Министерство образования Российской Федерации

Федеральное агентство по образованию

КАЗАНСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. А.Н. Туполева - КАИ

Кафедра АСОИУ

Отчет по лабораторным работам №1-8

по дисциплинам

«ТЕОРИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ» и

«МЕТОДЫ ТРАНСЛЯЦИИ»

Вариант №19

Выполнил:

Студент группы 4309

Муллин Б.И.

Проверила: Бикмуллина И.И.

Казань 2023

**Лабораторная работа № 1. Разработка транслитератора**

**Текст задания:**

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

**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 WindowsFormsApp1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

textBox1.AppendText("01ab" + "\r\n");

textBox1.AppendText("1 a");

int n = textBox1.Lines.Length;

}

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

{

}

private void button1\_Click(object sender, EventArgs e)

{

uLex Lex = new uLex();

Lex.strPSource = textBox1.Lines;

Lex.strPMessage = textBox2.Lines;

int x = textBox1.TextLength;

int y = textBox1.Lines.Length;

textBox2.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.clBracket: { s1 = "CloseBracket"; break; }

case TCharType.opBracket: { s1 = "OpenBracket"; break; }

case TCharType.expMark: { s1 = "ExclamationMark"; break; }

case TCharType.comma: { s1 = "Comma"; break; }

case TCharType.semicolon: { s1 = "Semicolon"; break; }

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

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

}

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

textBox2.Text += m; //добавляется в строку сообщение

}

}

catch (Exception exc)

{

textBox2.Text += exc.Message;

textBox1.Select();

textBox1.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Lex.intPSourceRowSelection; i++) n += textBox1.Lines[i].Length + 2;

n += Lex.intPSourceColSelection;

textBox1.SelectionLength = n;

}

}

private void Form1\_Load\_1(object sender, EventArgs e)

{

}

}

}

**uLex.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp1

{

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

public enum TCharType { Letter, Digit, EndRow, EndText, Space, ReservedSymbol, opBracket, clBracket, expMark, comma, semicolon }; // тип символа

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

class uLex

{

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

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

public TCharType enumFSelectionCharType;

public char chrFSelection;

private TState enumFState;

private int intFSourceRowSelection;

private int intFSourceColSelection;

private String strFLexicalUnit;

private TToken enumFToken;

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

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

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

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

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

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

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

public uLex()

{

}

public void GetSymbol() //метод класса лексический анализатор

{

// 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 == '(') enumFSelectionCharType = TCharType.opBracket;

else if (chrFSelection == ')') enumFSelectionCharType = TCharType.clBracket;

else if (chrFSelection == '!') enumFSelectionCharType = TCharType.expMark;

else if (chrFSelection == ',') enumFSelectionCharType = TCharType.comma;

else if (chrFSelection == ';') enumFSelectionCharType = TCharType.semicolon;

//else if (chrFSelection == '(' || chrFSelection == ')' || chrFSelection == ':' || chrFSelection == '-' || chrFSelection == ',' || chrFSelection == '.') enumFSelectionCharType = TCharType.ReservedSymbol;

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

enumFState = TState.Continue;

}

intFSourceColSelection++;

}

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();

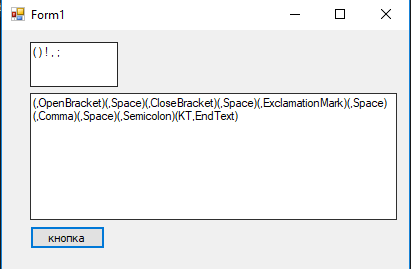
}

}

}

}

**Результат:**



**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.

|  |  |  |  |
| --- | --- | --- | --- |
| **19** | **(010)\*110(111)\*** | **(a|b|c|d)+** | **В порядке обратном алфавитному** |

**Первое слово:**

(010)\*110(111)\*

A → 0B | 1K

B → 1С

C → 0А

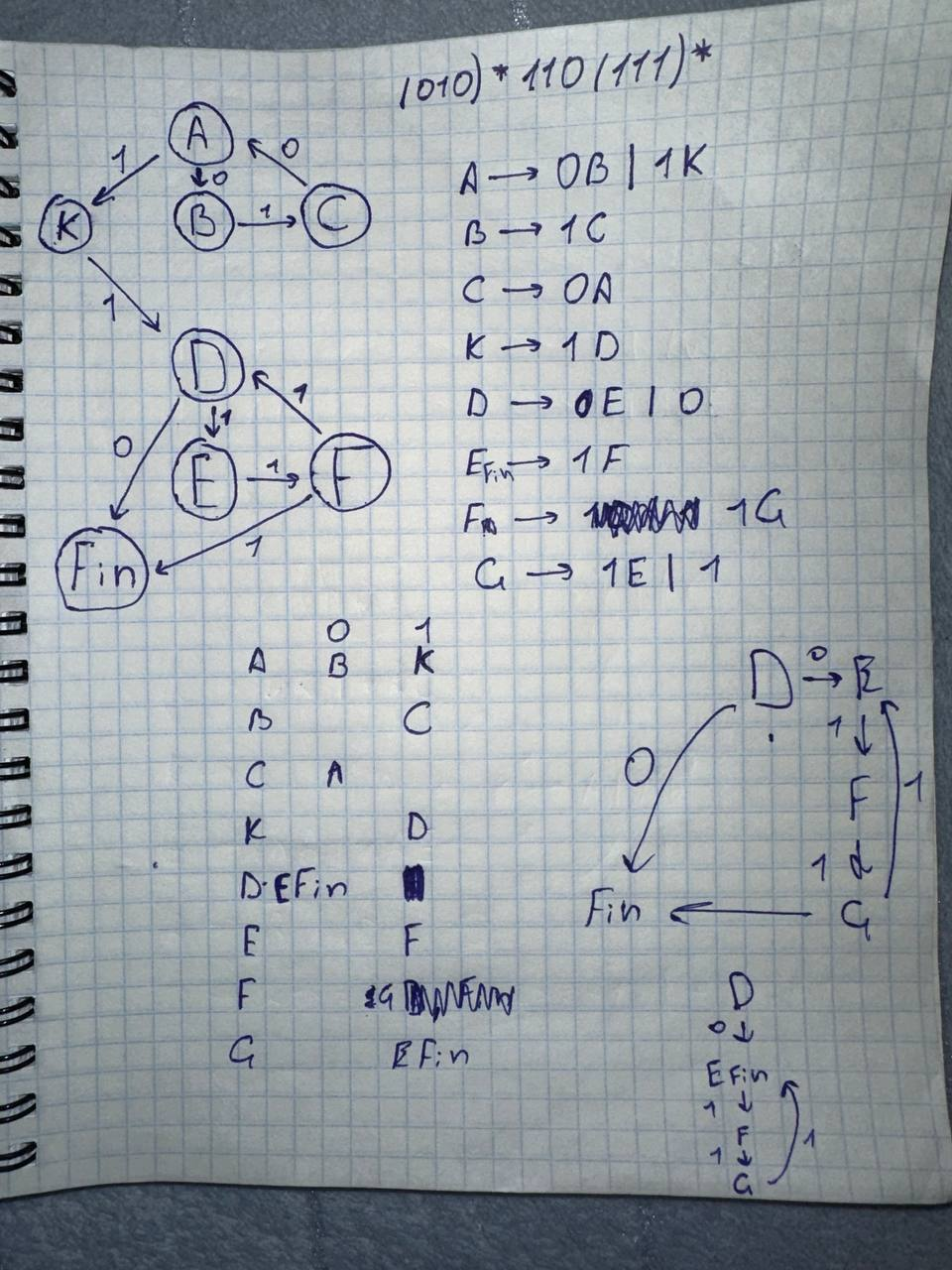
K → 1D

D → 0 | 0E

E → 1F

F → 1G

G → 1 | 1E



В математике и теории алгоритмов, матрица является детерминированной, если каждое значение в ней определяется однозначно на основе значений других элементов матрицы и правил операции. Это означает, что результат вычислений по такой матрице всегда будет одним и тем же, независимо от того, кто и когда выполняет вычисления.

Недетерминированная матрица, напротив, содержит элементы, значения которых могут быть случайными, неопределенными или неточно определенными. В результате вычислений по такой матрице, результат может варьироваться в зависимости от входных данных, случайных значений элементов или последовательности выполнения операций.

Примером детерминированной матрицы может служить квадратная матрица, обратная к которой вычисляется с помощью определителя и алгебраических методов решения систем линейных уравнений. Такая матрица всегда будет иметь единственное решение, которое можно найти по определенному алгоритму.

В качестве примера недетерминированной матрицы можно привести матрицу, в которой некоторые элементы представляют собой случайные числа. Если выполнить вычисления по такой матрице несколько раз, с разными случайными числами, результаты будут отличаться друг от друга.

Детерминированные матрицы и алгоритмы, работающие с ними, обычно более эффективны и предсказуемы, в то время как недетерминированные подходы могут обеспечивать большую гибкость и адаптивность.

**Недетерминированная матрица:**

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

**Детерминированная матрица:**

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| A | B | K |
| B |  | C |
| C | A |  |
| K |  | D |
| D | EFin |  |
| E |  | F |
| F |  | G |
| G |  | EFin |

**Второе слово:**

(a|b|c|d)+

В порядке обратном алфавитному

A → cB

B → b | bС

C → a | b | c | d | aC | bC | cC | dC

**Недетерминированная матрица:**

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

**Детерминированная матрица:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | a | b | c | d |
| AFin | AFin | BFin | CFin | DFin |
| BFin | AFin | BFin | CFin |  |
| CFin | AFin | BFin |  |  |
| DFin | AFin |  |  |  |

**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 тяп2

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

textBox1.AppendText("abcd");

}

private void button1\_Click(object sender, EventArgs e)

{

uLex Lex = new uLex();

Lex.strPSource = textBox1.Lines;

Lex.strPMessage = textBox2.Lines;

Lex.enumPState = TState.Start;

textBox2.Text = "";

try

{

Lex.GetSymbol();

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

String s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier: { s = "Identifier"; break; }

case TToken.lxmNumber: { s = "Numder"; break; }

case TToken.lxmEmpty: { s = "EndText"; break; }

}

textBox2.Text += "(" + Lex.strPLexicalUnit + "," + s + ")";

}

textBox2.Text = "Текст верный";

}

catch (Exception exc)

{

textBox2.Text += exc.Message;

textBox1.Select();

textBox1.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Lex.intPSourceRowSelection; i++) n += textBox1.Lines[i].Length + 2;

n += Lex.intPSourceColSelection;

textBox1.SelectionLength = n;

}

}

**uLex.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace тяп2

{

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

public enum TCharType { Letter, Digit, EndRow, EndText, Space, ReservedSymbol, opBracket, clBracket, expMark, comma, semicolon }; // тип символа

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

class uLex

{

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

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

public TCharType enumFSelectionCharType;

public char chrFSelection;

private TState enumFState;

private int intFSourceRowSelection;

private int intFSourceColSelection;

private String strFLexicalUnit;

private TToken enumFToken;

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

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

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

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

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

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

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

public uLex()

{

}

public void GetSymbol() //метод класса лексический анализатор

{

// 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 == '(') enumFSelectionCharType = TCharType.opBracket;

else if (chrFSelection == ')') enumFSelectionCharType = TCharType.clBracket;

else if (chrFSelection == '!') enumFSelectionCharType = TCharType.expMark;

else if (chrFSelection == ',') enumFSelectionCharType = TCharType.comma;

else if (chrFSelection == ';') enumFSelectionCharType = TCharType.semicolon;

//else if (chrFSelection == '(' || chrFSelection == ')' || chrFSelection == ':' || chrFSelection == '-' || chrFSelection == ',' || chrFSelection == '.') enumFSelectionCharType = TCharType.ReservedSymbol;

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

enumFState = TState.Continue;

}

intFSourceColSelection++;

}

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();

}

// Вариант 19

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// AFin | AFin | BFin | CFin | DFin |

// BFin | AFin | BFin | CFin | |

// CFin | AFin | BFin | | |

// DFin | AFin | | | |

DFin:

{

if (chrFSelection == 'a')

{

TakeSymbol();

goto AFin;

}

else if (chrFSelection == 'b')

{

TakeSymbol();

goto BFin;

}

else if (chrFSelection == 'c')

{

TakeSymbol();

goto CFin;

}

else if (chrFSelection == 'd')

{

TakeSymbol();

goto DFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

CFin:

{

if (chrFSelection == 'd')

{

throw new Exception("Ожидался b или c или a");

}

if (chrFSelection == 'b')

{

TakeSymbol();

goto BFin;

}

else if (chrFSelection == 'c')

{

TakeSymbol();

goto CFin;

}

else if (chrFSelection == 'a')

{

TakeSymbol();

goto AFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

BFin:

{

if (chrFSelection == 'd')

{

throw new Exception("Ожидался a или b");

}

else if (chrFSelection == 'c')

{

throw new Exception("Ожидался a или b");

}

if (chrFSelection == 'b')

{

TakeSymbol();

goto BFin;

}

else if (chrFSelection == 'a')

{

TakeSymbol();

goto AFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

AFin:

{

if (chrFSelection == 'd')

{

throw new Exception("Ожидался a");

}

else if (chrFSelection == 'b')

{

throw new Exception("Ожидался a");

}

else if (chrFSelection == 'c')

{

throw new Exception("Ожидался a");

}

if (chrFSelection == 'a')

{

TakeSymbol();

goto DFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

case TCharType.Digit:

{

// 0 1

// A | B | K |

// B | | C |

// C | A | |

// K | | D |

// D | EFin | |

// EFin | | F |

// F | | G |

// G | | EFin |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto B;

}

if (chrFSelection == '1')

{

TakeSymbol();

goto K;

}

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

B:

if (chrFSelection == '1')

{

TakeSymbol();

goto C;

}

else throw new Exception("Ожидался 1");

C:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидался 0");

K:

if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

else throw new Exception("Ожидался 1");

D:

if (chrFSelection == '0')

{

TakeSymbol();

goto EFin;

}

else throw new Exception("Ожидался 0");

EFin:

if (chrFSelection == '1')

{

TakeSymbol();

goto F;

}

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

else throw new Exception("Ожидался 0");

F:

if (chrFSelection == '1')

{

TakeSymbol();

goto G;

}

else throw new Exception("Ожидался 1");

G:

if (chrFSelection == '1')

{

TakeSymbol();

goto EFin;

}

else throw new Exception("Ожидался 1");

}

case TCharType.ReservedSymbol:

{

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

case TCharType.Space:

{

GetSymbol();

break;

}

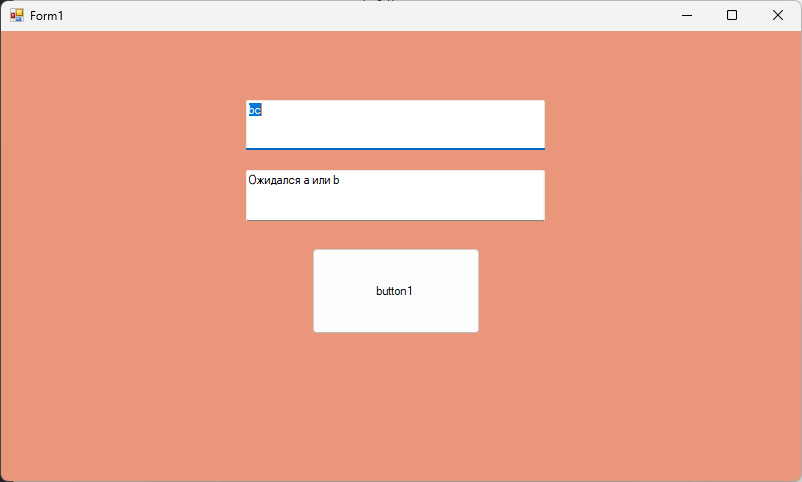
}

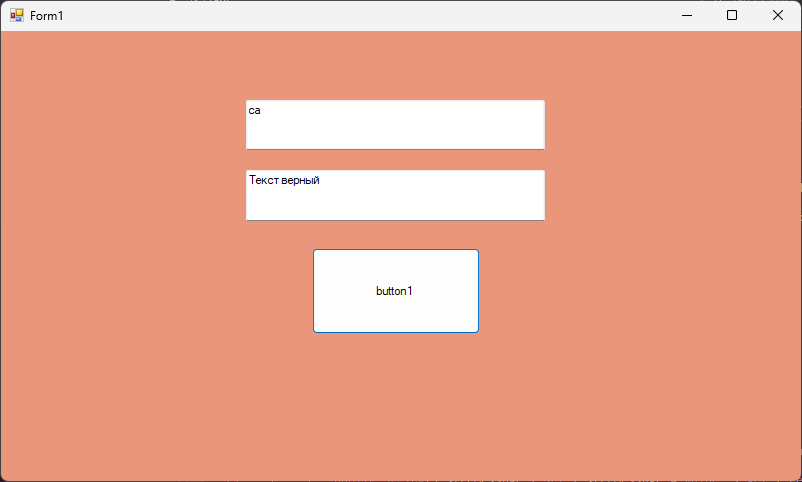
}

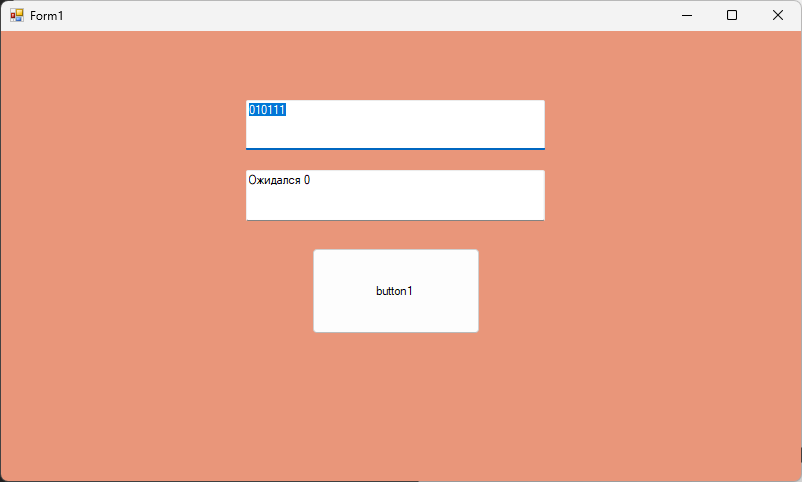
}

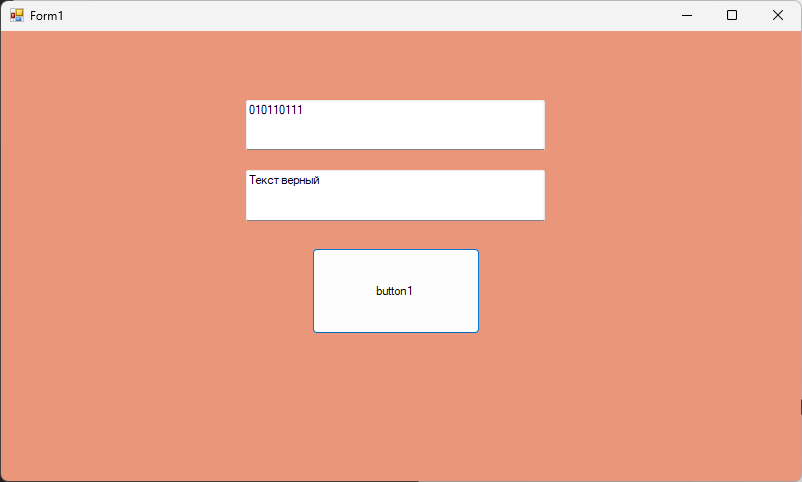
}

**Результаты выполнения:**









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

**Теория**

*Синтаксический анализатор* — это программа или часть программы, которая анализирует входной поток данных (обычно текст), проверяя его на соответствие определенной грамматике.

*Левая рекурсия* — это ситуация, когда правило в формальной грамматике ссылается на само себя в первом символе правой части. Такая рекурсия вызывает проблемы при анализе кода, так как может привести к бесконечному циклу.

*Чтобы избавиться от левой рекурсии*, ее можно заменить на правую с помощью специального алгоритма. Один из наиболее распространенных — это алгоритм, предложенный Ахо и Ульманом.

Предположим, что у вас есть грамматика, которая содержит левую рекурсию:

A -> Aα | β

где A — это некоторый нетерминальный символ, α и β — последовательности терминальных и нетерминальных символов, и α не может быть пустой.

Алгоритм Ахо и Ульмана удаляет левую рекурсию, заменяя эту грамматику следующим образом:

A -> βA'

A' -> αA' | ε

где ε обозначает пустую строку.

Этот алгоритм применим для прямой левой рекурсии. Однако, если грамматика содержит косвенную левую рекурсию, алгоритм должен быть модифицирован или применен в сочетании с другими подходами.

**Текст задания:**

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

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

Указания:

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

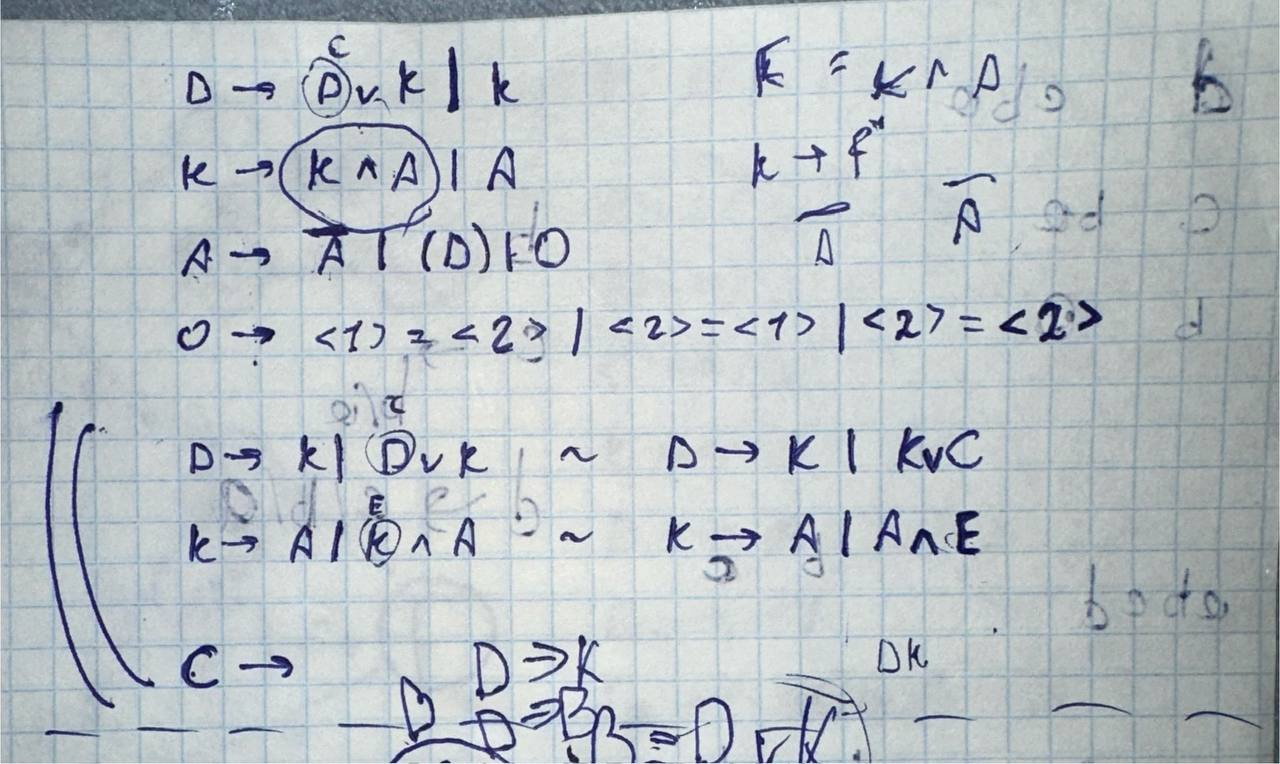
2. Оформить синтаксический анализатор в виде процедуры, функции или класса, которые при обращении обрабатывают весь исходный текст.

3. Если грамматика леворекурсивная, то устранить левую рекурсию.

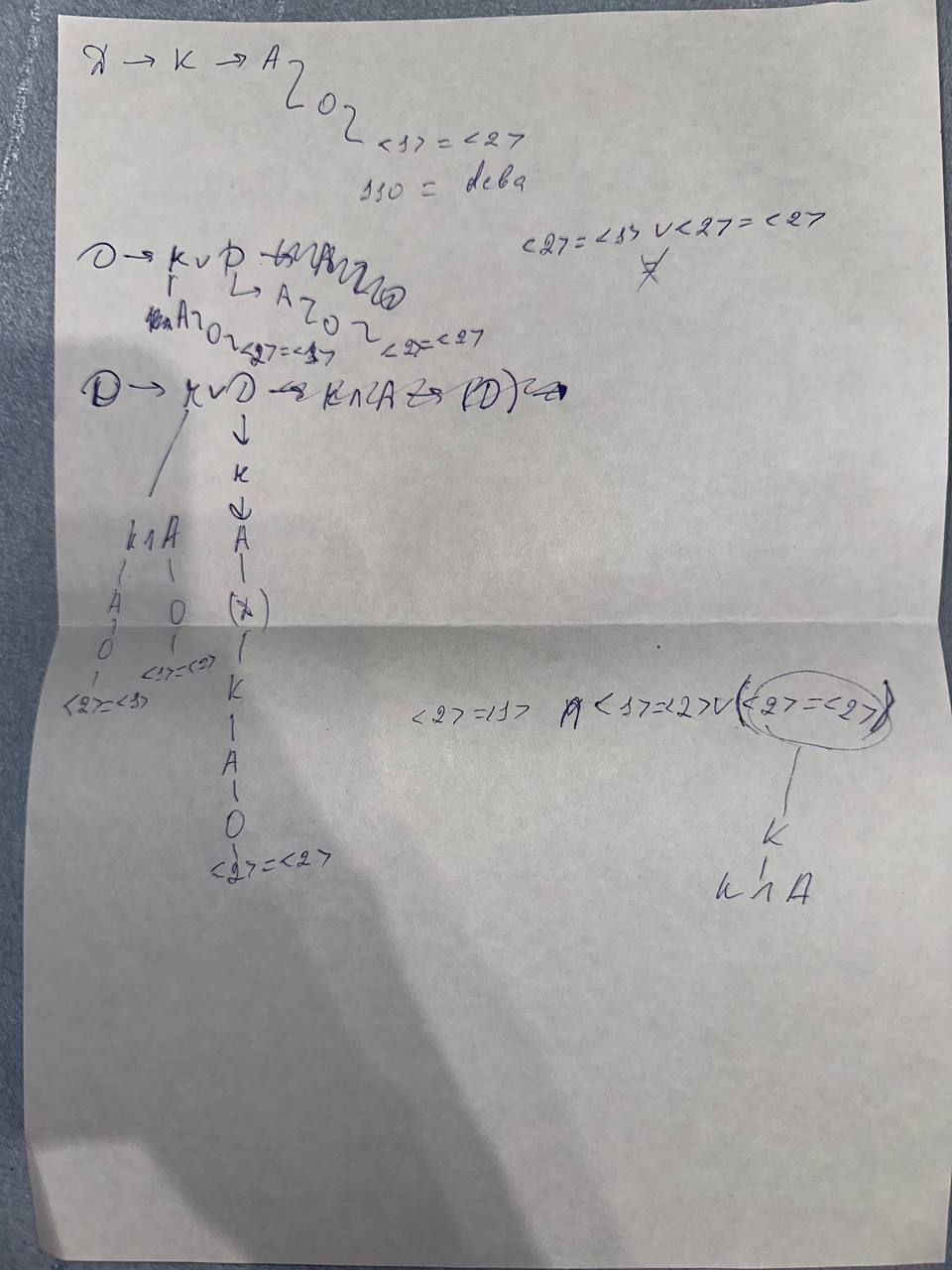
4. При обнаружении лексической ошибки целесообразно возбуждать исключительную ситуацию, которая будет обрабатываться в главной форме программы.

**Грамматика (вариант 19):**

**Избавление от левой рекурсии:**



Далее рассматриваем все варианты составления предложения от самого простого до самого легкого:



**Код программы:**

**Form1.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using nsSynt;

using nsLex;

namespace тяп\_лаба\_3

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

tbFSource.AppendText("c=010110111" + "\r\n");

int n = tbFSource.Lines.Length;

}

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.D();

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 Form1\_Load(object sender, EventArgs e)

{

}

}

}

**uLex.cs:**

using System;

using System.Collections.Generic;

using System.Text;

namespace nsLex

{

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

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

public enum TToken

{

mn, minus, plus, lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty,

lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma, lxmdt, lxmText, lxmtz, lxmr, lxmrs, lxmls

};

public class CLex //класс лексический анализатор

{

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

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

public TCharType enumFSelectionCharType;

public char chrFSelection;

private TState enumFState;

private int intFSourceRowSelection;

private int intFSourceColSelection;

private String strFLexicalUnit;

private TToken enumFToken;

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

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

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

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

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

public int intPSourceRowSelection

{

get { return intFSourceRowSelection; }

set

{

intFSourceRowSelection = value;

}

}

public int intPSourceColSelection

{

get { return intFSourceColSelection; }

set

{

intFSourceColSelection = value;

}

}

public CLex()

{

}

public void GetSymbol() //метод класса лексический анализатор

{

intFSourceColSelection++; // продвигаем номер колонки

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

{

intFSourceRowSelection++;

if (intFSourceRowSelection <= strFSource.Length - 1)

{

intFSourceColSelection = -1;

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndRow;

enumFState = TState.Continue;

}

else

{

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndText;

enumFState = TState.Finish;

}

}

else

{

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 == '\*' || chrFSelection == '-') enumFSelectionCharType = TCharType.ReservedSymbol;

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

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

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

else if (chrFSelection == ';' || chrFSelection == ',' || chrFSelection == '[' || chrFSelection == ']' || chrFSelection == '=' || chrFSelection == ':') enumFSelectionCharType = TCharType.ReservedSymbol;

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

enumFState = TState.Continue;

}

}

private void TakeSymbol()

{

char[] c = { chrFSelection };

String s = new string(c);

strFLexicalUnit += s;

GetSymbol();

}

public void NextToken()

{

strFLexicalUnit = "";

if (enumFState == TState.Start)

{

intFSourceRowSelection = 0;

intFSourceColSelection = -1;

GetSymbol();

}

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

TCharType.EndRow)

{

GetSymbol();

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

switch (enumFSelectionCharType)

{

case TCharType.Letter: //если начинается слово из букв a,b,c,d

{

// d c b a

// AF | AF | BF | CF | DF |

// BF | | BF | CF | DF |

// CF | | | CF | DF |

// DF | | | | DF |

AF:

{

if (chrFSelection == 'd')

{

TakeSymbol();

goto AF;

}

if (chrFSelection == 'c')

{

TakeSymbol();

goto BF;

}

if (chrFSelection == 'b')

{

TakeSymbol();

goto CF;

}

if (chrFSelection == 'a')

{

TakeSymbol();

goto DF;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

BF:

{

if (chrFSelection == 'd')

{

throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");

}

if (chrFSelection == 'c')

{

TakeSymbol();

goto BF;

}

if (chrFSelection == 'b')

{

TakeSymbol();

goto CF;

}

if (chrFSelection == 'a')

{

TakeSymbol();

goto DF;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

CF:

{

if (chrFSelection == 'd')

{

throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");

}

if (chrFSelection == 'c')

{

throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");

}

if (chrFSelection == 'b')

{

TakeSymbol();

goto CF;

}

if (chrFSelection == 'a')

{

TakeSymbol();

goto DF;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

DF:

{

if (chrFSelection == 'd')

{

throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");

}

if (chrFSelection == 'c')

{

throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");

}

if (chrFSelection == 'b')

{

throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");

}

if (chrFSelection == 'a')

{

TakeSymbol();

goto DF;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

case TCharType.Digit: //если начинается слово из букв – цифр 0 и 1

{

// 0 1

// A | B | C |

// B | | D |

// C | | E |

// D | A | |

// E | Ffin | |

// Ffin | | G |

// G | | H |

// H | | Ffin |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto B;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto C;

}

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

B:

if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

else throw new Exception("Ожидался 1");

C:

if (chrFSelection == '1')

{

TakeSymbol();

goto E;

}

else throw new Exception("Ожидался 1");

D:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидался 0");

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("Ожидался 1");

G:

if (chrFSelection == '1')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 1");

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.plus;

GetSymbol();

return;

}

if (chrFSelection == '\*')

{

enumFToken = TToken.mn;

GetSymbol();

return;

}

if (chrFSelection == '-')

{

enumFToken = TToken.minus;

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;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

}

}

}

**uSyntAnalyzer.cs**

using System;

using System.Collections.Generic;

using System.Text;

using System.Windows.Forms;

using nsLex;

using тяп\_лаба\_3;

namespace nsSynt

{

class uSyntAnalyzer

{

private String[] strFSource;

private String[] strFMessage;

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

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

public CLex Lex = new CLex();

public void D()

{

//string s = Convert.ToString(Lex.enumPToken);

//MessageBox.Show(s);

//Lex.NextToken();

K();

// Lex.NextToken();

if (Lex.enumPToken == TToken.plus)

{

C();

}

}

public void C()

{

if (Lex.enumPToken == TToken.plus)

{

Lex.NextToken();

K();

if (Lex.enumPToken == TToken.plus)

{

C();

}

}

//else throw new Exception("Ожидалcя +");

}

public void K()

{

A();

//Lex.NextToken();

if (Lex.enumPToken == TToken.mn)

{

B();

}

}

public void B()

{

if (Lex.enumPToken == TToken.mn)

{

Lex.NextToken();

A();

//Lex.NextToken();

if (Lex.enumPToken == TToken.mn)

{

B();

}

}

}

public void A()

{

if (Lex.enumPToken == TToken.minus)

{

Lex.NextToken();

A();

}

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

{

Lex.NextToken();

D();

//Lex.NextToken();

if (Lex.enumPToken == TToken.lxmRightParenth)

{

Lex.NextToken();

// throw new Exception("Konec slova");

}

else throw new Exception("Ожидалась )");

}

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

{

O();

}

else

{

throw new Exception("Ожидались - или ( или число");

}

}

public void O()

{

if (Lex.enumPToken == TToken.lxmNumber)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmr)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier)

{

//throw new Exception("Konec slova");

}

else throw new Exception("ожидалось число");

}

else throw new Exception("ожидалось =");

}

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

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmr)

{

Lex.NextToken();

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

{

//throw new Exception("Konec slova");

}

else throw new Exception("ожидалось число или слово");

}

else throw new Exception("ожидалось =");

}

else throw new Exception("Ожидалось число или слово");

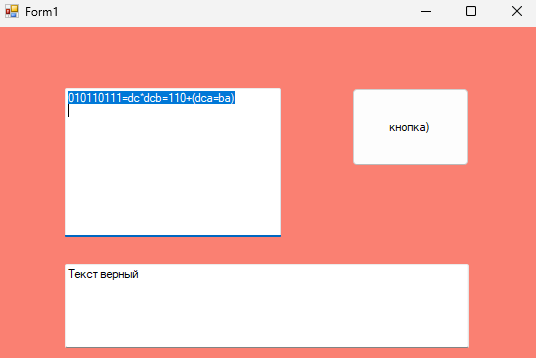
Lex.NextToken();

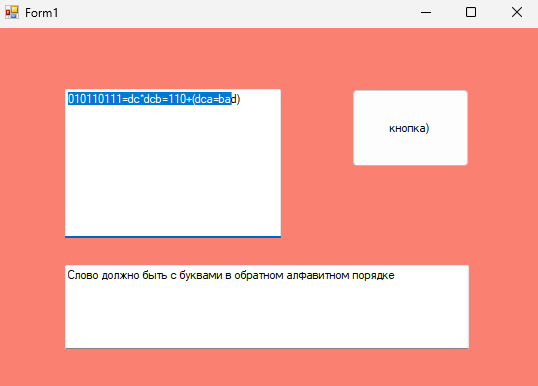
}

}

}

**Результаты работы программы:**



**Лабораторная работа № 4.**

**Введение табличного способа хранения слов**

**Теория**

*Коллизия* — это ситуация, когда два или более объекта имеют одинаковый хеш-код, но не являются идентичными. Как результат, они помещаются в одну и ту же ячейку хеш-таблицы, что приводит к конфликту.

