Министерство образования Российской Федерации

Федеральное агентство по образованию

КАЗАНСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. А.Н. Туполева - КАИ

Кафедра АСОИУ

Лабораторные работы №1-8

по дисциплине

«ТЕОРИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ» и «Методы трансляции»

Выполнил:

Студент группы 4309

Братилов Е.В.

Проверила: Бикмуллина И. И.

Казань 2023

**Лабораторная работа № 1. Разработка транслитератора**

**Текст задания:**

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

Form1.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

namespace WindowsFormsApp57

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

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

tbFSource.AppendText("1 a");

int n = tbFSource.Lines.Length;

}

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

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

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

Lex.NextToken();

String s = "";

String s1 = "";

switch (Lex.enumFSelectionCharType)

{

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

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

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

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

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

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

}

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

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

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

}

}

**Class1**

using System;

using System.Collections.Generic;

using System.Text;

namespace WindowsFormsApp57

{

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

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

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

public class CLex //класс лексический анализатор

{

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

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

public TCharType enumFSelectionCharType;

public char chrFSelection;

private TState enumFState;

private int intFSourceRowSelection;

private int intFSourceColSelection;

private String strFLexicalUnit;

private TToken enumFToken;

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

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

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

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

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

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

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

public CLex()

{

}

public void GetSymbol() //метод класса лексический анализатор

{

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

{

intFSourceRowSelection++;

if (intFSourceRowSelection <= strFSource.Length - 1)

{

intFSourceColSelection = -1;

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndRow;

enumFState = TState.Continue;

}

else

{

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndText;

enumFState = TState.Finish;

}

}

else

{

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

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

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

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

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

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

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

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

enumFState = TState.Continue;

}

intFSourceColSelection++; // продвигаем номер колонки

}

public void NextToken()

{

strFLexicalUnit = "";

if (enumFState == TState.Start)

{

intFSourceRowSelection = 0;

intFSourceColSelection = -1;

GetSymbol();

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

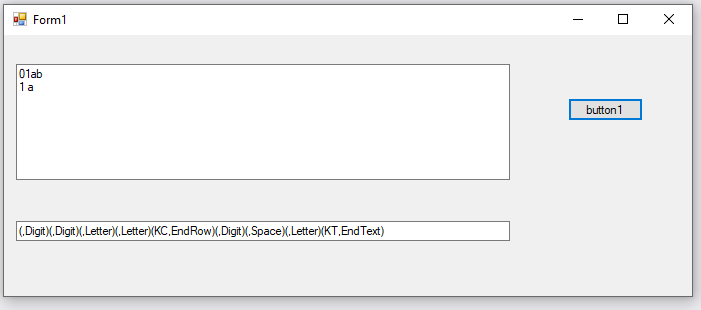
GetSymbol();

}

}

}

}



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

**Текст задания:**

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

|  |  |  |
| --- | --- | --- |
| (000)\*001(010)\* | (a|b|c|d)+ | В алфавитном порядке |

**Первое слово:**

0

(000)\*001(010)\*

0

A → 0B | 0D

0

0

B → 0C

1

C → 0A

0

D → 0E

1

E → 1FFin

00

0

FFin → 0G

G → 1H

00

H → 0FFin

10

**Недетерминированная матрица:**

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

**Граф:**

**Детерминированная матрица:**

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| A | BD |  |
| BD | CE |  |
| CE | A | Ffin |
| Ffin | G |  |
| E | Fin |  |
| G |  | H |
| H | Fin |  |

