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

Идрисов И.А.

Проверил: 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.

**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 TLP\_MOF\_4309\_Idrisov\_1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

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

tbFSource.AppendText("1 a");

int n = tbFSource.Lines.Length;

}

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

{

}

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; }

case TCharType.Undefined: { s1 = "Undefined"; 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.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

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

public enum TCharType { Letter, Digit, EndRow, EndText, Space, ReservedSymbol, Undefined }; // 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 = -1;

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

{

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

{

enumFSelectionCharType = TCharType.Undefined;

//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();

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

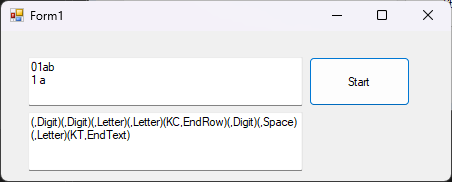
}

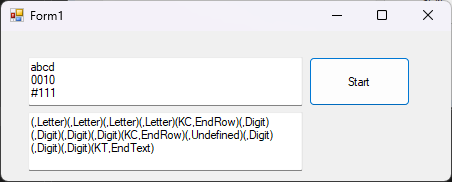
}

}

}

**Результаты тестирования:**





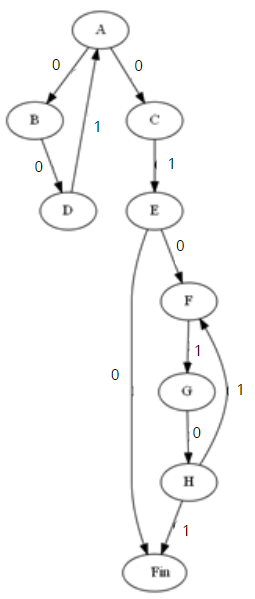
**Лабораторная работа № 2. Разработка лексического анализатора**

**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.



**Первое слово:**

(001)\*010(101)\*

A → 0B | 0C

B → 0D

C → 1Е

D → 1А

E → 0 | 0F

F → 1G

G → 0H

H → 1 | 1F

**Недетерминированная матрица:**

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

**Граф:**

**Детерминированная матрица:**

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| A | BС |  |
| BС | D | E |
| D |  | A |
| E |  | FFin |
| FFin |  | G |
| G | H |  |
| H |  | FFin |

**Второе слово:**

(a|b|c|d)+

Не должно заканчиваться на cd

A → a | b | c | d | aA | bA | cB | cB

B → a | b | c | aA | bA | cB | dA

CFin → a | b | c | d | aA | bA | cA | dA

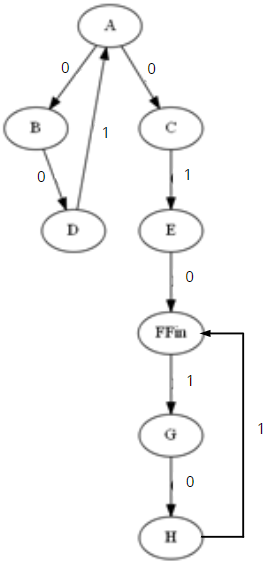
**Недетерминированная матрица:**

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

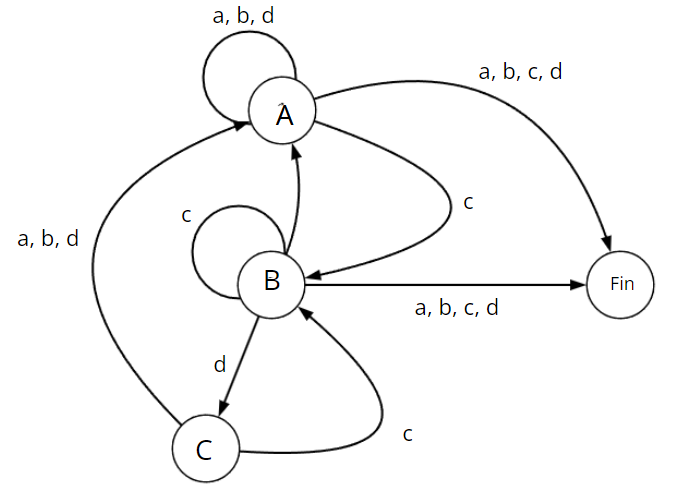
**Детерминированная матрица:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | a | b | c | d |
| AFin | AFin | AFin | BFin | AFin |
| BFin | AFin | AFin | BFin | C |
| C | AFin | AFin | BFin | AFin |

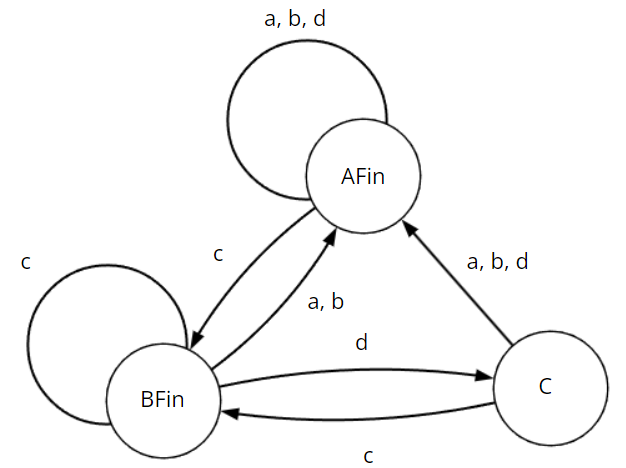
**Граф:**



**Граф:**



**Граф:**

****

**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 TLP\_MOF\_4309\_Idrisov\_1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

tbFSource.AppendText("abcd 001010101" + "\r\n");

int n = tbFSource.Lines.Length;

}

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

{

}

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();

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;

}

}

}

}

**uLex.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

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

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

public enum TToken { lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma, lxmText, lxmtz, lxmdt, lxmr, lxmrs, lxmls, lxmtd };

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 = -1;

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

{

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();

}

// Вариант 10

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// AFin|AFin|AFin|BFin|AFin|

// BFin|AFin|AFin|BFin|C |

// C |AFin|AFin|AFin|AFin|

AFin:

{

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

{

TakeSymbol();

goto AFin;

}

else if (chrFSelection == 'c')

{

TakeSymbol();

goto BFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

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

}

BFin:

{

if (chrFSelection == 'd')

{

TakeSymbol();

goto C;

}

else if (chrFSelection == 'c')

{

TakeSymbol();

goto BFin;

}

else if (chrFSelection == 'a' || chrFSelection == 'b')

{

TakeSymbol();

goto AFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

C:

{

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

{

TakeSymbol();

goto AFin;

