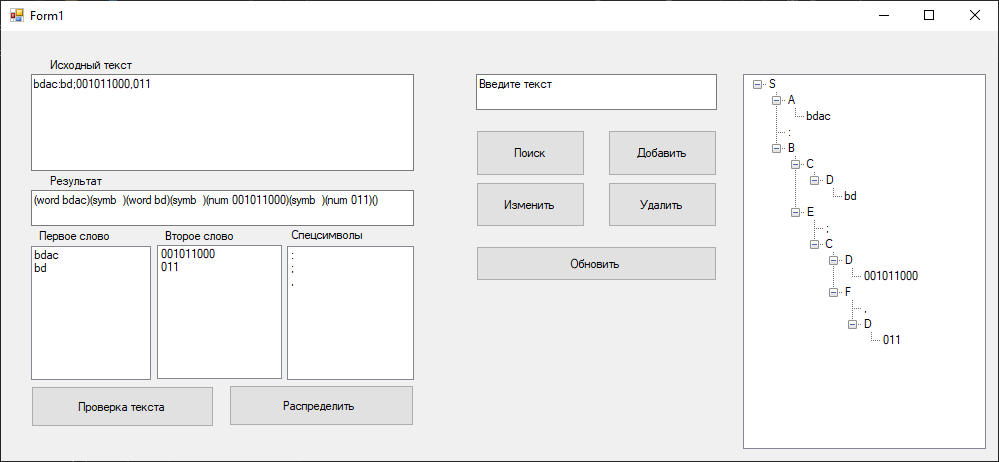
}

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

}

}

}



**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.Drawing;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

public partial class Form1 : Form

{

Dictionary<int, List<string>> hashTableIdentifier = new Dictionary<int, List<string>>();

Dictionary<int, List<string>> hashTableDigital = new Dictionary<int, List<string>>();

Dictionary<int, List<string>> hashTableRezerv = new Dictionary<int, List<string>>();

public MyHashFunction hashFunction = new MyHashFunction();

public CHashTableList htl = new CHashTableList(2);

public Form1()

{

InitializeComponent();

tbFSource.AppendText("bdac:bd;001011000,011" + "\r\n");

int n = tbFSource.Lines.Length;

}

private void Form1\_Load(object sender, EventArgs e)

{

}

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

{

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

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

//htl.

htl.TableToStringList(0, listTable);

foreach (var entry in hashTableIdentifier)

{

listBox1.Items.Add(string.Join(", ", entry.Value));

}

listTable.Clear();

htl.TableToStringList(1, listTable);

foreach (var entry in hashTableDigital)

{

listBox2.Items.Add(string.Join(", ", entry.Value));

}

listTable.Clear();

foreach (var entry in hashTableRezerv)

{

listBox3.Items.Add(string.Join(", ", entry.Value));

}

listTable.Clear();

}

private void resetBtnsColor()

{

searchBtn.ResetBackColor();

searchBtn.UseVisualStyleBackColor = true;

addBtn.ResetBackColor();

addBtn.UseVisualStyleBackColor = true;

deleteBtn.ResetBackColor();

deleteBtn.UseVisualStyleBackColor = true;

changeBtn.ResetBackColor();

changeBtn.UseVisualStyleBackColor = true;

}

private void btnFStart\_Click\_1(object sender, EventArgs e)

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

treeView1.Nodes.Clear();

Synt.Tree = treeView1;

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.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;

}

}

private void btnFRecord\_Click\_1(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(treeView1, treeView2);

Generator.RefactorTree(treeView2);

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:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmddt):

{

hashFunction.AddWord(hashTableRezerv, ":");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmdtcomma):

{

hashFunction.AddWord(hashTableRezerv, ";");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmComma):

{

hashFunction.AddWord(hashTableRezerv, ",");

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

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

{

TablesToMemo(this, 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;

}

}

private void searchBtn\_Click\_1(object sender, EventArgs e)