**Второе слово:**

b



b,c,d

a,b,c,d

(a|b|c|d)+

В алфавитном порядке

AFin → aAFin | bBFin | cCFin | dDFin | a | b | c | d

BFin → bBFin | cCFin | dDFin | b | c | d

CFin → cCFin | dDFin | c | d

DFin → dDFin | d

**Недетерминированная матрица:**

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

**Детерминированная матрица:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | a | b | c | d |
| ABCDFin | a,ABCDFin | b,ABCDFin | c,ABCDFin | d,ABCDFin |

c

c

d

c,d

d



d

d

Код программы:

using System;

using System.Collections.Generic;

using System.Text;

namespace WindowsFormsApp57

{

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

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

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

public class CLex //класс лексический анализатор

{

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

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

public TCharType enumFSelectionCharType;

public char chrFSelection;

private TState enumFState;

private int intFSourceRowSelection;

private int intFSourceColSelection;

private String strFLexicalUnit;

private TToken enumFToken;

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

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

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

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

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

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

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

public CLex()

{

}

public void GetSymbol() //метод класса лексический анализатор

{

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

{

intFSourceRowSelection++;

if (intFSourceRowSelection <= strFSource.Length - 1)

{

intFSourceColSelection = -1;

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndRow;

enumFState = TState.Continue;

}

else

{

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndText;

enumFState = TState.Finish;

}

}

else

{

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

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

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

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

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

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

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

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

enumFState = TState.Continue;

}

intFSourceColSelection++; // продвигаем номер колонки

}

private void TakeSymbol()

{

char[] c = { chrFSelection };

String s = new string(c);

strFLexicalUnit += s;

GetSymbol();

}

public void NextToken()

{

strFLexicalUnit = "";

if (enumFState == TState.Start)

{

intFSourceRowSelection = 0;

intFSourceColSelection = -1;

GetSymbol();

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

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

// BFin | |BFin | CFin | DFin |

// CFin | | | CFin | DFin |

// DFin | | | | DFin |

AFin:

{

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;

}

}

BFin:

{

if (chrFSelection == 'a')

{

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

}

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 == 'a')

{

throw new Exception("Ожидался c или d");

}

else if (chrFSelection == 'b')

{

throw new Exception("Ожидался c или d");

}

if (chrFSelection == 'c')

{

TakeSymbol();

goto CFin;

}

else if (chrFSelection == 'd')

{

TakeSymbol();

goto DFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

DFin:

{

if (chrFSelection == 'a')

{

throw new Exception("Ожидался d");

}

else if (chrFSelection == 'b')

{

throw new Exception("Ожидался d");

}

else if (chrFSelection == 'c')

{

throw new Exception("Ожидался d");

}

if (chrFSelection == 'd')

{

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 | BD | |

// BD | CE | |

// CE | A | FFin |

// FFin | G | |

// G | | H |

// H | FFin | |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto BD;

}

else throw new Exception("Ожидался 0");

BD:

if (chrFSelection == '0')

{

TakeSymbol();

goto CE;

}

else throw new Exception("Ожидался 0");

CE:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

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

FFin:

if (chrFSelection == '0')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидался 0");

G:

if (chrFSelection == '1')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидалась 1");

H:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидалась 1");

}

case TCharType.ReservedSymbol:

{

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

{

while (enumFSelectionCharType != TCharType.EndRow)

GetSymbol();

}

GetSymbol();

}

if (chrFSelection == '(')

{

enumFToken = TToken.lxmLeftParenth;

GetSymbol();

return;

}

if (chrFSelection == ')')

{

enumFToken = TToken.lxmRightParenth;

GetSymbol();