}

else throw new Exception("Слово не должно заканчиваться на cd");

}

}

case TCharType.Digit:

{

// 0 1

// A | BC | |

// BC | D | E |

// D | | A |

// E | |FFin |

// FFin | | G |

// G | H | |

// H | |FFin |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto BC;

}

else throw new Exception("Ожидался 0");

BC:

if (chrFSelection == '0')

{

TakeSymbol();

goto D;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto E;

}

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

D:

if (chrFSelection == '1')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидалась 1");

E:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0");

FFin:

if (chrFSelection == '1')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидалась 1");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 0");

H:

if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидалась 1");

}

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.lxmls;

GetSymbol();

return;

}

if (chrFSelection == ']')

{

enumFToken = TToken.lxmrs;

GetSymbol();

return;

}

if (chrFSelection == ',')

{

enumFToken = TToken.lxmComma;

GetSymbol();

return;

}

if (chrFSelection == ':')

{

enumFToken = TToken.lxmdt;

GetSymbol();

return;

}

if (chrFSelection == '=')

{

enumFToken = TToken.lxmr;

GetSymbol();

return;

}

if (chrFSelection == ';')

{

enumFToken = TToken.lxmtz;

GetSymbol();

return;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

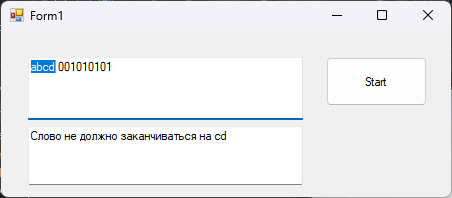
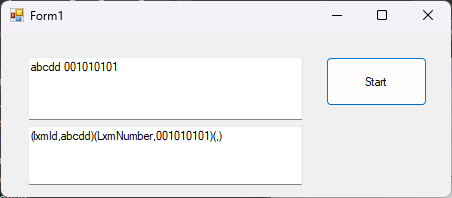
}

}

}

}

**Результаты тестирования:**

**Лабораторная работа № 3. Разработка контекстно-свободного (КС) синтаксического анализатора**

**Текст задания:**

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

**Грамматика:**

S−−> A ; S

S−−>A

A−−> A , B

A−−> B

B−−> <1>

B−−> <2> : <1>

**Освобождение от левой рекурсии:**

S → A | A;S

A → B | BC

C → ;B | ;BC

B → <1>

B → <2> : <1>

**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 TLP\_MOF\_4309\_Idrisov\_1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

tbFSource.AppendText("001010101" + "\r\n");

int n = tbFSource.Lines.Length;

}

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

{

}

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.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;

}

}

}

}

**uLex.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

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

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

public enum TToken { lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma, lxmText, lxmtz, lxmdt, lxmr, lxmrs, lxmls, lxmtd };

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 = -1;

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

{

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();

}

// Вариант 10

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// AFin|AFin|AFin|BFin|AFin|

// BFin|AFin|AFin|BFin|C |

// C |AFin|AFin|AFin|AFin|

AFin:

{

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

{

TakeSymbol();

goto AFin;

}

else if (chrFSelection == 'c')

{

TakeSymbol();

goto BFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

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

}

BFin:

{

if (chrFSelection == 'd')

{

TakeSymbol();

goto C;

}

else if (chrFSelection == 'c')

{

TakeSymbol();

goto BFin;

}

else if (chrFSelection == 'a' || chrFSelection == 'b')

{

TakeSymbol();

goto AFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

C:

{

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

{

TakeSymbol();

goto AFin;

}

else throw new Exception("Слово не должно заканчиваться на cd");

}

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

case TCharType.Digit:

{

// 0 1

// A | BC | |

// BC | D | E |

// D | | A |

// E | |FFin |

// FFin | | G |

// G | H | |

// H | |FFin |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto BC;

}

else throw new Exception("Ожидался 0");

BC:

if (chrFSelection == '0')

{

TakeSymbol();

goto D;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto E;

}

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

D:

if (chrFSelection == '1')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидалась 1");

E:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0");

FFin:

if (chrFSelection == '1')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидалась 1");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 0");

H:

if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидалась 1");

}

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.lxmls;

GetSymbol();

return;

}

if (chrFSelection == ']')

{

enumFToken = TToken.lxmrs;

GetSymbol();

return;

}

if (chrFSelection == ',')

{

enumFToken = TToken.lxmComma;

GetSymbol();

return;

}

if (chrFSelection == ':')

{

enumFToken = TToken.lxmdt;

GetSymbol();

return;

}

if (chrFSelection == '=')

{

enumFToken = TToken.lxmr;

GetSymbol();

return;

}

if (chrFSelection == ';')

{

enumFToken = TToken.lxmtz;

GetSymbol();

return;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

}

}

}

}

**uSyntAnalyzer.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

internal 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 uSyntAnalyzer()

{

}

public void S()

{

A();

while (Lex.enumPToken == TToken.lxmtz || Lex.enumPToken != TToken.lxmEmpty)

{

Lex.NextToken();

A();

}

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

}

public void C()

{

if (Lex.enumPToken == TToken.lxmComma)

{

Lex.NextToken();

B();

if (Lex.enumPToken == TToken.lxmComma)

{

C();

}

}

}

public void A()

{

B();

if (Lex.enumPToken == TToken.lxmComma)

{

C();

}

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

{

throw new Exception("ERROR");

}

}

public void B()

{

if (Lex.enumPToken == TToken.lxmNumber)

{

while (Lex.enumPToken == TToken.lxmNumber) Lex.NextToken();

}

else if (Lex.enumPToken == TToken.lxmIdentifier)

{

while (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

}

if (Lex.enumPToken == TToken.lxmdt)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

while (Lex.enumPToken == TToken.lxmNumber) Lex.NextToken();

}

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

}

else throw new Exception("Ожидалось :");

}

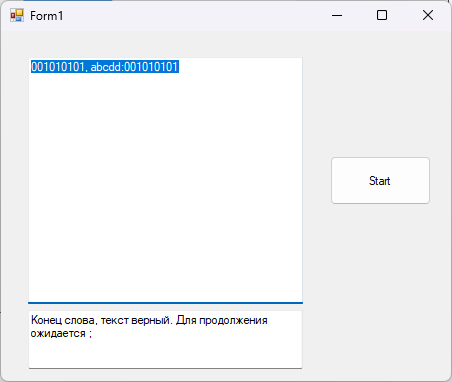
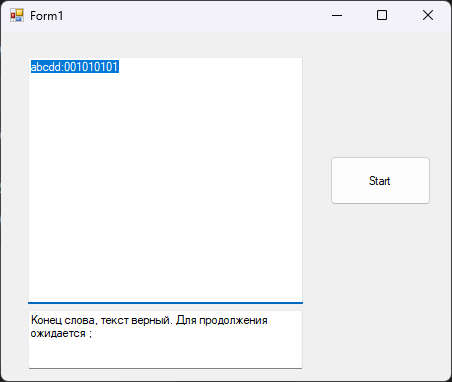
else throw new Exception("Ожидался идентификатор");