{

if (hashFunction.SearchWord(hashTableIdentifier, textBox1.Text.ToString()) == 1)

{

searchBtn.BackColor = Color.Green;

}

else

{

searchBtn.BackColor = Color.Red;

}

}

private void addBtn\_Click\_1(object sender, EventArgs e)

{

hashFunction.AddWord(hashTableIdentifier, textBox1.Text.ToString());

addBtn.BackColor = Color.Green;

}

private void deleteBtn\_Click\_1(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox1.SelectedItem.ToString()))

{

deleteBtn.BackColor = Color.Green;

}

else

{

deleteBtn.BackColor = Color.Red;

}

}

private void changeBtn\_Click\_1(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox1.SelectedItem.ToString()))

{

hashFunction.AddWord(hashTableIdentifier, textBox1.Text.ToString());

changeBtn.BackColor = Color.Green;

}

else

{

changeBtn.BackColor = Color.Red;

}

}

private void reloadBtn\_Click\_1(object sender, EventArgs e)

{

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

resetBtnsColor();

foreach (var entry in hashTableIdentifier)

{

listBox1.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableDigital)

{

listBox2.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableRezerv)

{

listBox3.Items.Add(string.Join(", ", entry.Value));

}

}

}

}

**Generator.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

internal 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);

ReNode(null, node, ref newNode);

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);

}

}

tree.ExpandAll();

}

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

{

bool skipToRoot = false;

foreach (TreeNode child in node.Nodes)

{

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);

}

}

}

public static void RefactorTree(TreeView tree)

{

TreeNode nodeTemp = null;

foreach (TreeNode node in tree.Nodes)

{

nodeTemp = RefactorTreeNodes(tree, node);

if (nodeTemp != null)

{

node.Nodes.Remove(nodeTemp);

tree.Nodes.Add(nodeTemp);

}

}

}

public static TreeNode RefactorTreeNodes(TreeView tree, TreeNode parent)

{

TreeNode nodeToMove = null;

TreeNode nodeTemp = null;

foreach (TreeNode node in parent.Nodes)

{

nodeTemp = RefactorTreeNodes(tree, node);

if (nodeTemp != null)

{

if (parent.Text == "S" && nodeTemp.Text == ":")

{

nodeToMove = nodeTemp;

}

else

if (parent.Text == "B" && nodeTemp.Text == ";")

{

nodeToMove = nodeTemp;

}

else

if (parent.Text == "C" && nodeTemp.Text == ",")

{

nodeToMove = nodeTemp;

}

else

{

node.Nodes.Remove(nodeTemp);

parent.Nodes.Add(nodeTemp);

}

}

if (parent.Text == "S" && node.Text == ":")

{

nodeToMove = node;

}

if (parent.Text == "E" && node.Text == ";")

{

nodeToMove = node;

}

if (parent.Text == "F" && node.Text == ",")

{

nodeToMove = node;

}

}

return nodeToMove;

//if (nodeToMove != null)

//{

// parent.Nodes.Remove(nodeToMove);