return;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

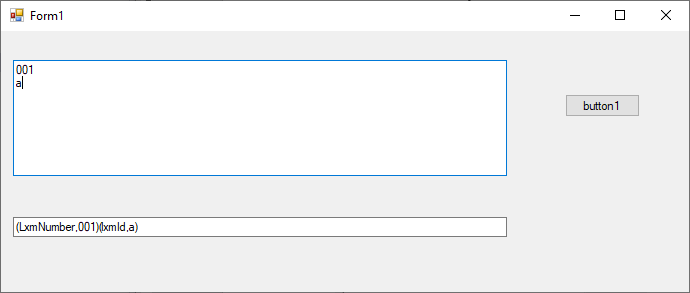
}

}

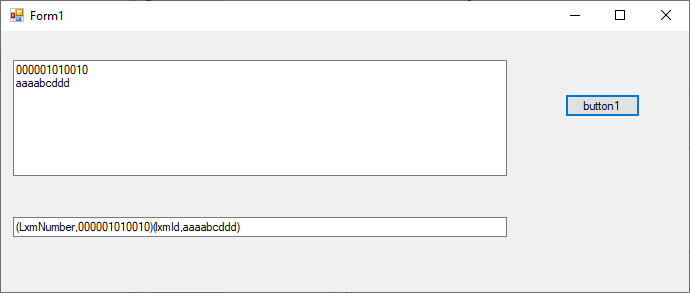
}

}

}







**Лабораторная работа № 3. Разработка транслитератора**

**Текст задания:**

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

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

Указания:

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

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

3. Если грамматика леворекурсивная, то устранить левую рекурсию.

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

**1 Вариант**

S → <2> (А) : - B .

A → <1>A

A → <1>

B → B, <2>

B → <2>

**Избавление от левой рекурсии:**

S → <2> (А) : - B .

A → <1>A

A → <1>

B → <2> | <2> C

C → ,<2> | ,<2> C

1: S 🡪 <2>(A):-B 🡪 <2>(<1>):-<2>

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

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

**Код программы**

**uSyntAnalyzer.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp57

{

class uSyntAnalyzer

{

private String[] strFSource;

private String[] strFMessage;

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

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

public CLex Lex = new CLex();

public void S()

{

if (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmLeftParenth)

{

Lex.NextToken();

A();

if (Lex.enumPToken == TToken.lxmRightParenth)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.twopoints)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.minus)

{

B();

}

else throw new Exception("Ожидалась -");

}

else throw new Exception("Ожидалась :");

}

else throw new Exception("Ожидалась )");

}

else throw new Exception("Ожидалась (");

}

else throw new Exception("Неверно. Введите буквы");

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

}

public void A()

{

if (Lex.enumPToken == TToken.lxmNumber)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.space)

A();

else

{

if (Lex.enumPToken == TToken.lxmNumber)

Lex.NextToken();

else throw new Exception("Неверно. Введите цифры");

}

}

else throw new Exception("Неверно. Введите цифры");

}

public void B()

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmComma)

{

C();

}

else

Lex.NextToken();

}

else throw new Exception("Ожидались буквы");

}

public void C()

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmComma)

{

Lex.NextToken();

C();

}

}

else throw new Exception("Ожидались буквы");

}

}

}

**Form1.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

namespace WindowsFormsApp57

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

tbFSource.AppendText("abc(001):-ab" + "\r\n");

int n = tbFSource.Lines.Length;

}

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

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.Lines;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.S();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Synt.Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

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

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

Lex.intPSourceColSelection = -1;

Lex.intPSourceRowSelection = 0;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

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

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

String s = "";

String s1 = "";

switch (Lex.enumPToken)

{

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

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

}

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

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

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Lex.intPSourceColSelection;

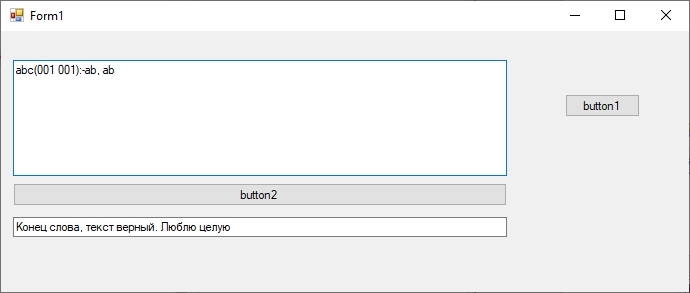
tbFSource.SelectionLength = n;