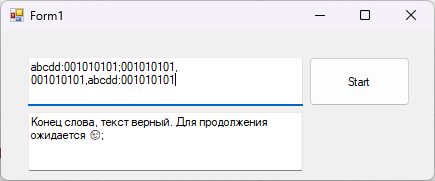
}

}

}

**Результаты тестирования:**





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

**Текст задания:**

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 TLP\_MOF\_4309\_Idrisov\_1

{

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("abcdd:001010101" + "\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.TableToStringList(0, listTable);

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

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

foreach (var entry in hashTableIdentifier)

{

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

}

listTable.Clear();

htl.TableToStringList(1, listTable);

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

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

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 btnFStart\_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;

}

}

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:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

s1 = "id " + 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.lxmdt):

{

hashFunction.AddWord(hashTableRezerv, ":");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmComma):

{

hashFunction.AddWord(hashTableRezerv, ",");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmtz):

{

hashFunction.AddWord(hashTableRezerv, ";");

s1 = "rez " + 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(object sender, EventArgs e)

{

if (hashFunction.SearchWord(hashTableIdentifier, listBox1.SelectedItem.ToString()) == 1)

{

searchBtn.BackColor = Color.Green;

}

else

{

searchBtn.BackColor = Color.Red;

}

}

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

{

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

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));

}

}

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

{

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

}

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

{

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

{

deleteBtn.BackColor = Color.Green;

}

else

{

deleteBtn.BackColor= Color.Red;

}

}

private void changeBtn\_Click(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;

}

}

}

}

**uLex.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

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

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

public enum TToken { lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma, lxmText, lxmtz, lxmdt, lxmr, lxmrs, lxmls, lxmtd };

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 = -1;

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

{

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();

}

// Вариант 10

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// AFin|AFin|AFin|BFin|AFin|

// BFin|AFin|AFin|BFin|C |

// C |AFin|AFin|AFin|AFin|

AFin:

{

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

{

TakeSymbol();

goto AFin;

}

else if (chrFSelection == 'c')

{

TakeSymbol();

goto BFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

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

}

BFin:

{

if (chrFSelection == 'd')

{

TakeSymbol();

goto C;

}

else if (chrFSelection == 'c')

{

TakeSymbol();

goto BFin;

}

else if (chrFSelection == 'a' || chrFSelection == 'b')

{

TakeSymbol();

goto AFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

C:

{

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

{

TakeSymbol();

goto AFin;

}

else throw new Exception("Слово не должно заканчиваться на cd");

}

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

case TCharType.Digit:

{

// 0 1

// A | BC | |

// BC | D | E |

// D | | A |

// E | |FFin |

// FFin | | G |

// G | H | |

// H | |FFin |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto BC;

}

else throw new Exception("Ожидался 0");

BC:

if (chrFSelection == '0')

{

TakeSymbol();

goto D;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto E;

}

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

D:

if (chrFSelection == '1')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидалась 1");

E:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0");

FFin:

if (chrFSelection == '1')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидалась 1");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 0");

H:

if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидалась 1");

}

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.lxmls;

GetSymbol();

return;

}

if (chrFSelection == ']')

{

enumFToken = TToken.lxmrs;

GetSymbol();

return;

}

if (chrFSelection == ',')

{

enumFToken = TToken.lxmComma;

GetSymbol();

return;

}

if (chrFSelection == ':')

{

enumFToken = TToken.lxmdt;

GetSymbol();

return;

}

if (chrFSelection == '=')

{

enumFToken = TToken.lxmr;

GetSymbol();

return;

}

if (chrFSelection == ';')

{

enumFToken = TToken.lxmtz;

GetSymbol();

return;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

}

}

}

}

**uSyntAnalyzer.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

internal 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 uSyntAnalyzer()

{

}

public void S()

{

A();

while (Lex.enumPToken == TToken.lxmtz || Lex.enumPToken != TToken.lxmEmpty)

{

Lex.NextToken();

A();

}

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

}

public void C()

{

if (Lex.enumPToken == TToken.lxmComma)

{

Lex.NextToken();

B();

if (Lex.enumPToken == TToken.lxmComma)

{

C();

}

}

}

public void A()

{

B();

if (Lex.enumPToken == TToken.lxmComma)

{

C();

}

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

{

throw new Exception("ERROR");

}

}

public void B()

{

if (Lex.enumPToken == TToken.lxmNumber)

{

while (Lex.enumPToken == TToken.lxmNumber) Lex.NextToken();

}

else if (Lex.enumPToken == TToken.lxmIdentifier)

{

while (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

}

if (Lex.enumPToken == TToken.lxmdt)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

while (Lex.enumPToken == TToken.lxmNumber) Lex.NextToken();

}

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

}

else throw new Exception("Ожидалось :");

}

else throw new Exception("Ожидался идентификатор");

}

}

}

**MyHashFunction.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

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;