В контексте хеш-функций, коллизия происходит, когда два разных входных значения дают одинаковый хеш.

*Чтобы избавиться от коллизии,* необходимо использовать правильный метод хеширования и уникальный алгоритм. Есть несколько подходов:

1. Линейное разрешение коллизий (линейное пробирование): В этом методе, если происходит коллизия, то следующая ячейка массива проверяется до тех пор, пока не будет найдена свободная ячейка. Если хеш-таблица окажется полностью заполненной, следовательно, будет невозможно добавлять в неё новые элементы. Так что при возникновении такой ситуации решением может быть динамическое увеличение размера хеш-таблицы, с одновременной её перестройкой.
2. Двойное хеширование: Этот метод борьбы с коллизиями основан на использовании двух хеш-функций для построения различных последовательностей исследования хеш-таблицы.
3. Метод цепочек: В этом методе, каждая ячейка хеш-таблицы представляет собой связный список всех элементов, хеш-функция которых возвращает адрес данной ячейки. В случае коллизии, новый элемент просто добавляется в конец списка.
4. Удаление элемента без пометок: если происходит коллизия при удалении элемента, то можно сдвигать все последующие элементы на определенное количество позиций назад. В этом случае важно учесть, что элементы с другим хешем должны оставаться на своем месте и в цепочке не должно быть "дырок".

Хеш-таблица - структура данных, которая позволяет хранить пары (ключ, значение) и выполнять три операции: вставку новой пары, поиск пары по ключу и удаление пары по ключу.

Хеш-таблица основана на принципе деления ключа на некоторые остатки, которые определяют местоположение ключа в таблице. Каждый ключ имеет свой уникальный хеш-код, который преобразуется в индекс в таблице.

Если два ключа имеют одинаковый хеш-код (т.е. они хешируются в одну и ту же позицию), то возникает коллизия. Для разрешения коллизий используются различные методы, такие как открытая адресация, квадратичное пробирование и двойное хеширование.

Хеш-таблицы имеют высокую скорость поиска и вставки, но у них есть проблема с коллизиями, которые могут снижать эффективность работы таблицы. Однако, правильный выбор хеширования и разрешение коллизий могут значительно улучшить производительность хеш-таблицы.

**Текст задания:**

Синтаксический анализатор размещает распознанные с помощью лексического анализатора слова в таблицы. Количество таблиц соответствует числу типов слов. В нашем случае нужны три таблицы: таблица идентификаторов (первый тип слова), таблица чисел (второй тип слова), таблица составленных из специальных символов слов (третий тип слова).

Таблицы организуются методом хеширования. Включить класс «Поиск слова в таблице методом хеширования» в разрабатываемый транслятор.

План работы:

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 представляет собой реализацию хэш-функции и метода рехэширования.

Хэш-функция HashFunctionWainberger принимает на вход строку strALexicalUnit и возвращает UInt32 значение. В начале устанавливается начальное значение h = 0. Затем происходит цикл, в котором происходит суммирование значений символов строки strALexicalUnit, приведенных к типу UInt32, по модулю 12. Полученное значение h возвращается как результат хэш-функции.

Метод ReHashFunctionLine принимает на вход значение h и строку strALexicalUnit, а затем осуществляет рехэширование для разрешения коллизий.

Если начальное значение h равно 0, оно устанавливается равным трети от размера массива arrFHashTable, а если h равно 1, оно устанавливается равным трем четвертям от размера массива arrFHashTable.

Затем инициализируются переменные i и hi, и устанавливается флаг boolVFinish в false.

Далее выполняется цикл do-while. Внутри цикла проверяется условие: если значение arrFHashTablehi равно 0, то boolVFinish устанавливается в true, иначе, если значение objFHeap.arrFHeapTablearrFHashTable[hi].strFLexicalUnit равно strALexicalUnit, также устанавливается boolVFinish в true. В противном случае инкрементируется переменная i, и вычисляется новое значение hi для рехэширования. Цикл продолжается, пока boolVFinish не станет true. Затем возвращается значение hi.

Таким образом, этот код реализует хэш-функцию и метод рехэширования для обработки коллизий.

**Код программы:**

**Form1.cs:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using nsSynt;

using nsLex;

namespace тяп\_лаба\_3

{

public partial class Form1 : Form

{

public CHashTableList htl = new CHashTableList(2);

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

{

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

listBox1.Items.Clear();

listBox2.Items.Clear();

htl.TableToStringList(0, listTable);

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

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

listTable.Clear();

htl.TableToStringList(1, listTable);

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

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

listTable.Clear();

}

public Form1()

{

InitializeComponent();

tbFSource.AppendText("(cba=cba)\*-010110111=cba+cba=cba+cba=010110111" + "\r\n");

int n = tbFSource.Lines.Length;

}

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.D();

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;

}

}

Кнопка удаления слова

private void button4\_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) проверка: слово - номер или идентификатор

{

case TToken.lxmIdentifier: если идентификатор

{

s1 = "id " + Lex.strPLexicalUnit;

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 0); слово удаляется из таблицы идентификаторов

// htl.DeleteLexicalCode(0);

TablesToMemo(this, e);

break;

}

case TToken.lxmNumber: если номер

{

s1 = "num " + Lex.strPLexicalUnit;

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 1); слово удаляется из таблицы номеров

// htl.DeleteLexicalCode(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;

}

}

Кнопка замены

private void button5\_Click(object sender, EventArgs e)

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource1.Lines;

Lex.strPMessage = tbFMessage1.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = tbFSource1.TextLength;

CLex Lex2 = new CLex();

Lex2.strPSource = tbFSource2.Lines;

Lex2.strPMessage = tbFMessage1.Lines;

Lex2.intPSourceColSelection = 0;

Lex2.intPSourceRowSelection = 0;

int x2 = tbFSource2.TextLength;

tbFMessage1.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

switch (Lex.enumPToken) Проверка 1-го слова на то, является ли оно номером или идентификатором

{

case TToken.lxmIdentifier: Если идентификатор то:

{

int b = 0;

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

{

while (Lex2.enumPState != TState.Finish)

{

Lex2.NextToken();

switch (Lex2.enumPToken) Проверка 1-го слова на то, является ли оно номером или идентификатором

{

case TToken.lxmIdentifier: Если является идентификатором, то:

{

int b1 = 0;

if (htl.AddLexicalUnit(Lex2.strPLexicalUnit, 0, ref b1)) вызывается функция добавления в таблицу идентификаторов 2-го слова

{

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 0); Удаляется 1-ое слово

TablesToMemo(this, e);

}

tbFMessage1.AppendText("(Идентификатор был изменен)");

break;

}

case TToken.lxmNumber: если 2-ое слово - номер

{

tbFMessage1.AppendText("(Нельзя заменить идентификатор на номер)");

break;

}

}

}

}

else tbFMessage1.AppendText("(Идентификатор для удаления не найден)");

break;

}

case TToken.lxmNumber: если 1-ое слово номер

{

int b = 0;

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

{

while (Lex2.enumPState != TState.Finish)

{

Lex2.NextToken();

switch (Lex2.enumPToken) проверка 2-го слова

{

case TToken.lxmIdentifier: если 2-ое слово идентификатор

{

tbFMessage1.AppendText("(Нельзя заменить номер на идентификатор)");

break;

}

case TToken.lxmNumber: если 2-ое слово номер

{

int b1 = 0;

if (htl.AddLexicalUnit(Lex2.strPLexicalUnit, 1, ref b1)) 1-ое слово добавляется в таблицу номеров

{

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 1); 2-ое слово удаляется

TablesToMemo(this, e);

}

tbFMessage1.AppendText("(Номер был изменен)");

break;

}

}

}

}

else tbFMessage1.AppendText("(Номер для удаления не найден)");

break;

}

}

}

}

catch (Exception exc)

{

tbFMessage1.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n1 = 0;

for (int i = 0; i < Lex.intPSourceRowSelection; i++) n1 += tbFSource.Lines[i].Length + 2;

n1 += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n1;

}

}

Кнопка добавления слов в таблицы

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

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

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

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 0, ref b)) Вызывается функция добавления слова в таблицу идентификаторов (0)

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

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

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 1, ref b)) Вызывается функция добавления слова в таблицу номеров (1)

{

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;

}

}

Кнопка посика слова в таблицах

private void button3\_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();

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

int b = 0;

if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, 0, ref b)) Вызов функции поиска, возвращает значение true если было найден такой идентификатор, иначе false

{

tbFMessage2.AppendText("(Найден идентификатор)");

}

else tbFMessage2.AppendText("(Идентификатор не найден)");

break;

}

case TToken.lxmNumber:

{

int b = 0;

if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, 1, ref b)) Вызов функции поиска, возвращает значение true если было найден такой номер, иначе false

{

tbFMessage2.AppendText("(Найден номер)");

}

else tbFMessage2.AppendText("(Номер не найден)");

break;

}

}

}

}

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.Text;

namespace nsLex

{

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

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

public enum TToken

{

mn, minus, plus, lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty,

lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma, lxmdt, lxmText, lxmtz, lxmr, lxmrs, lxmls

};

public class CLex //класс лексический анализатор

{

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

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

public TCharType enumFSelectionCharType;

public char chrFSelection;

private TState enumFState;

private int intFSourceRowSelection;

private int intFSourceColSelection;

private String strFLexicalUnit;

private TToken enumFToken;

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

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

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

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

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

public int intPSourceRowSelection

{

get { return intFSourceRowSelection; }

set

{

intFSourceRowSelection = value;

}

}

public int intPSourceColSelection

{

get { return intFSourceColSelection; }

set

{

intFSourceColSelection = value;

}

}

public CLex()

{

}

public void GetSymbol() //метод класса лексический анализатор

{

intFSourceColSelection++; // продвигаем номер колонки

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

{

intFSourceRowSelection++;

if (intFSourceRowSelection <= strFSource.Length - 1)

{

intFSourceColSelection = -1;

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndRow;

enumFState = TState.Continue;

}

else

{

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndText;

enumFState = TState.Finish;

}

}

else

{

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 == '\*' || chrFSelection == '-') enumFSelectionCharType =TCharType.ReservedSymbol;

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

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

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

else if (chrFSelection == ';' || chrFSelection == ',' || chrFSelection == '[' || chrFSelection == ']' || chrFSelection == '=' || chrFSelection == ':') enumFSelectionCharType = TCharType.ReservedSymbol;

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

enumFState = TState.Continue;

}

}

private void TakeSymbol()

{

char[] c = { chrFSelection };

String s = new string(c);

strFLexicalUnit += s;

GetSymbol();

}

public void NextToken()

{

strFLexicalUnit = "";

if (enumFState == TState.Start)

{

intFSourceRowSelection = 0;

intFSourceColSelection = -1;

GetSymbol();

}

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

{

GetSymbol();

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

switch (enumFSelectionCharType)

{

case TCharType.Letter: //если начинается слово из букв a,b,c,d

{

// d c b a

// AF | AF | BF | CF | DF |

// BF | | BF | CF | DF |

// CF | | | CF | DF |

// DF | | | | DF |

AF:

{

if (chrFSelection == 'd')

{

TakeSymbol();

goto AF;

}

if (chrFSelection == 'c')

{

TakeSymbol();

goto BF;

}

if (chrFSelection == 'b')

{

TakeSymbol();

goto CF;

}

if (chrFSelection == 'a')

{

TakeSymbol();

goto DF;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

BF:

{

if (chrFSelection == 'd')

{

throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");

}

if (chrFSelection == 'c')

{

TakeSymbol();

goto BF;

}

if (chrFSelection == 'b')

{

TakeSymbol();

goto CF;

}

if (chrFSelection == 'a')

{

TakeSymbol();

goto DF;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

CF:

{

if (chrFSelection == 'd')

{

throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");

}

if (chrFSelection == 'c')

{

throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");

}

if (chrFSelection == 'b')

{

TakeSymbol();

goto CF;

}

if (chrFSelection == 'a')

{

TakeSymbol();

goto DF;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

DF:

{

if (chrFSelection == 'd')

{

throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");

}

if (chrFSelection == 'c')

{

throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");

}

if (chrFSelection == 'b')

{

throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");

}

if (chrFSelection == 'a')

{

TakeSymbol();

goto DF;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

case TCharType.Digit: //если начинается слово из букв – цифр 0 и 1

{

// 0 1

// A | B | C |

// B | | D |

// C | | E |

// D | A | |

// E | Ffin | |

// Ffin | | G |

// G | | H |

// H | | Ffin |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto B;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto C;

}

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

B:

if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

else throw new Exception("Ожидался 1");

C:

if (chrFSelection == '1')

{

TakeSymbol();

goto E;

}

else throw new Exception("Ожидался 1");

D:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидался 0");

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("Ожидался 1");

G:

if (chrFSelection == '1')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 1");

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 == '(')

{

strFLexicalUnit = "(";

enumFToken = TToken.lxmLeftParenth;

GetSymbol();

return;

}

if (chrFSelection == '+')

{

strFLexicalUnit = "+";

enumFToken = TToken.plus;

GetSymbol();

return;

}

if (chrFSelection == '\*')

{

strFLexicalUnit = "\*";

enumFToken = TToken.mn;

GetSymbol();

return;

}

if (chrFSelection == '-')

{

enumFToken = TToken.minus;

GetSymbol();

return;

}

if (chrFSelection == ')')

{

strFLexicalUnit = ")";

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 == '=')

{

strFLexicalUnit = "=";

enumFToken = TToken.lxmr;

GetSymbol();

return;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

}

}

}

}

**uSyntAnalyzer.cs:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Windows.Forms;

using nsLex;

using тяп\_лаба\_3;

namespace nsSynt

{

class uSyntAnalyzer

{

private String[] strFSource;

private String[] strFMessage;

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

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

public CLex Lex = new CLex();

public void D()

{

//string s = Convert.ToString(Lex.enumPToken);

//MessageBox.Show(s);

//Lex.NextToken();

K();

// Lex.NextToken();

if (Lex.enumPToken == TToken.plus)

{

C();

}

}

public void C()

{

if (Lex.enumPToken == TToken.plus)

{

Lex.NextToken();

K();

if (Lex.enumPToken == TToken.plus)

{

C();

}

}

//else throw new Exception("Ожидалcя +");

}

public void K()

{

A();

//Lex.NextToken();

if (Lex.enumPToken == TToken.mn)

{

B();

}

}

public void B()

{

if (Lex.enumPToken == TToken.mn)

{

Lex.NextToken();

A();

//Lex.NextToken();

if (Lex.enumPToken == TToken.mn)

{

B();

}

}

}

public void A()

{

if (Lex.enumPToken == TToken.minus)

{

Lex.NextToken();

A();

}

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

{

Lex.NextToken();

D();

//Lex.NextToken();

if (Lex.enumPToken == TToken.lxmRightParenth)

{

Lex.NextToken();

// throw new Exception("Konec slova");

}

else throw new Exception("Ожидалась )");

}

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

{

O();

}

else

{

string s = Convert.ToString(Lex.enumPToken);

MessageBox.Show(s);

throw new Exception("Ожидались - или ( или число");

}

}

public void O()

{

if (Lex.enumPToken == TToken.lxmNumber)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmr)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier)

{

//throw new Exception("Konec slova");

}

else throw new Exception("ожидалось число");

}

else throw new Exception("ожидалось =");

}

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

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmr)

{

Lex.NextToken();

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

{

//throw new Exception("Konec slova");

}

else throw new Exception("ожидалось число или слово");

}

else throw new Exception("ожидалось =");

}

else throw new Exception("Ожидалось число или слово");

Lex.NextToken();

}

}

}

**THashTable.cs:**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace nsSynt

{

public class THashTable

{

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

private int intFCurrentPrimeNumber;

private int intFItemReserve;

private bool boolIsSaved;

public int intFHashIndex;

public int cardPTableSize { get { return arrFHashTable.Count; } }

public List<object> arrFUserTable = null;

static THeap objFHeap;

public THashTable(ref THeap objAHeap)

{

objFHeap = objAHeap;

Init(7);

intFItemReserve = 0;

}

public void Init(int count)

{

arrFHashTable.Clear();

Resize(arrFHashTable, count);

intFCurrentPrimeNumber = count;

}

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

{

if (size > list.Count)

while (size > list.Count)

list.Add(new object());

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

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

{

if (size > list.Count)

while (size > list.Count)

list.Add(new Int32());

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

int NextPrimeNumber(int cardAOldPrimeNumber)

{

int intVLowerBound, intVUpperBound, intVNextPrimeNumber;

bool boolVIsDivisor;

intVNextPrimeNumber = cardAOldPrimeNumber + cardAOldPrimeNumber / 10 + 1; // увеличиваем на 10 процентов

if ((intVNextPrimeNumber % 2) == 0) intVNextPrimeNumber++;

do

{

boolVIsDivisor = true; intVNextPrimeNumber = intVNextPrimeNumber + 2;

intVLowerBound = 3; intVUpperBound = intVNextPrimeNumber / 3 + 1; // диапазон делителей

while (boolVIsDivisor && (intVLowerBound < intVUpperBound))

{

if ((intVNextPrimeNumber % intVLowerBound) == 0) boolVIsDivisor = false;

else intVLowerBound = intVLowerBound + 2;

}

} while (!boolVIsDivisor);

return intVNextPrimeNumber;

}

//HashFunc

UInt32 HashFunction\_Wainberger(string strALexicalUnit)

{

UInt32 h = 0/\*, seed=131313\*/;

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

{

h += Convert.ToUInt32(strALexicalUnit[i] % 12); //Хэширование

}

return h;

}

int ReHashFunction\_Line(int h, string strALexicalUnit)

{

if (h == 0) h = arrFHashTable.Count / 3;

else if (h == 1) h = arrFHashTable.Count \* 3 / 4;

int i = 1, hi = h;

bool boolVFinish = false;

do

{

if (arrFHashTable[hi] == 0) boolVFinish = true;

else

if (objFHeap.arrFHeapTable[arrFHashTable[hi]].strFLexicalUnit == strALexicalUnit)

boolVFinish = true;

else

{

i++;

hi = (h % i) % (Int32)(arrFHashTable.Count); //Рехэширование(это метод для избавления от коллизии)

}

} while (!boolVFinish);

return hi;

}

public void HashIndex(string strALexicalUnit)

{

int h;

h = (Int32)HashFunction\_Wainberger(strALexicalUnit) % (Int32)(arrFHashTable.Count);

intFHashIndex = ReHashFunction\_Line(h, strALexicalUnit);

}

void TableReHashing()

{

int i, j;

List<int> cardarrVHashTableImage = new List<int>();

List<object> arrVUserTableImage = new List<object>();

Resize(cardarrVHashTableImage, arrFHashTable.Count);

if (arrFUserTable != null)

Resize(arrVUserTableImage, arrFHashTable.Count);

for (i = 0; i < arrFHashTable.Count; i++)

{

cardarrVHashTableImage[i] = arrFHashTable[i];

if (arrFUserTable != null) arrVUserTableImage[i] = arrFUserTable[i];

}

arrFHashTable.Clear();

if (arrFUserTable != null) arrFUserTable.Clear();

Resize(arrFHashTable, intFCurrentPrimeNumber);

if (arrFUserTable != null) Resize(arrFUserTable, intFCurrentPrimeNumber);

for (i = 0; i < cardarrVHashTableImage.Count; i++)

{

if (cardarrVHashTableImage[i] != 0)

{

j = cardarrVHashTableImage[i];

HashIndex(objFHeap.arrFHeapTable[j].strFLexicalUnit);

arrFHashTable[intFHashIndex] = j;

if (arrFUserTable != null) arrFUserTable[intFHashIndex] = arrVUserTableImage[i];

THeapItem Th2 = objFHeap.arrFHeapTable[j];

Th2.intFHashIndex = intFHashIndex;

objFHeap.arrFHeapTable[j] = Th2;

}

}

cardarrVHashTableImage.Clear();

if (arrFUserTable != null) arrVUserTableImage.Clear();

}

void Expansion()

{

intFCurrentPrimeNumber = NextPrimeNumber(intFCurrentPrimeNumber);

TableReHashing();

}

object GetUserPointer(int cardILexicalCode)

{

THeapItem Item = objFHeap.arrFHeapTable[cardILexicalCode];

if (Item.intFHashIndex >= cardPTableSize)

{

MessageBox.Show("Индекс пользовательского массива вышел за диапазон!");

return null;

}

else

{

return arrFUserTable[objFHeap.arrFHeapTable[cardILexicalCode].intFHashIndex];

}

}

void SetUserPointer(int cardILexicalCode, object ptrANewPoint)

{

if (objFHeap.arrFHeapTable[cardILexicalCode].intFHashIndex >= cardPTableSize)

MessageBox.Show("Индекс пользовательского массива вышел за диапазон!");

else

arrFUserTable[objFHeap.arrFHeapTable[cardILexicalCode].intFHashIndex] = ptrANewPoint;

}

public void SetUserTable()

{

arrFUserTable = new List<object>();

Resize(arrFUserTable, arrFHashTable.Count);

}

public bool SearchLexicalUnit(string strAlexicalUnit, ref int intALexicalCode) //Поиск

{

HashIndex(strAlexicalUnit);

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

else

{

intALexicalCode = arrFHashTable[intFHashIndex];

return true;

}

}

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

{

HashIndex(strALexicalUnit);

if (arrFHashTable[intFHashIndex] != 0)

{

intALexicalCode = arrFHashTable[intFHashIndex];

return true;

}

else

{

if ((intFItemReserve + 2) > (cardPTableSize \* 0.9))

{

Expansion();

HashIndex(strALexicalUnit);

}

objFHeap.AddLexicalUnit(strALexicalUnit, byteAHashTable, intFHashIndex, ref intALexicalCode);

arrFHashTable[intFHashIndex] = intALexicalCode;

intFItemReserve++;

return false;

}

}

public void DeleteLexicalUnit(string strAlexicalUnit)

{

HashIndex(strAlexicalUnit);

if (arrFHashTable[intFHashIndex] != 0)

{

if (arrFUserTable != null)

{

if (arrFUserTable[intFHashIndex] != null)

MessageBox.Show("Удаление из таблицы связанного данного");

else

{

objFHeap.DeleteLexicalUnit(arrFHashTable[intFHashIndex]);

arrFHashTable[intFHashIndex] = 0;

intFItemReserve--;

TableReHashing();

}

}

else

{

objFHeap.DeleteLexicalUnit(arrFHashTable[intFHashIndex]);

arrFHashTable[intFHashIndex] = 0;

intFItemReserve--;

TableReHashing();

}

}

}

public void DeleteLexicalCode(int cardALexicalCode)

{

int VHashIndex;

VHashIndex = objFHeap.arrFHeapTable[cardALexicalCode].intFHashIndex;

if (arrFHashTable[VHashIndex] != 0)

if (arrFUserTable.Count != 0)

if (arrFUserTable[VHashIndex] != null)

MessageBox.Show("Удаление из таблицы связанного данного");

else

{

objFHeap.DeleteLexicalUnit(cardALexicalCode);

arrFHashTable[VHashIndex] = 0;

intFItemReserve--;

TableReHashing();

}

else

{

objFHeap.DeleteLexicalUnit(cardALexicalCode);

arrFHashTable[VHashIndex] = 0;

intFItemReserve--;

TableReHashing();

}

}

public void Save(ref StreamWriter fl)

{

try

{

fl.WriteLine(cardPTableSize.ToString());

fl.WriteLine(intFItemReserve.ToString());

for (int i = 1; i < cardPTableSize; i++)

fl.Write("\t" + arrFHashTable[i].ToString());

fl.Write("\n");

boolIsSaved = true;

}

catch (InvalidCastException)

{ boolIsSaved = false; }

}

public void GetLexicalUnitList(ref List<string> sList)

{

for (int i = 0; i < arrFHashTable.Count; i++) if (arrFHashTable[i] != 0) sList.Add(objFHeap.arrFHeapTable[arrFHashTable[i]].strFLexicalUnit);

}

}

//HashTableList

public class CHashTableList

{

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

private bool boolFIsSaved;

public bool boolFIsLoaded;

private byte byteFTablesSize;

static THeap objFHeap = new THeap();

//------------------------------------------------------------------------------

public CHashTableList(byte byteATableCount)

{

this.byteFTablesSize = byteATableCount;

objFHeap = new THeap();

Resize(arrFHashTableList, byteATableCount);

}

//------------------------------------------------------------------------------

public byte GetTableNumber(int intALexicalCode)

{

return objFHeap.arrFHeapTable[intALexicalCode].byteFHashTable;

}

//------------------------------------------------------------------------------

public int GetTablesCount()

{

return arrFHashTableList.Count();

}

//------------------------------------------------------------------------------

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

{

if (size > list.Count)

while (size > list.Count)

{

list.Add(new THashTable(ref objFHeap));

}

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

//------------------------------------------------------------------------------

static void Resize(List<object> list, int intANewSize)

{

if (intANewSize > list.Count)

while (intANewSize > list.Count)

list.Add(new object());

else if (intANewSize < list.Count)

while (list.Count - intANewSize > 0)

list.RemoveAt(list.Count - 1);

}

//------------------------------------------------------------------------------

static void Resize(List<int> list, int intANewSize)

{

if (intANewSize > list.Count)

while (intANewSize > list.Count)

list.Add(new Int32());

else if (intANewSize < list.Count)

while (list.Count - intANewSize > 0)

list.RemoveAt(list.Count - 1);

}

//------------------------------------------------------------------------------

public object GetUserData(int intALexicalCode)

{

if ((0 < intALexicalCode) && (intALexicalCode < objFHeap.intPFreeItem))

return arrFHashTableList[GetTableNumber(intALexicalCode)].arrFUserTable[intALexicalCode];

else

{

MessageBox.Show("Неверно задан лексический код при чтении пользовательских данных");

return null;

}

}

//------------------------------------------------------------------------------

public void SetUserData(int intALexicalCode, object objAUserData)

{

if ((0 < intALexicalCode) && (intALexicalCode < objFHeap.intPFreeItem))

{

if (arrFHashTableList[GetTableNumber(intALexicalCode)].arrFUserTable.Count > 0)

arrFHashTableList[GetTableNumber(intALexicalCode)].arrFUserTable[intALexicalCode] = objAUserData;

else

MessageBox.Show("Попытка записи адреса в несозданный массив пользовательских данных!");

}

else MessageBox.Show("Неверно задан лексический код при записи пользовательских данных!");

}

//------------------------------------------------------------------------------

public string GetLexicalUnit(int intALexicalCode)

{

if ((0 < intALexicalCode) && (intALexicalCode < objFHeap.intPFreeItem)) return objFHeap.arrFHeapTable[intALexicalCode].strFLexicalUnit;

else

{

MessageBox.Show("Неверно задан лексический код при чтении пользовательских данных!");

return "";

}

}

//------------------------------------------------------------------------------

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

{

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

}

//------------------------------------------------------------------------------

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

{

if (byteATable >= arrFHashTableList.Count)

{

if (MessageBox.Show("Увеличить количество таблиц?", "Запрашиваемый индекс таблицы не существует.", MessageBoxButtons.YesNo) == DialogResult.Yes)

Resize(arrFHashTableList, byteATable + 1);

else

return false;

}

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

}

//------------------------------------------------------------------------------

public void DeleteLexicalUnit(string strALexicalUnit, byte byteATable)

{

arrFHashTableList[byteATable].DeleteLexicalUnit(strALexicalUnit);

}

//------------------------------------------------------------------------------

public void DeleteLexicalCode(int intALexicalCode)

{

short T = objFHeap.arrFHeapTable[intALexicalCode].byteFHashTable;

arrFHashTableList[T].DeleteLexicalCode(intALexicalCode);

}

//------------------------------------------------------------------------------

public void SetUserTable(byte byteATable)

{

arrFHashTableList[byteATable].SetUserTable();

}

//------------------------------------------------------------------------------

public void Expantion()

{

Resize(arrFHashTableList, ++byteFTablesSize);

}

//------------------------------------------------------------------------------

public void Save(string strAFileName)

{

try

{

StreamWriter fl = File.CreateText(strAFileName);

fl.WriteLine(byteFTablesSize.ToString());

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

fl.Write(arrFHashTableList[i].arrFHashTable.Count.ToString() + "\t");

fl.WriteLine("");

objFHeap.Save(ref fl);

boolFIsSaved = true;

fl.Close();

}

catch (InvalidDataException)

{ boolFIsSaved = false; }

}

//------------------------------------------------------------------------------

// отладка

public void HeapTableView(List<string> sList)

{

objFHeap.HeapTableView(sList);

}

//------------------------------------------------------------------------------

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

{

arrFHashTableList[byteATable].GetLexicalUnitList(ref sList);

}

//------------------------------------------------------------------------------

public int GetHashIndex(byte Table)

{

return arrFHashTableList[Table].intFHashIndex;

}

//------------------------------------------------------------------------------

}

public struct THeapItem

{

public string strFLexicalUnit;

public byte byteFHashTable;

public int intFHashIndex;

public THeapItem(string strALexicalUnit, byte byteATable, int intAHashIndex)

{

strFLexicalUnit = strALexicalUnit;

byteFHashTable = byteATable;

intFHashIndex = intAHashIndex;

}

}

public class THeap

{

public List<THeapItem> arrFHeapTable = new List<THeapItem>();

private List<int> arrFDeleted = new List<int>();

private int intFFreeItem;

bool boolIsSaved;

bool boolIsLoaded;

public bool boolPIsSaved { get { return boolIsSaved; } }

public bool boolPIsLoaded { get { return boolIsLoaded; } }

public int intPFreeItem { get { return intFFreeItem; } }

public THeap()

{

Init();

intFFreeItem = 1;

}

protected void Init()

{

arrFDeleted.Clear();

arrFHeapTable.Clear();

int cnt = 4;

Resize(arrFHeapTable, cnt);

}

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

{

if (size > list.Count)

while (size > list.Count)

list.Add(new THeapItem("", 0, 0));

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

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

{

if (size > list.Count)

while (size > list.Count)

list.Add(new Int32());

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

/\* static void Resize(List<char> list, int size)

{

if (size > list.Count)

while (size > list.Count)

list.Add('0');

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

\*/

public void Expansion()

{

int cardVSize = arrFHeapTable.Count;

cardVSize = cardVSize + cardVSize % 10 + 1;

Resize(arrFHeapTable, cardVSize);

Resize(arrFHeapTable, cardVSize);

}

public void AddLexicalUnit(string strALexicalUnit, byte byteAHashTable, int cardAHashIndex, ref int cardALexicalCode)

{

int intVIndex;

if (arrFDeleted.Count == 0)

{

intVIndex = intFFreeItem;

intFFreeItem++;

if (intFFreeItem >= (Int32)(arrFHeapTable.Count \* 0.9))

Expansion();

}

else

{

intVIndex = arrFDeleted[arrFDeleted.Count - 1];

Resize(arrFDeleted, arrFDeleted.Count - 1);

}

THeapItem Item = arrFHeapTable[intVIndex];

Item.strFLexicalUnit = strALexicalUnit;

Item.byteFHashTable = byteAHashTable;

Item.intFHashIndex = cardAHashIndex;

arrFHeapTable[intVIndex] = Item;

cardALexicalCode = intVIndex;

}

public void DeleteLexicalUnit(int cardALexicalCode)

{

int i;

if (arrFDeleted == null || !arrFDeleted.Any())

i = 0;

else i = arrFDeleted.Count();

Resize(arrFDeleted, i + 1);

arrFDeleted[i] = cardALexicalCode;

THeapItem Item = arrFHeapTable[cardALexicalCode];

Item.strFLexicalUnit = "";

Item.byteFHashTable = 0;

Item.intFHashIndex = 0;

}

public void Save(ref StreamWriter sw)

{

try

{

for (int i = 1; i < arrFHeapTable.Count; i++) //type?