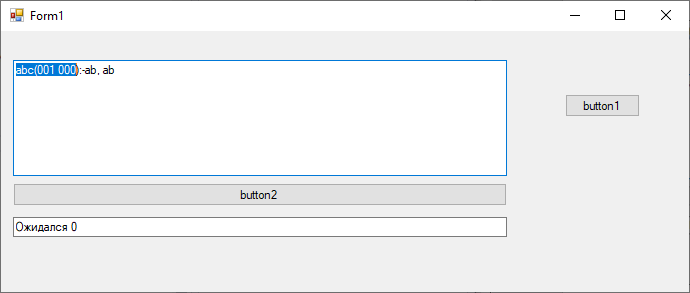
}

}

}

}





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

**Задание:**

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

**Код программы:**

**Form1.cs:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

namespace WindowsFormsApp57

{

public partial class Form1 : Form

{

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

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

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

public Class3 hashFunction = new Class3();

public CHashTableList htl = new CHashTableList(2);

public Form1()

{

InitializeComponent();

tbFSource.AppendText("abc(001):-ab" + "\r\n");

int n = tbFSource.Lines.Length;

}

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

{

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

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

htl.TableToStringList(0, listTable);

foreach (var entry in hashTableIdentifier)

{

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

}

listTable.Clear();

htl.TableToStringList(1, listTable);

foreach (var entry in hashTableDigital)

{

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

}

listTable.Clear();

foreach (var entry in hashTableRezerv)

{

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

}

listTable.Clear();

}

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

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.Lines;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.S();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Synt.Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

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

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmLeftParenth:

{

hashFunction.AddWord(hashTableRezerv, "(");

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmRightParenth:

{

hashFunction.AddWord(hashTableRezerv, ")");

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.twopoints:

{

hashFunction.AddWord(hashTableRezerv, ":");

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.minus:

{

hashFunction.AddWord(hashTableRezerv, "-");

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

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

{

TablesToMemo(this, e);

}

break;

}

}

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

tbFMessage.Text += m;

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

private void button3\_Click(object sender, EventArgs e)

{

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

{

MessageBox.Show("Успешно", "Поиск", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show("Ошибка", "Поиск", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

private void button4\_Click(object sender, EventArgs e)

{

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

MessageBox.Show("Успешно", "Добавление", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

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

{

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

{

MessageBox.Show("Успешно", "Удаление", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show("Ошибка", "Удаление", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

private void button6\_Click(object sender, EventArgs e)

{

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

{

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

MessageBox.Show("Успешно", "Изменение", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show("Ошибка", "Изменение", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

private void button7\_Click(object sender, EventArgs e)

{

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

foreach (var entry in hashTableIdentifier)

{

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

}

foreach (var entry in hashTableDigital)

{

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

}

foreach (var entry in hashTableRezerv)

{

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

}

}

}

}

**Hash.Tables.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 WindowsFormsApp57

{

public class TableItem

{

public int value;

public string lex;

public TableItem next;

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

{

value = v;

lex = s;

next = t;

}

}

public class THashTable

{

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

public int intFHashIndex;

static int tableSize = 20;

public THashTable()

{

Init(tableSize);

}

public void Init(int count)

{

arrFHashTable.Clear();

Resize(arrFHashTable, count);

}

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

{

if (size > list.Count)

while (size > list.Count)

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

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

int HashFunction(string strALexicalUnit)

{

int h = 0;

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

{

h += strALexicalUnit[i];

}

return h % tableSize;

}

public void HashIndex(string strALexicalUnit)

{

int h = HashFunction(strALexicalUnit);

intFHashIndex = h;

}

public bool AddLexicalUnit(string strALexicalUnit, ref int intALexicalCode)

{

HashIndex(strALexicalUnit);

intALexicalCode = intFHashIndex;

TableItem item = arrFHashTable[intFHashIndex];

TableItem prev = arrFHashTable[intFHashIndex];

bool exist = false;

if (prev.lex == strALexicalUnit)

{

exist = true;