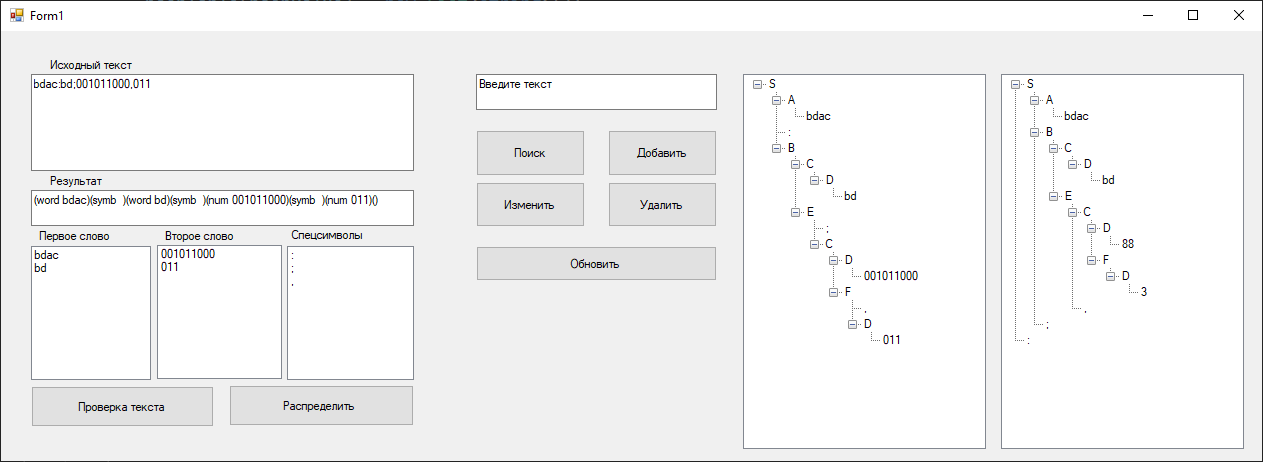
// tree.Nodes.Add(nodeToMove);

//}

}

}

}



**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 11:** Все числа должны быть разными.

Код программы:  
**uSyntAnalyzer.cs**

using System;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

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 TreeView Tree;

public void S()

{

TreeNode Temp = new TreeNode("S");

Tree.Nodes.Add(Temp);

A(Temp);

if (Lex.enumPToken == TToken.lxmddt)

{

Temp.Nodes.Add(new TreeNode(":"));

B(Temp);

}

else throw new Exception("Ожидалось :");

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

}

public void A(TreeNode Par)

{

TreeNode Temp = new TreeNode("A");

Par.Nodes.Add(Temp);

if (Lex.enumPToken == TToken.lxmIdentifier)

{

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

Lex.NextToken();

}

else throw new Exception("Ожидался идентификатор");

}

public void B(TreeNode Par)

{

TreeNode Temp = new TreeNode("B");

Par.Nodes.Add(Temp);

C(Temp);

//Lex.NextToken();

if (Lex.enumPToken == TToken.lxmdtcomma)

{

//Temp.Nodes.Add(new TreeNode(";"));

E(Temp);

}

}

public void E(TreeNode Par)

{

TreeNode Temp = new TreeNode("E");

Par.Nodes.Add(Temp);

if (Lex.enumPToken == TToken.lxmdtcomma)

{

Temp.Nodes.Add(new TreeNode(";"));

C(Temp);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmdtcomma)

{

//Temp.Nodes.Add(new TreeNode(";"));

E(Temp);

}

}

else throw new Exception("Ожидался ;");

}

public void C(TreeNode Par)

{

TreeNode Temp = new TreeNode("C");

Par.Nodes.Add(Temp);

D(Temp);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmComma)

{

//Temp.Nodes.Add(new TreeNode(","));

F(Temp);

}

}

public void F(TreeNode Par)

{

TreeNode Temp = new TreeNode("F");

Par.Nodes.Add(Temp);

if (Lex.enumPToken == TToken.lxmComma)

{

Temp.Nodes.Add(new TreeNode(","));

D(Temp);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmComma)

{

//Temp.Nodes.Add(new TreeNode(","));

F(Temp);

}

}

}

public void D(TreeNode Par)

{

TreeNode Temp = new TreeNode("D");

Par.Nodes.Add(Temp);

Lex.NextToken();

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

{

if (!IsNumberUnique(Lex.strPLexicalUnit) && Lex.enumPToken == TToken.lxmNumber)

{

throw new Exception("Число " + Lex.strPLexicalUnit + " повторилось!");

}

else

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

}

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

}

private bool IsNumberUnique(string Num)

{

foreach (TreeNode node in Tree.Nodes)

{

if (!IsNumberUniqueInNode(Num, node))

{

return false;

}

}

return true;

}

private bool IsNumberUniqueInNode(string Num, TreeNode node)

{

if (node.Text == Num)

{

return false;