{

if (arrFHeapTable[i].strFLexicalUnit == "")

break;

sw.Write(arrFHeapTable[i].strFLexicalUnit + "\t");

sw.Write(arrFHeapTable[i].byteFHashTable.ToString() + "\t");

sw.WriteLine(arrFHeapTable[i].intFHashIndex.ToString());

}

boolIsSaved = true;

}

catch (Exception) { boolIsSaved = false; }

}

public void Load(ref StreamReader sr)

{

try

{

Init();

int size = arrFHeapTable.Count;

int readSz = 0;

while (true)

{

string line = sr.ReadLine();

if (line == null)

break;

if (++readSz >= size)

{

size \*= 2;

Resize(arrFHeapTable, size);

}

char[] delim = { '\t'/\*,'\n'\*/ };

string[] lines = line.Split(delim);

THeapItem it = arrFHeapTable[readSz];

it.strFLexicalUnit = lines[0];

it.byteFHashTable = Convert.ToByte(lines[1]);

it.intFHashIndex = Convert.ToInt32(lines[2]);

arrFHeapTable[readSz] = it;

}

intFFreeItem = readSz + 1;

boolIsLoaded = true;

}

catch (InvalidCastException)

{ boolIsLoaded = false; }

}

THeapItem GetItem(int i)

{

if (i >= arrFHeapTable.Count)

{

MessageBox.Show("GetИндекс кучи вышел за диапазон!");

THeapItem Item = new THeapItem("", 0, 0);

return Item;

}

else return arrFHeapTable[i];

}

void SetItem(int i, THeapItem NewItem)

{

if (i >= arrFHeapTable.Count)

MessageBox.Show("SetИндекс кучи вышел за диапазон!");

else arrFHeapTable[i] = NewItem;

}

public void HeapTableView(List<string> sList)

{

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

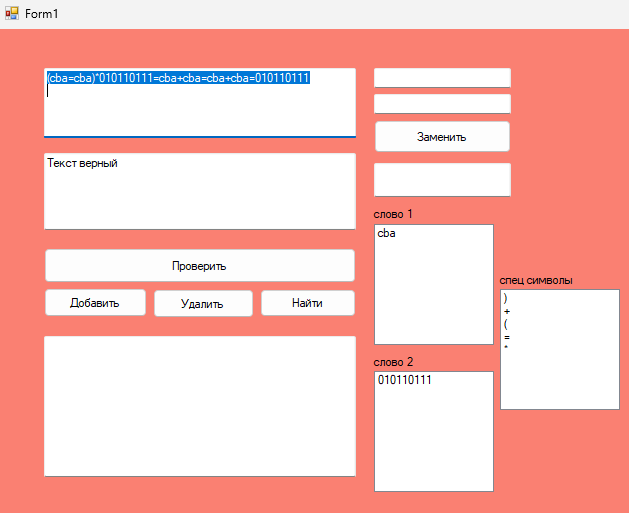
sList.Add(arrFHeapTable[i].strFLexicalUnit);

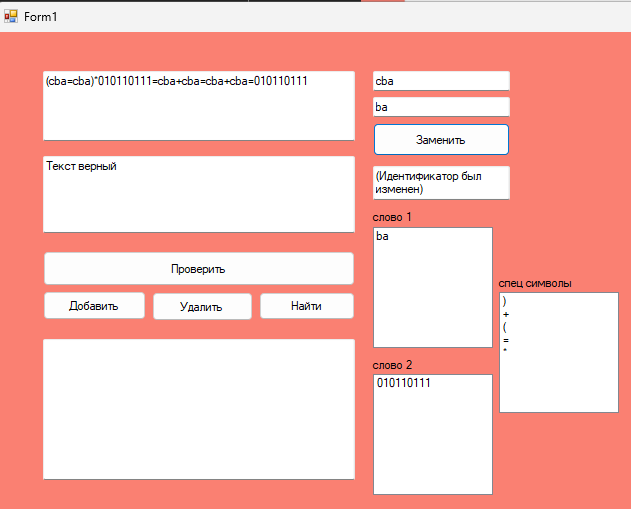
}

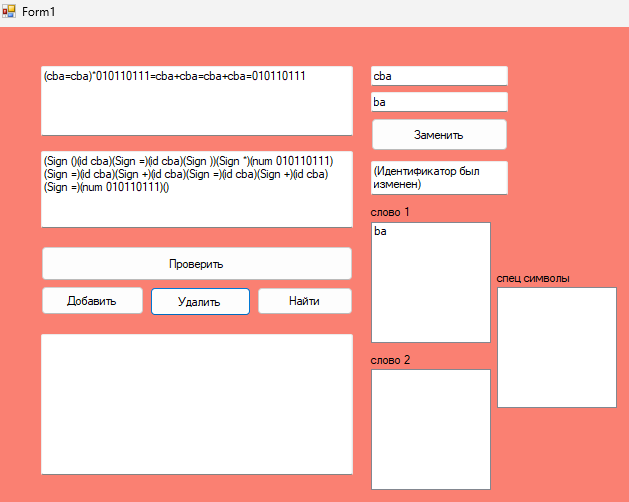
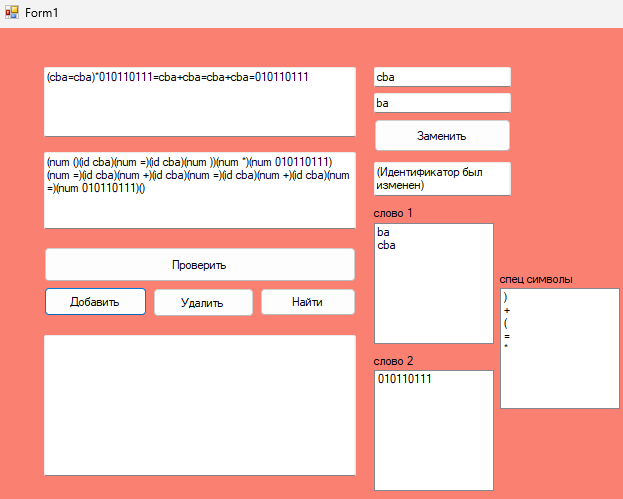
}

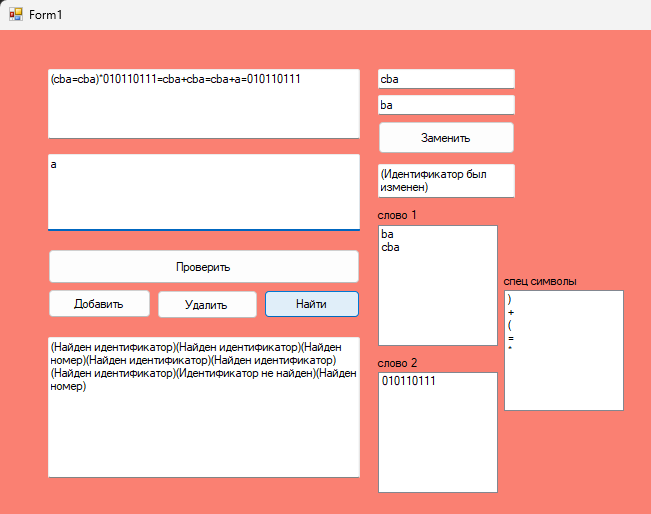
}

**Результаты работы программы:**







**Лабораторная работа № 5.**

**Построение синтаксического дерева**

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

**Краткое теоретическое обоснование:**

Построение синтаксического дерева (или абстрактного синтаксического дерева, AST) — это важная часть синтаксического анализа, которая преобразует текст программы или входные данные в структуру данных, которая представляет собой абстракцию синтаксической структуры этого текста. Синтаксическое дерево позволяет легко анализировать и обрабатывать синтаксическую структуру программы, а также выполнять различные виды статического анализа или генерации кода.

Вот основные шаги построения синтаксического дерева:

1**. Лексический анализ** (токенизация):

- Начните с лексического анализа входного текста или кода, который разбивает текст на лексемы (токены). Лексемы представляют собой минимальные логические единицы, такие как идентификаторы, ключевые слова, операторы и числа.

2. **Синтаксический анализ:**

- Затем применяется синтаксический анализатор для создания синтаксического дерева на основе лексем. Синтаксический анализатор использует контекстно-свободную грамматику, определенную для языка, чтобы определить структуру программы и создать соответствующее синтаксическое дерево.

- Синтаксический анализатор обрабатывает грамматические правила, которые определены в грамматике языка, и строит дерево, в котором узлы представляют синтаксические конструкции, а дуги (рёбра) указывают на связи между этими конструкциями.

3. **Построение синтаксического дерева:**

- В процессе синтаксического анализа начинается построение синтаксического дерева.

- Корень дерева обычно представляет программу в целом, а каждый узел дерева представляет собой определенную синтаксическую конструкцию, такую как выражение, оператор, условие и т. д.

- Для бинарных операторов, таких как арифметические операции или операции сравнения, левый и правый операнды представлены дочерними узлами этого оператора.

4. **Упрощение дерева (по необходимости):**

- В некоторых случаях синтаксическое дерево может быть упрощено или оптимизировано. Например, можно удалить узлы, представляющие ненужные промежуточные вычисления.

5. **Использование синтаксического дерева:**

- После построения синтаксического дерева, оно может использоваться для различных целей, таких как выполнение статического анализа, генерация промежуточного кода, компиляция, интерпретация и т. д.

- Синтаксическое дерево может быть также основой для построения абстрактных семантических деревьев (ASD), которые представляют собой более высокоуровневое представление смысла программы.

Построение синтаксического дерева является фундаментальным этапом при разработке компиляторов и интерпретаторов, а также при создании инструментов для анализа кода.

**Абстрактное синтаксическое дерево (AST):**

AST — это более абстрактное и упрощенное представление синтаксической структуры программы по сравнению с синтаксическим деревом. Оно убирает некоторые детали, такие как скобки и другие второстепенные элементы, и оставляет только важные узлы, представляющие смысл программы. AST обычно используется для дальнейшей обработки и анализа программы, такой как оптимизация и генерация кода.

**Аннотации и информация:**

Синтаксическое дерево может быть аннотировано дополнительной информацией, такой как типы данных, области видимости, местоположение в исходном коде и другие атрибуты. Эта информация полезна при выполнении семантического анализа и других видов анализа.

**Генерация кода:**

В контексте компиляторов, после построения синтаксического дерева можно использовать его для генерации промежуточного кода или целевого кода для целевой платформы.

**Оптимизация:**

Синтаксическое дерево может подвергаться различным видам оптимизаций, чтобы улучшить производительность программы. Например, можно проводить оптимизации вычислений, устранение избыточных вычислений и другие виды оптимизаций.

Рекурсивный спуск и генерация синтаксического анализатора:

Для построения синтаксического дерева можно использовать метод рекурсивного спуска, который часто используется при разработке синтаксических анализаторов. Рекурсивный спуск означает, что анализатор разбирает текст программы, вызывая себя рекурсивно для каждой синтаксической конструкции.

Многие инструменты для генерации синтаксических анализаторов (например, ANTLR, Bison, YACC) позволяют создавать код для синтаксического анализа на основе описания грамматики, что упрощает разработку.

Построение синтаксического дерева является фундаментальной частью компиляторов, интерпретаторов и других инструментов для анализа и обработки языка. Корректное и эффективное построение дерева позволяет проводить разнообразные виды анализа и манипуляций с программным кодом.

**Код программы:**

**THashTable.cs:**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace nsSynt

{

public class THashTable

{

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

private int intFCurrentPrimeNumber;

private int intFItemReserve;

private bool boolIsSaved;

public int intFHashIndex;

public int cardPTableSize { get { return arrFHashTable.Count; } }

public List<object> arrFUserTable = null;

static THeap objFHeap;

public THashTable(ref THeap objAHeap)

{

objFHeap = objAHeap;

Init(7);

intFItemReserve = 0;

}

public void Init(int count)

{

arrFHashTable.Clear();

Resize(arrFHashTable, count);

intFCurrentPrimeNumber = count;

}

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

{

if (size > list.Count)

while (size > list.Count)

list.Add(new object());

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

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

{

if (size > list.Count)

while (size > list.Count)

list.Add(new Int32());

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

int NextPrimeNumber(int cardAOldPrimeNumber)

{

int intVLowerBound, intVUpperBound, intVNextPrimeNumber;

bool boolVIsDivisor;

intVNextPrimeNumber = cardAOldPrimeNumber + cardAOldPrimeNumber / 10 + 1; // увеличиваем на 10 процентов

if ((intVNextPrimeNumber % 2) == 0) intVNextPrimeNumber++;

do

{

boolVIsDivisor = true; intVNextPrimeNumber = intVNextPrimeNumber + 2;

intVLowerBound = 3; intVUpperBound = intVNextPrimeNumber / 3 + 1; // диапазон делителей

while (boolVIsDivisor && (intVLowerBound < intVUpperBound))

{

if ((intVNextPrimeNumber % intVLowerBound) == 0) boolVIsDivisor = false;

else intVLowerBound = intVLowerBound + 2;

}

} while (!boolVIsDivisor);

return intVNextPrimeNumber;

}

//HashFunc

UInt32 HashFunction\_Wainberger(string strALexicalUnit)

{

UInt32 h = 0/\*, seed=131313\*/;

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

{

h += Convert.ToUInt32(strALexicalUnit[i] % 12); //Хэширование

}

return h;

}

int ReHashFunction\_Line(int h, string strALexicalUnit)

{

if (h == 0) h = arrFHashTable.Count / 3;

else if (h == 1) h = arrFHashTable.Count \* 3 / 4;

int i = 1, hi = h;

bool boolVFinish = false;

do

{

if (arrFHashTable[hi] == 0) boolVFinish = true;

else

if (objFHeap.arrFHeapTable[arrFHashTable[hi]].strFLexicalUnit == strALexicalUnit)

boolVFinish = true;

else

{

i++;

hi = (h % i) % (Int32)(arrFHashTable.Count); //Рехэширование(это метод для избавления от коллизии)

}

} while (!boolVFinish);

return hi;

}

public void HashIndex(string strALexicalUnit)

{

int h;

h = (Int32)HashFunction\_Wainberger(strALexicalUnit) % (Int32)(arrFHashTable.Count);

intFHashIndex = ReHashFunction\_Line(h, strALexicalUnit);

}

void TableReHashing()

{

int i, j;

List<int> cardarrVHashTableImage = new List<int>();

List<object> arrVUserTableImage = new List<object>();

Resize(cardarrVHashTableImage, arrFHashTable.Count);

if (arrFUserTable != null)

Resize(arrVUserTableImage, arrFHashTable.Count);

for (i = 0; i < arrFHashTable.Count; i++)

{

cardarrVHashTableImage[i] = arrFHashTable[i];

if (arrFUserTable != null) arrVUserTableImage[i] = arrFUserTable[i];

}

arrFHashTable.Clear();

if (arrFUserTable != null) arrFUserTable.Clear();

Resize(arrFHashTable, intFCurrentPrimeNumber);

if (arrFUserTable != null) Resize(arrFUserTable, intFCurrentPrimeNumber);

for (i = 0; i < cardarrVHashTableImage.Count; i++)

{

if (cardarrVHashTableImage[i] != 0)

{

j = cardarrVHashTableImage[i];

HashIndex(objFHeap.arrFHeapTable[j].strFLexicalUnit);

arrFHashTable[intFHashIndex] = j;

if (arrFUserTable != null) arrFUserTable[intFHashIndex] = arrVUserTableImage[i];

THeapItem Th2 = objFHeap.arrFHeapTable[j];

Th2.intFHashIndex = intFHashIndex;

objFHeap.arrFHeapTable[j] = Th2;

}

}

cardarrVHashTableImage.Clear();

if (arrFUserTable != null) arrVUserTableImage.Clear();

}

void Expansion()

{

intFCurrentPrimeNumber = NextPrimeNumber(intFCurrentPrimeNumber);

TableReHashing();

}

object GetUserPointer(int cardILexicalCode)

{

THeapItem Item = objFHeap.arrFHeapTable[cardILexicalCode];

if (Item.intFHashIndex >= cardPTableSize)

{

MessageBox.Show("Индекс пользовательского массива вышел за диапазон!");

return null;

}

else

{

return arrFUserTable[objFHeap.arrFHeapTable[cardILexicalCode].intFHashIndex];

}

}

void SetUserPointer(int cardILexicalCode, object ptrANewPoint)

{

if (objFHeap.arrFHeapTable[cardILexicalCode].intFHashIndex >= cardPTableSize)

MessageBox.Show("Индекс пользовательского массива вышел за диапазон!");

else

arrFUserTable[objFHeap.arrFHeapTable[cardILexicalCode].intFHashIndex] = ptrANewPoint;

}

public void SetUserTable()

{

arrFUserTable = new List<object>();

Resize(arrFUserTable, arrFHashTable.Count);

}

public bool SearchLexicalUnit(string strAlexicalUnit, ref int intALexicalCode) //Поиск

{

HashIndex(strAlexicalUnit);

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

else

{

intALexicalCode = arrFHashTable[intFHashIndex];

return true;

}

}

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

{

HashIndex(strALexicalUnit);

if (arrFHashTable[intFHashIndex] != 0)

{

intALexicalCode = arrFHashTable[intFHashIndex];

return true;

}

else

{

if ((intFItemReserve + 2) > (cardPTableSize \* 0.9))

{

Expansion();

HashIndex(strALexicalUnit);

}

objFHeap.AddLexicalUnit(strALexicalUnit, byteAHashTable, intFHashIndex, ref intALexicalCode);

arrFHashTable[intFHashIndex] = intALexicalCode;

intFItemReserve++;

return false;

}

}

public void DeleteLexicalUnit(string strAlexicalUnit)

{

HashIndex(strAlexicalUnit);

if (arrFHashTable[intFHashIndex] != 0)

{

if (arrFUserTable != null)

{

if (arrFUserTable[intFHashIndex] != null)

MessageBox.Show("Удаление из таблицы связанного данного");

else

{

objFHeap.DeleteLexicalUnit(arrFHashTable[intFHashIndex]);

arrFHashTable[intFHashIndex] = 0;

intFItemReserve--;

TableReHashing();

}

}

else

{

objFHeap.DeleteLexicalUnit(arrFHashTable[intFHashIndex]);

arrFHashTable[intFHashIndex] = 0;

intFItemReserve--;

TableReHashing();

}

}

}

public void DeleteLexicalCode(int cardALexicalCode)

{

int VHashIndex;

VHashIndex = objFHeap.arrFHeapTable[cardALexicalCode].intFHashIndex;

if (arrFHashTable[VHashIndex] != 0)

if (arrFUserTable.Count != 0)

if (arrFUserTable[VHashIndex] != null)

MessageBox.Show("Удаление из таблицы связанного данного");

else

{

objFHeap.DeleteLexicalUnit(cardALexicalCode);

arrFHashTable[VHashIndex] = 0;

intFItemReserve--;

TableReHashing();

}

else

{

objFHeap.DeleteLexicalUnit(cardALexicalCode);

arrFHashTable[VHashIndex] = 0;

intFItemReserve--;

TableReHashing();

}

}

public void Save(ref StreamWriter fl)

{

try

{

fl.WriteLine(cardPTableSize.ToString());

fl.WriteLine(intFItemReserve.ToString());

for (int i = 1; i < cardPTableSize; i++)

fl.Write("\t" + arrFHashTable[i].ToString());

fl.Write("\n");

boolIsSaved = true;

}

catch (InvalidCastException)

{ boolIsSaved = false; }

}

public void GetLexicalUnitList(ref List<string> sList)

{

for (int i = 0; i < arrFHashTable.Count; i++) if (arrFHashTable[i] != 0) sList.Add(objFHeap.arrFHeapTable[arrFHashTable[i]].strFLexicalUnit);

}

}

//HashTableList

public class CHashTableList

{

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

private bool boolFIsSaved;

public bool boolFIsLoaded;

private byte byteFTablesSize;

static THeap objFHeap = new THeap();

//------------------------------------------------------------------------------

public CHashTableList(byte byteATableCount)

{

this.byteFTablesSize = byteATableCount;

objFHeap = new THeap();

Resize(arrFHashTableList, byteATableCount);

}

//------------------------------------------------------------------------------

public byte GetTableNumber(int intALexicalCode)

{

return objFHeap.arrFHeapTable[intALexicalCode].byteFHashTable;

}

//------------------------------------------------------------------------------

public int GetTablesCount()

{

return arrFHashTableList.Count();

}

//------------------------------------------------------------------------------

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

{

if (size > list.Count)

while (size > list.Count)

{

list.Add(new THashTable(ref objFHeap));

}

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

//------------------------------------------------------------------------------

static void Resize(List<object> list, int intANewSize)

{

if (intANewSize > list.Count)

while (intANewSize > list.Count)

list.Add(new object());

else if (intANewSize < list.Count)

while (list.Count - intANewSize > 0)

list.RemoveAt(list.Count - 1);

}

//------------------------------------------------------------------------------

static void Resize(List<int> list, int intANewSize)

{

if (intANewSize > list.Count)

while (intANewSize > list.Count)

list.Add(new Int32());

else if (intANewSize < list.Count)

while (list.Count - intANewSize > 0)

list.RemoveAt(list.Count - 1);

}

//------------------------------------------------------------------------------

public object GetUserData(int intALexicalCode)

{

if ((0 < intALexicalCode) && (intALexicalCode < objFHeap.intPFreeItem))

return arrFHashTableList[GetTableNumber(intALexicalCode)].arrFUserTable[intALexicalCode];

else

{

MessageBox.Show("Неверно задан лексический код при чтении пользовательских данных");

return null;

}

}

//------------------------------------------------------------------------------

public void SetUserData(int intALexicalCode, object objAUserData)

{

if ((0 < intALexicalCode) && (intALexicalCode < objFHeap.intPFreeItem))

{

if (arrFHashTableList[GetTableNumber(intALexicalCode)].arrFUserTable.Count > 0)

arrFHashTableList[GetTableNumber(intALexicalCode)].arrFUserTable[intALexicalCode] = objAUserData;

else

MessageBox.Show("Попытка записи адреса в несозданный массив пользовательских данных!");

}

else MessageBox.Show("Неверно задан лексический код при записи пользовательских данных!");

}

//------------------------------------------------------------------------------

public string GetLexicalUnit(int intALexicalCode)

{

if ((0 < intALexicalCode) && (intALexicalCode < objFHeap.intPFreeItem)) return objFHeap.arrFHeapTable[intALexicalCode].strFLexicalUnit;

else

{

MessageBox.Show("Неверно задан лексический код при чтении пользовательских данных!");

return "";

}

}

//------------------------------------------------------------------------------

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

{

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

}

//------------------------------------------------------------------------------

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

{

if (byteATable >= arrFHashTableList.Count)

{

if (MessageBox.Show("Увеличить количество таблиц?", "Запрашиваемый индекс таблицы не существует.", MessageBoxButtons.YesNo) == DialogResult.Yes)

Resize(arrFHashTableList, byteATable + 1);

else

return false;

}

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

}

//------------------------------------------------------------------------------

public void DeleteLexicalUnit(string strALexicalUnit, byte byteATable)

{

arrFHashTableList[byteATable].DeleteLexicalUnit(strALexicalUnit);

}

//------------------------------------------------------------------------------

public void DeleteLexicalCode(int intALexicalCode)

{

short T = objFHeap.arrFHeapTable[intALexicalCode].byteFHashTable;

arrFHashTableList[T].DeleteLexicalCode(intALexicalCode);

}

//------------------------------------------------------------------------------

public void SetUserTable(byte byteATable)

{

arrFHashTableList[byteATable].SetUserTable();

}

//------------------------------------------------------------------------------

public void Expantion()

{

Resize(arrFHashTableList, ++byteFTablesSize);

}

//------------------------------------------------------------------------------

public void Save(string strAFileName)

{

try

{

StreamWriter fl = File.CreateText(strAFileName);

fl.WriteLine(byteFTablesSize.ToString());

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

fl.Write(arrFHashTableList[i].arrFHashTable.Count.ToString() + "\t");

fl.WriteLine("");

objFHeap.Save(ref fl);

boolFIsSaved = true;

fl.Close();

}

catch (InvalidDataException)

{ boolFIsSaved = false; }

}

//------------------------------------------------------------------------------

// отладка

public void HeapTableView(List<string> sList)

{

objFHeap.HeapTableView(sList);

}

//------------------------------------------------------------------------------

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

{

arrFHashTableList[byteATable].GetLexicalUnitList(ref sList);

}

//------------------------------------------------------------------------------

public int GetHashIndex(byte Table)

{

return arrFHashTableList[Table].intFHashIndex;

}

//------------------------------------------------------------------------------

}

public struct THeapItem

{

public string strFLexicalUnit;

public byte byteFHashTable;

public int intFHashIndex;

public THeapItem(string strALexicalUnit, byte byteATable, int intAHashIndex)

{

strFLexicalUnit = strALexicalUnit;

byteFHashTable = byteATable;

intFHashIndex = intAHashIndex;

}

}

public class THeap

{

public List<THeapItem> arrFHeapTable = new List<THeapItem>();

private List<int> arrFDeleted = new List<int>();

private int intFFreeItem;

bool boolIsSaved;

bool boolIsLoaded;

public bool boolPIsSaved { get { return boolIsSaved; } }

public bool boolPIsLoaded { get { return boolIsLoaded; } }

public int intPFreeItem { get { return intFFreeItem; } }

public THeap()

{

Init();

intFFreeItem = 1;

}

protected void Init()

{

arrFDeleted.Clear();

arrFHeapTable.Clear();

int cnt = 4;

Resize(arrFHeapTable, cnt);

}

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

{

if (size > list.Count)

while (size > list.Count)

list.Add(new THeapItem("", 0, 0));

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

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

{

if (size > list.Count)

while (size > list.Count)

list.Add(new Int32());

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

/\* static void Resize(List<char> list, int size)

{

if (size > list.Count)

while (size > list.Count)

list.Add('0');

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

\*/

public void Expansion()

{

int cardVSize = arrFHeapTable.Count;

cardVSize = cardVSize + cardVSize % 10 + 1;

Resize(arrFHeapTable, cardVSize);

Resize(arrFHeapTable, cardVSize);

}

public void AddLexicalUnit(string strALexicalUnit, byte byteAHashTable, int cardAHashIndex, ref int cardALexicalCode)

{

int intVIndex;

if (arrFDeleted.Count == 0)

{

intVIndex = intFFreeItem;

intFFreeItem++;

if (intFFreeItem >= (Int32)(arrFHeapTable.Count \* 0.9))

Expansion();

}

else

{

intVIndex = arrFDeleted[arrFDeleted.Count - 1];

Resize(arrFDeleted, arrFDeleted.Count - 1);

}

THeapItem Item = arrFHeapTable[intVIndex];

Item.strFLexicalUnit = strALexicalUnit;

Item.byteFHashTable = byteAHashTable;

Item.intFHashIndex = cardAHashIndex;

arrFHeapTable[intVIndex] = Item;

cardALexicalCode = intVIndex;

}

public void DeleteLexicalUnit(int cardALexicalCode)

{

int i;

if (arrFDeleted == null || !arrFDeleted.Any())

i = 0;

else i = arrFDeleted.Count();

Resize(arrFDeleted, i + 1);

arrFDeleted[i] = cardALexicalCode;

THeapItem Item = arrFHeapTable[cardALexicalCode];

Item.strFLexicalUnit = "";

Item.byteFHashTable = 0;

Item.intFHashIndex = 0;

}

public void Save(ref StreamWriter sw)

{

try

{

for (int i = 1; i < arrFHeapTable.Count; i++) //type?

{

if (arrFHeapTable[i].strFLexicalUnit == "")

break;

sw.Write(arrFHeapTable[i].strFLexicalUnit + "\t");

sw.Write(arrFHeapTable[i].byteFHashTable.ToString() + "\t");

sw.WriteLine(arrFHeapTable[i].intFHashIndex.ToString());

}

boolIsSaved = true;

}

catch (Exception) { boolIsSaved = false; }

}

public void Load(ref StreamReader sr)

{

try

{

Init();

int size = arrFHeapTable.Count;

int readSz = 0;

while (true)

{

string line = sr.ReadLine();

if (line == null)

break;

if (++readSz >= size)

{

size \*= 2;

Resize(arrFHeapTable, size);

}

char[] delim = { '\t'/\*,'\n'\*/ };

string[] lines = line.Split(delim);

THeapItem it = arrFHeapTable[readSz];

it.strFLexicalUnit = lines[0];

it.byteFHashTable = Convert.ToByte(lines[1]);

it.intFHashIndex = Convert.ToInt32(lines[2]);

arrFHeapTable[readSz] = it;

}

intFFreeItem = readSz + 1;

boolIsLoaded = true;

}

catch (InvalidCastException)

{ boolIsLoaded = false; }

}

THeapItem GetItem(int i)

{

if (i >= arrFHeapTable.Count)

{

MessageBox.Show("GetИндекс кучи вышел за диапазон!");

THeapItem Item = new THeapItem("", 0, 0);

return Item;

}

else return arrFHeapTable[i];

}

void SetItem(int i, THeapItem NewItem)

{

if (i >= arrFHeapTable.Count)

MessageBox.Show("SetИндекс кучи вышел за диапазон!");

else arrFHeapTable[i] = NewItem;

}

public void HeapTableView(List<string> sList)

{

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

sList.Add(arrFHeapTable[i].strFLexicalUnit);

}

}

}

**uLex.cs:**

**using System;**

**using System.Collections.Generic;**

**using System.Text;**

**namespace nsLex**

**{**

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

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

**public enum TToken**

**{**

**mn, minus, plus, lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty,**

**lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma, lxmdt, lxmText, lxmtz, lxmr, lxmrs, lxmls**

**};**

**public class CLex //класс лексический анализатор**

**{**

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

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

**public TCharType enumFSelectionCharType;**

**public char chrFSelection;**

**private TState enumFState;**

**private int intFSourceRowSelection;**

**private int intFSourceColSelection;**

**private String strFLexicalUnit;**

**private TToken enumFToken;**

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

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

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

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

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

**public int intPSourceRowSelection**

**{**

**get { return intFSourceRowSelection; }**

**set**

**{**

**intFSourceRowSelection = value;**

**}**

**}**

**public int intPSourceColSelection**

**{**

**get { return intFSourceColSelection; }**

**set**

**{**

**intFSourceColSelection = value;**

**}**

**}**

**public CLex()**

**{**

**}**

**public void GetSymbol() //метод класса лексический анализатор**

**{**

**intFSourceColSelection++; // продвигаем номер колонки**

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

**{**

**intFSourceRowSelection++;**

**if (intFSourceRowSelection <= strFSource.Length - 1)**

**{**

**intFSourceColSelection = -1;**

**chrFSelection = '\0';**

**enumFSelectionCharType = TCharType.EndRow;**

**enumFState = TState.Continue;**

**}**

**else**

**{**

**chrFSelection = '\0';**

**enumFSelectionCharType = TCharType.EndText;**

**enumFState = TState.Finish;**

**}**

**}**

**else**

**{**

**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 == '\*' || chrFSelection == '-') enumFSelectionCharType =TCharType.ReservedSymbol;**

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

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

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

**else if (chrFSelection == ';' || chrFSelection == ',' || chrFSelection == '[' || chrFSelection == ']' || chrFSelection == '=' || chrFSelection == ':') enumFSelectionCharType = TCharType.ReservedSymbol;**

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

**enumFState = TState.Continue;**

**}**

**}**

**private void TakeSymbol()**

**{**

**char[] c = { chrFSelection };**

**String s = new string(c);**

**strFLexicalUnit += s;**

**GetSymbol();**

**}**

**public void NextToken()**

**{**

**strFLexicalUnit = "";**

**if (enumFState == TState.Start)**

**{**

**intFSourceRowSelection = 0;**

**intFSourceColSelection = -1;**

**GetSymbol();**

**}**

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

**{**

**GetSymbol();**

**}**

**if (chrFSelection == '/')**

**{**

**GetSymbol();**

**if (chrFSelection == '/')**

**while (enumFSelectionCharType != TCharType.EndRow)**

**{**

**GetSymbol();**

**}**

**GetSymbol();**

**}**

**switch (enumFSelectionCharType)**

**{**

**case TCharType.Letter: //если начинается слово из букв a,b,c,d**

**{**

**// d c b a**

**// AF | AF | BF | CF | DF |**

**// BF | | BF | CF | DF |**

**// CF | | | CF | DF |**

**// DF | | | | DF |**

**AF:**

**{**

**if (chrFSelection == 'd')**

**{**

**TakeSymbol();**

**goto AF;**

**}**

**if (chrFSelection == 'c')**

**{**

**TakeSymbol();**

**goto BF;**

**}**

**if (chrFSelection == 'b')**

**{**

**TakeSymbol();**

**goto CF;**

**}**

**if (chrFSelection == 'a')**

**{**

**TakeSymbol();**

**goto DF;**

**}**

**else**

**{**

**enumFToken = TToken.lxmIdentifier;**

**return;**

**}**

**}**

**BF:**

**{**

**if (chrFSelection == 'd')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'c')**

**{**

**TakeSymbol();**

**goto BF;**

**}**

**if (chrFSelection == 'b')**

**{**

**TakeSymbol();**

**goto CF;**

**}**

**if (chrFSelection == 'a')**

**{**

**TakeSymbol();**

**goto DF;**

**}**

**else**

**{**

**enumFToken = TToken.lxmIdentifier;**

**return;**

**}**

**}**

**CF:**

**{**

**if (chrFSelection == 'd')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'c')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'b')**

**{**

**TakeSymbol();**

**goto CF;**

**}**

**if (chrFSelection == 'a')**

**{**

**TakeSymbol();**

**goto DF;**

**}**

**else**

**{**

**enumFToken = TToken.lxmIdentifier;**

**return;**

**}**

**}**

**DF:**

**{**

**if (chrFSelection == 'd')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'c')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'b')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'a')**

**{**

**TakeSymbol();**

**goto DF;**

**}**

**else**

**{**

**enumFToken = TToken.lxmIdentifier;**

**return;**

**}**

**}**

**}**

**if (chrFSelection == '/')**

**{**

**GetSymbol();**

**if (chrFSelection == '/')**

**while (enumFSelectionCharType != TCharType.EndRow)**

**{**

**GetSymbol();**

**}**

**GetSymbol();**

**}**

**case TCharType.Digit: //если начинается слово из букв – цифр 0 и 1**

**{**

**// 0 1**

**// A | B | C |**

**// B | | D |**

**// C | | E |**

**// D | A | |**

**// E | Ffin | |**

**// Ffin | | G |**

**// G | | H |**

**// H | | Ffin |**

**A:**

**if (chrFSelection == '0')**

**{**

**TakeSymbol();**

**goto B;**

**}**

**else if (chrFSelection == '1')**

**{**

**TakeSymbol();**

**goto C;**

**}**

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

**B:**

**if (chrFSelection == '1')**

**{**

**TakeSymbol();**

**goto D;**

**}**

**else throw new Exception("Ожидался 1");**

**C:**

**if (chrFSelection == '1')**

**{**

**TakeSymbol();**

**goto E;**

**}**

**else throw new Exception("Ожидался 1");**

**D:**

**if (chrFSelection == '0')**

**{**

**TakeSymbol();**

**goto A;**

**}**

**else throw new Exception("Ожидался 0");**

**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("Ожидался 1");**

**G:**

**if (chrFSelection == '1')**

**{**

**TakeSymbol();**

**goto H;**

**}**

**else throw new Exception("Ожидался 1");**

**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 == '(')**

**{**

**strFLexicalUnit = "(";**

**enumFToken = TToken.lxmLeftParenth;**

**GetSymbol();**

**return;**

**}**

**if (chrFSelection == '+')**

**{**

**strFLexicalUnit = "+";**

**enumFToken = TToken.plus;**

**GetSymbol();**

**return;**

**}**

**if (chrFSelection == '\*')**

**{**

**strFLexicalUnit = "\*";**

**enumFToken = TToken.mn;**

**GetSymbol();**

**return;**

**}**

**if (chrFSelection == '-')**

**{**

**enumFToken = TToken.minus;**

**GetSymbol();**

**return;**

**}**

**if (chrFSelection == ')')**

**{**

**strFLexicalUnit = ")";**

**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 == '=')**

**{**

**strFLexicalUnit = "=";**

**enumFToken = TToken.lxmr;**

**GetSymbol();**

**return;**

**}**

**break;**

**}**

**case TCharType.EndText:**

**{**

**enumFToken = TToken.lxmEmpty;**

**break;**

**}**

**}**

**}**

**}**

**}**

**uSyntAnalyzer.cs:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Windows.Forms;

using nsLex;

using тяп\_лаба\_3;

namespace nsSynt

{

class uSyntAnalyzer

{

private String[] strFSource;

private String[] strFMessage;