}

while (prev.next != null)

{

prev = prev.next;

if (prev.lex == strALexicalUnit)

{

exist = true;

}

}

if (!exist)

{

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

{

item.value = intALexicalCode;

item.lex = strALexicalUnit;

}

else

{

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

prev.next = newItem;

}

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

return true;

}

return false;

}

public bool DeleteLexicalUnit(string strALexicalUnit)

{

HashIndex(strALexicalUnit);

int indx = intFHashIndex;

if (arrFHashTable[indx] != null)

{

TableItem item = arrFHashTable[indx];

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

{

item = item.next;

}

if (item.lex == strALexicalUnit)

{

if (item.next == null)

{

TableItem prev = arrFHashTable[indx];

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

{

prev = prev.next;

}

prev.next = null;

item.value = 0;

item.lex = "";

}

else

{

TableItem prev = arrFHashTable[indx];

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

{

prev = prev.next;

}

item.value = 0;

item.lex = "";

prev.next = item.next;

}

return true;

}

}

return false;

}

public void GetLexicalUnitList(ref List<string> sList)

{

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

{

TableItem item = arrFHashTable[i];

while (item != null)

{

if (item.lex != "")

{

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

}

item = item.next;

}

}

}

}

}

**HashTablesList.cs**

using System.Collections.Generic;

using System.Windows.Forms;

namespace WindowsFormsApp57

{

public class CHashTableList

{

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

private byte byteFTablesSize;

public CHashTableList(byte byteATableCount)

{

this.byteFTablesSize = byteATableCount;

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

{

arrFHashTableList.Add(new THashTable());

}

}

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

{

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

}

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

{

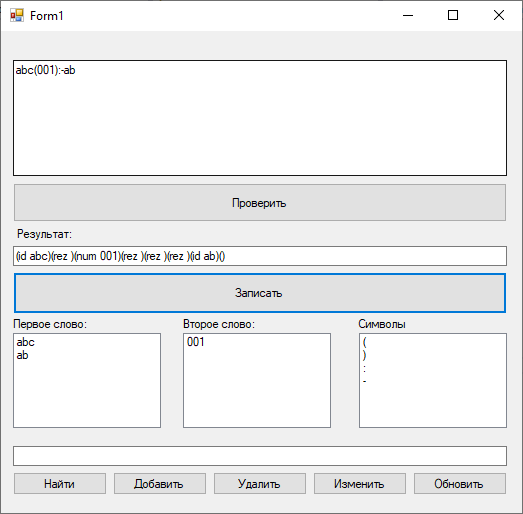
arrFHashTableList[byteATable].GetLexicalUnitList(ref sList);