}

foreach (TreeNode child in node.Nodes)

{

if (!IsNumberUniqueInNode(Num, child))

{

return false;

}

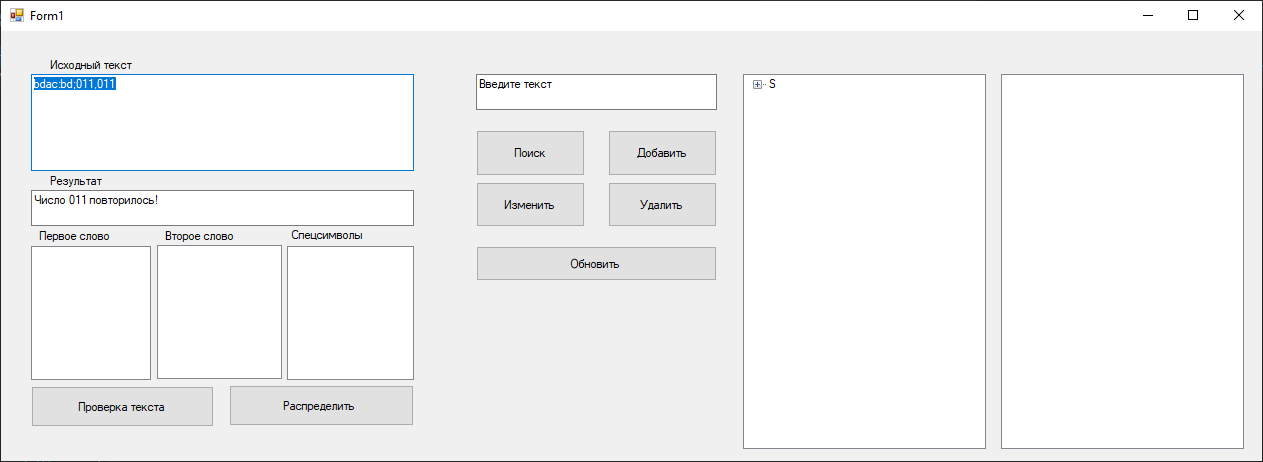
}

return true;

}

}

}



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

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

**Вариант:** После идентификатора bd не может быть число 001011000

Код программы:

**Form1.cs**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

public partial class Form1 : Form

{

Dictionary<int, List<string>> hashTableIdentifier = new Dictionary<int, List<string>>();

Dictionary<int, List<string>> hashTableDigital = new Dictionary<int, List<string>>();

Dictionary<int, List<string>> hashTableRezerv = new Dictionary<int, List<string>>();

public MyHashFunction hashFunction = new MyHashFunction();

public CHashTableList htl = new CHashTableList(2);

public Form1()

{

InitializeComponent();

tbFSource.AppendText("bdac:bd;001011000,011" + "\r\n");

int n = tbFSource.Lines.Length;

}

private void Form1\_Load(object sender, EventArgs e)

{

}

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

{

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

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

//htl.

htl.TableToStringList(0, listTable);

foreach (var entry in hashTableIdentifier)

{

listBox1.Items.Add(string.Join(", ", entry.Value));

}

listTable.Clear();

htl.TableToStringList(1, listTable);

foreach (var entry in hashTableDigital)

{

listBox2.Items.Add(string.Join(", ", entry.Value));

}

listTable.Clear();

foreach (var entry in hashTableRezerv)

{

listBox3.Items.Add(string.Join(", ", entry.Value));

}

listTable.Clear();

}

private void resetBtnsColor()

{

searchBtn.ResetBackColor();

searchBtn.UseVisualStyleBackColor = true;

addBtn.ResetBackColor();

addBtn.UseVisualStyleBackColor = true;

deleteBtn.ResetBackColor();

deleteBtn.UseVisualStyleBackColor = true;

changeBtn.ResetBackColor();

changeBtn.UseVisualStyleBackColor = true;

}

private void btnFStart\_Click\_1(object sender, EventArgs e)