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

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

public CLex Lex = new CLex();

public TreeView tree;

public void D(TreeNode parent)

{

TreeNode node = new TreeNode("D");

parent.Nodes.Add(node);

//string s = Convert.ToString(Lex.enumPToken);

//MessageBox.Show(s);

//Lex.NextToken();

K(node);

// Lex.NextToken();

if (Lex.enumPToken == TToken.plus)

{

node.Nodes.Add("+");

C(node);

}

}

public void C(TreeNode parent)

{

TreeNode node = new TreeNode("C");

parent.Nodes.Add(node);

if (Lex.enumPToken == TToken.plus)

{

//node.Nodes.Add("+");

Lex.NextToken();

K(node);

if (Lex.enumPToken == TToken.plus)

{

node.Nodes.Add("+");

C(node);

}

}

//else throw new Exception("Ожидалcя +");

}

public void K(TreeNode parent)

{

TreeNode node = new TreeNode("K");

parent.Nodes.Add(node);

A(node);

//Lex.NextToken();

if (Lex.enumPToken == TToken.mn)

{

//node.Nodes.Add("=");

B(node);

}

}

public void B(TreeNode parent)

{

TreeNode node = new TreeNode("B");

parent.Nodes.Add(node);

if (Lex.enumPToken == TToken.mn)

{

node.Nodes.Add("\*");

Lex.NextToken();

A(node);

//Lex.NextToken();

if (Lex.enumPToken == TToken.mn)

{

node.Nodes.Add("\*");

B(node);

}

}

}

public void A(TreeNode parent)

{

TreeNode node = new TreeNode("A");

parent.Nodes.Add(node);

if (Lex.enumPToken == TToken.minus)

{

node.Nodes.Add("-");

Lex.NextToken();

A(node);

}

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

{

node.Nodes.Add("(");

Lex.NextToken();

D(node);

//Lex.NextToken();

if (Lex.enumPToken == TToken.lxmRightParenth)

{

node.Nodes.Add(")");

Lex.NextToken();

// throw new Exception("Konec slova");

}

else throw new Exception("Ожидалась )");

}

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

{

//node.Nodes.Add(Lex.strPLexicalUnit);

O(node);

}

else

{

string s = Convert.ToString(Lex.enumPToken);

MessageBox.Show(s);

throw new Exception("Ожидались - или ( или число");

}

}

public void O(TreeNode parent)

{

TreeNode node = new TreeNode("O");

parent.Nodes.Add(node);

if (Lex.enumPToken == TToken.lxmNumber)

{

node.Nodes.Add(Lex.strPLexicalUnit);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmr)

{

node.Nodes.Add("=");

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier)

{

node.Nodes.Add(Lex.strPLexicalUnit);

//throw new Exception("Konec slova");

}

else throw new Exception("ожидалось число");

}

else throw new Exception("ожидалось =");

}

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

{

node.Nodes.Add(Lex.strPLexicalUnit);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmr)

{

node.Nodes.Add("=");

Lex.NextToken();

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

{

node.Nodes.Add(Lex.strPLexicalUnit);

//throw new Exception("Konec slova");

}

else throw new Exception("ожидалось число или слово");

}

else throw new Exception("ожидалось =");

}

else throw new Exception("Ожидалось число или слово");

Lex.NextToken();

}

}

}

**Form1.cs:**

**using System;**

**using System.Collections.Generic;**

**using System.ComponentModel;**

**using System.Data;**

**using System.Drawing;**

**using System.Text;**

**using System.Windows.Forms;**

**using nsSynt;**

**using nsLex;**

**namespace тяп\_лаба\_3**

**{**

**public partial class Form1 : Form**

**{**

**public CHashTableList htl = new CHashTableList(3);**

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

**{**

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

**listBox1.Items.Clear();**

**listBox2.Items.Clear();**

**listBox3.Items.Clear();**

**htl.TableToStringList(0, listTable);**

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

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

**listTable.Clear();**

**htl.TableToStringList(1, listTable);**

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

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

**listTable.Clear();**

**htl.TableToStringList(2, listTable);**

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

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

**listTable.Clear();**

**}**

**public Form1()**

**{**

**InitializeComponent();**

**tbFSource.AppendText("(cba=cba)\*010110111=cba+cba=cba+cba=010110111" + "\r\n");**

**int n = tbFSource.Lines.Length;**

**}**

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

**{**

**tbFMessage.Clear();**

**uSyntAnalyzer Synt = new uSyntAnalyzer();**

**treeView1.Nodes.Clear();**

**Synt.tree = treeView1;**

**Synt.Lex.strPSource = tbFSource.Lines;**

**Synt.Lex.strPMessage = tbFMessage.Lines;**

**Synt.Lex.enumPState = TState.Start;**

**try**

**{**

**TreeNode start = new TreeNode("Start");**

**treeView1.Nodes.Add(start);**

**Synt.Lex.NextToken();**

**Synt.D(start);**

**Synt.tree.ExpandAll();**

**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;**

**}**

**}**

**//Кнопка удаления слова**

**private void button4\_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) //проверка: слово - номер или идентификатор**

**{**

**case TToken.lxmIdentifier: //если идентификатор**

**{**

**s1 = "id " + Lex.strPLexicalUnit;**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 0); //слово удаляется из таблицы идентификаторов**

**htl.DeleteLexicalCode(0);**

**TablesToMemo(this, e);**

**break;**

**}**

**case TToken.lxmNumber: //если номер**

**{**

**s1 = "num " + Lex.strPLexicalUnit;**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 1); //слово удаляется из таблицы номеров**

**htl.DeleteLexicalCode(0);**

**TablesToMemo(this, e);**

**break;**

**}**

**case TToken.lxmLeftParenth: //если номер**

**{**

**s1 = "Sign " + Lex.strPLexicalUnit;**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 2); //слово удаляется из таблицы номеров**

**htl.DeleteLexicalCode(0);**

**TablesToMemo(this, e);**

**break;**

**}**

**case TToken.lxmr: //если номер**

**{**

**s1 = "Sign " + Lex.strPLexicalUnit;**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 2); //слово удаляется из таблицы номеров**

**htl.DeleteLexicalCode(0);**

**TablesToMemo(this, e);**

**break;**

**}**

**case TToken.lxmRightParenth: //если номер**

**{**

**s1 = "Sign " + Lex.strPLexicalUnit;**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 2); //слово удаляется из таблицы номеров**

**htl.DeleteLexicalCode(0);**

**TablesToMemo(this, e);**

**break;**

**}**

**case TToken.mn: //если номер**

**{**

**s1 = "Sign " + Lex.strPLexicalUnit;**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 2); //слово удаляется из таблицы номеров**

**htl.DeleteLexicalCode(0);**

**TablesToMemo(this, e);**

**break;**

**}**

**case TToken.plus: //если номер**

**{**

**s1 = "Sign " + Lex.strPLexicalUnit;**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 2); //слово удаляется из таблицы номеров**

**htl.DeleteLexicalCode(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;**

**}**

**}**

**//Кнопка замены**

**private void button5\_Click(object sender, EventArgs e)**

**{**

**CLex Lex = new CLex();**

**Lex.strPSource = tbFSource1.Lines;**

**Lex.strPMessage = tbFMessage1.Lines;**

**Lex.intPSourceColSelection = 0;**

**Lex.intPSourceRowSelection = 0;**

**int x = tbFSource1.TextLength;**

**CLex Lex2 = new CLex();**

**Lex2.strPSource = tbFSource2.Lines;**

**Lex2.strPMessage = tbFMessage1.Lines;**

**Lex2.intPSourceColSelection = 0;**

**Lex2.intPSourceRowSelection = 0;**

**int x2 = tbFSource2.TextLength;**

**tbFMessage1.Text = "";**

**try**

**{**

**while (Lex.enumPState != TState.Finish)**

**{**

**Lex.NextToken();**

**switch (Lex.enumPToken) //Проверка 1 - го слова на то, является ли оно номером или идентификатором**

**{**

**case TToken.lxmIdentifier: //Если идентификатор то:**

**{**

**int b = 0;**

**if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, 0, ref b))**

**{**

**while (Lex2.enumPState != TState.Finish)**

**{**

**Lex2.NextToken();**

**switch (Lex2.enumPToken) //Проверка 1 - го слова на то, является ли оно номером или идентификатором**

**{**

**case TToken.lxmIdentifier: //Если является идентификатором, то:**

**{**

**int b1 = 0;**

**if (htl.AddLexicalUnit(Lex2.strPLexicalUnit, 0, ref b1)) //вызывается функция добавления в таблицу идентификаторов 2 - го слова**

**{**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 0); //Удаляется 1 - ое слово**

**TablesToMemo(this, e);**

**}**

**tbFMessage1.AppendText("(Идентификатор был изменен)");**

**break;**

**}**

**case TToken.lxmNumber: //если 2 - ое слово - номер**

**{**

**tbFMessage1.AppendText("(Нельзя заменить идентификатор на номер)");**

**break;**

**}**

**}**

**}**

**}**

**else tbFMessage1.AppendText("(Идентификатор для удаления не найден)");**

**break;**

**}**

**case TToken.lxmNumber: //если 1 - ое слово номер**

**{**

**int b = 0;**

**if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, 1, ref b))**

**{**

**while (Lex2.enumPState != TState.Finish)**

**{**

**Lex2.NextToken();**

**switch (Lex2.enumPToken) //проверка 2 - го слова**

**{**

**case TToken.lxmIdentifier://если 2 - ое слово идентификатор**

**{**

**tbFMessage1.AppendText("(Нельзя заменить номер на идентификатор)");**

**break;**

**}**

**case TToken.lxmNumber: // если 2 - ое слово номер**

**{**

**int b1 = 0;**

**if (htl.AddLexicalUnit(Lex2.strPLexicalUnit, 1, ref b1))//1 - ое слово добавляется в таблицу номеров**

**{**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 1);// 2 - ое слово удаляется**

**TablesToMemo(this, e);**

**}**

**tbFMessage1.AppendText("(Номер был изменен)");**

**break;**

**}**

**}**

**}**

**}**

**else tbFMessage1.AppendText("(Номер для удаления не найден)");**

**break;**

**}**

**}**

**}**

**}**

**catch (Exception exc)**

**{**

**tbFMessage1.Text += exc.Message;**

**tbFSource.Select();**

**tbFSource.SelectionStart = 0;**

**int n1 = 0;**

**for (int i = 0; i < Lex.intPSourceRowSelection; i++) n1 += tbFSource.Lines[i].Length + 2;**

**n1 += Lex.intPSourceColSelection;**

**tbFSource.SelectionLength = n1;**

**}**

**}**

**// Кнопка добавления слов в таблицы**

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

**{**

**CLex Lex = new CLex();**

**Lex.strPSource = tbFSource.Lines;**

**Lex.strPMessage = tbFMessage.Lines;**

**Lex.intPSourceColSelection = 0;**

**Lex.intPSourceRowSelection = 0;**

**int x = tbFSource.TextLength;**

**int y = tbFSource.Lines.Length;**

**tbFMessage.Text = "";**

**try**

**{**

**while (Lex.enumPState != TState.Finish)**

**{**

**Lex.NextToken();**

**string s1 = "", s = "";**

**switch (Lex.enumPToken)**

**{**

**case TToken.lxmIdentifier:**

**{**

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

**if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 0, ref b)) //Вызывается функция добавления слова в таблицу идентификаторов(0)**

**{**

**TablesToMemo(this, e);**

**}**

**break;**

**}**

**case TToken.lxmNumber:**

**{**

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

**if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 1, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)**

**{**

**TablesToMemo(this, e);**

**}**

**break;**

**}**

**case TToken.lxmLeftParenth:**

**{**

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

**if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)**

**{**

**TablesToMemo(this, e);**

**}**

**break;**

**}**

**case TToken.lxmRightParenth:**

**{**

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

**if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)**

**{**

**TablesToMemo(this, e);**

**}**

**break;**

**}**

**case TToken.lxmr:**

**{**

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

**if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)**

**{**

**TablesToMemo(this, e);**

**}**

**break;**

**}**

**case TToken.mn:**

**{**

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

**if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)**

**{**

**TablesToMemo(this, e);**

**}**

**break;**

**}**

**case TToken.plus:**

**{**

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

**if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)**

**{**

**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;**

**}**

**}**

**//Кнопка посика слова в таблицах**

**private void button3\_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();**

**switch (Lex.enumPToken)**

**{**

**case TToken.lxmIdentifier:**

**{**

**int b = 0;**

**if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, 0, ref b)) //Вызов функции поиска, возвращает значение true если было найден такой идентификатор, иначе false**

**{**

**tbFMessage2.AppendText("(Найден идентификатор)");**

**}**

**else tbFMessage2.AppendText("(Идентификатор не найден)");**

**break;**

**}**

**case TToken.lxmNumber:**

**{**

**int b = 0;**

**if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, 1, ref b)) //Вызов функции поиска, возвращает значение true если было найден такой номер, иначе false**

**{**

**tbFMessage2.AppendText("(Найден номер)");**

**}**

**else tbFMessage2.AppendText("(Номер не найден)");**

**break;**

**}**

**}**

**}**

**}**

**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;**

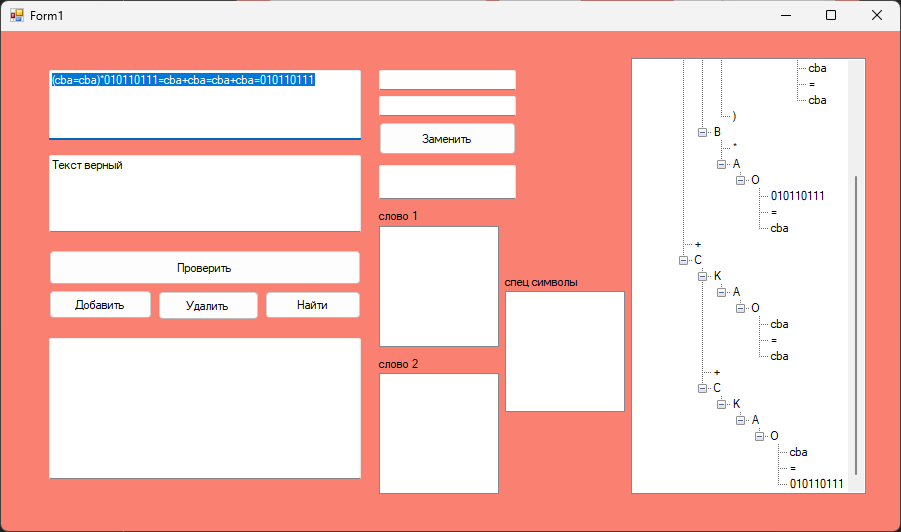
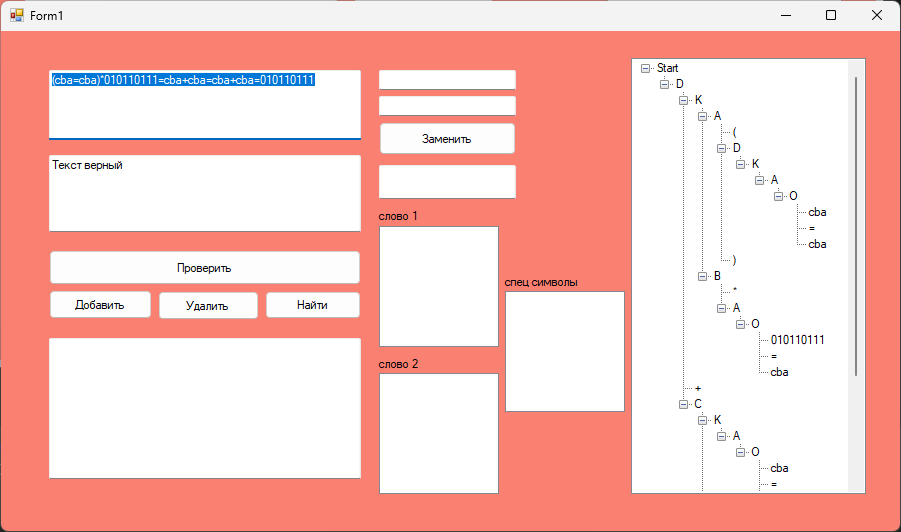
**n += Lex.intPSourceColSelection;**

**tbFSource.SelectionLength = n;**

**}**

**}**

**Результат:**



**Лабораторная работа № 6. Разработка генератора**

**Задание:**

1. Перевести все числа в десятичное представление.

2. Выполнить вывод исходного текста в структурированном виде.

В моем коде выполняется перевод чисел и идентификаторов на 1 уровень ветки вверх, с переводом чисел из двоичного в десятичный вид, а также удалением самой нижней ветки. Сделано это в классе TreeConverter.cs

Этот код представляет собой часть программы на C#, которая преобразует бинарные числа в десятичные числа в древовидной структуре данных. Давайте разберем его построчно.

1. private void ConvertBinaryToDecimal(TreeNode node): Это объявление метода ConvertBinaryToDecimal, который принимает в качестве параметра объект типа TreeNode.

2. if (node.Text == \"O\"): Это условное выражение, которое проверяет значение свойства Text объекта node. Если оно равно "O", выполняется следующий блок кода.

3. if (node.Nodes.Count == 3): Это условное выражение проверяет количество узлов внутри текущего узла. Если их количество равно 3, выполняется следующий блок кода.

4. foreach (TreeNode node2 in node.Nodes): Это цикл foreach, который перебирает все узлы внутри текущего узла.

5. string binaryValue = node2.Text;: Создается переменная binaryValue, которая содержит значение свойства Text объекта node2.

6. if (IsBin(binaryValue)): Это условное выражение вызывает метод IsBin для проверки, является ли значение binaryValue бинарным числом.

7. int decimalValue = Convert.ToInt32(binaryValue, 2);: Если значение binaryValue является бинарным числом, оно преобразуется в десятичное число с помощью метода Convert.ToInt32.

8. node2.Text = decimalValue.ToString();: Значение Text объекта node2 заменяется на его десятичное представление.

9. node.Parent.Nodes.Add(node2.Text);: Десятичное значение добавляется в список узлов родительского узла.

10. node.Remove();: Узел node удаляется.

11. foreach (TreeNode childNode in node.Nodes): Это цикл foreach, который перебирает все дочерние узлы объекта node и вызывает метод ConvertBinaryToDecimal для каждого из них.

12. private bool IsBin(string value): Это объявление метода IsBin, который проверяет, является ли значение value бинарным числом.

В целом, этот код рекурсивно итерирует по каждому узлу в древовидной структуре данных, преобразуя бинарные числа в их десятичное представление и обновляя значения узлов.

**Краткое теоретическое обоснование:**

**Структурированное представление в TreeView:**

TreeView - это виджет (компонент пользовательского интерфейса) в графических пользовательских интерфейсах (GUI), который используется для отображения и управления иерархическими данными в виде древовидной структуры. В структурированном представлении TreeView узлы дерева могут быть организованы и иерархически связаны между собой.

**Основные элементы структурированного представления TreeView включают в себя:**

Узлы: Каждый узел представляет собой элемент в дереве и может содержать данные и/или другие дочерние узлы.

**Дерево:** Совокупность всех узлов и их отношений между собой формирует дерево, которое отображается в виде иерархической структуры.

**Разделители и иконки:** TreeView обычно использует разделители для отображения уровней вложенности и иконки для обозначения типа узлов (например, папки или файлы).

Структурированный TreeView может использоваться для представления различных типов данных, таких как файловая система, структуры документов, иерархии каталогов и многое другое. Пользователи могут разворачивать и сворачивать ветви дерева для просмотра и управления данными.

**Разработка генератора:**

Разработка генератора относится к созданию программного инструмента, который генерирует код или другой выходной результат на основе некоторых исходных данных или спецификаций. Генераторы могут использоваться для автоматизации и упрощения процессов разработки, а также для создания кода или документации.

Примеры задач, для которых могут быть разработаны генераторы, включают в себя:

Генерация исходного кода на основе спецификации (например, генераторы кода для языков программирования).

Генерация документации на основе комментариев в исходном коде (например, генераторы документации).

Генерация структур данных или конфигурационных файлов из описаний.

Генерация отчетов и отображений из данных.

**Разработка генератора обычно включает в себя следующие шаги:**

1. Определение исходных данных или спецификаций, на основе которых будет производиться генерация.
2. Создание алгоритма или логики генерации.
3. Разработка генератора, который реализует этот алгоритм.
4. Тестирование и отладка генератора.
5. Интеграция генератора в рабочий процесс разработки или другой инструмент.

Генераторы могут быть написаны на различных языках программирования и часто используются в разработке программного обеспечения, чтобы сократить рутинную работу и обеспечить согласованность и точность в выходных данных.

**Код программы:**

**TreeConventer.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace тяп\_лаба\_3

{

internal class TreeConverter

{

public void ConvertBinaryToDecimal(TreeView tree)

{

foreach (TreeNode node in tree.Nodes)

{

ConvertBinaryToDecimal(node);

}

}

private void ConvertBinaryToDecimal(TreeNode node)

{

if (node.Text == "O")

{

if (node.Nodes.Count == 3)

{

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;

}

}

}

**THashTable.cs:**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace nsSynt

{

public class THashTable

{

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

private int intFCurrentPrimeNumber;

private int intFItemReserve;

private bool boolIsSaved;

public int intFHashIndex;

public int cardPTableSize { get { return arrFHashTable.Count; } }

public List<object> arrFUserTable = null;

static THeap objFHeap;

public THashTable(ref THeap objAHeap)

{

objFHeap = objAHeap;

Init(7);

intFItemReserve = 0;

}

public void Init(int count)

{

arrFHashTable.Clear();

Resize(arrFHashTable, count);

intFCurrentPrimeNumber = count;

}

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

{

if (size > list.Count)

while (size > list.Count)

list.Add(new object());

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

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

{

if (size > list.Count)

while (size > list.Count)

list.Add(new Int32());

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

int NextPrimeNumber(int cardAOldPrimeNumber)

{

int intVLowerBound, intVUpperBound, intVNextPrimeNumber;

bool boolVIsDivisor;

intVNextPrimeNumber = cardAOldPrimeNumber + cardAOldPrimeNumber / 10 + 1; // увеличиваем на 10 процентов

if ((intVNextPrimeNumber % 2) == 0) intVNextPrimeNumber++;

do

{

boolVIsDivisor = true; intVNextPrimeNumber = intVNextPrimeNumber + 2;

intVLowerBound = 3; intVUpperBound = intVNextPrimeNumber / 3 + 1; // диапазон делителей

while (boolVIsDivisor && (intVLowerBound < intVUpperBound))

{

if ((intVNextPrimeNumber % intVLowerBound) == 0) boolVIsDivisor = false;

else intVLowerBound = intVLowerBound + 2;

}

} while (!boolVIsDivisor);

return intVNextPrimeNumber;

}

//HashFunc

UInt32 HashFunction\_Wainberger(string strALexicalUnit)

{

UInt32 h = 0/\*, seed=131313\*/;

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

{

h += Convert.ToUInt32(strALexicalUnit[i] % 12); //Хэширование

}

return h;

}

int ReHashFunction\_Line(int h, string strALexicalUnit)

{

if (h == 0) h = arrFHashTable.Count / 3;

else if (h == 1) h = arrFHashTable.Count \* 3 / 4;

int i = 1, hi = h;

bool boolVFinish = false;

do

{

if (arrFHashTable[hi] == 0) boolVFinish = true;

else

if (objFHeap.arrFHeapTable[arrFHashTable[hi]].strFLexicalUnit == strALexicalUnit)

boolVFinish = true;

else

{

i++;

hi = (h % i) % (Int32)(arrFHashTable.Count); //Рехэширование(это метод для избавления от коллизии)

}

} while (!boolVFinish);

return hi;

}

public void HashIndex(string strALexicalUnit)

{

int h;

h = (Int32)HashFunction\_Wainberger(strALexicalUnit) % (Int32)(arrFHashTable.Count);

intFHashIndex = ReHashFunction\_Line(h, strALexicalUnit);

}

void TableReHashing()

{

int i, j;

List<int> cardarrVHashTableImage = new List<int>();

List<object> arrVUserTableImage = new List<object>();

Resize(cardarrVHashTableImage, arrFHashTable.Count);

if (arrFUserTable != null)

Resize(arrVUserTableImage, arrFHashTable.Count);

for (i = 0; i < arrFHashTable.Count; i++)

{

cardarrVHashTableImage[i] = arrFHashTable[i];

if (arrFUserTable != null) arrVUserTableImage[i] = arrFUserTable[i];

}

arrFHashTable.Clear();

if (arrFUserTable != null) arrFUserTable.Clear();

Resize(arrFHashTable, intFCurrentPrimeNumber);

if (arrFUserTable != null) Resize(arrFUserTable, intFCurrentPrimeNumber);

for (i = 0; i < cardarrVHashTableImage.Count; i++)

{

if (cardarrVHashTableImage[i] != 0)

{

j = cardarrVHashTableImage[i];

HashIndex(objFHeap.arrFHeapTable[j].strFLexicalUnit);

arrFHashTable[intFHashIndex] = j;

if (arrFUserTable != null) arrFUserTable[intFHashIndex] = arrVUserTableImage[i];

THeapItem Th2 = objFHeap.arrFHeapTable[j];

Th2.intFHashIndex = intFHashIndex;

objFHeap.arrFHeapTable[j] = Th2;

}

}

cardarrVHashTableImage.Clear();

if (arrFUserTable != null) arrVUserTableImage.Clear();

}

void Expansion()

{

intFCurrentPrimeNumber = NextPrimeNumber(intFCurrentPrimeNumber);

TableReHashing();

}

object GetUserPointer(int cardILexicalCode)

{

THeapItem Item = objFHeap.arrFHeapTable[cardILexicalCode];

if (Item.intFHashIndex >= cardPTableSize)

{

MessageBox.Show("Индекс пользовательского массива вышел за диапазон!");

return null;

}

else

{

return arrFUserTable[objFHeap.arrFHeapTable[cardILexicalCode].intFHashIndex];

}

}

void SetUserPointer(int cardILexicalCode, object ptrANewPoint)

{

if (objFHeap.arrFHeapTable[cardILexicalCode].intFHashIndex >= cardPTableSize)

MessageBox.Show("Индекс пользовательского массива вышел за диапазон!");

else

arrFUserTable[objFHeap.arrFHeapTable[cardILexicalCode].intFHashIndex] = ptrANewPoint;

}

public void SetUserTable()

{

arrFUserTable = new List<object>();

Resize(arrFUserTable, arrFHashTable.Count);

}

public bool SearchLexicalUnit(string strAlexicalUnit, ref int intALexicalCode) //Поиск

{

HashIndex(strAlexicalUnit);

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

else

{

intALexicalCode = arrFHashTable[intFHashIndex];

return true;

}

}

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

{

HashIndex(strALexicalUnit);

if (arrFHashTable[intFHashIndex] != 0)

{

intALexicalCode = arrFHashTable[intFHashIndex];

return true;

}

else

{

if ((intFItemReserve + 2) > (cardPTableSize \* 0.9))

{

Expansion();

HashIndex(strALexicalUnit);

}

objFHeap.AddLexicalUnit(strALexicalUnit, byteAHashTable, intFHashIndex, ref intALexicalCode);

arrFHashTable[intFHashIndex] = intALexicalCode;

intFItemReserve++;

return false;

}

}

public void DeleteLexicalUnit(string strAlexicalUnit)

{

HashIndex(strAlexicalUnit);

if (arrFHashTable[intFHashIndex] != 0)

{

if (arrFUserTable != null)

{

if (arrFUserTable[intFHashIndex] != null)

MessageBox.Show("Удаление из таблицы связанного данного");

else

{

objFHeap.DeleteLexicalUnit(arrFHashTable[intFHashIndex]);

arrFHashTable[intFHashIndex] = 0;

intFItemReserve--;

TableReHashing();

}

}

else

{

objFHeap.DeleteLexicalUnit(arrFHashTable[intFHashIndex]);

arrFHashTable[intFHashIndex] = 0;

intFItemReserve--;

TableReHashing();

}

}

}

public void DeleteLexicalCode(int cardALexicalCode)

{

int VHashIndex;

VHashIndex = objFHeap.arrFHeapTable[cardALexicalCode].intFHashIndex;

if (arrFHashTable[VHashIndex] != 0)

if (arrFUserTable.Count != 0)

if (arrFUserTable[VHashIndex] != null)

MessageBox.Show("Удаление из таблицы связанного данного");

else

{

objFHeap.DeleteLexicalUnit(cardALexicalCode);

arrFHashTable[VHashIndex] = 0;

intFItemReserve--;

TableReHashing();

}

else

{

objFHeap.DeleteLexicalUnit(cardALexicalCode);

arrFHashTable[VHashIndex] = 0;

intFItemReserve--;

TableReHashing();

}

}

public void Save(ref StreamWriter fl)

{

try

{

fl.WriteLine(cardPTableSize.ToString());

fl.WriteLine(intFItemReserve.ToString());

for (int i = 1; i < cardPTableSize; i++)

fl.Write("\t" + arrFHashTable[i].ToString());

fl.Write("\n");

boolIsSaved = true;

}

catch (InvalidCastException)

{ boolIsSaved = false; }

}

public void GetLexicalUnitList(ref List<string> sList)

{

for (int i = 0; i < arrFHashTable.Count; i++) if (arrFHashTable[i] != 0) sList.Add(objFHeap.arrFHeapTable[arrFHashTable[i]].strFLexicalUnit);

}

}

//HashTableList

public class CHashTableList

{

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

private bool boolFIsSaved;

public bool boolFIsLoaded;

private byte byteFTablesSize;

static THeap objFHeap = new THeap();

//------------------------------------------------------------------------------

public CHashTableList(byte byteATableCount)

{

this.byteFTablesSize = byteATableCount;

objFHeap = new THeap();

Resize(arrFHashTableList, byteATableCount);

}

//------------------------------------------------------------------------------

public byte GetTableNumber(int intALexicalCode)

{

return objFHeap.arrFHeapTable[intALexicalCode].byteFHashTable;

}

//------------------------------------------------------------------------------

public int GetTablesCount()

{

return arrFHashTableList.Count();

}

//------------------------------------------------------------------------------

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

{

if (size > list.Count)

while (size > list.Count)

{

list.Add(new THashTable(ref objFHeap));

}

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

//------------------------------------------------------------------------------

static void Resize(List<object> list, int intANewSize)

{

if (intANewSize > list.Count)

while (intANewSize > list.Count)

list.Add(new object());

else if (intANewSize < list.Count)

while (list.Count - intANewSize > 0)

list.RemoveAt(list.Count - 1);

}

//------------------------------------------------------------------------------

static void Resize(List<int> list, int intANewSize)

{

if (intANewSize > list.Count)

while (intANewSize > list.Count)

list.Add(new Int32());

else if (intANewSize < list.Count)

while (list.Count - intANewSize > 0)

list.RemoveAt(list.Count - 1);

}

//------------------------------------------------------------------------------

public object GetUserData(int intALexicalCode)

{

if ((0 < intALexicalCode) && (intALexicalCode < objFHeap.intPFreeItem))

return arrFHashTableList[GetTableNumber(intALexicalCode)].arrFUserTable[intALexicalCode];

else

{

MessageBox.Show("Неверно задан лексический код при чтении пользовательских данных");

return null;

}

}

//------------------------------------------------------------------------------

public void SetUserData(int intALexicalCode, object objAUserData)

{

if ((0 < intALexicalCode) && (intALexicalCode < objFHeap.intPFreeItem))

{

if (arrFHashTableList[GetTableNumber(intALexicalCode)].arrFUserTable.Count > 0)

arrFHashTableList[GetTableNumber(intALexicalCode)].arrFUserTable[intALexicalCode] = objAUserData;

else

MessageBox.Show("Попытка записи адреса в несозданный массив пользовательских данных!");

}

else MessageBox.Show("Неверно задан лексический код при записи пользовательских данных!");

}

//------------------------------------------------------------------------------

public string GetLexicalUnit(int intALexicalCode)

{

if ((0 < intALexicalCode) && (intALexicalCode < objFHeap.intPFreeItem)) return objFHeap.arrFHeapTable[intALexicalCode].strFLexicalUnit;

else

{

MessageBox.Show("Неверно задан лексический код при чтении пользовательских данных!");

return "";

}

}

//------------------------------------------------------------------------------

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

{

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

}

//------------------------------------------------------------------------------

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

{

if (byteATable >= arrFHashTableList.Count)

{

if (MessageBox.Show("Увеличить количество таблиц?", "Запрашиваемый индекс таблицы не существует.", MessageBoxButtons.YesNo) == DialogResult.Yes)

Resize(arrFHashTableList, byteATable + 1);

else

return false;

}

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

}

//------------------------------------------------------------------------------

public void DeleteLexicalUnit(string strALexicalUnit, byte byteATable)

{

arrFHashTableList[byteATable].DeleteLexicalUnit(strALexicalUnit);

}

//------------------------------------------------------------------------------

public void DeleteLexicalCode(int intALexicalCode)

{

short T = objFHeap.arrFHeapTable[intALexicalCode].byteFHashTable;

arrFHashTableList[T].DeleteLexicalCode(intALexicalCode);

}

//------------------------------------------------------------------------------

public void SetUserTable(byte byteATable)

{

arrFHashTableList[byteATable].SetUserTable();

}

//------------------------------------------------------------------------------

public void Expantion()

{

Resize(arrFHashTableList, ++byteFTablesSize);

}

//------------------------------------------------------------------------------

public void Save(string strAFileName)

{

try

{

StreamWriter fl = File.CreateText(strAFileName);

fl.WriteLine(byteFTablesSize.ToString());

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

fl.Write(arrFHashTableList[i].arrFHashTable.Count.ToString() + "\t");

fl.WriteLine("");

objFHeap.Save(ref fl);

boolFIsSaved = true;

fl.Close();

}

catch (InvalidDataException)

{ boolFIsSaved = false; }

}

//------------------------------------------------------------------------------

// отладка

public void HeapTableView(List<string> sList)

{

objFHeap.HeapTableView(sList);

}

//------------------------------------------------------------------------------

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

{

arrFHashTableList[byteATable].GetLexicalUnitList(ref sList);

}

//------------------------------------------------------------------------------

public int GetHashIndex(byte Table)

{

return arrFHashTableList[Table].intFHashIndex;

}

//------------------------------------------------------------------------------

}

public struct THeapItem

{

public string strFLexicalUnit;

public byte byteFHashTable;

public int intFHashIndex;

public THeapItem(string strALexicalUnit, byte byteATable, int intAHashIndex)

{

strFLexicalUnit = strALexicalUnit;

byteFHashTable = byteATable;

intFHashIndex = intAHashIndex;

}

}

public class THeap

{

public List<THeapItem> arrFHeapTable = new List<THeapItem>();

private List<int> arrFDeleted = new List<int>();

private int intFFreeItem;

bool boolIsSaved;

bool boolIsLoaded;

public bool boolPIsSaved { get { return boolIsSaved; } }

public bool boolPIsLoaded { get { return boolIsLoaded; } }

public int intPFreeItem { get { return intFFreeItem; } }

public THeap()

{

Init();

intFFreeItem = 1;

}

protected void Init()

{

arrFDeleted.Clear();

arrFHeapTable.Clear();

int cnt = 4;

Resize(arrFHeapTable, cnt);

}

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

{

if (size > list.Count)

while (size > list.Count)

list.Add(new THeapItem("", 0, 0));

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

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

{

if (size > list.Count)

while (size > list.Count)

list.Add(new Int32());

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

/\* static void Resize(List<char> list, int size)

{

if (size > list.Count)

while (size > list.Count)

list.Add('0');

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

\*/

public void Expansion()

{

int cardVSize = arrFHeapTable.Count;

cardVSize = cardVSize + cardVSize % 10 + 1;

Resize(arrFHeapTable, cardVSize);

Resize(arrFHeapTable, cardVSize);

}

public void AddLexicalUnit(string strALexicalUnit, byte byteAHashTable, int cardAHashIndex, ref int cardALexicalCode)

{

int intVIndex;

if (arrFDeleted.Count == 0)

{

intVIndex = intFFreeItem;

intFFreeItem++;

if (intFFreeItem >= (Int32)(arrFHeapTable.Count \* 0.9))

Expansion();

}

else

{

intVIndex = arrFDeleted[arrFDeleted.Count - 1];

Resize(arrFDeleted, arrFDeleted.Count - 1);

}

THeapItem Item = arrFHeapTable[intVIndex];

Item.strFLexicalUnit = strALexicalUnit;

Item.byteFHashTable = byteAHashTable;

Item.intFHashIndex = cardAHashIndex;

arrFHeapTable[intVIndex] = Item;

cardALexicalCode = intVIndex;

}

public void DeleteLexicalUnit(int cardALexicalCode)

{

int i;

if (arrFDeleted == null || !arrFDeleted.Any())

i = 0;

else i = arrFDeleted.Count();

Resize(arrFDeleted, i + 1);

arrFDeleted[i] = cardALexicalCode;

THeapItem Item = arrFHeapTable[cardALexicalCode];

Item.strFLexicalUnit = "";

Item.byteFHashTable = 0;

Item.intFHashIndex = 0;

}

public void Save(ref StreamWriter sw)

{

try

{

for (int i = 1; i < arrFHeapTable.Count; i++) //type?