}

}

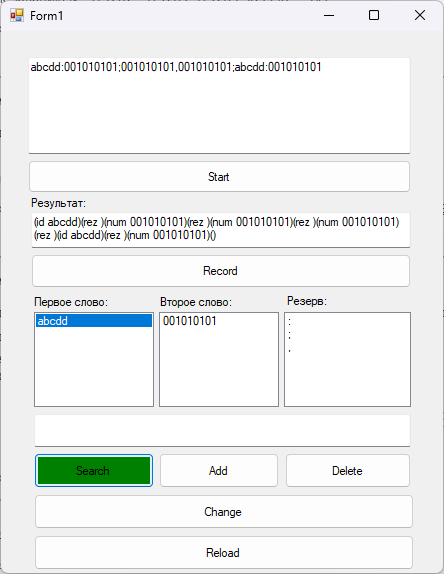
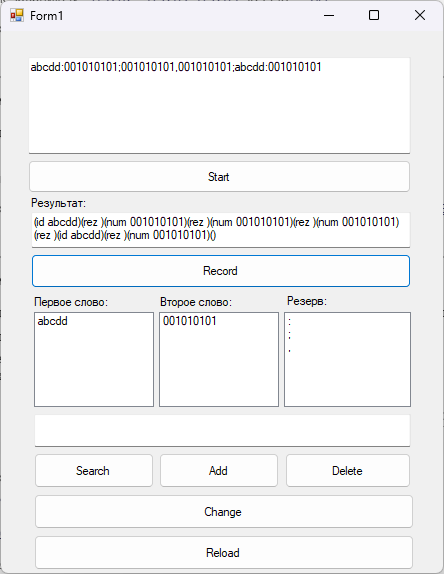
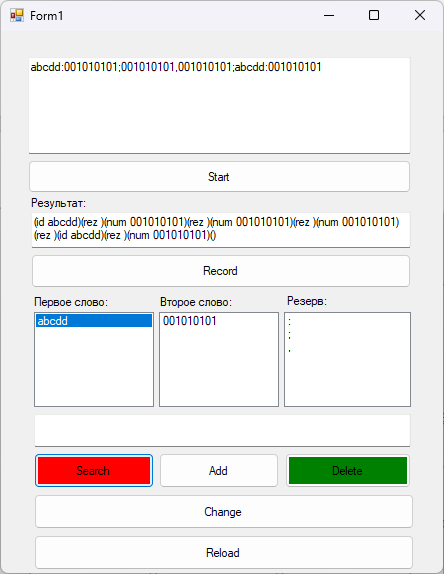
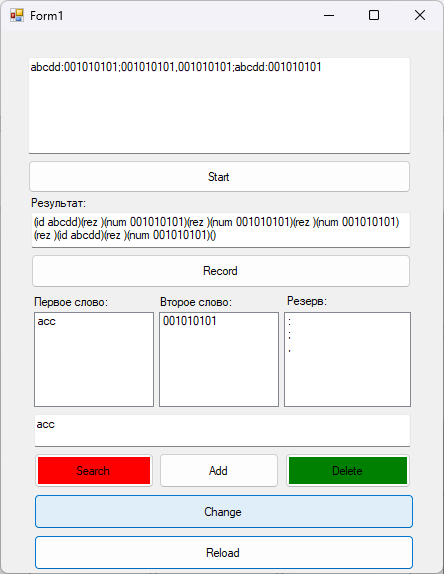
return false;

}

}

}

**Результаты тестирования:**

Лабораторная работа № 6. Разработка генератора (2 часа)

Задание. 1. Перевести все числа в десятичное представление.

Задание 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.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

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("001010101,abcdd:001010101;001010101,001010101;abcdd:001010101" + "\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.TableToStringList(0, listTable);

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

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

foreach (var entry in hashTableIdentifier)

{

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

}

listTable.Clear();

htl.TableToStringList(1, listTable);

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

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

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 btnFStart\_Click(object sender, EventArgs e)

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer(SyntTree); // Используем существующий экземпляр treeView2

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(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 = "id " + 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.lxmdt):

{

hashFunction.AddWord(hashTableRezerv, ":");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmComma):

{

hashFunction.AddWord(hashTableRezerv, ",");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmtz):

{

hashFunction.AddWord(hashTableRezerv, ";");

s1 = "rez " + 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(object sender, EventArgs e)

{

if (hashFunction.SearchWord(hashTableIdentifier, listBox1.SelectedItem.ToString()) == 1)

{

searchBtn.BackColor = Color.Green;

}

else

{

searchBtn.BackColor = Color.Red;

}

}

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

{

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

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));

}

SyntTree.Nodes.Clear();

}

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

{

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

}

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

{

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

{

deleteBtn.BackColor = Color.Green;

}

else

{

deleteBtn.BackColor= Color.Red;

}

}

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

{

TreeConverter treeConverter = new TreeConverter();

treeConverter.ConvertBinaryToDecimal(SyntTree);

}

}

}

**TreeConverter.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Xml.Linq;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

internal class TreeConverter

{

public void ConvertBinaryToDecimal(TreeView tree)

{

foreach (TreeNode node in tree.Nodes)

{

ConvertBinaryToDecimal(node);

}

}

private void ConvertBinaryToDecimal(TreeNode node)

{

if (node.Text == "B")

{

if (node.Nodes.Count == 1)

{

foreach (TreeNode node2 in node.Nodes)

{

string binaryValue = node2.Text;

if (IsBin(binaryValue))

{

int decimalValue = Convert.ToInt32(binaryValue, 2);

node2.Text = decimalValue.ToString();

}

}

}

if (node.Nodes.Count > 1)

{

foreach (TreeNode node2 in node.Nodes)

{

string binaryValue = node2.Text;

if (IsBin(binaryValue))

{

int decimalValue = Convert.ToInt32(binaryValue, 2);

node2.Text = decimalValue.ToString();

}

node.Parent.Nodes.Add(node2.Text);

}

node.Remove();

}

}

foreach (TreeNode childNode in node.Nodes)

{

ConvertBinaryToDecimal(childNode);

}

}

private bool IsBin(string value)

{

foreach (var c in value)

if (c != '0' && c != '1')

return false;

return true;

}

}

}

**uSyntAnalyzer.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

internal 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();

private TreeView tree; // Declare TreeView for function calls

public uSyntAnalyzer(TreeView treeView)

{

tree = treeView;

}

public int i = -1;

public void S()

{

TreeNode parent = new TreeNode("S");

tree.Nodes.Add(parent);

Console.WriteLine("S");

A(parent);

Console.WriteLine(Lex.enumPToken);

while (Lex.enumPToken == TToken.lxmtz || Lex.enumPToken != TToken.lxmEmpty)

{

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

TreeNode highParent = new TreeNode("S");

parent.Nodes.Add(highParent);

Lex.NextToken();

A(highParent);

}

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

}

public void C(TreeNode highParent)

{

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

Console.WriteLine("C");

if (Lex.enumPToken == TToken.lxmComma)

{

Lex.NextToken();

B(highParent);

if (Lex.enumPToken == TToken.lxmComma)

{

C(highParent);

}

}

}

public void A(TreeNode highParent)

{

Console.WriteLine("A");

TreeNode parent = new TreeNode("A");

highParent.Nodes.Add(parent);

B(parent);

if (Lex.enumPToken == TToken.lxmComma)

{

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

C(parent);

}

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

{

throw new Exception("ERROR");

}

}

public void B(TreeNode highParent)

{

TreeNode parent = new TreeNode("B");

highParent.Nodes.Add(parent);

Console.WriteLine("B");

if (Lex.enumPToken == TToken.lxmNumber)

{

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

while (Lex.enumPToken == TToken.lxmNumber) Lex.NextToken();

}

else if (Lex.enumPToken == TToken.lxmIdentifier)

{

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

while (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

}

if (Lex.enumPToken == TToken.lxmdt)

{

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

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

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

while (Lex.enumPToken == TToken.lxmNumber) Lex.NextToken();