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

treeView1.Nodes.Clear();

Synt.Tree = treeView1;

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.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;

}

}

private void btnFRecord\_Click\_1(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(treeView1, treeView2);

Generator.RefactorTree(treeView2);

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:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmddt):

{

hashFunction.AddWord(hashTableRezerv, ":");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmdtcomma):

{

hashFunction.AddWord(hashTableRezerv, ";");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmComma):

{

hashFunction.AddWord(hashTableRezerv, ",");

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

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

{

TablesToMemo(this, 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;

}

}

private void searchBtn\_Click\_1(object sender, EventArgs e)

{

if (hashFunction.SearchWord(hashTableIdentifier, textBox1.Text.ToString()) == 1)

{

searchBtn.BackColor = Color.Green;

}

else

{

searchBtn.BackColor = Color.Red;

}

}

private void addBtn\_Click\_1(object sender, EventArgs e)

{

hashFunction.AddWord(hashTableIdentifier, textBox1.Text.ToString());

addBtn.BackColor = Color.Green;

}

private void deleteBtn\_Click\_1(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox1.SelectedItem.ToString()))

{

deleteBtn.BackColor = Color.Green;

}

else

{

deleteBtn.BackColor = Color.Red;

}

}

private void changeBtn\_Click\_1(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox1.SelectedItem.ToString()))

{

hashFunction.AddWord(hashTableIdentifier, textBox1.Text.ToString());

changeBtn.BackColor = Color.Green;

}

else

{

changeBtn.BackColor = Color.Red;

}

}

private void reloadBtn\_Click\_1(object sender, EventArgs e)

{

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

resetBtnsColor();

foreach (var entry in hashTableIdentifier)

{

listBox1.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableDigital)

{

listBox2.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableRezerv)

{

listBox3.Items.Add(string.Join(", ", entry.Value));

}

}

}

}

**HashTableList.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

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 WindowsFormsApp4

{

internal 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);

ReNode(null, node, ref newNode);

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);

}

}

tree.ExpandAll();

}

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

{

bool skipToRoot = false;

foreach (TreeNode child in node.Nodes)

{

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);

}

}

}

public static void RefactorTree(TreeView tree)

{

TreeNode nodeTemp = null;

foreach (TreeNode node in tree.Nodes)

{

nodeTemp = RefactorTreeNodes(tree, node);

if (nodeTemp != null)

{

node.Nodes.Remove(nodeTemp);

tree.Nodes.Add(nodeTemp);

}

}

}

public static TreeNode RefactorTreeNodes(TreeView tree, TreeNode parent)

{

TreeNode nodeToMove = null;

TreeNode nodeTemp = null;

foreach (TreeNode node in parent.Nodes)

{

nodeTemp = RefactorTreeNodes(tree, node);

if (nodeTemp != null)

{

if (parent.Text == "S" && nodeTemp.Text == ":")

{

nodeToMove = nodeTemp;

}

else

if (parent.Text == "B" && nodeTemp.Text == ";")

{

nodeToMove = nodeTemp;

}

else

if (parent.Text == "C" && nodeTemp.Text == ",")

{

nodeToMove = nodeTemp;

}

else

{

node.Nodes.Remove(nodeTemp);

parent.Nodes.Add(nodeTemp);

}

}

if (parent.Text == "S" && node.Text == ":")

{

nodeToMove = node;

}

if (parent.Text == "E" && node.Text == ";")

{

nodeToMove = node;

}

if (parent.Text == "F" && node.Text == ",")

{

nodeToMove = node;

}

}

return nodeToMove;

}

}

}

**uSemantAnalyzer.cs**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

class uSemantAnalyzer

{

public int i = 0;

public string strIndentifier;

public string strDigital;

private TreeView tree;

public uSemantAnalyzer()

{

}

public uSemantAnalyzer(TreeView treeView)

{

tree = treeView;

TreeController(tree);

}

public void TreeController(TreeView tree)

{

foreach (TreeNode node in tree.Nodes)