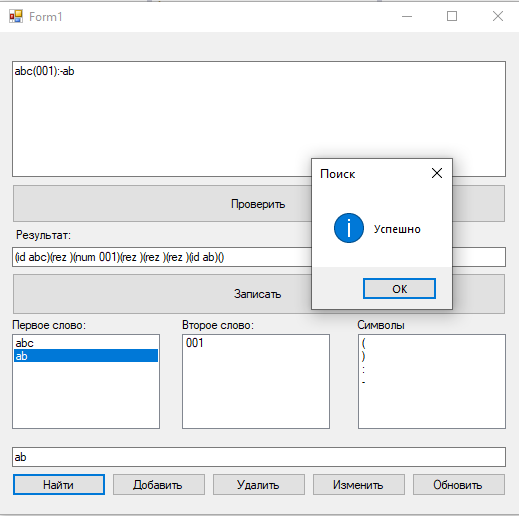
}

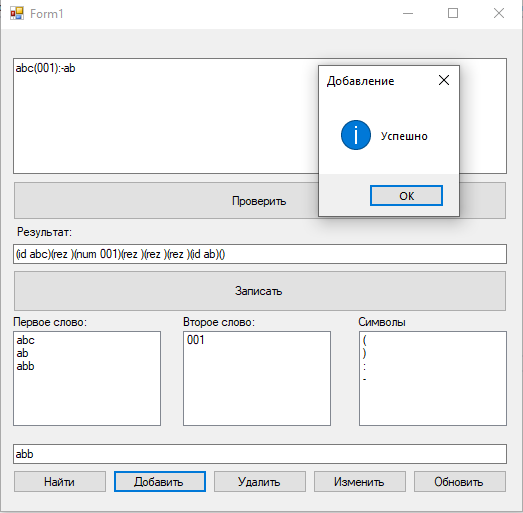
}

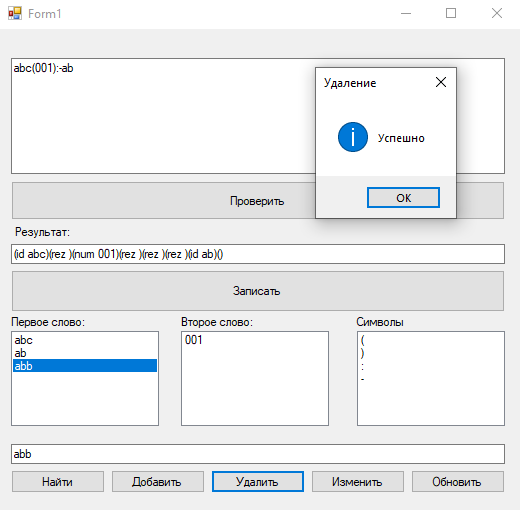
}

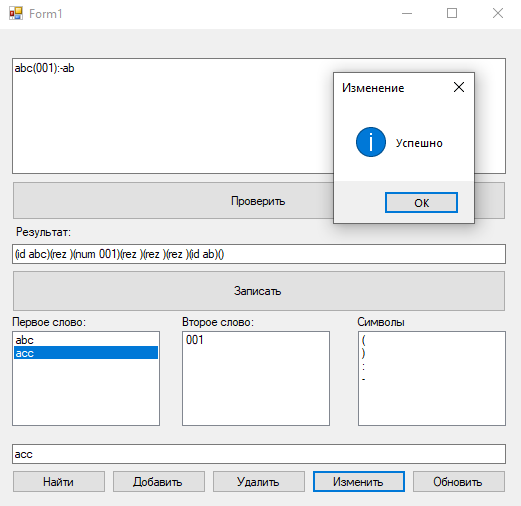
**Class3.cs**











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

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

**Код программы:**

**Form1.cs:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

namespace WindowsFormsApp57

{

public partial class Form1 : Form

{

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

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

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

public Class3 hashFunction = new Class3();

public CHashTableList htl = new CHashTableList(2);

public Form1()

{

InitializeComponent();

tbFSource.AppendText("abc(001):-ab, ab, ab" + "\r\n");

int n = tbFSource.Lines.Length;

}

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

{

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

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

htl.TableToStringList(0, listTable);

foreach (var entry in hashTableIdentifier)

{

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

}

listTable.Clear();

htl.TableToStringList(1, listTable);

foreach (var entry in hashTableDigital)

{

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

}

listTable.Clear();

foreach (var entry in hashTableRezerv)

{

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

}

listTable.Clear();

}

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

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

treeView1.Nodes.Clear();

Synt.derevo = treeView1;

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.Lines;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.S();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Synt.Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

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

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmLeftParenth:

{

hashFunction.AddWord(hashTableRezerv, "(");

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmRightParenth:

{

hashFunction.AddWord(hashTableRezerv, ")");

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.twopoints:

{

hashFunction.AddWord(hashTableRezerv, ":");

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.minus:

{

hashFunction.AddWord(hashTableRezerv, "-");

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

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

{

TablesToMemo(this, e);

}

break;

}

}

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

tbFMessage.Text += m;

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

private void button3\_Click(object sender, EventArgs e)