}

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

}

else throw new Exception("Ожидалось :");

}

else throw new Exception("Ожидался идентификатор");

}

}

}

**uLex.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

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

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

public enum TToken { lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma, lxmText, lxmtz, lxmdt, lxmr, lxmrs, lxmls, lxmtd };

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 = -1;

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

{

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();

}

// Вариант 10

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// AFin|AFin|AFin|BFin|AFin|

// BFin|AFin|AFin|BFin|C |

// C |AFin|AFin|AFin|AFin|

AFin:

{

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

{

TakeSymbol();

goto AFin;

}

else if (chrFSelection == 'c')

{

TakeSymbol();

goto BFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

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

}

BFin:

{

if (chrFSelection == 'd')

{

TakeSymbol();

goto C;

}

else if (chrFSelection == 'c')

{

TakeSymbol();

goto BFin;

}

else if (chrFSelection == 'a' || chrFSelection == 'b')

{

TakeSymbol();

goto AFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

C:

{

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

{

TakeSymbol();

goto AFin;

}

else throw new Exception("Слово не должно заканчиваться на cd");

}

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

case TCharType.Digit:

{

// 0 1

// A | BC | |

// BC | D | E |

// D | | A |

// E | |FFin |

// FFin | | G |

// G | H | |

// H | |FFin |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto BC;

}

else throw new Exception("Ожидался 0");

BC:

if (chrFSelection == '0')

{

TakeSymbol();

goto D;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto E;

}

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

D:

if (chrFSelection == '1')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидалась 1");

E:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0");

FFin:

if (chrFSelection == '1')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидалась 1");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 0");

H:

if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидалась 1");

}

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.lxmls;

GetSymbol();

return;

}

if (chrFSelection == ']')

{

enumFToken = TToken.lxmrs;

GetSymbol();

return;

}

if (chrFSelection == ',')

{

enumFToken = TToken.lxmComma;

GetSymbol();

return;

}

if (chrFSelection == ':')

{

enumFToken = TToken.lxmdt;

GetSymbol();

return;

}

if (chrFSelection == '=')

{

enumFToken = TToken.lxmr;

GetSymbol();

return;

}

if (chrFSelection == ';')

{

enumFToken = TToken.lxmtz;

GetSymbol();

return;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

}

}

}

}

**myHashFunction.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

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;

}

}

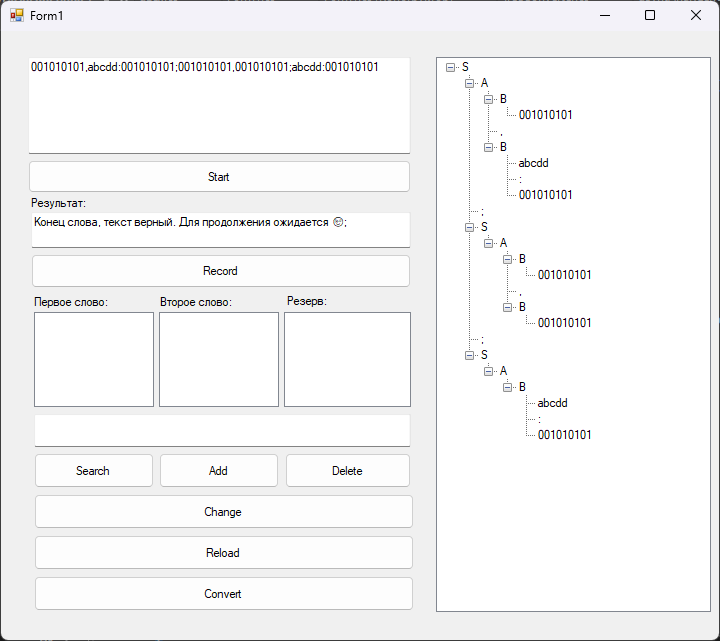
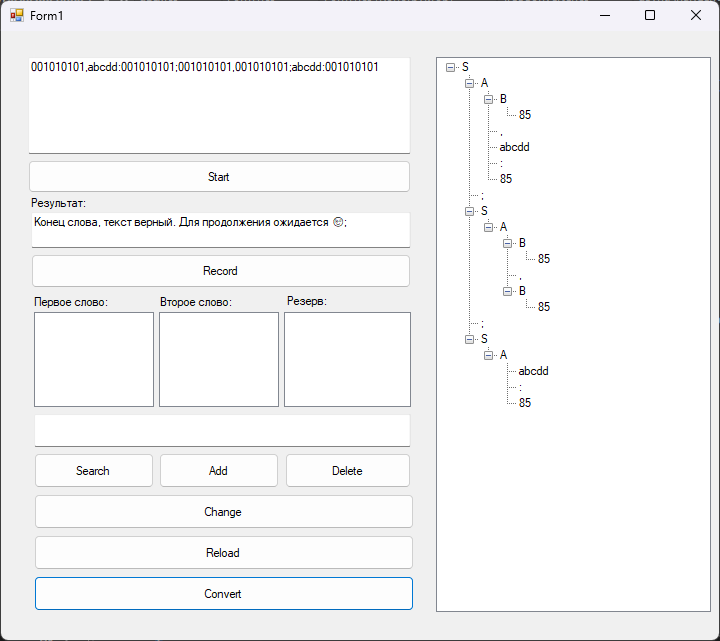
return false;

}

}

}

**Результат работы программы:**

Лабораторная работа № 6. 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 10: 

Код программы:

**Form1.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

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 Form1()

{

InitializeComponent();

tbFSource.AppendText("001010101,abcdd:001010101;001010101,001010101;abcdd:001010101" + "\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();

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

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

foreach (var entry in hashTableIdentifier)

{

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

}

listTable.Clear();

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

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

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 btnFStart\_Click(object sender, EventArgs e)

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer(SyntTree); // Используем существующий экземпляр treeView2

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

TablesToMemo(this, e);

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

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

TablesToMemo(this, e);

break;

}

case (TToken.lxmdt):

{

hashFunction.AddWord(hashTableRezerv, ":");

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

TablesToMemo(this, e);

break;

}

case (TToken.lxmComma):

{

hashFunction.AddWord(hashTableRezerv, ",");

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

TablesToMemo(this, e);

break;

}

case (TToken.lxmtz):

{

hashFunction.AddWord(hashTableRezerv, ";");

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

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(object sender, EventArgs e)

{

if (hashFunction.SearchWord(hashTableIdentifier, listBox1.SelectedItem.ToString()) == 1)

{

searchBtn.BackColor = Color.Green;

}

else

{

searchBtn.BackColor = Color.Red;

}

}

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

{

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

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));

}

SyntTree.Nodes.Clear();