{

TreeController(node);

}

}

public void TreeController(TreeNode node)

{

if (node.Text == "B")

{

if (node.Nodes.Count == 2)

{

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

strDigital = node.Nodes[1].Nodes[1].Nodes[0].Nodes[0].Text.ToString();

Check(strIndentifier, strDigital, node);

}

}

foreach (TreeNode childNode in node.Nodes)

{

TreeController(childNode);

}

}

private void Check(string ident, string digit, TreeNode node)

{

if (ident == "bd" && digit == "001011000")

{

tree.SelectedNode = node;

tree.SelectedNode.BackColor = Color.Red;

throw new Exception("Условие не выполнено");

}

}

}

}

**Hash.Tables.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

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;

namespace WindowsFormsApp4

{

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, lxmText, lxmdt, lxmddt, lxmmns, lxmvsclsgn, lxmdtcomma, lxmls, lxmrs, lxmdlr };

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 void GetSymbol() //метод класса лексический анализатор

{

intFSourceColSelection++;

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

{

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 (chrFSelection == '(' || chrFSelection == ')' || chrFSelection == ':' || chrFSelection == '-' || chrFSelection == ',' || chrFSelection == '.' || chrFSelection == '!' || chrFSelection == ';' || chrFSelection == '[' || chrFSelection == ']' || 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);

strFLexicalUnit += s;

GetSymbol();

}

public void NextToken()

{

strFLexicalUnit = "";

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 b c d

// A | | B | | |

// B | | | |CFin|

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

A:

{

if (chrFSelection == 'b')

{

TakeSymbol();

goto B;

}

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

}

B:

{

if (chrFSelection == 'd')

{

TakeSymbol();

goto CFin;

}

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

}

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 | C |

// B | D | |

// C | E | |

// D | A | |

// E | |FFin |

// FFin | | G |

// G | H | |

// 0 1

// A | BD | |

// BD | C | E |

// C | | A |

// E | |FFin |

// FFin | G | |

// G | H | |

// H |FFin | |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto BD;

}

else throw new Exception("Ожидался 0");

BD:

if (chrFSelection == '0')

{

TakeSymbol();

goto C;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto E;

}

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

C:

if (chrFSelection == '1')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидался 1");

E:

if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидалась 1");

FFin:

if (chrFSelection == '0')

{

TakeSymbol();

goto G;

}

else if (enumFSelectionCharType != TCharType.Digit) { enumFToken = TToken.lxmNumber; return; }

else throw new Exception("Ожидалась 0");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 0");

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.lxmddt;

GetSymbol();

return;

}

if (chrFSelection == '-')

{

enumFToken = TToken.lxmmns;

GetSymbol();

return;

}

if (chrFSelection == ',')

{

enumFToken = TToken.lxmComma;

GetSymbol();

return;

}

if (chrFSelection == '.')

{

enumFToken = TToken.lxmdt;

GetSymbol();

return;

}

if (chrFSelection == '!')

{

enumFToken = TToken.lxmvsclsgn;

GetSymbol();

return;

}

if (chrFSelection == ';')

{

enumFToken = TToken.lxmdtcomma;

GetSymbol();

return;

}

if (chrFSelection == '[')

{

enumFToken = TToken.lxmls;

GetSymbol();

return;

}

if (chrFSelection == ']')

{

enumFToken = TToken.lxmrs;

GetSymbol();

return;

}

if (chrFSelection == '$')

{

enumFToken = TToken.lxmdlr;

GetSymbol();

return;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

}

}

}

}

**uSyntAnalyzer.cs**

using System;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

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 TreeView Tree;

public void S()

{

TreeNode Temp = new TreeNode("S");

Tree.Nodes.Add(Temp);

A(Temp);

if (Lex.enumPToken == TToken.lxmddt)

{

Temp.Nodes.Add(new TreeNode(":"));

B(Temp);

}

else throw new Exception("Ожидалось :");

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

}

public void A(TreeNode Par)