{

if (arrFHeapTable[i].strFLexicalUnit == "")

break;

sw.Write(arrFHeapTable[i].strFLexicalUnit + "\t");

sw.Write(arrFHeapTable[i].byteFHashTable.ToString() + "\t");

sw.WriteLine(arrFHeapTable[i].intFHashIndex.ToString());

}

boolIsSaved = true;

}

catch (Exception) { boolIsSaved = false; }

}

public void Load(ref StreamReader sr)

{

try

{

Init();

int size = arrFHeapTable.Count;

int readSz = 0;

while (true)

{

string line = sr.ReadLine();

if (line == null)

break;

if (++readSz >= size)

{

size \*= 2;

Resize(arrFHeapTable, size);

}

char[] delim = { '\t'/\*,'\n'\*/ };

string[] lines = line.Split(delim);

THeapItem it = arrFHeapTable[readSz];

it.strFLexicalUnit = lines[0];

it.byteFHashTable = Convert.ToByte(lines[1]);

it.intFHashIndex = Convert.ToInt32(lines[2]);

arrFHeapTable[readSz] = it;

}

intFFreeItem = readSz + 1;

boolIsLoaded = true;

}

catch (InvalidCastException)

{ boolIsLoaded = false; }

}

THeapItem GetItem(int i)

{

if (i >= arrFHeapTable.Count)

{

MessageBox.Show("GetИндекс кучи вышел за диапазон!");

THeapItem Item = new THeapItem("", 0, 0);

return Item;

}

else return arrFHeapTable[i];

}

void SetItem(int i, THeapItem NewItem)

{

if (i >= arrFHeapTable.Count)

MessageBox.Show("SetИндекс кучи вышел за диапазон!");

else arrFHeapTable[i] = NewItem;

}

public void HeapTableView(List<string> sList)

{

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

sList.Add(arrFHeapTable[i].strFLexicalUnit);

}

}

}

**uLex.cs:**

**using System;**

**using System.Collections.Generic;**

**using System.Text;**

**namespace nsLex**

**{**

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

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

**public enum TToken**

**{**

**mn, minus, plus, lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty,**

**lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma, lxmdt, lxmText, lxmtz, lxmr, lxmrs, lxmls**

**};**

**public class CLex //класс лексический анализатор**

**{**

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

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

**public TCharType enumFSelectionCharType;**

**public char chrFSelection;**

**private TState enumFState;**

**private int intFSourceRowSelection;**

**private int intFSourceColSelection;**

**private String strFLexicalUnit;**

**private TToken enumFToken;**

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

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

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

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

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

**public int intPSourceRowSelection**

**{**

**get { return intFSourceRowSelection; }**

**set**

**{**

**intFSourceRowSelection = value;**

**}**

**}**

**public int intPSourceColSelection**

**{**

**get { return intFSourceColSelection; }**

**set**

**{**

**intFSourceColSelection = value;**

**}**

**}**

**public CLex()**

**{**

**}**

**public void GetSymbol() //метод класса лексический анализатор**

**{**

**intFSourceColSelection++; // продвигаем номер колонки**

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

**{**

**intFSourceRowSelection++;**

**if (intFSourceRowSelection <= strFSource.Length - 1)**

**{**

**intFSourceColSelection = -1;**

**chrFSelection = '\0';**

**enumFSelectionCharType = TCharType.EndRow;**

**enumFState = TState.Continue;**

**}**

**else**

**{**

**chrFSelection = '\0';**

**enumFSelectionCharType = TCharType.EndText;**

**enumFState = TState.Finish;**

**}**

**}**

**else**

**{**

**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 == '\*' || chrFSelection == '-') enumFSelectionCharType =TCharType.ReservedSymbol;**

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

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

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

**else if (chrFSelection == ';' || chrFSelection == ',' || chrFSelection == '[' || chrFSelection == ']' || chrFSelection == '=' || chrFSelection == ':') enumFSelectionCharType = TCharType.ReservedSymbol;**

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

**enumFState = TState.Continue;**

**}**

**}**

**private void TakeSymbol()**

**{**

**char[] c = { chrFSelection };**

**String s = new string(c);**

**strFLexicalUnit += s;**

**GetSymbol();**

**}**

**public void NextToken()**

**{**

**strFLexicalUnit = "";**

**if (enumFState == TState.Start)**

**{**

**intFSourceRowSelection = 0;**

**intFSourceColSelection = -1;**

**GetSymbol();**

**}**

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

**{**

**GetSymbol();**

**}**

**if (chrFSelection == '/')**

**{**

**GetSymbol();**

**if (chrFSelection == '/')**

**while (enumFSelectionCharType != TCharType.EndRow)**

**{**

**GetSymbol();**

**}**

**GetSymbol();**

**}**

**switch (enumFSelectionCharType)**

**{**

**case TCharType.Letter: //если начинается слово из букв a,b,c,d**

**{**

**// d c b a**

**// AF | AF | BF | CF | DF |**

**// BF | | BF | CF | DF |**

**// CF | | | CF | DF |**

**// DF | | | | DF |**

**AF:**

**{**

**if (chrFSelection == 'd')**

**{**

**TakeSymbol();**

**goto AF;**

**}**

**if (chrFSelection == 'c')**

**{**

**TakeSymbol();**

**goto BF;**

**}**

**if (chrFSelection == 'b')**

**{**

**TakeSymbol();**

**goto CF;**

**}**

**if (chrFSelection == 'a')**

**{**

**TakeSymbol();**

**goto DF;**

**}**

**else**

**{**

**enumFToken = TToken.lxmIdentifier;**

**return;**

**}**

**}**

**BF:**

**{**

**if (chrFSelection == 'd')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'c')**

**{**

**TakeSymbol();**

**goto BF;**

**}**

**if (chrFSelection == 'b')**

**{**

**TakeSymbol();**

**goto CF;**

**}**

**if (chrFSelection == 'a')**

**{**

**TakeSymbol();**

**goto DF;**

**}**

**else**

**{**

**enumFToken = TToken.lxmIdentifier;**

**return;**

**}**

**}**

**CF:**

**{**

**if (chrFSelection == 'd')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'c')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'b')**

**{**

**TakeSymbol();**

**goto CF;**

**}**

**if (chrFSelection == 'a')**

**{**

**TakeSymbol();**

**goto DF;**

**}**

**else**

**{**

**enumFToken = TToken.lxmIdentifier;**

**return;**

**}**

**}**

**DF:**

**{**

**if (chrFSelection == 'd')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'c')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'b')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'a')**

**{**

**TakeSymbol();**

**goto DF;**

**}**

**else**

**{**

**enumFToken = TToken.lxmIdentifier;**

**return;**

**}**

**}**

**}**

**if (chrFSelection == '/')**

**{**

**GetSymbol();**

**if (chrFSelection == '/')**

**while (enumFSelectionCharType != TCharType.EndRow)**

**{**

**GetSymbol();**

**}**

**GetSymbol();**

**}**

**case TCharType.Digit: //если начинается слово из букв – цифр 0 и 1**

**{**

**// 0 1**

**// A | B | C |**

**// B | | D |**

**// C | | E |**

**// D | A | |**

**// E | Ffin | |**

**// Ffin | | G |**

**// G | | H |**

**// H | | Ffin |**

**A:**

**if (chrFSelection == '0')**

**{**

**TakeSymbol();**

**goto B;**

**}**

**else if (chrFSelection == '1')**

**{**

**TakeSymbol();**

**goto C;**

**}**

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

**B:**

**if (chrFSelection == '1')**

**{**

**TakeSymbol();**

**goto D;**

**}**

**else throw new Exception("Ожидался 1");**

**C:**

**if (chrFSelection == '1')**

**{**

**TakeSymbol();**

**goto E;**

**}**

**else throw new Exception("Ожидался 1");**

**D:**

**if (chrFSelection == '0')**

**{**

**TakeSymbol();**

**goto A;**

**}**

**else throw new Exception("Ожидался 0");**

**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("Ожидался 1");**

**G:**

**if (chrFSelection == '1')**

**{**

**TakeSymbol();**

**goto H;**

**}**

**else throw new Exception("Ожидался 1");**

**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 == '(')**

**{**

**strFLexicalUnit = "(";**

**enumFToken = TToken.lxmLeftParenth;**

**GetSymbol();**

**return;**

**}**

**if (chrFSelection == '+')**

**{**

**strFLexicalUnit = "+";**

**enumFToken = TToken.plus;**

**GetSymbol();**

**return;**

**}**

**if (chrFSelection == '\*')**

**{**

**strFLexicalUnit = "\*";**

**enumFToken = TToken.mn;**

**GetSymbol();**

**return;**

**}**

**if (chrFSelection == '-')**

**{**

**enumFToken = TToken.minus;**

**GetSymbol();**

**return;**

**}**

**if (chrFSelection == ')')**

**{**

**strFLexicalUnit = ")";**

**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 == '=')**

**{**

**strFLexicalUnit = "=";**

**enumFToken = TToken.lxmr;**

**GetSymbol();**

**return;**

**}**

**break;**

**}**

**case TCharType.EndText:**

**{**

**enumFToken = TToken.lxmEmpty;**

**break;**

**}**

**}**

**}**

**}**

**}**

**uSyntAnalyzer.cs:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Windows.Forms;

using nsLex;

using тяп\_лаба\_3;

namespace nsSynt

{

class uSyntAnalyzer

{

private String[] strFSource;

private String[] strFMessage;

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

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

public CLex Lex = new CLex();

public TreeView tree;

public void D(TreeNode parent)

{

TreeNode node = new TreeNode("D");

parent.Nodes.Add(node);

//string s = Convert.ToString(Lex.enumPToken);

//MessageBox.Show(s);

//Lex.NextToken();

K(node);

// Lex.NextToken();

if (Lex.enumPToken == TToken.plus)

{

node.Nodes.Add("+");

C(node);

}

}

public void C(TreeNode parent)

{

TreeNode node = new TreeNode("C");

parent.Nodes.Add(node);

if (Lex.enumPToken == TToken.plus)

{

//node.Nodes.Add("+");

Lex.NextToken();

K(node);

if (Lex.enumPToken == TToken.plus)

{

node.Nodes.Add("+");

C(node);

}

}

//else throw new Exception("Ожидалcя +");

}

public void K(TreeNode parent)

{

TreeNode node = new TreeNode("K");

parent.Nodes.Add(node);

A(node);

//Lex.NextToken();

if (Lex.enumPToken == TToken.mn)

{

//node.Nodes.Add("=");

B(node);

}

}

public void B(TreeNode parent)

{

TreeNode node = new TreeNode("B");

parent.Nodes.Add(node);

if (Lex.enumPToken == TToken.mn)

{

node.Nodes.Add("\*");

Lex.NextToken();

A(node);

//Lex.NextToken();

if (Lex.enumPToken == TToken.mn)

{

node.Nodes.Add("\*");

B(node);

}

}

}

public void A(TreeNode parent)

{

TreeNode node = new TreeNode("A");

parent.Nodes.Add(node);

if (Lex.enumPToken == TToken.minus)

{

node.Nodes.Add("-");

Lex.NextToken();

A(node);

}

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

{

node.Nodes.Add("(");

Lex.NextToken();

D(node);

//Lex.NextToken();

if (Lex.enumPToken == TToken.lxmRightParenth)

{

node.Nodes.Add(")");

Lex.NextToken();

// throw new Exception("Konec slova");

}

else throw new Exception("Ожидалась )");

}

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

{

//node.Nodes.Add(Lex.strPLexicalUnit);

O(node);

}

else

{

string s = Convert.ToString(Lex.enumPToken);

MessageBox.Show(s);

throw new Exception("Ожидались - или ( или число");

}

}

public void O(TreeNode parent)

{

TreeNode node = new TreeNode("O");

parent.Nodes.Add(node);

if (Lex.enumPToken == TToken.lxmNumber)

{

node.Nodes.Add(Lex.strPLexicalUnit);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmr)

{

node.Nodes.Add("=");

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier)

{

node.Nodes.Add(Lex.strPLexicalUnit);

//throw new Exception("Konec slova");

}

else throw new Exception("ожидалось число");

}

else throw new Exception("ожидалось =");

}

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

{

node.Nodes.Add(Lex.strPLexicalUnit);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmr)

{

node.Nodes.Add("=");

Lex.NextToken();

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

{

node.Nodes.Add(Lex.strPLexicalUnit);

//throw new Exception("Konec slova");

}

else throw new Exception("ожидалось число или слово");

}

else throw new Exception("ожидалось =");

}

else throw new Exception("Ожидалось число или слово");

Lex.NextToken();

}

}

}

**Form1.cs:**

**using System;**

**using System.Collections.Generic;**

**using System.ComponentModel;**

**using System.Data;**

**using System.Drawing;**

**using System.Text;**

**using System.Windows.Forms;**

**using nsSynt;**

**using nsLex;**

**namespace тяп\_лаба\_3**

**{**

**public partial class Form1 : Form**

**{**

**public CHashTableList htl = new CHashTableList(3);**

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

**{**

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

**listBox1.Items.Clear();**

**listBox2.Items.Clear();**

**listBox3.Items.Clear();**

**htl.TableToStringList(0, listTable);**

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

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

**listTable.Clear();**

**htl.TableToStringList(1, listTable);**

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

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

**listTable.Clear();**

**htl.TableToStringList(2, listTable);**

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

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

**listTable.Clear();**

**}**

**public Form1()**

**{**

**InitializeComponent();**

**tbFSource.AppendText("(cba=cba)\*010110111=cba+cba=cba+cba=010110111" + "\r\n");**

**int n = tbFSource.Lines.Length;**

**}**

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

**{**

**tbFMessage.Clear();**

**uSyntAnalyzer Synt = new uSyntAnalyzer();**

**treeView1.Nodes.Clear();**

**Synt.tree = treeView1;**

**Synt.Lex.strPSource = tbFSource.Lines;**

**Synt.Lex.strPMessage = tbFMessage.Lines;**

**Synt.Lex.enumPState = TState.Start;**

**try**

**{**

**TreeNode start = new TreeNode("Start");**

**treeView1.Nodes.Add(start);**

**Synt.Lex.NextToken();**

**Synt.D(start);**

**Synt.tree.ExpandAll();**

**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;**

**}**

**}**

**//Кнопка удаления слова**

**private void button4\_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) //проверка: слово - номер или идентификатор**

**{**

**case TToken.lxmIdentifier: //если идентификатор**

**{**

**s1 = "id " + Lex.strPLexicalUnit;**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 0); //слово удаляется из таблицы идентификаторов**

**htl.DeleteLexicalCode(0);**

**TablesToMemo(this, e);**

**break;**

**}**

**case TToken.lxmNumber: //если номер**

**{**

**s1 = "num " + Lex.strPLexicalUnit;**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 1); //слово удаляется из таблицы номеров**

**htl.DeleteLexicalCode(0);**

**TablesToMemo(this, e);**

**break;**

**}**

**case TToken.lxmLeftParenth: //если номер**

**{**

**s1 = "Sign " + Lex.strPLexicalUnit;**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 2); //слово удаляется из таблицы номеров**

**htl.DeleteLexicalCode(0);**

**TablesToMemo(this, e);**

**break;**

**}**

**case TToken.lxmr: //если номер**

**{**

**s1 = "Sign " + Lex.strPLexicalUnit;**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 2); //слово удаляется из таблицы номеров**

**htl.DeleteLexicalCode(0);**

**TablesToMemo(this, e);**

**break;**

**}**

**case TToken.lxmRightParenth: //если номер**

**{**

**s1 = "Sign " + Lex.strPLexicalUnit;**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 2); //слово удаляется из таблицы номеров**

**htl.DeleteLexicalCode(0);**

**TablesToMemo(this, e);**

**break;**

**}**

**case TToken.mn: //если номер**

**{**

**s1 = "Sign " + Lex.strPLexicalUnit;**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 2); //слово удаляется из таблицы номеров**

**htl.DeleteLexicalCode(0);**

**TablesToMemo(this, e);**

**break;**

**}**

**case TToken.plus: //если номер**

**{**

**s1 = "Sign " + Lex.strPLexicalUnit;**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 2); //слово удаляется из таблицы номеров**

**htl.DeleteLexicalCode(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;**

**}**

**}**

**//Кнопка замены**

**private void button5\_Click(object sender, EventArgs e)**

**{**

**CLex Lex = new CLex();**

**Lex.strPSource = tbFSource1.Lines;**

**Lex.strPMessage = tbFMessage1.Lines;**

**Lex.intPSourceColSelection = 0;**

**Lex.intPSourceRowSelection = 0;**

**int x = tbFSource1.TextLength;**

**CLex Lex2 = new CLex();**

**Lex2.strPSource = tbFSource2.Lines;**

**Lex2.strPMessage = tbFMessage1.Lines;**

**Lex2.intPSourceColSelection = 0;**

**Lex2.intPSourceRowSelection = 0;**

**int x2 = tbFSource2.TextLength;**

**tbFMessage1.Text = "";**

**try**

**{**

**while (Lex.enumPState != TState.Finish)**

**{**

**Lex.NextToken();**

**switch (Lex.enumPToken) //Проверка 1 - го слова на то, является ли оно номером или идентификатором**

**{**

**case TToken.lxmIdentifier: //Если идентификатор то:**

**{**

**int b = 0;**

**if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, 0, ref b))**

**{**

**while (Lex2.enumPState != TState.Finish)**

**{**

**Lex2.NextToken();**

**switch (Lex2.enumPToken) //Проверка 1 - го слова на то, является ли оно номером или идентификатором**

**{**

**case TToken.lxmIdentifier: //Если является идентификатором, то:**

**{**

**int b1 = 0;**

**if (htl.AddLexicalUnit(Lex2.strPLexicalUnit, 0, ref b1)) //вызывается функция добавления в таблицу идентификаторов 2 - го слова**

**{**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 0); //Удаляется 1 - ое слово**

**TablesToMemo(this, e);**

**}**

**tbFMessage1.AppendText("(Идентификатор был изменен)");**

**break;**

**}**

**case TToken.lxmNumber: //если 2 - ое слово - номер**

**{**

**tbFMessage1.AppendText("(Нельзя заменить идентификатор на номер)");**

**break;**

**}**

**}**

**}**

**}**

**else tbFMessage1.AppendText("(Идентификатор для удаления не найден)");**

**break;**

**}**

**case TToken.lxmNumber: //если 1 - ое слово номер**

**{**

**int b = 0;**

**if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, 1, ref b))**

**{**

**while (Lex2.enumPState != TState.Finish)**

**{**

**Lex2.NextToken();**

**switch (Lex2.enumPToken) //проверка 2 - го слова**

**{**

**case TToken.lxmIdentifier://если 2 - ое слово идентификатор**

**{**

**tbFMessage1.AppendText("(Нельзя заменить номер на идентификатор)");**

**break;**

**}**

**case TToken.lxmNumber: // если 2 - ое слово номер**

**{**

**int b1 = 0;**

**if (htl.AddLexicalUnit(Lex2.strPLexicalUnit, 1, ref b1))//1 - ое слово добавляется в таблицу номеров**

**{**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 1);// 2 - ое слово удаляется**

**TablesToMemo(this, e);**

**}**

**tbFMessage1.AppendText("(Номер был изменен)");**

**break;**

**}**

**}**

**}**

**}**

**else tbFMessage1.AppendText("(Номер для удаления не найден)");**

**break;**

**}**

**}**

**}**

**}**

**catch (Exception exc)**

**{**

**tbFMessage1.Text += exc.Message;**

**tbFSource.Select();**

**tbFSource.SelectionStart = 0;**

**int n1 = 0;**

**for (int i = 0; i < Lex.intPSourceRowSelection; i++) n1 += tbFSource.Lines[i].Length + 2;**

**n1 += Lex.intPSourceColSelection;**

**tbFSource.SelectionLength = n1;**

**}**

**}**

**// Кнопка добавления слов в таблицы**

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

**{**

**CLex Lex = new CLex();**

**Lex.strPSource = tbFSource.Lines;**

**Lex.strPMessage = tbFMessage.Lines;**

**Lex.intPSourceColSelection = 0;**

**Lex.intPSourceRowSelection = 0;**

**int x = tbFSource.TextLength;**

**int y = tbFSource.Lines.Length;**

**tbFMessage.Text = "";**

**try**

**{**

**while (Lex.enumPState != TState.Finish)**

**{**

**Lex.NextToken();**

**string s1 = "", s = "";**

**switch (Lex.enumPToken)**

**{**

**case TToken.lxmIdentifier:**

**{**

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

**if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 0, ref b)) //Вызывается функция добавления слова в таблицу идентификаторов(0)**

**{**

**TablesToMemo(this, e);**

**}**

**break;**

**}**

**case TToken.lxmNumber:**

**{**

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

**if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 1, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)**

**{**

**TablesToMemo(this, e);**

**}**

**break;**

**}**

**case TToken.lxmLeftParenth:**

**{**

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

**if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)**

**{**

**TablesToMemo(this, e);**

**}**

**break;**

**}**

**case TToken.lxmRightParenth:**

**{**

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

**if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)**

**{**

**TablesToMemo(this, e);**

**}**

**break;**

**}**

**case TToken.lxmr:**

**{**

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

**if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)**

**{**

**TablesToMemo(this, e);**

**}**

**break;**

**}**

**case TToken.mn:**

**{**

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

**if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)**

**{**

**TablesToMemo(this, e);**

**}**

**break;**

**}**

**case TToken.plus:**

**{**

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

**if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)**

**{**

**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;**

**}**

**}**

**//Кнопка посика слова в таблицах**

**private void button3\_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();**

**switch (Lex.enumPToken)**

**{**

**case TToken.lxmIdentifier:**

**{**

**int b = 0;**

**if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, 0, ref b)) //Вызов функции поиска, возвращает значение true если было найден такой идентификатор, иначе false**

**{**

**tbFMessage2.AppendText("(Найден идентификатор)");**

**}**

**else tbFMessage2.AppendText("(Идентификатор не найден)");**

**break;**

**}**

**case TToken.lxmNumber:**

**{**

**int b = 0;**

**if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, 1, ref b)) //Вызов функции поиска, возвращает значение true если было найден такой номер, иначе false**

**{**

**tbFMessage2.AppendText("(Найден номер)");**

**}**

**else tbFMessage2.AppendText("(Номер не найден)");**

**break;**

**}**

**}**

**}**

**}**

**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 button6\_Click(object sender, EventArgs e)

{

TreeConverter treeConverter = new TreeConverter();

treeConverter.ConvertBinaryToDecimal(treeView1);

}

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

**{**

**}**

**private void tbFSource2\_TextChanged(object sender, EventArgs e)**

**{**

**}**

**private void tbFSource1\_TextChanged(object sender, EventArgs e)**

**{**

**}**

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

**{**

**}**

**private void label3\_Click(object sender, EventArgs e)**

**{**

**}**

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

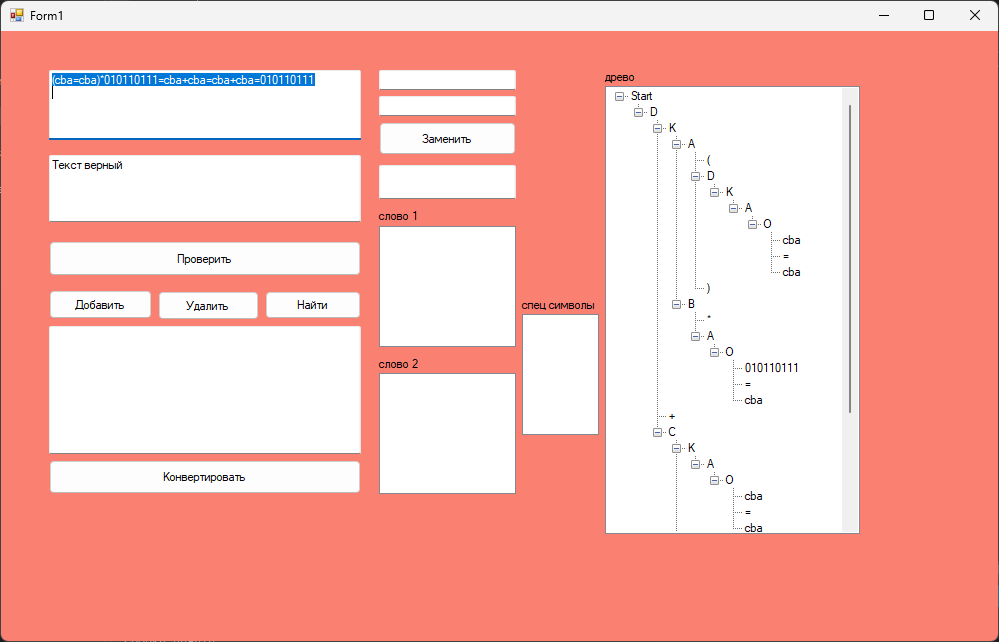
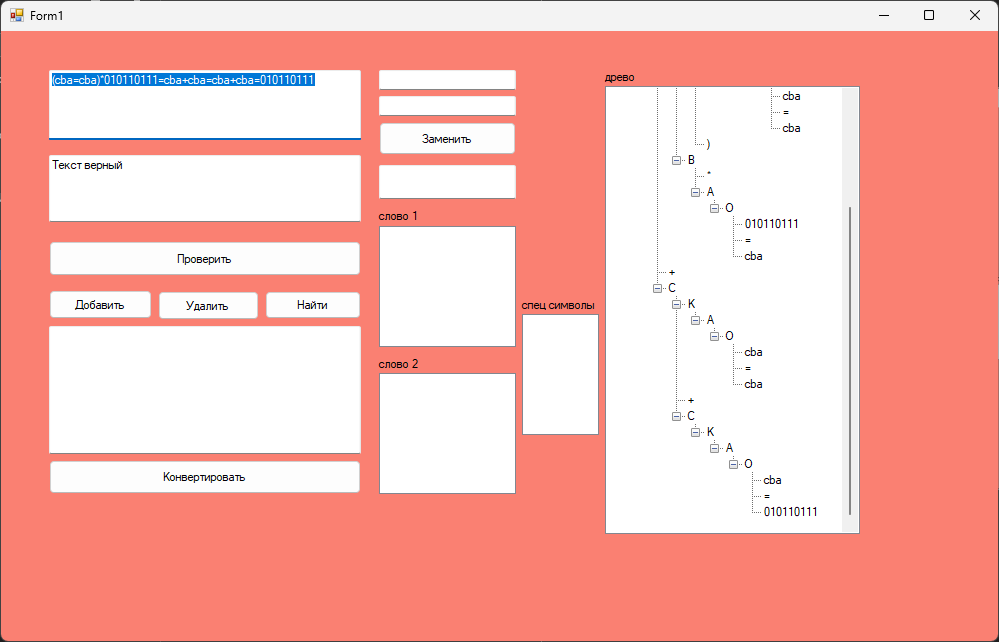
**{**

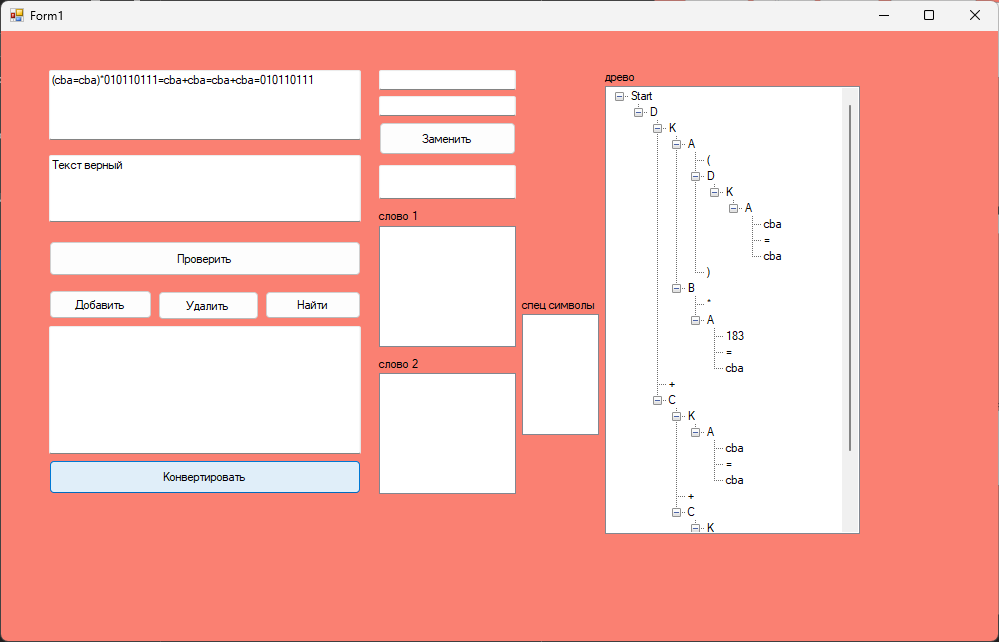
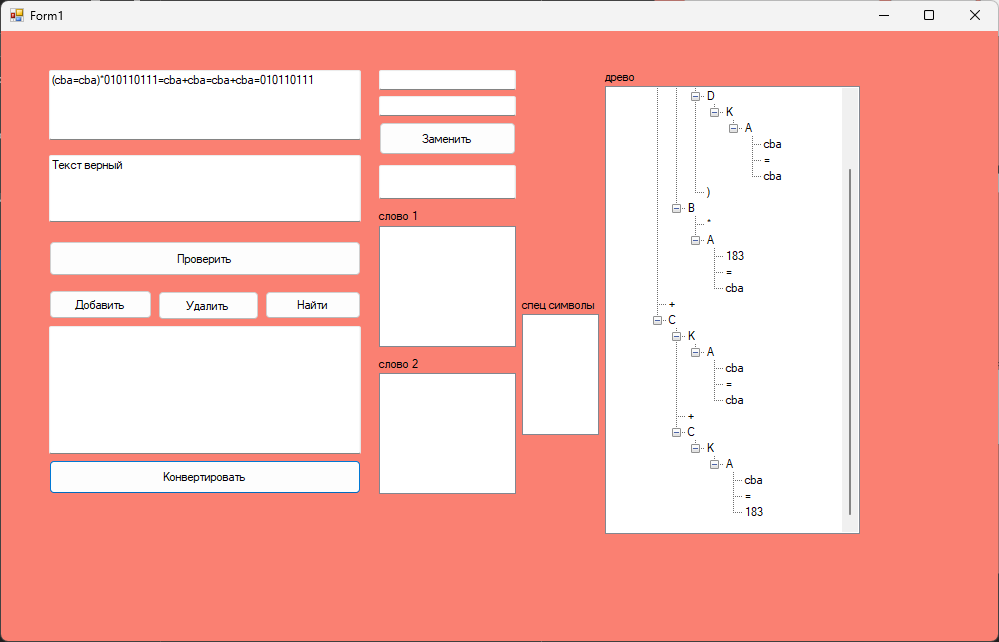
**}**

**}**

**}**

**Результат:**



**Лабораторная работа № 7.**

**Разработка контекстного анализатора**

**Задание:**

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

**Вариант 19:** 

Задание выполнено в конце кода класса uSyntAnalyzer.cs с помощью функции Dublicates, которая служит для поиска дубликатов в TreeView, который представляет собой иерархическую структуру узлов.

1. Метод Duplicates принимает TreeView и строку s в качестве параметров.

2. В начале метода происходит перебор всех узлов верхнего уровня в TreeView с помощью цикла foreach.

3. Для каждого узла вызывается метод Duplicates, который принимает узел и строку s в качестве параметров.

4. В методе Duplicates для узла проверяется, содержит ли он текст "O". Если да, то проверяется, не пустой ли он, и если не пустой, то происходит перебор всех его дочерних узлов.

5. Для каждого дочернего узла происходит проверка его текста на равенство строке s. Если текст совпадает со строкой s, то генерируется исключение с сообщением о повторении идентификатора.

6. После этого происходит рекурсивный вызов метода Duplicates для каждого дочернего узла, чтобы проверить дубликаты на более глубоких уровнях иерархии.

Таким образом, данный код выполняет рекурсивный поиск дубликатов строки s в узлах с текстом "O" и его потомках в TreeView.