}

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

{

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

}

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

{

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

{

deleteBtn.BackColor = Color.Green;

}

else

{

deleteBtn.BackColor= Color.Red;

}

}

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

{

TreeConverter treeConverter = new TreeConverter();

treeConverter.ConvertBinaryToDecimal(SyntTree);

}

}

}

**uSyntAnalyzer.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

internal 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();

private TreeView tree; // Declare TreeView for function calls

public uSyntAnalyzer(TreeView treeView)

{

tree = treeView;

}

public int i = -1;

public void S()

{

TreeNode parent = new TreeNode("S");

tree.Nodes.Add(parent);

Console.WriteLine("S");

A(parent);

Console.WriteLine(Lex.enumPToken);

while (Lex.enumPToken == TToken.lxmtz || Lex.enumPToken != TToken.lxmEmpty)

{

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

TreeNode highParent = new TreeNode("S");

parent.Nodes.Add(highParent);

Lex.NextToken();

A(highParent);

}

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

}

public void C(TreeNode highParent)

{

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

Console.WriteLine("C");

if (Lex.enumPToken == TToken.lxmComma)

{

Lex.NextToken();

B(highParent);

if (Lex.enumPToken == TToken.lxmComma)

{

C(highParent);

}

}

}

public void A(TreeNode highParent)

{

Console.WriteLine("A");

TreeNode parent = new TreeNode("A");

highParent.Nodes.Add(parent);

B(parent);

if (Lex.enumPToken == TToken.lxmComma)

{

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

C(parent);

}

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

{

throw new Exception("ERROR");

}

}

public void B(TreeNode highParent)

{

TreeNode parent = new TreeNode("B");

highParent.Nodes.Add(parent);

Console.WriteLine("B");

if (Lex.enumPToken == TToken.lxmNumber)

{

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

while (Lex.enumPToken == TToken.lxmNumber) Lex.NextToken();

}

else if (Lex.enumPToken == TToken.lxmIdentifier)

{

Duplicates(tree, Lex.strPLexicalUnit);

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

//if (myHashFunction.SearchWord(hashTableIdentifier, Lex.strPLexicalUnit) == 0)

//{

// myHashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

//}

//else

//{

// tree.Nodes.Clear();

// throw new Exception(Lex.strPLexicalUnit + " повторяется");

//}

while (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

}

if (Lex.enumPToken == TToken.lxmdt)

{

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

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

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

while (Lex.enumPToken == TToken.lxmNumber) Lex.NextToken();

}

else

{

tree.Nodes.Clear();

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

}

}

else

{

tree.Nodes.Clear();

throw new Exception("Ожидалось :");

}

}

else

{

tree.Nodes.Clear();

throw new Exception("Ожидался идентификатор");

}

}

public void Duplicates(TreeView tree, string s)

{

foreach (TreeNode node in tree.Nodes) // Рассматриваем всевозможные ветви (node) нашего TreeView

{

Duplicates(node, s);

}

}

private void Duplicates(TreeNode node, string s)

{

if (node.Text == "B") // Если мы попали в ветку под названием B, именно там в моём варианте хранятся лексические юниты

{

if (node.Nodes.Count > 0) // Если эта ветка не пустая

{

foreach (TreeNode node2 in node.Nodes) // Перебираем все подветви нода с названием B, тем самым рассматривая каждый лексический юнит

{

if (node2.Text == s)

{

throw new Exception("Значение " + s + " повторяется");

}

}

}

}

foreach (TreeNode childNode in node.Nodes)

{

Duplicates(childNode, s); // Тут мы разделяем опять ветвь (node) на ещё более мелкие, то есть углубляемся

}

}

}

}

**uLex.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

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

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

public enum TToken { lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma, lxmText, lxmtz, lxmdt, lxmr, lxmrs, lxmls, lxmtd };

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 = -1;

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

{

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();

}

// Вариант 10

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// AFin|AFin|AFin|BFin|AFin|

// BFin|AFin|AFin|BFin|C |

// C |AFin|AFin|AFin|AFin|

AFin:

{

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

{

TakeSymbol();

goto AFin;

}

else if (chrFSelection == 'c')

{

TakeSymbol();

goto BFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

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

}

BFin:

{

if (chrFSelection == 'd')

{

TakeSymbol();

goto C;

}

else if (chrFSelection == 'c')

{

TakeSymbol();

goto BFin;

}

else if (chrFSelection == 'a' || chrFSelection == 'b')

{

TakeSymbol();

goto AFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

C:

{

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

{

TakeSymbol();

goto AFin;

}

else throw new Exception("Слово не должно заканчиваться на cd");

}

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

case TCharType.Digit:

{

// 0 1

// A | BC | |

// BC | D | E |

// D | | A |

// E | |FFin |

// FFin | | G |

// G | H | |

// H | |FFin |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto BC;

}

else throw new Exception("Ожидался 0");

BC:

if (chrFSelection == '0')

{

TakeSymbol();

goto D;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto E;

}

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

D:

if (chrFSelection == '1')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидалась 1");

E:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0");

FFin:

if (chrFSelection == '1')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидалась 1");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 0");

H:

if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидалась 1");

}

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.lxmls;

GetSymbol();

return;

}

if (chrFSelection == ']')

{

enumFToken = TToken.lxmrs;

GetSymbol();

return;

}

if (chrFSelection == ',')

{

enumFToken = TToken.lxmComma;

GetSymbol();

return;

}

if (chrFSelection == ':')

{

enumFToken = TToken.lxmdt;

GetSymbol();

return;

}

if (chrFSelection == '=')

{

enumFToken = TToken.lxmr;

GetSymbol();

return;

}

if (chrFSelection == ';')

{

enumFToken = TToken.lxmtz;

GetSymbol();

return;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

}

}

}

}

**MyHashFunction.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

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))

{

List<string> words = hashTable[hashValue];

if (words.Contains(word)) return 1;

}

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;

}

}

}

**TreeConverter.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Xml.Linq;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

internal class TreeConverter

{

public void ConvertBinaryToDecimal(TreeView tree)

{

foreach (TreeNode node in tree.Nodes)

{

ConvertBinaryToDecimal(node);

}

}

private void ConvertBinaryToDecimal(TreeNode node)

{

if (node.Text == "B")

{

if (node.Nodes.Count == 1)

{

foreach (TreeNode node2 in node.Nodes)

{

string binaryValue = node2.Text;

if (IsBin(binaryValue))

{

int decimalValue = Convert.ToInt32(binaryValue, 2);

node2.Text = decimalValue.ToString();

}

}

}

if (node.Nodes.Count > 1)

{

foreach (TreeNode node2 in node.Nodes)