{

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

{

MessageBox.Show("Успешно", "Поиск", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show("Ошибка", "Поиск", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

private void button4\_Click(object sender, EventArgs e)

{

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

MessageBox.Show("Успешно", "Добавление", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

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

{

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

{

MessageBox.Show("Успешно", "Удаление", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show("Ошибка", "Удаление", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

private void button6\_Click(object sender, EventArgs e)

{

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

{

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

MessageBox.Show("Успешно", "Изменение", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show("Ошибка", "Изменение", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

private void button7\_Click(object sender, EventArgs e)

{

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

foreach (var entry in hashTableIdentifier)

{

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

}

foreach (var entry in hashTableDigital)

{

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

}

foreach (var entry in hashTableRezerv)

{

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

}

}

}

}

**Class2.cs**

using System;

using System.Windows.Forms;

namespace WindowsFormsApp57

{

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 derevo;

public void S()

{

TreeNode tek = new TreeNode("S");

derevo.Nodes.Add(tek);

if (Lex.enumPToken == TToken.lxmIdentifier)

{

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

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmLeftParenth)

{

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

Lex.NextToken();

A(tek);

if (Lex.enumPToken == TToken.lxmRightParenth)

{

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

Lex.NextToken();

if (Lex.enumPToken == TToken.twopoints)

{

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

Lex.NextToken();

if (Lex.enumPToken == TToken.minus)

{

tek.Nodes.Add(new TreeNode("-"));

B(tek);

}

else throw new Exception("Ожидалась -");

}

else throw new Exception("Ожидалась :");

}

else throw new Exception("Ожидалась )");

}

else throw new Exception("Ожидалась (");

}

else throw new Exception("Неверно. Введите буквы");

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

}

public void A(TreeNode papa)

{

TreeNode tek = new TreeNode("A");

papa.Nodes.Add(tek);

if (Lex.enumPToken == TToken.lxmNumber)

{

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

Lex.NextToken();

if (Lex.enumPToken == TToken.space)

{

tek.Nodes.Add(new TreeNode("\_"));

A(tek);

}

else

{

if (Lex.enumPToken == TToken.lxmNumber)

{

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

Lex.NextToken();

}

}

}

else throw new Exception("Неверно. Введите цифры");

}

public void B(TreeNode papa)

{

TreeNode tek = new TreeNode("B");

papa.Nodes.Add(tek);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier)

{

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

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmComma)

{

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

C(tek);

}

else

Lex.NextToken();

}

else throw new Exception("Ожидались буквы");

}

public void C(TreeNode papa)

{

TreeNode tek = new TreeNode("C");

papa.Nodes.Add(tek);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier)

{

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

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmComma)

{

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

//Lex.NextToken();

C(tek);

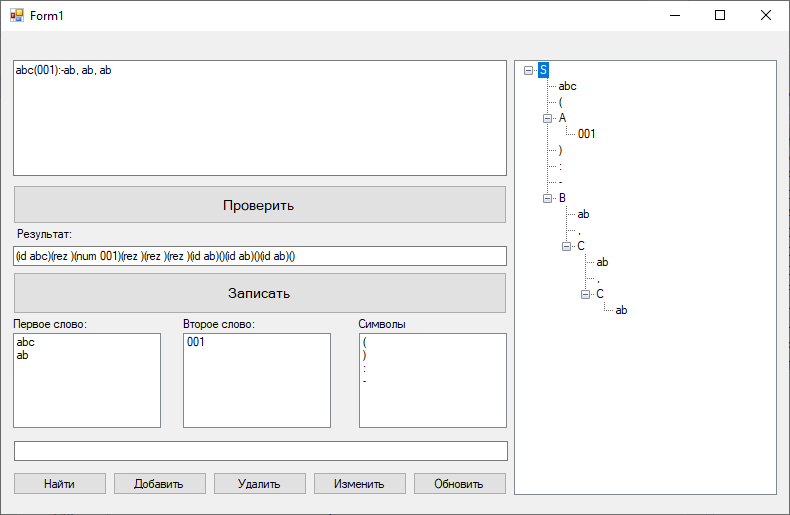
}

}

else throw new Exception("Ожидались буквы");

}

}

} 

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

**Задание:**

1. Перевести все числа в десятичное представление.