**Краткое теоретическое обоснование:**

Контекстный анализатор (иногда также называемый семантическим анализатором) является важным компонентом в процессе компиляции и анализа программных кодов. Его основная задача - проводить анализ программы после синтаксического анализа и проверять семантические правила языка. Разработка контекстного анализатора включает в себя несколько этапов:

**Определение семантических правил:**

Начните с определения семантических правил для вашего языка программирования. Семантические правила определяют, какие операции и действия разрешены в программе, и какие ограничения применяются к типам данных и выражениям.

**Аннотация синтаксического дерева:**

В контекстном анализе сначала строится синтаксическое дерево (или абстрактное синтаксическое дерево) на основе синтаксического анализа. Синтаксическое дерево представляет собой иерархическую структуру программы.

Затем синтаксическое дерево аннотируется семантической информацией. Это включает в себя привязку имен к их объявлениям (разрешение идентификаторов), вычисление типов данных выражений и другие проверки семантической корректности.

**Проверки семантических ошибок:**

Контекстный анализатор проводит проверки на наличие семантических ошибок в программе. Эти ошибки могут включать в себя попытку использования необъявленных переменных, присвоение значений переменным с неподходящими типами данных и другие нарушения семантических правил.

**Управление областями видимости:**

Контекстный анализатор управляет областями видимости переменных и других идентификаторов в программе. Это включает в себя правила для блоков кода, функций и других областей, где действует привязка имен.

**Вычисление констант:**

В некоторых языках программирования контекстный анализатор может также проводить вычисление значений констант на этапе компиляции.

**Генерация семантического дерева или промежуточного представления:**

После успешной проверки семантических правил, контекстный анализатор может сгенерировать семантическое дерево или промежуточное представление программы, которое будет использоваться для дальнейшего анализа и генерации кода.

**Обработка специфических семантических возможностей:**

В зависимости от языка программирования и его особенностей, контекстный анализатор может также обрабатывать специфические семантические конструкции, такие как наследование классов, обработку исключений и другие.

Разработка контекстного анализатора является сложным и важным этапом в создании компиляторов и интерпретаторов. Он обеспечивает проверку корректности программы с точки зрения семантики и создает основу для последующих этапов компиляции или интерпретации.

**Проверка типов данных:**

Один из основных аспектов контекстного анализа - это проверка соответствия типов данных. Контекстный анализатор определяет типы данных выражений и операций, а также убеждается в их совместимости. Например, он проверяет, что операторы, применяемые к переменным или значениям, имеют совместимые типы.

**Обработка объявлений и областей видимости:**

Контекстный анализатор управляет объявлениями переменных, функций, классов и других идентификаторов в программе. Он гарантирует, что переменные объявлены до их использования, что имена уникальны в рамках своих областей видимости и что правила областей видимости соблюдаются.

**Разрешение идентификаторов:**

Одной из задач контекстного анализатора является разрешение идентификаторов. Это процесс, при котором анализатор связывает каждое использование идентификатора с его объявлением, чтобы определить, на что именно ссылается данное использование.

**Проверка семантических правил:**

Контекстный анализатор применяет семантические правила языка программирования. Эти правила могут включать в себя ограничения на использование ключевых слов, операторов и структур данных.

**Оптимизации и определение констант:**

Некоторые контекстные анализаторы также проводят оптимизации кода на этапе компиляции, например, вычисление значений констант. Это может повысить производительность программы или сгенерированного кода.

**Обработка специфичных семантических конструкций:**

Контекстный анализатор может обрабатывать специфические семантические конструкции языка программирования, такие как наследование и полиморфизм в объектно-ориентированных языках или работу с указателями в языках C/C++.

**Выдача сообщений об ошибках:**

Контекстный анализатор генерирует сообщения об ошибках, если находит нарушения семантических правил. Эти сообщения об ошибках помогают разработчикам исправить проблемы в коде.

**Код программы:**

**TreeConventer.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace тяп\_лаба\_3

{

internal class TreeConverter

{

public void ConvertBinaryToDecimal(TreeView tree)

{

foreach (TreeNode node in tree.Nodes)

{

ConvertBinaryToDecimal(node);

}

}

private void ConvertBinaryToDecimal(TreeNode node)

{

if (node.Text == "O")

{

if (node.Nodes.Count == 3)

{

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;

}

}

}

**THashTable.cs:**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace nsSynt

{

public class THashTable

{

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

private int intFCurrentPrimeNumber;

private int intFItemReserve;

private bool boolIsSaved;

public int intFHashIndex;

public int cardPTableSize { get { return arrFHashTable.Count; } }

public List<object> arrFUserTable = null;

static THeap objFHeap;

public THashTable(ref THeap objAHeap)

{

objFHeap = objAHeap;

Init(7);

intFItemReserve = 0;

}

public void Init(int count)

{

arrFHashTable.Clear();

Resize(arrFHashTable, count);

intFCurrentPrimeNumber = count;

}

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

{

if (size > list.Count)

while (size > list.Count)

list.Add(new object());

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

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

{

if (size > list.Count)

while (size > list.Count)

list.Add(new Int32());

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

int NextPrimeNumber(int cardAOldPrimeNumber)

{

int intVLowerBound, intVUpperBound, intVNextPrimeNumber;

bool boolVIsDivisor;

intVNextPrimeNumber = cardAOldPrimeNumber + cardAOldPrimeNumber / 10 + 1; // увеличиваем на 10 процентов

if ((intVNextPrimeNumber % 2) == 0) intVNextPrimeNumber++;

do

{

boolVIsDivisor = true; intVNextPrimeNumber = intVNextPrimeNumber + 2;

intVLowerBound = 3; intVUpperBound = intVNextPrimeNumber / 3 + 1; // диапазон делителей

while (boolVIsDivisor && (intVLowerBound < intVUpperBound))

{

if ((intVNextPrimeNumber % intVLowerBound) == 0) boolVIsDivisor = false;

else intVLowerBound = intVLowerBound + 2;

}

} while (!boolVIsDivisor);

return intVNextPrimeNumber;

}

//HashFunc

UInt32 HashFunction\_Wainberger(string strALexicalUnit)

{

UInt32 h = 0/\*, seed=131313\*/;

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

{

h += Convert.ToUInt32(strALexicalUnit[i] % 12); //Хэширование

}

return h;

}

int ReHashFunction\_Line(int h, string strALexicalUnit)

{

if (h == 0) h = arrFHashTable.Count / 3;

else if (h == 1) h = arrFHashTable.Count \* 3 / 4;

int i = 1, hi = h;

bool boolVFinish = false;

do

{

if (arrFHashTable[hi] == 0) boolVFinish = true;

else

if (objFHeap.arrFHeapTable[arrFHashTable[hi]].strFLexicalUnit == strALexicalUnit)

boolVFinish = true;

else

{

i++;

hi = (h % i) % (Int32)(arrFHashTable.Count); //Рехэширование(это метод для избавления от коллизии)

}

} while (!boolVFinish);

return hi;

}

public void HashIndex(string strALexicalUnit)

{

int h;

h = (Int32)HashFunction\_Wainberger(strALexicalUnit) % (Int32)(arrFHashTable.Count);

intFHashIndex = ReHashFunction\_Line(h, strALexicalUnit);

}

void TableReHashing()

{

int i, j;

List<int> cardarrVHashTableImage = new List<int>();

List<object> arrVUserTableImage = new List<object>();

Resize(cardarrVHashTableImage, arrFHashTable.Count);

if (arrFUserTable != null)

Resize(arrVUserTableImage, arrFHashTable.Count);

for (i = 0; i < arrFHashTable.Count; i++)

{

cardarrVHashTableImage[i] = arrFHashTable[i];

if (arrFUserTable != null) arrVUserTableImage[i] = arrFUserTable[i];

}

arrFHashTable.Clear();

if (arrFUserTable != null) arrFUserTable.Clear();

Resize(arrFHashTable, intFCurrentPrimeNumber);

if (arrFUserTable != null) Resize(arrFUserTable, intFCurrentPrimeNumber);

for (i = 0; i < cardarrVHashTableImage.Count; i++)

{

if (cardarrVHashTableImage[i] != 0)

{

j = cardarrVHashTableImage[i];

HashIndex(objFHeap.arrFHeapTable[j].strFLexicalUnit);

arrFHashTable[intFHashIndex] = j;

if (arrFUserTable != null) arrFUserTable[intFHashIndex] = arrVUserTableImage[i];

THeapItem Th2 = objFHeap.arrFHeapTable[j];

Th2.intFHashIndex = intFHashIndex;

objFHeap.arrFHeapTable[j] = Th2;

}

}

cardarrVHashTableImage.Clear();

if (arrFUserTable != null) arrVUserTableImage.Clear();

}

void Expansion()

{

intFCurrentPrimeNumber = NextPrimeNumber(intFCurrentPrimeNumber);

TableReHashing();

}

object GetUserPointer(int cardILexicalCode)

{

THeapItem Item = objFHeap.arrFHeapTable[cardILexicalCode];

if (Item.intFHashIndex >= cardPTableSize)

{

MessageBox.Show("Индекс пользовательского массива вышел за диапазон!");

return null;

}

else

{

return arrFUserTable[objFHeap.arrFHeapTable[cardILexicalCode].intFHashIndex];

}

}

void SetUserPointer(int cardILexicalCode, object ptrANewPoint)

{

if (objFHeap.arrFHeapTable[cardILexicalCode].intFHashIndex >= cardPTableSize)

MessageBox.Show("Индекс пользовательского массива вышел за диапазон!");

else

arrFUserTable[objFHeap.arrFHeapTable[cardILexicalCode].intFHashIndex] = ptrANewPoint;

}

public void SetUserTable()

{

arrFUserTable = new List<object>();

Resize(arrFUserTable, arrFHashTable.Count);

}

public bool SearchLexicalUnit(string strAlexicalUnit, ref int intALexicalCode) //Поиск

{

HashIndex(strAlexicalUnit);

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

else

{

intALexicalCode = arrFHashTable[intFHashIndex];

return true;

}

}

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

{

HashIndex(strALexicalUnit);

if (arrFHashTable[intFHashIndex] != 0)

{

intALexicalCode = arrFHashTable[intFHashIndex];

return true;

}

else

{

if ((intFItemReserve + 2) > (cardPTableSize \* 0.9))

{

Expansion();

HashIndex(strALexicalUnit);

}

objFHeap.AddLexicalUnit(strALexicalUnit, byteAHashTable, intFHashIndex, ref intALexicalCode);

arrFHashTable[intFHashIndex] = intALexicalCode;

intFItemReserve++;

return false;

}

}

public void DeleteLexicalUnit(string strAlexicalUnit)

{

HashIndex(strAlexicalUnit);

if (arrFHashTable[intFHashIndex] != 0)

{

if (arrFUserTable != null)

{

if (arrFUserTable[intFHashIndex] != null)

MessageBox.Show("Удаление из таблицы связанного данного");

else

{

objFHeap.DeleteLexicalUnit(arrFHashTable[intFHashIndex]);

arrFHashTable[intFHashIndex] = 0;

intFItemReserve--;

TableReHashing();

}

}

else

{

objFHeap.DeleteLexicalUnit(arrFHashTable[intFHashIndex]);

arrFHashTable[intFHashIndex] = 0;

intFItemReserve--;

TableReHashing();

}

}

}

public void DeleteLexicalCode(int cardALexicalCode)

{

int VHashIndex;

VHashIndex = objFHeap.arrFHeapTable[cardALexicalCode].intFHashIndex;

if (arrFHashTable[VHashIndex] != 0)

if (arrFUserTable.Count != 0)

if (arrFUserTable[VHashIndex] != null)

MessageBox.Show("Удаление из таблицы связанного данного");

else

{

objFHeap.DeleteLexicalUnit(cardALexicalCode);

arrFHashTable[VHashIndex] = 0;

intFItemReserve--;

TableReHashing();

}

else

{

objFHeap.DeleteLexicalUnit(cardALexicalCode);

arrFHashTable[VHashIndex] = 0;

intFItemReserve--;

TableReHashing();

}

}

public void Save(ref StreamWriter fl)

{

try

{

fl.WriteLine(cardPTableSize.ToString());

fl.WriteLine(intFItemReserve.ToString());

for (int i = 1; i < cardPTableSize; i++)

fl.Write("\t" + arrFHashTable[i].ToString());

fl.Write("\n");

boolIsSaved = true;

}

catch (InvalidCastException)

{ boolIsSaved = false; }

}

public void GetLexicalUnitList(ref List<string> sList)

{

for (int i = 0; i < arrFHashTable.Count; i++) if (arrFHashTable[i] != 0) sList.Add(objFHeap.arrFHeapTable[arrFHashTable[i]].strFLexicalUnit);

}

}

//HashTableList

public class CHashTableList

{

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

private bool boolFIsSaved;

public bool boolFIsLoaded;

private byte byteFTablesSize;

static THeap objFHeap = new THeap();

//------------------------------------------------------------------------------

public CHashTableList(byte byteATableCount)

{

this.byteFTablesSize = byteATableCount;

objFHeap = new THeap();

Resize(arrFHashTableList, byteATableCount);

}

//------------------------------------------------------------------------------

public byte GetTableNumber(int intALexicalCode)

{

return objFHeap.arrFHeapTable[intALexicalCode].byteFHashTable;

}

//------------------------------------------------------------------------------

public int GetTablesCount()

{

return arrFHashTableList.Count();

}

//------------------------------------------------------------------------------

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

{

if (size > list.Count)

while (size > list.Count)

{

list.Add(new THashTable(ref objFHeap));

}

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

//------------------------------------------------------------------------------

static void Resize(List<object> list, int intANewSize)

{

if (intANewSize > list.Count)

while (intANewSize > list.Count)

list.Add(new object());

else if (intANewSize < list.Count)

while (list.Count - intANewSize > 0)

list.RemoveAt(list.Count - 1);

}

//------------------------------------------------------------------------------

static void Resize(List<int> list, int intANewSize)

{

if (intANewSize > list.Count)

while (intANewSize > list.Count)

list.Add(new Int32());

else if (intANewSize < list.Count)

while (list.Count - intANewSize > 0)

list.RemoveAt(list.Count - 1);

}

//------------------------------------------------------------------------------

public object GetUserData(int intALexicalCode)

{

if ((0 < intALexicalCode) && (intALexicalCode < objFHeap.intPFreeItem))

return arrFHashTableList[GetTableNumber(intALexicalCode)].arrFUserTable[intALexicalCode];

else

{

MessageBox.Show("Неверно задан лексический код при чтении пользовательских данных");

return null;

}

}

//------------------------------------------------------------------------------

public void SetUserData(int intALexicalCode, object objAUserData)

{

if ((0 < intALexicalCode) && (intALexicalCode < objFHeap.intPFreeItem))

{

if (arrFHashTableList[GetTableNumber(intALexicalCode)].arrFUserTable.Count > 0)

arrFHashTableList[GetTableNumber(intALexicalCode)].arrFUserTable[intALexicalCode] = objAUserData;

else

MessageBox.Show("Попытка записи адреса в несозданный массив пользовательских данных!");

}

else MessageBox.Show("Неверно задан лексический код при записи пользовательских данных!");

}

//------------------------------------------------------------------------------

public string GetLexicalUnit(int intALexicalCode)

{

if ((0 < intALexicalCode) && (intALexicalCode < objFHeap.intPFreeItem)) return objFHeap.arrFHeapTable[intALexicalCode].strFLexicalUnit;

else

{

MessageBox.Show("Неверно задан лексический код при чтении пользовательских данных!");

return "";

}

}

//------------------------------------------------------------------------------

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

{

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

}

//------------------------------------------------------------------------------

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

{

if (byteATable >= arrFHashTableList.Count)

{

if (MessageBox.Show("Увеличить количество таблиц?", "Запрашиваемый индекс таблицы не существует.", MessageBoxButtons.YesNo) == DialogResult.Yes)

Resize(arrFHashTableList, byteATable + 1);

else

return false;

}

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

}

//------------------------------------------------------------------------------

public void DeleteLexicalUnit(string strALexicalUnit, byte byteATable)

{

arrFHashTableList[byteATable].DeleteLexicalUnit(strALexicalUnit);

}

//------------------------------------------------------------------------------

public void DeleteLexicalCode(int intALexicalCode)

{

short T = objFHeap.arrFHeapTable[intALexicalCode].byteFHashTable;

arrFHashTableList[T].DeleteLexicalCode(intALexicalCode);

}

//------------------------------------------------------------------------------

public void SetUserTable(byte byteATable)

{

arrFHashTableList[byteATable].SetUserTable();

}

//------------------------------------------------------------------------------

public void Expantion()

{

Resize(arrFHashTableList, ++byteFTablesSize);

}

//------------------------------------------------------------------------------

public void Save(string strAFileName)

{

try

{

StreamWriter fl = File.CreateText(strAFileName);

fl.WriteLine(byteFTablesSize.ToString());

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

fl.Write(arrFHashTableList[i].arrFHashTable.Count.ToString() + "\t");

fl.WriteLine("");

objFHeap.Save(ref fl);

boolFIsSaved = true;

fl.Close();

}

catch (InvalidDataException)

{ boolFIsSaved = false; }

}

//------------------------------------------------------------------------------

// отладка

public void HeapTableView(List<string> sList)

{

objFHeap.HeapTableView(sList);

}

//------------------------------------------------------------------------------

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

{

arrFHashTableList[byteATable].GetLexicalUnitList(ref sList);

}

//------------------------------------------------------------------------------

public int GetHashIndex(byte Table)

{

return arrFHashTableList[Table].intFHashIndex;

}

//------------------------------------------------------------------------------

}

public struct THeapItem

{

public string strFLexicalUnit;

public byte byteFHashTable;

public int intFHashIndex;

public THeapItem(string strALexicalUnit, byte byteATable, int intAHashIndex)

{

strFLexicalUnit = strALexicalUnit;

byteFHashTable = byteATable;

intFHashIndex = intAHashIndex;

}

}

public class THeap

{

public List<THeapItem> arrFHeapTable = new List<THeapItem>();

private List<int> arrFDeleted = new List<int>();

private int intFFreeItem;

bool boolIsSaved;

bool boolIsLoaded;

public bool boolPIsSaved { get { return boolIsSaved; } }

public bool boolPIsLoaded { get { return boolIsLoaded; } }

public int intPFreeItem { get { return intFFreeItem; } }

public THeap()

{

Init();

intFFreeItem = 1;

}

protected void Init()

{

arrFDeleted.Clear();

arrFHeapTable.Clear();

int cnt = 4;

Resize(arrFHeapTable, cnt);

}

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

{

if (size > list.Count)

while (size > list.Count)

list.Add(new THeapItem("", 0, 0));

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

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

{

if (size > list.Count)

while (size > list.Count)

list.Add(new Int32());

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

/\* static void Resize(List<char> list, int size)

{

if (size > list.Count)

while (size > list.Count)

list.Add('0');

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

\*/

public void Expansion()

{

int cardVSize = arrFHeapTable.Count;

cardVSize = cardVSize + cardVSize % 10 + 1;

Resize(arrFHeapTable, cardVSize);

Resize(arrFHeapTable, cardVSize);

}

public void AddLexicalUnit(string strALexicalUnit, byte byteAHashTable, int cardAHashIndex, ref int cardALexicalCode)

{

int intVIndex;

if (arrFDeleted.Count == 0)

{

intVIndex = intFFreeItem;

intFFreeItem++;

if (intFFreeItem >= (Int32)(arrFHeapTable.Count \* 0.9))

Expansion();

}

else

{

intVIndex = arrFDeleted[arrFDeleted.Count - 1];

Resize(arrFDeleted, arrFDeleted.Count - 1);

}

THeapItem Item = arrFHeapTable[intVIndex];

Item.strFLexicalUnit = strALexicalUnit;

Item.byteFHashTable = byteAHashTable;

Item.intFHashIndex = cardAHashIndex;

arrFHeapTable[intVIndex] = Item;

cardALexicalCode = intVIndex;

}

public void DeleteLexicalUnit(int cardALexicalCode)

{

int i;

if (arrFDeleted == null || !arrFDeleted.Any())

i = 0;

else i = arrFDeleted.Count();

Resize(arrFDeleted, i + 1);

arrFDeleted[i] = cardALexicalCode;

THeapItem Item = arrFHeapTable[cardALexicalCode];

Item.strFLexicalUnit = "";

Item.byteFHashTable = 0;

Item.intFHashIndex = 0;

}

public void Save(ref StreamWriter sw)

{

try

{

for (int i = 1; i < arrFHeapTable.Count; i++) //type?

{

if (arrFHeapTable[i].strFLexicalUnit == "")

break;

sw.Write(arrFHeapTable[i].strFLexicalUnit + "\t");

sw.Write(arrFHeapTable[i].byteFHashTable.ToString() + "\t");

sw.WriteLine(arrFHeapTable[i].intFHashIndex.ToString());

}

boolIsSaved = true;

}

catch (Exception) { boolIsSaved = false; }

}

public void Load(ref StreamReader sr)

{

try

{

Init();

int size = arrFHeapTable.Count;

int readSz = 0;

while (true)

{

string line = sr.ReadLine();

if (line == null)

break;

if (++readSz >= size)

{

size \*= 2;

Resize(arrFHeapTable, size);

}

char[] delim = { '\t'/\*,'\n'\*/ };

string[] lines = line.Split(delim);

THeapItem it = arrFHeapTable[readSz];

it.strFLexicalUnit = lines[0];

it.byteFHashTable = Convert.ToByte(lines[1]);

it.intFHashIndex = Convert.ToInt32(lines[2]);

arrFHeapTable[readSz] = it;

}

intFFreeItem = readSz + 1;

boolIsLoaded = true;

}

catch (InvalidCastException)

{ boolIsLoaded = false; }

}

THeapItem GetItem(int i)

{

if (i >= arrFHeapTable.Count)

{

MessageBox.Show("GetИндекс кучи вышел за диапазон!");

THeapItem Item = new THeapItem("", 0, 0);

return Item;

}

else return arrFHeapTable[i];

}

void SetItem(int i, THeapItem NewItem)

{

if (i >= arrFHeapTable.Count)

MessageBox.Show("SetИндекс кучи вышел за диапазон!");

else arrFHeapTable[i] = NewItem;

}

public void HeapTableView(List<string> sList)

{

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

sList.Add(arrFHeapTable[i].strFLexicalUnit);

}

}

}

**uLex.cs:**

**using System;**

**using System.Collections.Generic;**

**using System.Text;**

**namespace nsLex**

**{**

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

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

**public enum TToken**

**{**

**mn, minus, plus, lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty,**

**lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma, lxmdt, lxmText, lxmtz, lxmr, lxmrs, lxmls**

**};**

**public class CLex //класс лексический анализатор**

**{**

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

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

**public TCharType enumFSelectionCharType;**

**public char chrFSelection;**

**private TState enumFState;**

**private int intFSourceRowSelection;**

**private int intFSourceColSelection;**

**private String strFLexicalUnit;**

**private TToken enumFToken;**

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

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

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

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

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

**public int intPSourceRowSelection**

**{**

**get { return intFSourceRowSelection; }**

**set**

**{**

**intFSourceRowSelection = value;**

**}**

**}**

**public int intPSourceColSelection**

**{**

**get { return intFSourceColSelection; }**

**set**

**{**

**intFSourceColSelection = value;**

**}**

**}**

**public CLex()**

**{**

**}**

**public void GetSymbol() //метод класса лексический анализатор**

**{**

**intFSourceColSelection++; // продвигаем номер колонки**

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

**{**

**intFSourceRowSelection++;**

**if (intFSourceRowSelection <= strFSource.Length - 1)**

**{**

**intFSourceColSelection = -1;**

**chrFSelection = '\0';**

**enumFSelectionCharType = TCharType.EndRow;**

**enumFState = TState.Continue;**

**}**

**else**

**{**

**chrFSelection = '\0';**

**enumFSelectionCharType = TCharType.EndText;**

**enumFState = TState.Finish;**

**}**

**}**

**else**

**{**

**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 == '\*' || chrFSelection == '-') enumFSelectionCharType =TCharType.ReservedSymbol;**

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

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

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

**else if (chrFSelection == ';' || chrFSelection == ',' || chrFSelection == '[' || chrFSelection == ']' || chrFSelection == '=' || chrFSelection == ':') enumFSelectionCharType = TCharType.ReservedSymbol;**

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

**enumFState = TState.Continue;**

**}**

**}**

**private void TakeSymbol()**

**{**

**char[] c = { chrFSelection };**

**String s = new string(c);**

**strFLexicalUnit += s;**

**GetSymbol();**

**}**

**public void NextToken()**

**{**

**strFLexicalUnit = "";**

**if (enumFState == TState.Start)**

**{**

**intFSourceRowSelection = 0;**

**intFSourceColSelection = -1;**

**GetSymbol();**

**}**

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

**{**

**GetSymbol();**

**}**

**if (chrFSelection == '/')**

**{**

**GetSymbol();**

**if (chrFSelection == '/')**

**while (enumFSelectionCharType != TCharType.EndRow)**

**{**

**GetSymbol();**

**}**

**GetSymbol();**

**}**

**switch (enumFSelectionCharType)**

**{**

**case TCharType.Letter: //если начинается слово из букв a,b,c,d**

**{**

**// d c b a**

**// AF | AF | BF | CF | DF |**

**// BF | | BF | CF | DF |**

**// CF | | | CF | DF |**

**// DF | | | | DF |**

**AF:**

**{**

**if (chrFSelection == 'd')**

**{**

**TakeSymbol();**

**goto AF;**

**}**

**if (chrFSelection == 'c')**

**{**

**TakeSymbol();**

**goto BF;**

**}**

**if (chrFSelection == 'b')**

**{**

**TakeSymbol();**

**goto CF;**

**}**

**if (chrFSelection == 'a')**

**{**

**TakeSymbol();**

**goto DF;**

**}**

**else**

**{**

**enumFToken = TToken.lxmIdentifier;**

**return;**

**}**

**}**

**BF:**

**{**

**if (chrFSelection == 'd')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'c')**

**{**

**TakeSymbol();**

**goto BF;**

**}**

**if (chrFSelection == 'b')**

**{**

**TakeSymbol();**

**goto CF;**

**}**

**if (chrFSelection == 'a')**

**{**

**TakeSymbol();**

**goto DF;**

**}**

**else**

**{**

**enumFToken = TToken.lxmIdentifier;**

**return;**

**}**

**}**

**CF:**

**{**

**if (chrFSelection == 'd')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'c')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'b')**

**{**

**TakeSymbol();**

**goto CF;**

**}**

**if (chrFSelection == 'a')**

**{**

**TakeSymbol();**

**goto DF;**

**}**

**else**

**{**

**enumFToken = TToken.lxmIdentifier;**

**return;**

**}**

**}**

**DF:**

**{**

**if (chrFSelection == 'd')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'c')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'b')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'a')**

**{**

**TakeSymbol();**

**goto DF;**

**}**

**else**

**{**

**enumFToken = TToken.lxmIdentifier;**

**return;**

**}**

**}**

**}**

**if (chrFSelection == '/')**

**{**

**GetSymbol();**

**if (chrFSelection == '/')**

**while (enumFSelectionCharType != TCharType.EndRow)**

**{**

**GetSymbol();**

**}**

**GetSymbol();**

**}**

**case TCharType.Digit: //если начинается слово из букв – цифр 0 и 1**

**{**

**// 0 1**

**// A | B | C |**

**// B | | D |**

**// C | | E |**

**// D | A | |**

**// E | Ffin | |**

**// Ffin | | G |**

**// G | | H |**

**// H | | Ffin |**

**A:**

**if (chrFSelection == '0')**

**{**

**TakeSymbol();**

**goto B;**

**}**

**else if (chrFSelection == '1')**

**{**

**TakeSymbol();**

**goto C;**

**}**

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

**B:**

**if (chrFSelection == '1')**

**{**

**TakeSymbol();**

**goto D;**

**}**

**else throw new Exception("Ожидался 1");**

**C:**

**if (chrFSelection == '1')**

**{**

**TakeSymbol();**

**goto E;**

**}**

**else throw new Exception("Ожидался 1");**

**D:**

**if (chrFSelection == '0')**

**{**

**TakeSymbol();**

**goto A;**

**}**

**else throw new Exception("Ожидался 0");**

**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("Ожидался 1");**

**G:**

**if (chrFSelection == '1')**

**{**

**TakeSymbol();**

**goto H;**

**}**

**else throw new Exception("Ожидался 1");**

**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 == '(')**

**{**

**strFLexicalUnit = "(";**

**enumFToken = TToken.lxmLeftParenth;**

**GetSymbol();**

**return;**

**}**

**if (chrFSelection == '+')**

**{**

**strFLexicalUnit = "+";**

**enumFToken = TToken.plus;**

**GetSymbol();**

**return;**

**}**

**if (chrFSelection == '\*')**

**{**

**strFLexicalUnit = "\*";**

**enumFToken = TToken.mn;**

**GetSymbol();**

**return;**

**}**

**if (chrFSelection == '-')**

**{**

**enumFToken = TToken.minus;**

**GetSymbol();**

**return;**

**}**

**if (chrFSelection == ')')**

**{**

**strFLexicalUnit = ")";**

**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 == '=')**

**{**

**strFLexicalUnit = "=";**

**enumFToken = TToken.lxmr;**

**GetSymbol();**

**return;**

**}**

**break;**

**}**

**case TCharType.EndText:**

**{**

**enumFToken = TToken.lxmEmpty;**

**break;**

**}**

**}**

**}**

**}**

**}**

**uSyntAnalyzer.cs:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Windows.Forms;

using nsLex;

using тяп\_лаба\_3;

namespace nsSynt

{

class uSyntAnalyzer

{

private String[] strFSource;

private String[] strFMessage;

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

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

public CLex Lex = new CLex();

public TToken firstToken;

/\*public List<string> numTokenList = new List<string>();

public bool noDuplications = true; \*/

public TreeView tree;

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

{

if (token == TToken.lxmNumber)

{

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

{

if (input == numTokenList[i])

{

noDuplications = false;

}

}

numTokenList.Add(input);

}

parent.Nodes.Add(new TreeNode(input)); \*/

public void D(TreeNode parent)

{

TreeNode node = new TreeNode("D");

parent.Nodes.Add(node);

//string s = Convert.ToString(Lex.enumPToken);

//MessageBox.Show(s);

//Lex.NextToken();

K(node);

// Lex.NextToken();

if (Lex.enumPToken == TToken.plus)

{

node.Nodes.Add("+");

C(node);

}

}

public void C(TreeNode parent)

{

TreeNode node = new TreeNode("C");

parent.Nodes.Add(node);

if (Lex.enumPToken == TToken.plus)

{

//node.Nodes.Add("+");

Lex.NextToken();

K(node);

if (Lex.enumPToken == TToken.plus)

{

node.Nodes.Add("+");

C(node);

}

}

//else throw new Exception("Ожидалcя +");

}

public void K(TreeNode parent)

{

TreeNode node = new TreeNode("K");

parent.Nodes.Add(node);

A(node);

//Lex.NextToken();

if (Lex.enumPToken == TToken.mn)

{

//node.Nodes.Add("=");

B(node);

}

}

public void B(TreeNode parent)

{

TreeNode node = new TreeNode("B");

parent.Nodes.Add(node);

if (Lex.enumPToken == TToken.mn)

{

node.Nodes.Add("\*");

Lex.NextToken();

A(node);

//Lex.NextToken();

if (Lex.enumPToken == TToken.mn)

{

node.Nodes.Add("\*");

B(node);

}

}

}

public void A(TreeNode parent)

{

TreeNode node = new TreeNode("A");

parent.Nodes.Add(node);

if (Lex.enumPToken == TToken.minus)

{

node.Nodes.Add("-");

Lex.NextToken();

A(node);

}

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

{

node.Nodes.Add("(");

Lex.NextToken();

D(node);

//Lex.NextToken();

if (Lex.enumPToken == TToken.lxmRightParenth)

{

node.Nodes.Add(")");

Lex.NextToken();

// throw new Exception("Konec slova");

}

else throw new Exception("Ожидалась )");

}

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

{

//node.Nodes.Add(Lex.strPLexicalUnit);

O(node);

}

else

{

string s = Convert.ToString(Lex.enumPToken);

MessageBox.Show(s);

throw new Exception("Ожидались - или ( или число");

}

}

public void O(TreeNode parent)

{

TreeNode node = new TreeNode("O");

parent.Nodes.Add(node);

if (Lex.enumPToken == TToken.lxmNumber)

{

node.Nodes.Add(Lex.strPLexicalUnit);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmr)

{

node.Nodes.Add("=");

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier)

{

node.Nodes.Add(Lex.strPLexicalUnit);

//throw new Exception("Konec slova");

}

else throw new Exception("ожидалось число");

}

else throw new Exception("ожидалось =");

}

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

{

Duplicates(tree, Lex.strPLexicalUnit);

node.Nodes.Add(Lex.strPLexicalUnit);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmr)

{

node.Nodes.Add("=");

Lex.NextToken();

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

{

if (Lex.enumPToken == TToken.lxmIdentifier)

{

Duplicates(tree, Lex.strPLexicalUnit);

}

node.Nodes.Add(Lex.strPLexicalUnit);

//throw new Exception("Konec slova");

}

else throw new Exception("ожидалось число или слово");

}

else throw new Exception("ожидалось =");

}

else throw new Exception("Ожидалось число или слово");

Lex.NextToken();

}

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 == "O") // Если попали в ветку под названием O, там 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 O, тем самым рассматривая каждый лексический юнит

{

if (node2.Text == s)

{

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

}

}

}

}

foreach (TreeNode childNode in node.Nodes)

{

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

}

}

}

}

**Form1.cs:**

**using System;**

**using System.Collections.Generic;**

**using System.ComponentModel;**

**using System.Data;**

**using System.Drawing;**

**using System.Text;**

**using System.Windows.Forms;**

**using nsSynt;**

**using nsLex;**

**namespace тяп\_лаба\_3**

**{**

**public partial class Form1 : Form**

**{**

**public CHashTableList htl = new CHashTableList(3);**

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

**{**

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

**listBox1.Items.Clear();**

**listBox2.Items.Clear();**

**listBox3.Items.Clear();**

**htl.TableToStringList(0, listTable);**

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

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

**listTable.Clear();**

**htl.TableToStringList(1, listTable);**

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

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

**listTable.Clear();**

**htl.TableToStringList(2, listTable);**

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

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

**listTable.Clear();**

**}**

**public Form1()**

**{**

**InitializeComponent();**

**tbFSource.AppendText("(cba=cba)\*010110111=cba+cba=cba+cba=010110111" + "\r\n");**

**int n = tbFSource.Lines.Length;**

**}**

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

**{**

**tbFMessage.Clear();**

**uSyntAnalyzer Synt = new uSyntAnalyzer();**

**treeView1.Nodes.Clear();**

**Synt.tree = treeView1;**

**Synt.Lex.strPSource = tbFSource.Lines;**

**Synt.Lex.strPMessage = tbFMessage.Lines;**

**Synt.Lex.enumPState = TState.Start;**

**try**

**{**

**TreeNode start = new TreeNode("Start");**

**treeView1.Nodes.Add(start);**

**Synt.Lex.NextToken();**

**Synt.D(start);**

**Synt.tree.ExpandAll();**

**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;**

**}**

**}**

**//Кнопка удаления слова**

**private void button4\_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) //проверка: слово - номер или идентификатор**