{

string binaryValue = node2.Text;

if (IsBin(binaryValue))

{

int decimalValue = Convert.ToInt32(binaryValue, 2);

node2.Text = decimalValue.ToString();

}

node.Parent.Nodes.Add(node2.Text);

}

node.Remove();

}

}

foreach (TreeNode childNode in node.Nodes)

{

ConvertBinaryToDecimal(childNode);

}

}

private bool IsBin(string value)

{

foreach (var c in value)

if (c != '0' && c != '1')

return false;

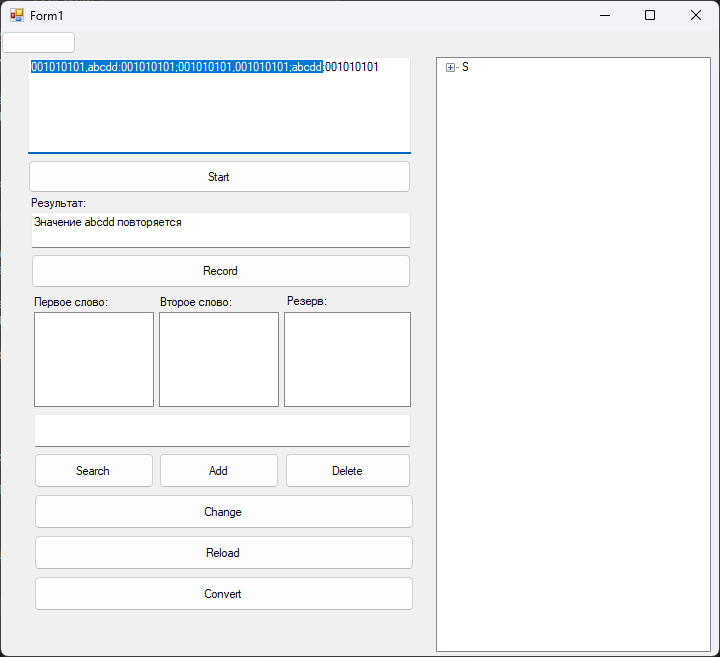
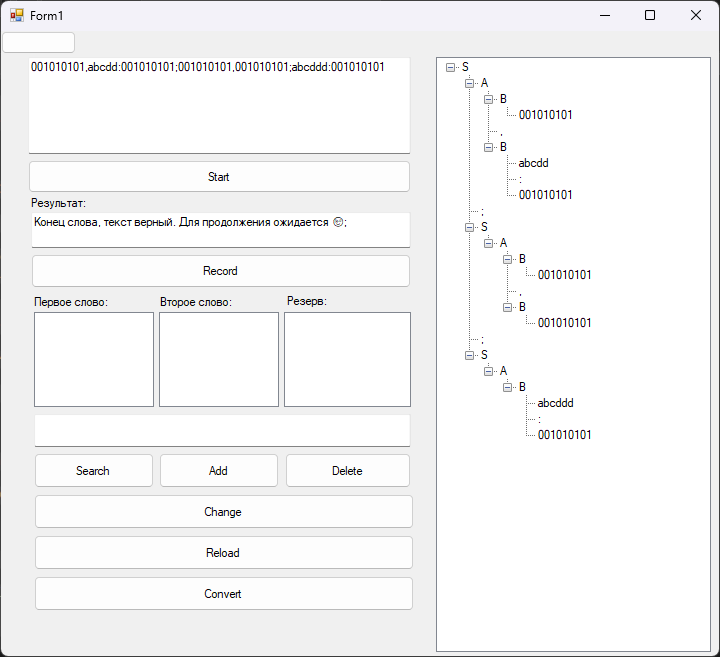
return true;

}

}

}

**Результат работы кода:**

Лабораторная работа №8.

Разработка семантического анализатор.

Задание: Разработать семантический анализатор. Выполнить проверку внеконтекстной грамматики.

Вариант: Проверка равенства количества d в конце идентификатора с количеством дополнительных блоков двоичной записи в конструкциях вида <2> : <1>

Код программы:

**Form1.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

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 Form1()

{

InitializeComponent();

tbFSource.AppendText("001010101,abcdd:001010101;001010101,001010101;abcdd:001010101" + "\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();

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

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

foreach (var entry in hashTableIdentifier)

{

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

}

listTable.Clear();

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

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

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 btnFStart\_Click(object sender, EventArgs e)

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer(SyntTree); // Используем существующий экземпляр treeView2

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

TablesToMemo(this, e);

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

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

TablesToMemo(this, e);

break;

}

case (TToken.lxmdt):

{

hashFunction.AddWord(hashTableRezerv, ":");

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

TablesToMemo(this, e);

break;

}

case (TToken.lxmComma):

{

hashFunction.AddWord(hashTableRezerv, ",");

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

TablesToMemo(this, e);

break;

}

case (TToken.lxmtz):

{

hashFunction.AddWord(hashTableRezerv, ";");

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

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(object sender, EventArgs e)

{

if (hashFunction.SearchWord(hashTableIdentifier, listBox1.SelectedItem.ToString()) == 1)

{

searchBtn.BackColor = Color.Green;

}

else

{

searchBtn.BackColor = Color.Red;

}

}

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

{

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

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));

}

SyntTree.Nodes.Clear();

}

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

{

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

}

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

{

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

{

deleteBtn.BackColor = Color.Green;

}

else

{

deleteBtn.BackColor= Color.Red;

}

}

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

{

TreeConverter treeConverter = new TreeConverter();

treeConverter.ConvertBinaryToDecimal(SyntTree);

}

}

}

**uSyntAnalyzer.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

internal 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();

private TreeView tree; // Declare TreeView for function calls

public uSyntAnalyzer(TreeView treeView)

{

tree = treeView;

}

public int i = -1;

public void S()

{

TreeNode parent = new TreeNode("S");

tree.Nodes.Add(parent);

Console.WriteLine("S");

A(parent);

Console.WriteLine(Lex.enumPToken);

while (Lex.enumPToken == TToken.lxmtz || Lex.enumPToken != TToken.lxmEmpty)

{

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

TreeNode highParent = new TreeNode("S");

parent.Nodes.Add(highParent);

Lex.NextToken();

A(highParent);

}

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

}

public void C(TreeNode highParent)

{

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

Console.WriteLine("C");

if (Lex.enumPToken == TToken.lxmComma)

{

Lex.NextToken();

B(highParent);

if (Lex.enumPToken == TToken.lxmComma)

{

C(highParent);

}

}

}

public void A(TreeNode highParent)

{

Console.WriteLine("A");

TreeNode parent = new TreeNode("A");

highParent.Nodes.Add(parent);