{

TreeNode Temp = new TreeNode("A");

Par.Nodes.Add(Temp);

if (Lex.enumPToken == TToken.lxmIdentifier)

{

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

Lex.NextToken();

}

else throw new Exception("Ожидался идентификатор");

}

public void B(TreeNode Par)

{

TreeNode Temp = new TreeNode("B");

Par.Nodes.Add(Temp);

C(Temp);

//Lex.NextToken();

if (Lex.enumPToken == TToken.lxmdtcomma)

{

//Temp.Nodes.Add(new TreeNode(";"));

E(Temp);

}

}

public void E(TreeNode Par)

{

TreeNode Temp = new TreeNode("E");

Par.Nodes.Add(Temp);

if (Lex.enumPToken == TToken.lxmdtcomma)

{

Temp.Nodes.Add(new TreeNode(";"));

C(Temp);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmdtcomma)

{

//Temp.Nodes.Add(new TreeNode(";"));

E(Temp);

}

}

else throw new Exception("Ожидался ;");

}

public void C(TreeNode Par)

{

TreeNode Temp = new TreeNode("C");

Par.Nodes.Add(Temp);

D(Temp);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmComma)

{

//Temp.Nodes.Add(new TreeNode(","));

F(Temp);

}

}

public void F(TreeNode Par)

{

TreeNode Temp = new TreeNode("F");

Par.Nodes.Add(Temp);

if (Lex.enumPToken == TToken.lxmComma)

{

Temp.Nodes.Add(new TreeNode(","));

D(Temp);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmComma)

{

//Temp.Nodes.Add(new TreeNode(","));

F(Temp);

}

}

}

public void D(TreeNode Par)

{

TreeNode Temp = new TreeNode("D");

Par.Nodes.Add(Temp);

Lex.NextToken();

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

{

if (!IsNumberUnique(Lex.strPLexicalUnit) && Lex.enumPToken == TToken.lxmNumber)

{

throw new Exception("Число " + Lex.strPLexicalUnit + " повторилось!");

}

else

{

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

uSemantAnalyzer uSemant = new uSemantAnalyzer(Tree);

}

}

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

}

private bool IsNumberUnique(string Num)

{

foreach (TreeNode node in Tree.Nodes)

{

if (!IsNumberUniqueInNode(Num, node))

{

return false;

}

}

return true;

}

private bool IsNumberUniqueInNode(string Num, TreeNode node)

{

if (node.Text == Num)

{

return false;

}

foreach (TreeNode child in node.Nodes)

{

if (!IsNumberUniqueInNode(Num, child))

{

return false;

}

}

return true;

}

}

}

**MyHashFunction.cs**

using System;

using System.Collections.Generic;

namespace WindowsFormsApp4

{

public class MyHashFunction

{

public int HashFunction(string word)

{

int hashValue = 0;

foreach (char c in word)

{

hashValue += (int)c;

}

return hashValue;

}

public void AddWord(Dictionary<int, List<string>> hashTable, string word)

{

int hashValue = HashFunction(word);

if (!hashTable.ContainsKey(hashValue))

{

hashTable[hashValue] = new List<string>();

}

else

{

if (hashTable[hashValue].Contains(word))

{

Console.WriteLine("Слово уже существует: " + word);

return;

}

}

hashTable[hashValue].Add(word);

}

public int SearchWord(Dictionary<int, List<string>> hashTable, string word)

{

int hashValue = HashFunction(word);

if (hashTable.ContainsKey(hashValue))

{

return 1;

}

else

{

return 0;

}

}

public bool RemoveWord(Dictionary<int, List<string>> hashTable, string word)

{

int hashValue = HashFunction(word);

if (hashTable.ContainsKey(hashValue))

{

List<string> words = hashTable[hashValue];

if (words.Contains(word))

{

words.Remove(word);

if (words.Count == 0)

{

hashTable.Remove(hashValue);

}

return true;

}

}

return false;

}

}

}