**{**

**case TToken.lxmIdentifier: //если идентификатор**

**{**

**s1 = "id " + Lex.strPLexicalUnit;**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 0); //слово удаляется из таблицы идентификаторов**

**htl.DeleteLexicalCode(0);**

**TablesToMemo(this, e);**

**break;**

**}**

**case TToken.lxmNumber: //если номер**

**{**

**s1 = "num " + Lex.strPLexicalUnit;**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 1); //слово удаляется из таблицы номеров**

**htl.DeleteLexicalCode(0);**

**TablesToMemo(this, e);**

**break;**

**}**

**case TToken.lxmLeftParenth: //если номер**

**{**

**s1 = "Sign " + Lex.strPLexicalUnit;**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 2); //слово удаляется из таблицы номеров**

**htl.DeleteLexicalCode(0);**

**TablesToMemo(this, e);**

**break;**

**}**

**case TToken.lxmr: //если номер**

**{**

**s1 = "Sign " + Lex.strPLexicalUnit;**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 2); //слово удаляется из таблицы номеров**

**htl.DeleteLexicalCode(0);**

**TablesToMemo(this, e);**

**break;**

**}**

**case TToken.lxmRightParenth: //если номер**

**{**

**s1 = "Sign " + Lex.strPLexicalUnit;**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 2); //слово удаляется из таблицы номеров**

**htl.DeleteLexicalCode(0);**

**TablesToMemo(this, e);**

**break;**

**}**

**case TToken.mn: //если номер**

**{**

**s1 = "Sign " + Lex.strPLexicalUnit;**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 2); //слово удаляется из таблицы номеров**

**htl.DeleteLexicalCode(0);**

**TablesToMemo(this, e);**

**break;**

**}**

**case TToken.plus: //если номер**

**{**

**s1 = "Sign " + Lex.strPLexicalUnit;**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 2); //слово удаляется из таблицы номеров**

**htl.DeleteLexicalCode(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;**

**}**

**}**

**//Кнопка замены**

**private void button5\_Click(object sender, EventArgs e)**

**{**

**CLex Lex = new CLex();**

**Lex.strPSource = tbFSource1.Lines;**

**Lex.strPMessage = tbFMessage1.Lines;**

**Lex.intPSourceColSelection = 0;**

**Lex.intPSourceRowSelection = 0;**

**int x = tbFSource1.TextLength;**

**CLex Lex2 = new CLex();**

**Lex2.strPSource = tbFSource2.Lines;**

**Lex2.strPMessage = tbFMessage1.Lines;**

**Lex2.intPSourceColSelection = 0;**

**Lex2.intPSourceRowSelection = 0;**

**int x2 = tbFSource2.TextLength;**

**tbFMessage1.Text = "";**

**try**

**{**

**while (Lex.enumPState != TState.Finish)**

**{**

**Lex.NextToken();**

**switch (Lex.enumPToken) //Проверка 1 - го слова на то, является ли оно номером или идентификатором**

**{**

**case TToken.lxmIdentifier: //Если идентификатор то:**

**{**

**int b = 0;**

**if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, 0, ref b))**

**{**

**while (Lex2.enumPState != TState.Finish)**

**{**

**Lex2.NextToken();**

**switch (Lex2.enumPToken) //Проверка 1 - го слова на то, является ли оно номером или идентификатором**

**{**

**case TToken.lxmIdentifier: //Если является идентификатором, то:**

**{**

**int b1 = 0;**

**if (htl.AddLexicalUnit(Lex2.strPLexicalUnit, 0, ref b1)) //вызывается функция добавления в таблицу идентификаторов 2 - го слова**

**{**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 0); //Удаляется 1 - ое слово**

**TablesToMemo(this, e);**

**}**

**tbFMessage1.AppendText("(Идентификатор был изменен)");**

**break;**

**}**

**case TToken.lxmNumber: //если 2 - ое слово - номер**

**{**

**tbFMessage1.AppendText("(Нельзя заменить идентификатор на номер)");**

**break;**

**}**

**}**

**}**

**}**

**else tbFMessage1.AppendText("(Идентификатор для удаления не найден)");**

**break;**

**}**

**case TToken.lxmNumber: //если 1 - ое слово номер**

**{**

**int b = 0;**

**if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, 1, ref b))**

**{**

**while (Lex2.enumPState != TState.Finish)**

**{**

**Lex2.NextToken();**

**switch (Lex2.enumPToken) //проверка 2 - го слова**

**{**

**case TToken.lxmIdentifier://если 2 - ое слово идентификатор**

**{**

**tbFMessage1.AppendText("(Нельзя заменить номер на идентификатор)");**

**break;**

**}**

**case TToken.lxmNumber: // если 2 - ое слово номер**

**{**

**int b1 = 0;**

**if (htl.AddLexicalUnit(Lex2.strPLexicalUnit, 1, ref b1))//1 - ое слово добавляется в таблицу номеров**

**{**

**htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 1);// 2 - ое слово удаляется**

**TablesToMemo(this, e);**

**}**

**tbFMessage1.AppendText("(Номер был изменен)");**

**break;**

**}**

**}**

**}**

**}**

**else tbFMessage1.AppendText("(Номер для удаления не найден)");**

**break;**

**}**

**}**

**}**

**}**

**catch (Exception exc)**

**{**

**tbFMessage1.Text += exc.Message;**

**tbFSource.Select();**

**tbFSource.SelectionStart = 0;**

**int n1 = 0;**

**for (int i = 0; i < Lex.intPSourceRowSelection; i++) n1 += tbFSource.Lines[i].Length + 2;**

**n1 += Lex.intPSourceColSelection;**

**tbFSource.SelectionLength = n1;**

**}**

**}**

**// Кнопка добавления слов в таблицы**

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

**{**

**CLex Lex = new CLex();**

**Lex.strPSource = tbFSource.Lines;**

**Lex.strPMessage = tbFMessage.Lines;**

**Lex.intPSourceColSelection = 0;**

**Lex.intPSourceRowSelection = 0;**

**int x = tbFSource.TextLength;**

**int y = tbFSource.Lines.Length;**

**tbFMessage.Text = "";**

**try**

**{**

**while (Lex.enumPState != TState.Finish)**

**{**

**Lex.NextToken();**

**string s1 = "", s = "";**

**switch (Lex.enumPToken)**

**{**

**case TToken.lxmIdentifier:**

**{**

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

**if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 0, ref b)) //Вызывается функция добавления слова в таблицу идентификаторов(0)**

**{**

**TablesToMemo(this, e);**

**}**

**break;**

**}**

**case TToken.lxmNumber:**

**{**

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

**if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 1, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)**

**{**

**TablesToMemo(this, e);**

**}**

**break;**

**}**

**case TToken.lxmLeftParenth:**

**{**

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

**if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)**

**{**

**TablesToMemo(this, e);**

**}**

**break;**

**}**

**case TToken.lxmRightParenth:**

**{**

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

**if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)**

**{**

**TablesToMemo(this, e);**

**}**

**break;**

**}**

**case TToken.lxmr:**

**{**

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

**if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)**

**{**

**TablesToMemo(this, e);**

**}**

**break;**

**}**

**case TToken.mn:**

**{**

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

**if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)**

**{**

**TablesToMemo(this, e);**

**}**

**break;**

**}**

**case TToken.plus:**

**{**

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

**if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)**

**{**

**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;**

**}**

**}**

**//Кнопка посика слова в таблицах**

**private void button3\_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();**

**switch (Lex.enumPToken)**

**{**

**case TToken.lxmIdentifier:**

**{**

**int b = 0;**

**if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, 0, ref b)) //Вызов функции поиска, возвращает значение true если было найден такой идентификатор, иначе false**

**{**

**tbFMessage2.AppendText("(Найден идентификатор)");**

**}**

**else tbFMessage2.AppendText("(Идентификатор не найден)");**

**break;**

**}**

**case TToken.lxmNumber:**

**{**

**int b = 0;**

**if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, 1, ref b)) //Вызов функции поиска, возвращает значение true если было найден такой номер, иначе false**

**{**

**tbFMessage2.AppendText("(Найден номер)");**

**}**

**else tbFMessage2.AppendText("(Номер не найден)");**

**break;**

**}**

**}**

**}**

**}**

**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 button6\_Click(object sender, EventArgs e)

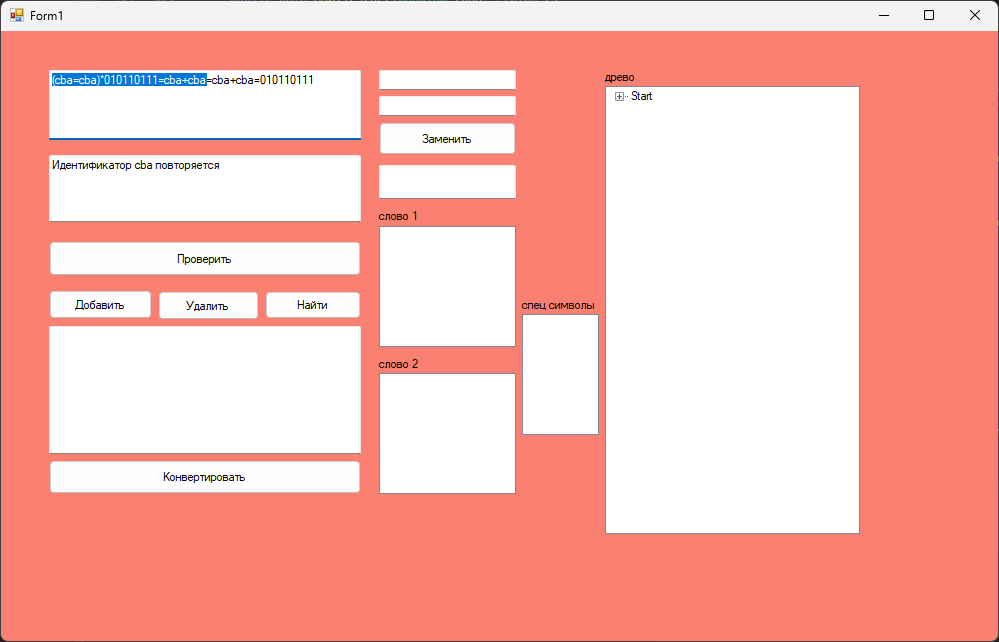
{

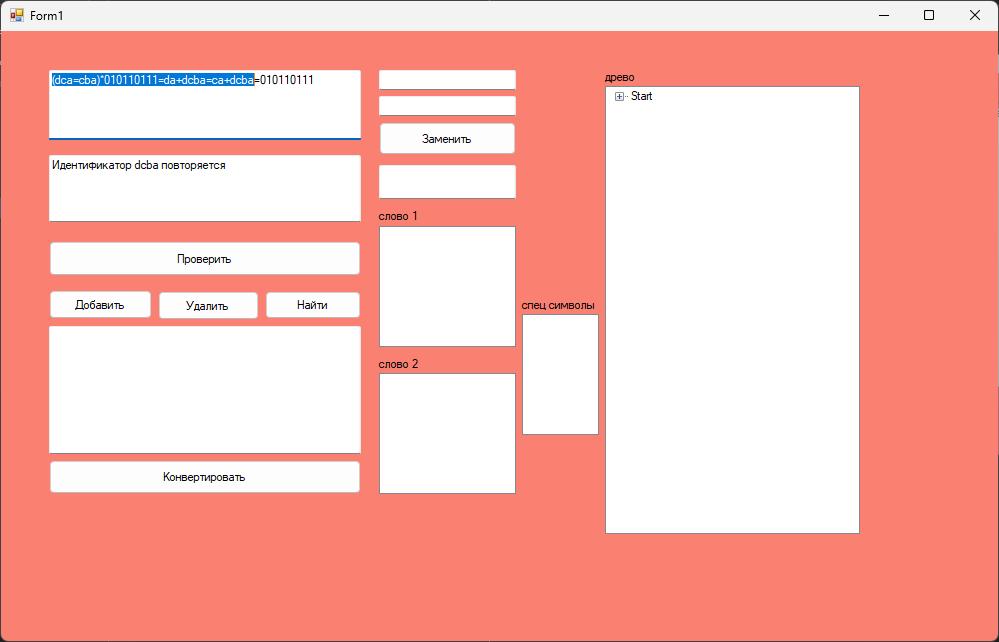
TreeConverter treeConverter = new TreeConverter();

treeConverter.ConvertBinaryToDecimal(treeView1);

}

**Результат:**





**Лабораторная работа № 8.**

**Разработка семантического анализатора.**

**Задание:**

Разработать семантический анализатор. Выполнить проверку внеконтекстной грамматики.

**Вариант задания 1**: Проверка идентификатора слова <1> на наличие не менее 2 букв ‘d’ и не более 1 буквы ‘c’, проверка того, что длина идентификатора слова <1> будет больше десятичного представления идентификатора числа <2>.

**Вариант задания 2:** Если идентификатором <1> является слово “ccb”, то идентификатором <2> должно является только слово “cccba”

Задание выполнено в классе **uSemantAnalyzer.cs,** показаны оба варианта выполнения задания. Объяснение кода выполнено в самом классе с помощью комментариев зеленого цвета.

**Краткое теоретическое обоснование:**

Семантика – раздел лингвистики, изучающий смысловое значение единиц языка. Процесс человеческого мышления, как и язык, который представляет собой инструмент выражения мыслей, является очень гибким и трудно поддается формализации. Поэтому семантический анализ по праву считается самым сложным этапом автоматической обработки текстов.

Создание новых методов семантического анализа текстов откроет новые возможности и позволит существенно продвинуться в решении многих задач компьютерной лингвистики, таких как машинный перевод, автореферирование, классификация текстов и других. Не менее актуальна разработка новых инструментов, позволяющих автоматизировать семантический анализ.

Основные семантические ошибки:

* Несоответствие типов;
* Необъявленная переменная
* Использование зарезервированного идентификатора
* Многократное объявление переменной в области видимости
* Доступ к переменной вне области
* Фактическое и формальное несоответствие параметров

Чтобы помочь в своей работе, семантический анализатор обычно строит и поддерживает структуру данных *таблицы символов*, которая сопоставляет каждый идентификатор с информацией, известной о нем. Помимо прочего, эта информация включает в себя тип идентификатора, внутреннюю структуру (если таковая имеется) и область действия (часть программы, в которой он действителен).

Используя таблицу символов, семантический анализатор применяет большое разнообразие правил, которые не улавливаются иерархической структурой контекстно-свободной грамматики и дерева синтаксического анализа. Например, в C он проверяет, что

■ Каждый идентификатор объявляется перед его использованием.

■ Идентификатор не используется в неподходящем контексте (вызов целого числа в качестве подпрограммы, добавление строки к целому числу, ссылка на поле неправильного типа структуры и т. Д.).

■ Вызовы подпрограмм обеспечивают правильное количество и типы аргументов.

■ Метки на плечах оператора switch являются различными константами.

■ Любая функция с типом возврата non-void возвращает значение явно.

Во многих компиляторах работа семантического анализатора принимает форму *семантических подпрограмм действий*, вызываемых синтаксическим анализатором, когда он понимает, что достиг определенной точки в пределах грамматического правила.

Семантический анализаториспользует синтаксическое дерево и информацию из таблицы символов для проверки исходной программы на семантическую согласованность с определением языка. Он также собирает информацию о типах и сохраняет ее в синтаксическом дереве или в таблице символов для последующего использования в процессе генерации промежуточного кода.

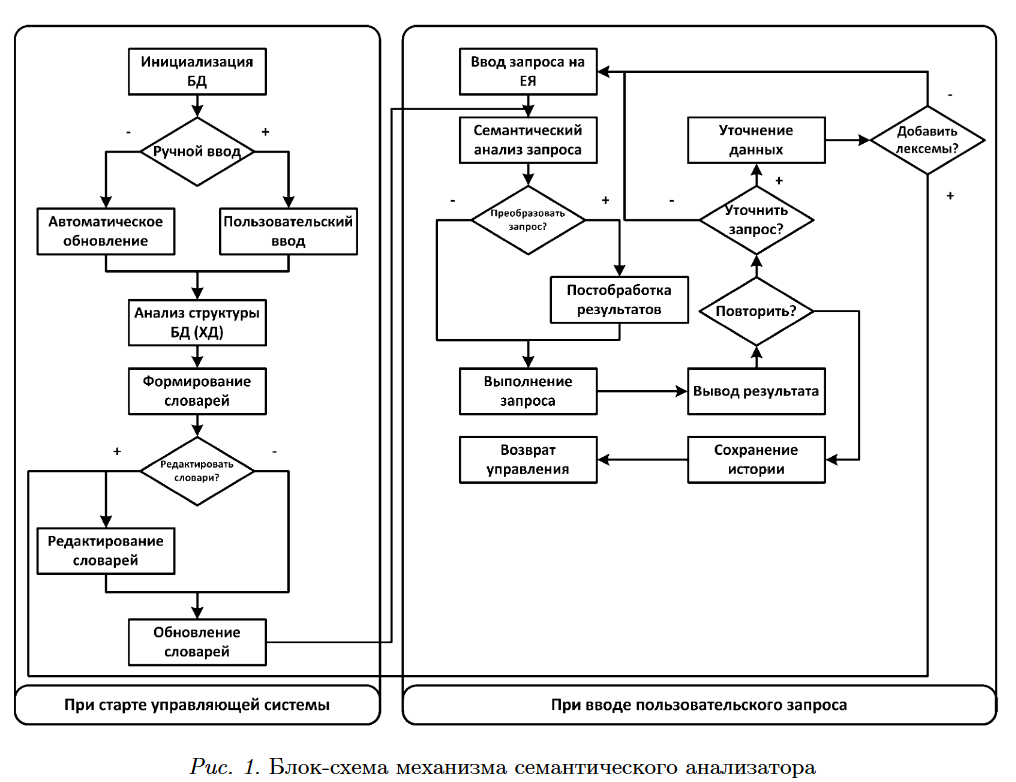
Важной частью семантического анализа является *проверка типов*, когда компилятор проверяет, имеет ли каждый оператор операнды соответствующего типа. Например, многие определения языков программирования требуют, чтобы индекс массива был целым числом; компилятор должен сообщить об ошибке, если в качестве индекса массива используется число с плавающей точкой.

Рассмотрим пример, описание типа переменной в Си вида int a,b=5,c[10] синтаксически реализуется группой правил, задающих цикл описания отдельных переменных, а общий тип и полученный список определяется отдельным правилом. Но между ними имеется связь, состоящая в том, что семантика типа int распространяется на все элементы списка. Отсюда имеем различные варианты реализации семантических процедур:

* если синтаксическая единица реализуется в одном правиле, то ее семантическая обработка., а также генерация кода или интерпретация могут быть выполнены независимой семантической процедурой. Взаимодействие с другими процедурами происходит по указанной выше схеме: процедура использует ссылки на семантику символов правой части (потомков поддерева для этого правила), формирует и возвращает ссылку на семантику левой части (корневой вершины поддерева). То же самое можно сказать и способе передачи сгенерированного кода или результата интерпретации;
* если синтаксическая единица реализуется несколькими правилами, то такая стройная картина нарушается. Например, может потребоваться более сложное взаимодействие семантических процедур (не только вверх-вниз по дереву, но и между смежными вершинами и т. п.). Для передачи результатов между процедурами могут использоваться не только ссылки на записи в семантических таблицах, но и более сложные структуры данных.

В приведенном выше примере правила, соединяющее общий тип описания (int) со списком переменных, имеет вид X::=TL и получает от первого нетерминала ссылку в семантической таблице типов, а от второго нетерминала множество ссылок (массив, список) на записи в семантической таблице переменных. Семантическая процедура правила должна дополнить все цепочки описаний типа данных в этом множестве записей указанным общим типов (причем по семантике определения типа данных в Си он должен быть дописан в конец цепочки).

Рассмотрим алгоритм функционирования предлагаемого механизма семантического анализатора в виде блок-схемы (рис. 1).



Прокомментируем работу алгоритма.

1. При инициализации система анализирует БД или ХД и на основе анализа описания данных, представленного при помощи XML-файла либо встроенного механизма платформы АИС, строит структуру используемых объектов.  
2. На основе построенной структуры объектов и их связей формируется проблемно-ориентированное подмножество ЕЯ.

3. Удаленный пользователь или оператор консоли (далее по тексту используется термин «оператор терминала» – ОТ) вводит запрос на ЕЯ на основе заполненных словарей.

4. Используя механизм семантического анализатора выполняется анализ введенного ОТ запроса на ЕЯ и строится соответствующий ему запрос на языке 1CQ.

5. Если АИС не поддерживает язык запросов 1CQ, то запрос конвертируется в стандарт языка SQL (переводится на английский язык, корректируется синтаксис и т. п.)

6. Имея в распоряжении код на языках запросов 1CQ или SQL – машинный запрос, платформа АИС выполняет его и выдает результат ОТ.

**Код программы:**

**uSemantAnalyzer.cs: (1 вариант)**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace тяп\_лаба\_3

{

internal class uSemantAnalyzer

{

public int i = 0;

public string strSlovo;

public string strNum;

private TreeView tree;

public uSemantAnalyzer()

{

}

public uSemantAnalyzer(TreeView treeView) /\* - Принимает объект TreeView в качестве параметра и инициализирует поле tree.

- Вызывает метод TreeController для анализа всего дерева. \*/

{

tree = treeView;

TreeController(tree);

}

public void TreeController(TreeView tree) /\* - Проходит по всем узлам верхнего уровня в дереве tree и вызывает перегруженный метод TreeController для каждого узла. \*/

{

foreach (TreeNode node in tree.Nodes)

{

TreeController(node);

}

}

public void TreeController(TreeNode node) /\* - Анализирует каждый узел дерева.

- Если узел имеет текст "O" и содержит более одного дочернего узла:

- Извлекает значения из узлов дерева.

- Вызывает метод Check для проверки условий и окрашивает узел в случае ошибки.

- Рекурсивно вызывает себя для каждого дочернего узла. \*/

{

if (node.Text == "O")

{

if (node.Nodes.Count > 1)

{

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

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

Check(strSlovo, strNum, node);

}

}

foreach (TreeNode childNode in node.Nodes)

{

TreeController(childNode);

}

}

private void Check(string Slovo, string num, TreeNode node) /\* - Производит проверку условий семантики.

- Переводит строку num в целое число numToInt.

- Если в идентификаторе (строка id) менее двух букв 'd' и больше одной буквы 'c', а также длина идентификатора меньше либо равна десятичному представлению числа numToInt:

- Окрашивает узел в дереве в розовый цвет и выбрасывает исключение с соответствующим сообщением.

- Выполняется для любого возможного случая\*/

{

int numToInt = Convert.ToInt32(num, 2);

if (Slovo.Count(i => i == 'd') <= 2 || Slovo.Count(i => i == 'c') > 1)

{

if (numToInt >= Slovo.Length)

{

tree.SelectedNode = node;

tree.SelectedNode.BackColor = Color.HotPink;

throw new Exception($"В идентификаторе {Slovo} меньше двух букв 'd' или больше одной буквы 'c' \n, и его длина меньше десятичного представления числа {strNum}");

}

else

{

tree.SelectedNode = node;

tree.SelectedNode.BackColor = Color.HotPink;

throw new Exception($"В идентификаторе {Slovo} меньше двух букв 'd' или больше одной буквы 'c'");

}

}

else if (numToInt >= Slovo.Length)

{

tree.SelectedNode = node;

tree.SelectedNode.BackColor = Color.HotPink;

throw new Exception($"Длина идентификатора {Slovo} меньше десятичного представления числа {strNum}");

}

if (Slovo.Count(i => i == 'd') > 2 && Slovo.Count(i => i == 'c') <= 1 && numToInt <= Slovo.Length)

{

throw new Exception($"Текст верный , cемантический анализатор тоже доволен");

}

}

}

}

**uSemantAnalyzer.cs: (2 вариант)**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace тяп\_лаба\_3

{

internal class uSemantAnalyzer

{

public int i = 0;

public string strSlovo;

public string strNum;

private TreeView tree;

public uSemantAnalyzer()

{

}

public uSemantAnalyzer(TreeView treeView) /\* - Принимает объект TreeView в качестве параметра и инициализирует поле tree.

- Вызывает метод TreeController для анализа всего дерева. \*/

{

tree = treeView;

TreeController(tree);

}

public void TreeController(TreeView tree) /\* - Проходит по всем узлам верхнего уровня в дереве tree и вызывает перегруженный метод TreeController для каждого узла. \*/

{

foreach (TreeNode node in tree.Nodes)

{

TreeController(node);

}

}

public void TreeController(TreeNode node) /\* - Анализирует каждый узел дерева.

- Если узел имеет текст "O" и содержит более одного дочернего узла:

- Извлекает значения из узлов дерева.

- Вызывает метод Check для проверки условий и окрашивает узел в случае ошибки.

- Рекурсивно вызывает себя для каждого дочернего узла. \*/

{

if (node.Text == "O")

{

if (node.Nodes.Count > 1)

{

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

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

Check(strSlovo, strNum, node);

}

}

foreach (TreeNode childNode in node.Nodes)

{

TreeController(childNode);

}

}

private void Check(string Slovo, string num, TreeNode node) /\* - Производит проверку условий семантики.

- Переводит строку num в целое число numToInt.

- Выполняет проверку значения каждого слова

- Окрашивает узел в дереве в розовый цвет и выбрасывает исключение с соответствующим сообщением.

- Выполняется для любого возможного случая\*/

{

if (Slovo == "ccb")

{

if (num != "cccba")

{

tree.SelectedNode = node;

tree.SelectedNode.BackColor = Color.HotPink;

throw new Exception($"после слова ccb должно идти только слово cccba");

}

if (num == "cccba")

{

tree.SelectedNode = node;

tree.SelectedNode.BackColor = Color.HotPink;

throw new Exception($"Текст верный , cемантический анализатор доволен");

}

}

}

}

}

**Form1.cs:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using nsSynt;

using nsLex;

namespace тяп\_лаба\_3

{

public partial class Form1 : Form

{

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

public CHashTableList htl = new CHashTableList(3);

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

{

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

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

htl.TableToStringList(0, listTable);

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

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

listTable.Clear();

htl.TableToStringList(1, listTable);

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

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

listTable.Clear();

htl.TableToStringList(2, listTable);

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

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

listTable.Clear();

}

public Form1()

{

InitializeComponent();

tbFSource.AppendText("(dddcbaa=110)\*010110111=d+ccb=cccba+ddca=110111" + "\r\n");

int n = tbFSource.Lines.Length;

}

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

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

treeView1.Nodes.Clear();

Synt.tree = treeView1;

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.Lines;

Synt.Lex.enumPState = TState.Start;

try

{

TreeNode start = new TreeNode("Start");

treeView1.Nodes.Add(start);

Synt.Lex.NextToken();

Synt.D(start);

Synt.tree.ExpandAll();

//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;

}

SemantFunction();

}

public void SemantFunction()

{

try

{

uSemantAnalyzer Semant = new uSemantAnalyzer(treeView1);

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

}

}

//Кнопка удаления слова

private void button4\_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) //проверка: слово - номер или идентификатор

{

case TToken.lxmIdentifier: //если идентификатор

{

s1 = "id " + Lex.strPLexicalUnit;

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 0); //слово удаляется из таблицы идентификаторов

htl.DeleteLexicalCode(0);

TablesToMemo(this, e);

break;

}

case TToken.lxmNumber: //если номер

{

s1 = "num " + Lex.strPLexicalUnit;

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 1); //слово удаляется из таблицы номеров

htl.DeleteLexicalCode(0);

TablesToMemo(this, e);

break;

}

case TToken.lxmLeftParenth: //если номер

{

s1 = "Sign " + Lex.strPLexicalUnit;

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 2); //слово удаляется из таблицы номеров

htl.DeleteLexicalCode(0);

TablesToMemo(this, e);

break;

}

case TToken.lxmr: //если номер

{

s1 = "Sign " + Lex.strPLexicalUnit;

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 2); //слово удаляется из таблицы номеров

htl.DeleteLexicalCode(0);

TablesToMemo(this, e);

break;

}

case TToken.lxmRightParenth: //если номер

{

s1 = "Sign " + Lex.strPLexicalUnit;

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 2); //слово удаляется из таблицы номеров

htl.DeleteLexicalCode(0);

TablesToMemo(this, e);

break;

}

case TToken.mn: //если номер

{

s1 = "Sign " + Lex.strPLexicalUnit;

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 2); //слово удаляется из таблицы номеров

htl.DeleteLexicalCode(0);

TablesToMemo(this, e);

break;

}

case TToken.plus: //если номер

{

s1 = "Sign " + Lex.strPLexicalUnit;

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 2); //слово удаляется из таблицы номеров

htl.DeleteLexicalCode(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;

}

}

//Кнопка замены

private void button5\_Click(object sender, EventArgs e)

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource1.Lines;

Lex.strPMessage = tbFMessage1.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = tbFSource1.TextLength;

CLex Lex2 = new CLex();

Lex2.strPSource = tbFSource2.Lines;

Lex2.strPMessage = tbFMessage1.Lines;

Lex2.intPSourceColSelection = 0;

Lex2.intPSourceRowSelection = 0;

int x2 = tbFSource2.TextLength;

tbFMessage1.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

switch (Lex.enumPToken) //Проверка 1 - го слова на то, является ли оно номером или идентификатором

{

case TToken.lxmIdentifier: //Если идентификатор то:

{

int b = 0;

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

{

while (Lex2.enumPState != TState.Finish)

{

Lex2.NextToken();

switch (Lex2.enumPToken) //Проверка 1 - го слова на то, является ли оно номером или идентификатором

{

case TToken.lxmIdentifier: //Если является идентификатором, то:

{

int b1 = 0;

if (htl.AddLexicalUnit(Lex2.strPLexicalUnit, 0, ref b1)) //вызывается функция добавления в таблицу идентификаторов 2 - го слова

{

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 0); //Удаляется 1 - ое слово

TablesToMemo(this, e);

}

tbFMessage1.AppendText("(Идентификатор был изменен)");

break;

}

case TToken.lxmNumber: //если 2 - ое слово - номер

{

tbFMessage1.AppendText("(Нельзя заменить идентификатор на номер)");

break;

}

}

}

}

else tbFMessage1.AppendText("(Идентификатор для удаления не найден)");

break;

}

case TToken.lxmNumber: //если 1 - ое слово номер

{

int b = 0;

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

{

while (Lex2.enumPState != TState.Finish)

{

Lex2.NextToken();

switch (Lex2.enumPToken) //проверка 2 - го слова

{

case TToken.lxmIdentifier://если 2 - ое слово идентификатор

{

tbFMessage1.AppendText("(Нельзя заменить номер на идентификатор)");

break;

}

case TToken.lxmNumber: // если 2 - ое слово номер

{

int b1 = 0;

if (htl.AddLexicalUnit(Lex2.strPLexicalUnit, 1, ref b1))//1 - ое слово добавляется в таблицу номеров

{

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 1);// 2 - ое слово удаляется

TablesToMemo(this, e);

}

tbFMessage1.AppendText("(Номер был изменен)");

break;

}

}

}

}

else tbFMessage1.AppendText("(Номер для удаления не найден)");

break;

}

}

}

}

catch (Exception exc)

{

tbFMessage1.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n1 = 0;

for (int i = 0; i < Lex.intPSourceRowSelection; i++) n1 += tbFSource.Lines[i].Length + 2;

n1 += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n1;

}

}

// Кнопка добавления слов в таблицы

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

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

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

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 0, ref b)) //Вызывается функция добавления слова в таблицу идентификаторов(0)

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

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

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 1, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmLeftParenth:

{

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

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmRightParenth:

{

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

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmr:

{

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

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)

{

TablesToMemo(this, e);

}

break;

}

case TToken.mn:

{

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

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)

{

TablesToMemo(this, e);

}

break;

}

case TToken.plus:

{

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

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 2, ref b)) //Вызывается функция добавления слова в таблицу номеров(1)

{

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;

}

}

//Кнопка поиска слова в таблицах

private void button3\_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();

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

int b = 0;

if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, 0, ref b)) //Вызов функции поиска, возвращает значение true если было найден такой идентификатор, иначе false

{

tbFMessage2.AppendText("(Найден идентификатор)");

}

else tbFMessage2.AppendText("(Идентификатор не найден)");

break;

}

case TToken.lxmNumber:

{

int b = 0;

if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, 1, ref b)) //Вызов функции поиска, возвращает значение true если было найден такой номер, иначе false

{

tbFMessage2.AppendText("(Найден номер)");

}

else tbFMessage2.AppendText("(Номер не найден)");

break;

}

}

}

}

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 button6\_Click(object sender, EventArgs e)

{

TreeConverter treeConverter = new TreeConverter();

treeConverter.ConvertBinaryToDecimal(treeView1);

}

**TreeConventer.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace тяп\_лаба\_3

{

internal class TreeConverter

{

public void ConvertBinaryToDecimal(TreeView tree)

{

foreach (TreeNode node in tree.Nodes)

{

ConvertBinaryToDecimal(node);

}

}

private void ConvertBinaryToDecimal(TreeNode node)

{

if (node.Text == "O")

{

if (node.Nodes.Count == 3)

{

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;

}

}

}

**THashTable.cs:**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace nsSynt

{

public class THashTable

{

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

private int intFCurrentPrimeNumber;

private int intFItemReserve;

private bool boolIsSaved;

public int intFHashIndex;

public int cardPTableSize { get { return arrFHashTable.Count; } }

public List<object> arrFUserTable = null;

static THeap objFHeap;

public THashTable(ref THeap objAHeap)

{

objFHeap = objAHeap;

Init(7);

intFItemReserve = 0;

}

public void Init(int count)

{

arrFHashTable.Clear();

Resize(arrFHashTable, count);

intFCurrentPrimeNumber = count;

}

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

{

if (size > list.Count)

while (size > list.Count)

list.Add(new object());

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

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

{

if (size > list.Count)

while (size > list.Count)

list.Add(new Int32());

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

int NextPrimeNumber(int cardAOldPrimeNumber)

{

int intVLowerBound, intVUpperBound, intVNextPrimeNumber;

bool boolVIsDivisor;

intVNextPrimeNumber = cardAOldPrimeNumber + cardAOldPrimeNumber / 10 + 1; // увеличиваем на 10 процентов

if ((intVNextPrimeNumber % 2) == 0) intVNextPrimeNumber++;

do

{

boolVIsDivisor = true; intVNextPrimeNumber = intVNextPrimeNumber + 2;

intVLowerBound = 3; intVUpperBound = intVNextPrimeNumber / 3 + 1; // диапазон делителей

while (boolVIsDivisor && (intVLowerBound < intVUpperBound))