B(parent);

if (Lex.enumPToken == TToken.lxmComma)

{

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

C(parent);

}

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

{

throw new Exception("ERROR");

}

}

public void B(TreeNode highParent)

{

TreeNode parent = new TreeNode("B");

highParent.Nodes.Add(parent);

Console.WriteLine("B");

if (Lex.enumPToken == TToken.lxmNumber)

{

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

while (Lex.enumPToken == TToken.lxmNumber) Lex.NextToken();

}

else if (Lex.enumPToken == TToken.lxmIdentifier)

{

Duplicates(tree, Lex.strPLexicalUnit);

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

//if (myHashFunction.SearchWord(hashTableIdentifier, Lex.strPLexicalUnit) == 0)

//{

// myHashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

//}

//else

//{

// tree.Nodes.Clear();

// throw new Exception(Lex.strPLexicalUnit + " повторяется");

//}

while (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

}

if (Lex.enumPToken == TToken.lxmdt)

{

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

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

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

while (Lex.enumPToken == TToken.lxmNumber) Lex.NextToken();

}

else

{

tree.Nodes.Clear();

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

}

}

else

{

tree.Nodes.Clear();

throw new Exception("Ожидалось :");

}

}

else

{

tree.Nodes.Clear();

throw new Exception("Ожидался идентификатор");

}

}

public void Duplicates(TreeView tree, string s)

{

foreach (TreeNode node in tree.Nodes) // Рассматриваем всевозможные ветви (node) нашего TreeView

{

Duplicates(node, s);

}

}

private void Duplicates(TreeNode node, string s)

{

if (node.Text == "B") // Если мы попали в ветку под названием B, именно там в моём варианте хранятся лексические юниты

{

if (node.Nodes.Count > 0) // Если эта ветка не пустая

{

foreach (TreeNode node2 in node.Nodes) // Перебираем все подветви нода с названием B, тем самым рассматривая каждый лексический юнит

{

if (node2.Text == s)

{

throw new Exception("Значение " + s + " повторяется");

}

}

}

}

foreach (TreeNode childNode in node.Nodes)

{

Duplicates(childNode, s); // Тут мы разделяем опять ветвь (node) на ещё более мелкие, то есть углубляемся

}

}

}

}

**uLex.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

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

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

public enum TToken { lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma, lxmText, lxmtz, lxmdt, lxmr, lxmrs, lxmls, lxmtd };

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 = -1;

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

{

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();

}

// Вариант 10

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// AFin|AFin|AFin|BFin|AFin|

// BFin|AFin|AFin|BFin|C |

// C |AFin|AFin|AFin|AFin|

AFin:

{

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

{

TakeSymbol();

goto AFin;

}

else if (chrFSelection == 'c')

{

TakeSymbol();

goto BFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

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

}

BFin:

{

if (chrFSelection == 'd')

{

TakeSymbol();

goto C;

}

else if (chrFSelection == 'c')

{

TakeSymbol();

goto BFin;

}

else if (chrFSelection == 'a' || chrFSelection == 'b')

{

TakeSymbol();

goto AFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

C:

{

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

{

TakeSymbol();

goto AFin;

}

else throw new Exception("Слово не должно заканчиваться на cd");

}

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

case TCharType.Digit:

{

// 0 1

// A | BC | |

// BC | D | E |

// D | | A |

// E | |FFin |

// FFin | | G |

// G | H | |

// H | |FFin |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto BC;

}

else throw new Exception("Ожидался 0");

BC:

if (chrFSelection == '0')

{

TakeSymbol();

goto D;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto E;

}

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

D:

if (chrFSelection == '1')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидалась 1");

E:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0");

FFin:

if (chrFSelection == '1')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидалась 1");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 0");

H:

if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидалась 1");

}

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.lxmls;

GetSymbol();

return;

}

if (chrFSelection == ']')

{

enumFToken = TToken.lxmrs;

GetSymbol();

return;

}

if (chrFSelection == ',')

{

enumFToken = TToken.lxmComma;

GetSymbol();

return;

}

if (chrFSelection == ':')

{

enumFToken = TToken.lxmdt;

GetSymbol();

return;

}

if (chrFSelection == '=')

{

enumFToken = TToken.lxmr;

GetSymbol();

return;

}

if (chrFSelection == ';')

{

enumFToken = TToken.lxmtz;

GetSymbol();

return;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

}

}

}

}

**MyHashFunction.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

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))

{

List<string> words = hashTable[hashValue];

if (words.Contains(word)) return 1;

}

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;

}

}

}

**TreeConverter.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Xml.Linq;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

internal class TreeConverter

{

public void ConvertBinaryToDecimal(TreeView tree)

{

foreach (TreeNode node in tree.Nodes)

{

ConvertBinaryToDecimal(node);

}

}

private void ConvertBinaryToDecimal(TreeNode node)

{

if (node.Text == "B")

{

if (node.Nodes.Count == 1)

{

foreach (TreeNode node2 in node.Nodes)

{

string binaryValue = node2.Text;

if (IsBin(binaryValue))

{

int decimalValue = Convert.ToInt32(binaryValue, 2);

node2.Text = decimalValue.ToString();

}

}

}

if (node.Nodes.Count > 1)

{

foreach (TreeNode node2 in node.Nodes)

{

string binaryValue = node2.Text;

if (IsBin(binaryValue))

{

int decimalValue = Convert.ToInt32(binaryValue, 2);

node2.Text = decimalValue.ToString();

}

node.Parent.Nodes.Add(node2.Text);

}

node.Remove();

}

}

foreach (TreeNode childNode in node.Nodes)

{

ConvertBinaryToDecimal(childNode);

}

}

private bool IsBin(string value)

{

foreach (var c in value)

if (c != '0' && c != '1')

return false;

return true;

}

}

}

**uSemantAnalyzer.cs**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Security.AccessControl;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace TLP\_MOF\_4309\_Idrisov\_1

{

public 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 > 1)

{

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

strDigital = node.Nodes[2].Text.ToString();

Check(strIndentifier, strDigital, node);

}

}

foreach (TreeNode childNode in node.Nodes)

{

TreeController(childNode);

}

}

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

{

int countI = 0;

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

{

if (ident[i] == 'd')

{

countI++;

}

else

{

break; // Прерываем цикл, если не встречаем 'd'

}

}

int countD = digit.Length / 3 - 1;

Console.WriteLine(countI);

Console.WriteLine(countD);

if (countD != countI)

{

tree.SelectedNode = node;

tree.SelectedNode.BackColor = Color.Blue;

throw new Exception("Error: count d <> count dop digit");

}

}

}

}

**Результат работы кода:** 