{

if ((intVNextPrimeNumber % intVLowerBound) == 0) boolVIsDivisor = false;

else intVLowerBound = intVLowerBound + 2;

}

} while (!boolVIsDivisor);

return intVNextPrimeNumber;

}

//HashFunc

UInt32 HashFunction\_Wainberger(string strALexicalUnit)

{

UInt32 h = 0/\*, seed=131313\*/;

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

{

h += Convert.ToUInt32(strALexicalUnit[i] % 12); //Хэширование

}

return h;

}

int ReHashFunction\_Line(int h, string strALexicalUnit)

{

if (h == 0) h = arrFHashTable.Count / 3;

else if (h == 1) h = arrFHashTable.Count \* 3 / 4;

int i = 1, hi = h;

bool boolVFinish = false;

do

{

if (arrFHashTable[hi] == 0) boolVFinish = true;

else

if (objFHeap.arrFHeapTable[arrFHashTable[hi]].strFLexicalUnit == strALexicalUnit)

boolVFinish = true;

else

{

i++;

hi = (h % i) % (Int32)(arrFHashTable.Count); //Рехэширование(это метод для избавления от коллизии)

}

} while (!boolVFinish);

return hi;

}

public void HashIndex(string strALexicalUnit)

{

int h;

h = (Int32)HashFunction\_Wainberger(strALexicalUnit) % (Int32)(arrFHashTable.Count);

intFHashIndex = ReHashFunction\_Line(h, strALexicalUnit);

}

void TableReHashing()

{

int i, j;

List<int> cardarrVHashTableImage = new List<int>();

List<object> arrVUserTableImage = new List<object>();

Resize(cardarrVHashTableImage, arrFHashTable.Count);

if (arrFUserTable != null)

Resize(arrVUserTableImage, arrFHashTable.Count);

for (i = 0; i < arrFHashTable.Count; i++)

{

cardarrVHashTableImage[i] = arrFHashTable[i];

if (arrFUserTable != null) arrVUserTableImage[i] = arrFUserTable[i];

}

arrFHashTable.Clear();

if (arrFUserTable != null) arrFUserTable.Clear();

Resize(arrFHashTable, intFCurrentPrimeNumber);

if (arrFUserTable != null) Resize(arrFUserTable, intFCurrentPrimeNumber);

for (i = 0; i < cardarrVHashTableImage.Count; i++)

{

if (cardarrVHashTableImage[i] != 0)

{

j = cardarrVHashTableImage[i];

HashIndex(objFHeap.arrFHeapTable[j].strFLexicalUnit);

arrFHashTable[intFHashIndex] = j;

if (arrFUserTable != null) arrFUserTable[intFHashIndex] = arrVUserTableImage[i];

THeapItem Th2 = objFHeap.arrFHeapTable[j];

Th2.intFHashIndex = intFHashIndex;

objFHeap.arrFHeapTable[j] = Th2;

}

}

cardarrVHashTableImage.Clear();

if (arrFUserTable != null) arrVUserTableImage.Clear();

}

void Expansion()

{

intFCurrentPrimeNumber = NextPrimeNumber(intFCurrentPrimeNumber);

TableReHashing();

}

object GetUserPointer(int cardILexicalCode)

{

THeapItem Item = objFHeap.arrFHeapTable[cardILexicalCode];

if (Item.intFHashIndex >= cardPTableSize)

{

MessageBox.Show("Индекс пользовательского массива вышел за диапазон!");

return null;

}

else

{

return arrFUserTable[objFHeap.arrFHeapTable[cardILexicalCode].intFHashIndex];

}

}

void SetUserPointer(int cardILexicalCode, object ptrANewPoint)

{

if (objFHeap.arrFHeapTable[cardILexicalCode].intFHashIndex >= cardPTableSize)

MessageBox.Show("Индекс пользовательского массива вышел за диапазон!");

else

arrFUserTable[objFHeap.arrFHeapTable[cardILexicalCode].intFHashIndex] = ptrANewPoint;

}

public void SetUserTable()

{

arrFUserTable = new List<object>();

Resize(arrFUserTable, arrFHashTable.Count);

}

public bool SearchLexicalUnit(string strAlexicalUnit, ref int intALexicalCode) //Поиск

{

HashIndex(strAlexicalUnit);

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

else

{

intALexicalCode = arrFHashTable[intFHashIndex];

return true;

}

}

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

{

HashIndex(strALexicalUnit);

if (arrFHashTable[intFHashIndex] != 0)

{

intALexicalCode = arrFHashTable[intFHashIndex];

return true;

}

else

{

if ((intFItemReserve + 2) > (cardPTableSize \* 0.9))

{

Expansion();

HashIndex(strALexicalUnit);

}

objFHeap.AddLexicalUnit(strALexicalUnit, byteAHashTable, intFHashIndex, ref intALexicalCode);

arrFHashTable[intFHashIndex] = intALexicalCode;

intFItemReserve++;

return false;

}

}

public void DeleteLexicalUnit(string strAlexicalUnit)

{

HashIndex(strAlexicalUnit);

if (arrFHashTable[intFHashIndex] != 0)

{

if (arrFUserTable != null)

{

if (arrFUserTable[intFHashIndex] != null)

MessageBox.Show("Удаление из таблицы связанного данного");

else

{

objFHeap.DeleteLexicalUnit(arrFHashTable[intFHashIndex]);

arrFHashTable[intFHashIndex] = 0;

intFItemReserve--;

TableReHashing();

}

}

else

{

objFHeap.DeleteLexicalUnit(arrFHashTable[intFHashIndex]);

arrFHashTable[intFHashIndex] = 0;

intFItemReserve--;

TableReHashing();

}

}

}

public void DeleteLexicalCode(int cardALexicalCode)

{

int VHashIndex;

VHashIndex = objFHeap.arrFHeapTable[cardALexicalCode].intFHashIndex;

if (arrFHashTable[VHashIndex] != 0)

if (arrFUserTable.Count != 0)

if (arrFUserTable[VHashIndex] != null)

MessageBox.Show("Удаление из таблицы связанного данного");

else

{

objFHeap.DeleteLexicalUnit(cardALexicalCode);

arrFHashTable[VHashIndex] = 0;

intFItemReserve--;

TableReHashing();

}

else

{

objFHeap.DeleteLexicalUnit(cardALexicalCode);

arrFHashTable[VHashIndex] = 0;

intFItemReserve--;

TableReHashing();

}

}

public void Save(ref StreamWriter fl)

{

try

{

fl.WriteLine(cardPTableSize.ToString());

fl.WriteLine(intFItemReserve.ToString());

for (int i = 1; i < cardPTableSize; i++)

fl.Write("\t" + arrFHashTable[i].ToString());

fl.Write("\n");

boolIsSaved = true;

}

catch (InvalidCastException)

{ boolIsSaved = false; }

}

public void GetLexicalUnitList(ref List<string> sList)

{

for (int i = 0; i < arrFHashTable.Count; i++) if (arrFHashTable[i] != 0) sList.Add(objFHeap.arrFHeapTable[arrFHashTable[i]].strFLexicalUnit);

}

}

//HashTableList

public class CHashTableList

{

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

private bool boolFIsSaved;

public bool boolFIsLoaded;

private byte byteFTablesSize;

static THeap objFHeap = new THeap();

//------------------------------------------------------------------------------

public CHashTableList(byte byteATableCount)

{

this.byteFTablesSize = byteATableCount;

objFHeap = new THeap();

Resize(arrFHashTableList, byteATableCount);

}

//------------------------------------------------------------------------------

public byte GetTableNumber(int intALexicalCode)

{

return objFHeap.arrFHeapTable[intALexicalCode].byteFHashTable;

}

//------------------------------------------------------------------------------

public int GetTablesCount()

{

return arrFHashTableList.Count();

}

//------------------------------------------------------------------------------

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

{

if (size > list.Count)

while (size > list.Count)

{

list.Add(new THashTable(ref objFHeap));

}

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

//------------------------------------------------------------------------------

static void Resize(List<object> list, int intANewSize)

{

if (intANewSize > list.Count)

while (intANewSize > list.Count)

list.Add(new object());

else if (intANewSize < list.Count)

while (list.Count - intANewSize > 0)

list.RemoveAt(list.Count - 1);

}

//------------------------------------------------------------------------------

static void Resize(List<int> list, int intANewSize)

{

if (intANewSize > list.Count)

while (intANewSize > list.Count)

list.Add(new Int32());

else if (intANewSize < list.Count)

while (list.Count - intANewSize > 0)

list.RemoveAt(list.Count - 1);

}

//------------------------------------------------------------------------------

public object GetUserData(int intALexicalCode)

{

if ((0 < intALexicalCode) && (intALexicalCode < objFHeap.intPFreeItem))

return arrFHashTableList[GetTableNumber(intALexicalCode)].arrFUserTable[intALexicalCode];

else

{

MessageBox.Show("Неверно задан лексический код при чтении пользовательских данных");

return null;

}

}

//------------------------------------------------------------------------------

public void SetUserData(int intALexicalCode, object objAUserData)

{

if ((0 < intALexicalCode) && (intALexicalCode < objFHeap.intPFreeItem))

{

if (arrFHashTableList[GetTableNumber(intALexicalCode)].arrFUserTable.Count > 0)

arrFHashTableList[GetTableNumber(intALexicalCode)].arrFUserTable[intALexicalCode] = objAUserData;

else

MessageBox.Show("Попытка записи адреса в несозданный массив пользовательских данных!");

}

else MessageBox.Show("Неверно задан лексический код при записи пользовательских данных!");

}

//------------------------------------------------------------------------------

public string GetLexicalUnit(int intALexicalCode)

{

if ((0 < intALexicalCode) && (intALexicalCode < objFHeap.intPFreeItem)) return objFHeap.arrFHeapTable[intALexicalCode].strFLexicalUnit;

else

{

MessageBox.Show("Неверно задан лексический код при чтении пользовательских данных!");

return "";

}

}

//------------------------------------------------------------------------------

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

{

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

}

//------------------------------------------------------------------------------

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

{

if (byteATable >= arrFHashTableList.Count)

{

if (MessageBox.Show("Увеличить количество таблиц?", "Запрашиваемый индекс таблицы не существует.", MessageBoxButtons.YesNo) == DialogResult.Yes)

Resize(arrFHashTableList, byteATable + 1);

else

return false;

}

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

}

//------------------------------------------------------------------------------

public void DeleteLexicalUnit(string strALexicalUnit, byte byteATable)

{

arrFHashTableList[byteATable].DeleteLexicalUnit(strALexicalUnit);

}

//------------------------------------------------------------------------------

public void DeleteLexicalCode(int intALexicalCode)

{

short T = objFHeap.arrFHeapTable[intALexicalCode].byteFHashTable;

arrFHashTableList[T].DeleteLexicalCode(intALexicalCode);

}

//------------------------------------------------------------------------------

public void SetUserTable(byte byteATable)

{

arrFHashTableList[byteATable].SetUserTable();

}

//------------------------------------------------------------------------------

public void Expantion()

{

Resize(arrFHashTableList, ++byteFTablesSize);

}

//------------------------------------------------------------------------------

public void Save(string strAFileName)

{

try

{

StreamWriter fl = File.CreateText(strAFileName);

fl.WriteLine(byteFTablesSize.ToString());

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

fl.Write(arrFHashTableList[i].arrFHashTable.Count.ToString() + "\t");

fl.WriteLine("");

objFHeap.Save(ref fl);

boolFIsSaved = true;

fl.Close();

}

catch (InvalidDataException)

{ boolFIsSaved = false; }

}

//------------------------------------------------------------------------------

// отладка

public void HeapTableView(List<string> sList)

{

objFHeap.HeapTableView(sList);

}

//------------------------------------------------------------------------------

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

{

arrFHashTableList[byteATable].GetLexicalUnitList(ref sList);

}

//------------------------------------------------------------------------------

public int GetHashIndex(byte Table)

{

return arrFHashTableList[Table].intFHashIndex;

}

//------------------------------------------------------------------------------

}

public struct THeapItem

{

public string strFLexicalUnit;

public byte byteFHashTable;

public int intFHashIndex;

public THeapItem(string strALexicalUnit, byte byteATable, int intAHashIndex)

{

strFLexicalUnit = strALexicalUnit;

byteFHashTable = byteATable;

intFHashIndex = intAHashIndex;

}

}

public class THeap

{

public List<THeapItem> arrFHeapTable = new List<THeapItem>();

private List<int> arrFDeleted = new List<int>();

private int intFFreeItem;

bool boolIsSaved;

bool boolIsLoaded;

public bool boolPIsSaved { get { return boolIsSaved; } }

public bool boolPIsLoaded { get { return boolIsLoaded; } }

public int intPFreeItem { get { return intFFreeItem; } }

public THeap()

{

Init();

intFFreeItem = 1;

}

protected void Init()

{

arrFDeleted.Clear();

arrFHeapTable.Clear();

int cnt = 4;

Resize(arrFHeapTable, cnt);

}

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

{

if (size > list.Count)

while (size > list.Count)

list.Add(new THeapItem("", 0, 0));

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

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

{

if (size > list.Count)

while (size > list.Count)

list.Add(new Int32());

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

/\* static void Resize(List<char> list, int size)

{

if (size > list.Count)

while (size > list.Count)

list.Add('0');

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

\*/

public void Expansion()

{

int cardVSize = arrFHeapTable.Count;

cardVSize = cardVSize + cardVSize % 10 + 1;

Resize(arrFHeapTable, cardVSize);

Resize(arrFHeapTable, cardVSize);

}

public void AddLexicalUnit(string strALexicalUnit, byte byteAHashTable, int cardAHashIndex, ref int cardALexicalCode)

{

int intVIndex;

if (arrFDeleted.Count == 0)

{

intVIndex = intFFreeItem;

intFFreeItem++;

if (intFFreeItem >= (Int32)(arrFHeapTable.Count \* 0.9))

Expansion();

}

else

{

intVIndex = arrFDeleted[arrFDeleted.Count - 1];

Resize(arrFDeleted, arrFDeleted.Count - 1);

}

THeapItem Item = arrFHeapTable[intVIndex];

Item.strFLexicalUnit = strALexicalUnit;

Item.byteFHashTable = byteAHashTable;

Item.intFHashIndex = cardAHashIndex;

arrFHeapTable[intVIndex] = Item;

cardALexicalCode = intVIndex;

}

public void DeleteLexicalUnit(int cardALexicalCode)

{

int i;

if (arrFDeleted == null || !arrFDeleted.Any())

i = 0;

else i = arrFDeleted.Count();

Resize(arrFDeleted, i + 1);

arrFDeleted[i] = cardALexicalCode;

THeapItem Item = arrFHeapTable[cardALexicalCode];

Item.strFLexicalUnit = "";

Item.byteFHashTable = 0;

Item.intFHashIndex = 0;

}

public void Save(ref StreamWriter sw)

{

try

{

for (int i = 1; i < arrFHeapTable.Count; i++) //type?

{

if (arrFHeapTable[i].strFLexicalUnit == "")

break;

sw.Write(arrFHeapTable[i].strFLexicalUnit + "\t");

sw.Write(arrFHeapTable[i].byteFHashTable.ToString() + "\t");

sw.WriteLine(arrFHeapTable[i].intFHashIndex.ToString());

}

boolIsSaved = true;

}

catch (Exception) { boolIsSaved = false; }

}

public void Load(ref StreamReader sr)

{

try

{

Init();

int size = arrFHeapTable.Count;

int readSz = 0;

while (true)

{

string line = sr.ReadLine();

if (line == null)

break;

if (++readSz >= size)

{

size \*= 2;

Resize(arrFHeapTable, size);

}

char[] delim = { '\t'/\*,'\n'\*/ };

string[] lines = line.Split(delim);

THeapItem it = arrFHeapTable[readSz];

it.strFLexicalUnit = lines[0];

it.byteFHashTable = Convert.ToByte(lines[1]);

it.intFHashIndex = Convert.ToInt32(lines[2]);

arrFHeapTable[readSz] = it;

}

intFFreeItem = readSz + 1;

boolIsLoaded = true;

}

catch (InvalidCastException)

{ boolIsLoaded = false; }

}

THeapItem GetItem(int i)

{

if (i >= arrFHeapTable.Count)

{

MessageBox.Show("GetИндекс кучи вышел за диапазон!");

THeapItem Item = new THeapItem("", 0, 0);

return Item;

}

else return arrFHeapTable[i];

}

void SetItem(int i, THeapItem NewItem)

{

if (i >= arrFHeapTable.Count)

MessageBox.Show("SetИндекс кучи вышел за диапазон!");

else arrFHeapTable[i] = NewItem;

}

public void HeapTableView(List<string> sList)

{

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

sList.Add(arrFHeapTable[i].strFLexicalUnit);

}

}

}

**uLex.cs:**

**using System;**

**using System.Collections.Generic;**

**using System.Text;**

**namespace nsLex**

**{**

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

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

**public enum TToken**

**{**

**mn, minus, plus, lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty,**

**lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma, lxmdt, lxmText, lxmtz, lxmr, lxmrs, lxmls**

**};**

**public class CLex //класс лексический анализатор**

**{**

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

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

**public TCharType enumFSelectionCharType;**

**public char chrFSelection;**

**private TState enumFState;**

**private int intFSourceRowSelection;**

**private int intFSourceColSelection;**

**private String strFLexicalUnit;**

**private TToken enumFToken;**

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

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

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

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

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

**public int intPSourceRowSelection**

**{**

**get { return intFSourceRowSelection; }**

**set**

**{**

**intFSourceRowSelection = value;**

**}**

**}**

**public int intPSourceColSelection**

**{**

**get { return intFSourceColSelection; }**

**set**

**{**

**intFSourceColSelection = value;**

**}**

**}**

**public CLex()**

**{**

**}**

**public void GetSymbol() //метод класса лексический анализатор**

**{**

**intFSourceColSelection++; // продвигаем номер колонки**

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

**{**

**intFSourceRowSelection++;**

**if (intFSourceRowSelection <= strFSource.Length - 1)**

**{**

**intFSourceColSelection = -1;**

**chrFSelection = '\0';**

**enumFSelectionCharType = TCharType.EndRow;**

**enumFState = TState.Continue;**

**}**

**else**

**{**

**chrFSelection = '\0';**

**enumFSelectionCharType = TCharType.EndText;**

**enumFState = TState.Finish;**

**}**

**}**

**else**

**{**

**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 == '\*' || chrFSelection == '-') enumFSelectionCharType =TCharType.ReservedSymbol;**

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

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

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

**else if (chrFSelection == ';' || chrFSelection == ',' || chrFSelection == '[' || chrFSelection == ']' || chrFSelection == '=' || chrFSelection == ':') enumFSelectionCharType = TCharType.ReservedSymbol;**

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

**enumFState = TState.Continue;**

**}**

**}**

**private void TakeSymbol()**

**{**

**char[] c = { chrFSelection };**

**String s = new string(c);**

**strFLexicalUnit += s;**

**GetSymbol();**

**}**

**public void NextToken()**

**{**

**strFLexicalUnit = "";**

**if (enumFState == TState.Start)**

**{**

**intFSourceRowSelection = 0;**

**intFSourceColSelection = -1;**

**GetSymbol();**

**}**

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

**{**

**GetSymbol();**

**}**

**if (chrFSelection == '/')**

**{**

**GetSymbol();**

**if (chrFSelection == '/')**

**while (enumFSelectionCharType != TCharType.EndRow)**

**{**

**GetSymbol();**

**}**

**GetSymbol();**

**}**

**switch (enumFSelectionCharType)**

**{**

**case TCharType.Letter: //если начинается слово из букв a,b,c,d**

**{**

**// d c b a**

**// AF | AF | BF | CF | DF |**

**// BF | | BF | CF | DF |**

**// CF | | | CF | DF |**

**// DF | | | | DF |**

**AF:**

**{**

**if (chrFSelection == 'd')**

**{**

**TakeSymbol();**

**goto AF;**

**}**

**if (chrFSelection == 'c')**

**{**

**TakeSymbol();**

**goto BF;**

**}**

**if (chrFSelection == 'b')**

**{**

**TakeSymbol();**

**goto CF;**

**}**

**if (chrFSelection == 'a')**

**{**

**TakeSymbol();**

**goto DF;**

**}**

**else**

**{**

**enumFToken = TToken.lxmIdentifier;**

**return;**

**}**

**}**

**BF:**

**{**

**if (chrFSelection == 'd')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'c')**

**{**

**TakeSymbol();**

**goto BF;**

**}**

**if (chrFSelection == 'b')**

**{**

**TakeSymbol();**

**goto CF;**

**}**

**if (chrFSelection == 'a')**

**{**

**TakeSymbol();**

**goto DF;**

**}**

**else**

**{**

**enumFToken = TToken.lxmIdentifier;**

**return;**

**}**

**}**

**CF:**

**{**

**if (chrFSelection == 'd')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'c')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'b')**

**{**

**TakeSymbol();**

**goto CF;**

**}**

**if (chrFSelection == 'a')**

**{**

**TakeSymbol();**

**goto DF;**

**}**

**else**

**{**

**enumFToken = TToken.lxmIdentifier;**

**return;**

**}**

**}**

**DF:**

**{**

**if (chrFSelection == 'd')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'c')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'b')**

**{**

**throw new Exception("Слово должно быть с буквами в обратном алфавитном порядке");**

**}**

**if (chrFSelection == 'a')**

**{**

**TakeSymbol();**

**goto DF;**

**}**

**else**

**{**

**enumFToken = TToken.lxmIdentifier;**

**return;**

**}**

**}**

**}**

**if (chrFSelection == '/')**

**{**

**GetSymbol();**

**if (chrFSelection == '/')**

**while (enumFSelectionCharType != TCharType.EndRow)**

**{**

**GetSymbol();**

**}**

**GetSymbol();**

**}**

**case TCharType.Digit: //если начинается слово из букв – цифр 0 и 1**

**{**

**// 0 1**

**// A | B | C |**

**// B | | D |**

**// C | | E |**

**// D | A | |**

**// E | Ffin | |**

**// Ffin | | G |**

**// G | | H |**

**// H | | Ffin |**

**A:**

**if (chrFSelection == '0')**

**{**

**TakeSymbol();**

**goto B;**

**}**

**else if (chrFSelection == '1')**

**{**

**TakeSymbol();**

**goto C;**

**}**

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

**B:**

**if (chrFSelection == '1')**

**{**

**TakeSymbol();**

**goto D;**

**}**

**else throw new Exception("Ожидался 1");**

**C:**

**if (chrFSelection == '1')**

**{**

**TakeSymbol();**

**goto E;**

**}**

**else throw new Exception("Ожидался 1");**

**D:**

**if (chrFSelection == '0')**

**{**

**TakeSymbol();**

**goto A;**

**}**

**else throw new Exception("Ожидался 0");**

**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("Ожидался 1");**

**G:**

**if (chrFSelection == '1')**

**{**

**TakeSymbol();**

**goto H;**

**}**

**else throw new Exception("Ожидался 1");**

**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 == '(')**

**{**

**strFLexicalUnit = "(";**

**enumFToken = TToken.lxmLeftParenth;**

**GetSymbol();**

**return;**

**}**

**if (chrFSelection == '+')**

**{**

**strFLexicalUnit = "+";**

**enumFToken = TToken.plus;**

**GetSymbol();**

**return;**

**}**

**if (chrFSelection == '\*')**

**{**

**strFLexicalUnit = "\*";**

**enumFToken = TToken.mn;**

**GetSymbol();**

**return;**

**}**

**if (chrFSelection == '-')**

**{**

**enumFToken = TToken.minus;**

**GetSymbol();**

**return;**

**}**

**if (chrFSelection == ')')**

**{**

**strFLexicalUnit = ")";**

**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 == '=')**

**{**

**strFLexicalUnit = "=";**

**enumFToken = TToken.lxmr;**

**GetSymbol();**

**return;**

**}**

**break;**

**}**

**case TCharType.EndText:**

**{**

**enumFToken = TToken.lxmEmpty;**

**break;**

**}**

**}**

**}**

**}**

**}**

**uSyntAnalyzer.cs:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Windows.Forms;

using nsLex;

using тяп\_лаба\_3;

namespace nsSynt

{

class uSyntAnalyzer

{

private String[] strFSource;

private String[] strFMessage;

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

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

public CLex Lex = new CLex();

public TToken firstToken;

/\*public List<string> numTokenList = new List<string>();

public bool noDuplications = true; \*/

public TreeView tree;

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

{

if (token == TToken.lxmNumber)

{

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

{

if (input == numTokenList[i])

{

noDuplications = false;

}

}

numTokenList.Add(input);

}

parent.Nodes.Add(new TreeNode(input)); \*/

public void D(TreeNode parent)

{

TreeNode node = new TreeNode("D");

parent.Nodes.Add(node);

//string s = Convert.ToString(Lex.enumPToken);

//MessageBox.Show(s);

//Lex.NextToken();

K(node);

// Lex.NextToken();

if (Lex.enumPToken == TToken.plus)

{

node.Nodes.Add("+");

C(node);

}

}

public void C(TreeNode parent)

{

TreeNode node = new TreeNode("C");

parent.Nodes.Add(node);

if (Lex.enumPToken == TToken.plus)

{

//node.Nodes.Add("+");

Lex.NextToken();

K(node);

if (Lex.enumPToken == TToken.plus)

{

node.Nodes.Add("+");

C(node);

}

}

//else throw new Exception("Ожидалcя +");

}

public void K(TreeNode parent)

{

TreeNode node = new TreeNode("K");

parent.Nodes.Add(node);

A(node);

//Lex.NextToken();

if (Lex.enumPToken == TToken.mn)

{

//node.Nodes.Add("=");

B(node);

}

}

public void B(TreeNode parent)

{

TreeNode node = new TreeNode("B");

parent.Nodes.Add(node);

if (Lex.enumPToken == TToken.mn)

{

node.Nodes.Add("\*");

Lex.NextToken();

A(node);

//Lex.NextToken();

if (Lex.enumPToken == TToken.mn)

{

node.Nodes.Add("\*");

B(node);

}

}

}

public void A(TreeNode parent)

{

TreeNode node = new TreeNode("A");

parent.Nodes.Add(node);

if (Lex.enumPToken == TToken.minus)

{

node.Nodes.Add("-");

Lex.NextToken();

A(node);

}

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

{

node.Nodes.Add("(");

Lex.NextToken();

D(node);

//Lex.NextToken();

if (Lex.enumPToken == TToken.lxmRightParenth)

{

node.Nodes.Add(")");

Lex.NextToken();

// throw new Exception("Konec slova");

}

else throw new Exception("Ожидалась )");

}

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

{

//node.Nodes.Add(Lex.strPLexicalUnit);

O(node);

}

else

{

string s = Convert.ToString(Lex.enumPToken);

MessageBox.Show(s);

throw new Exception("Ожидались - или ( или число");

}

}

public void O(TreeNode parent)

{

TreeNode node = new TreeNode("O");

parent.Nodes.Add(node);

if (Lex.enumPToken == TToken.lxmNumber)

{

node.Nodes.Add(Lex.strPLexicalUnit);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmr)

{

node.Nodes.Add("=");

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier)

{

node.Nodes.Add(Lex.strPLexicalUnit);

//throw new Exception("Konec slova");

}

else throw new Exception("ожидалось число");

}

else throw new Exception("ожидалось =");

}

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

{

Duplicates(tree, Lex.strPLexicalUnit);

node.Nodes.Add(Lex.strPLexicalUnit);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmr)

{

node.Nodes.Add("=");

Lex.NextToken();

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

{

if (Lex.enumPToken == TToken.lxmIdentifier)

{

Duplicates(tree, Lex.strPLexicalUnit);

}

node.Nodes.Add(Lex.strPLexicalUnit);

//throw new Exception("Konec slova");

}

else throw new Exception("ожидалось число или слово");

}

else throw new Exception("ожидалось =");

}

else throw new Exception("Ожидалось число или слово");

Lex.NextToken();

}

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 == "O") // Если попали в ветку под названием O, там 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 O, тем самым рассматривая каждый лексический юнит

{

if (node2.Text == s)

{

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

}

}

}

}

foreach (TreeNode childNode in node.Nodes)

{

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

}

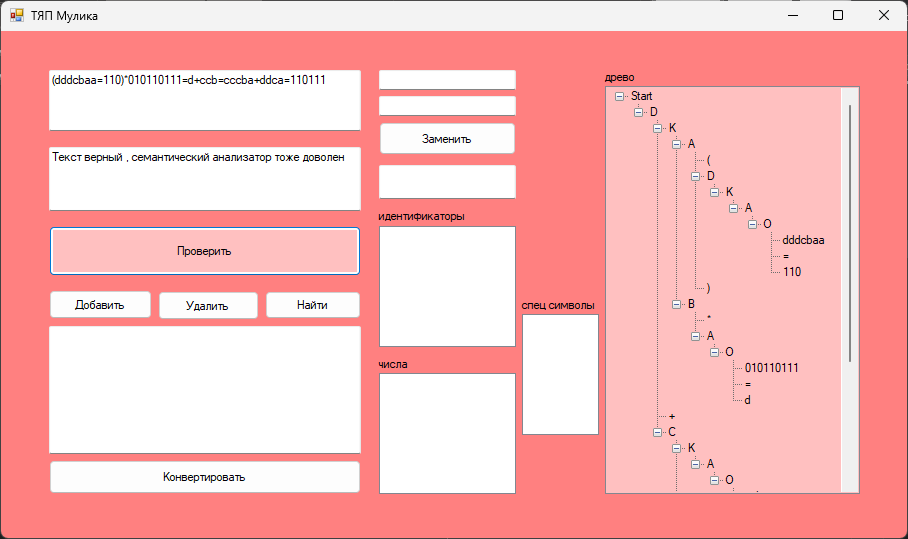
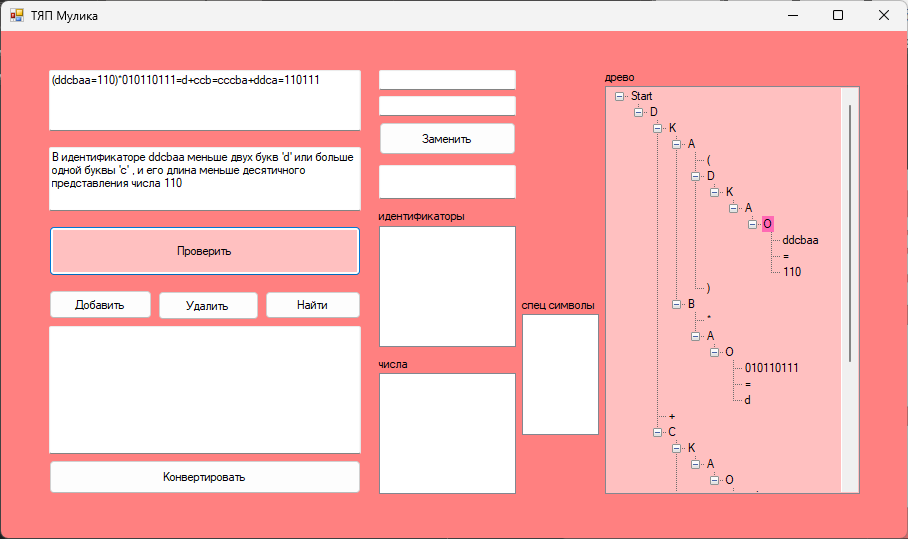
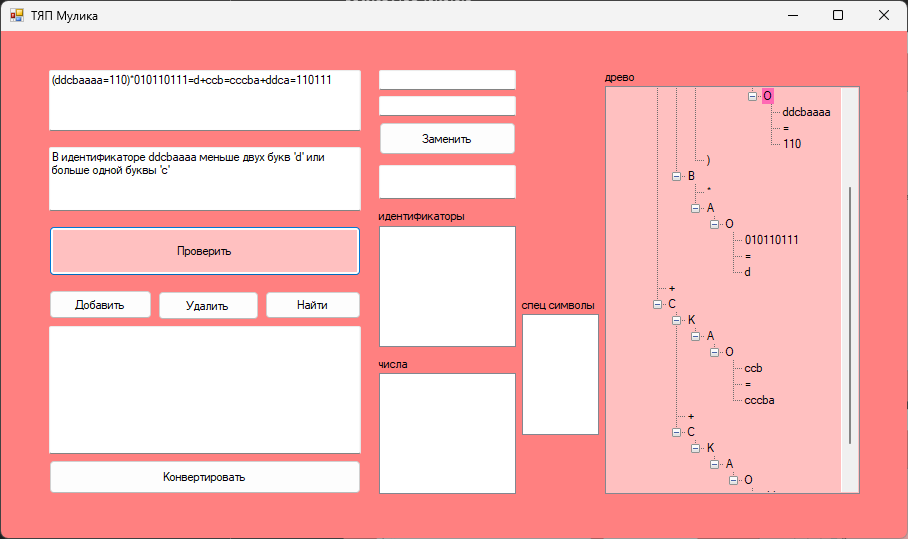
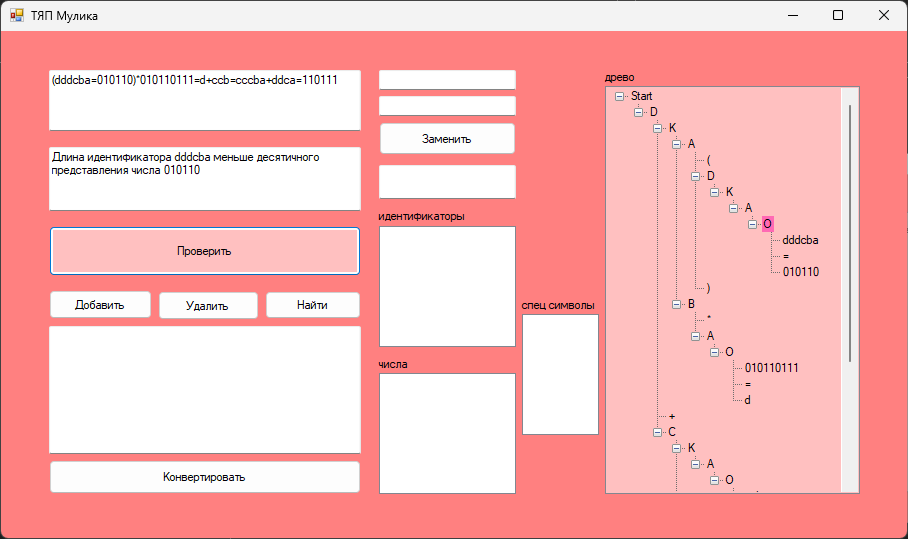
}

}

}

**Результат:**

**1 вариант**

**2 вариант:**