2. Выполнить вывод исходного текста в структурированном виде.

Код программы:

Form1.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Reflection.Emit;

using System.Text;

using System.Windows.Forms;

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

namespace WindowsFormsApp57

{

public partial class Form1 : Form

{

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

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

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

public Class3 hashFunction = new Class3();

public CHashTableList htl = new CHashTableList(2);

public Form1()

{

InitializeComponent();

tbFSource.AppendText("abc(001):-ab, ab, ab" + "\r\n");

int n = tbFSource.Lines.Length;

}

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

{

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

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

htl.TableToStringList(0, listTable);

foreach (var entry in hashTableIdentifier)

{

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

}

listTable.Clear();

htl.TableToStringList(1, listTable);

foreach (var entry in hashTableDigital)

{

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

}

listTable.Clear();

foreach (var entry in hashTableRezerv)

{

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

}

listTable.Clear();

}

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

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

treeView1.Nodes.Clear();

Synt.derevo = treeView1;

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.Lines;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.S();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Synt.Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

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

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

Generator gen = new Generator();

gen.Restruct(treeView1, treeView2);

Generator.RefactorTree(treeView2);

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmLeftParenth:

{

hashFunction.AddWord(hashTableRezerv, "(");

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmRightParenth:

{

hashFunction.AddWord(hashTableRezerv, ")");

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.twopoints:

{

hashFunction.AddWord(hashTableRezerv, ":");

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.minus:

{

hashFunction.AddWord(hashTableRezerv, "-");

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

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

{

TablesToMemo(this, e);

}

break;

}

}

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

tbFMessage.Text += m;

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

private void button3\_Click(object sender, EventArgs e)

{

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

{

MessageBox.Show("Успешно", "Поиск", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show("Ошибка", "Поиск", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

private void button4\_Click(object sender, EventArgs e)

{

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

MessageBox.Show("Успешно", "Добавление", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

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

{

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

{

MessageBox.Show("Успешно", "Удаление", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show("Ошибка", "Удаление", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

private void button6\_Click(object sender, EventArgs e)

{

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

{

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

MessageBox.Show("Успешно", "Изменение", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show("Ошибка", "Изменение", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

private void button7\_Click(object sender, EventArgs e)

{

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

foreach (var entry in hashTableIdentifier)

{

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

}

foreach (var entry in hashTableDigital)

{

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

}

foreach (var entry in hashTableRezerv)

{

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

}

}

}

}

Generator.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp57

{

public class Generator

{

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

{

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

foreach (TreeNode node in tree.Nodes)

{

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

}

return treeContent;

}

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

{

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

foreach (TreeNode child in node.Nodes)

{

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

}

if (edit)

{

int value = -1;

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

{

int newValue = ConvertToBase10(value);

node.Text = newValue.ToString();

}

}

nodeContent.Add(node.Text);

return nodeContent;

}

public int ConvertToBase10(int num)

{

int k = 0;

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

{

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

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

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

}

return k;

}

public TreeView currentTree;

public int depth = 0;

public void Restruct(TreeView box, TreeView tree)

{

currentTree = tree;

tree.Nodes.Clear();

TreeNode sNode = null;

foreach (TreeNode node in box.Nodes)

{

TreeNode newNode = new TreeNode(node.Text);

ReNode(null, node, ref newNode);

tree.Nodes.Add(newNode);

}

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

foreach (TreeNode node in tree.Nodes)

{

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

{

remove.Add(node);

}

}

tree.ExpandAll();

}

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

{

bool skipToRoot = false;

foreach (TreeNode child in node.Nodes)

{

TreeNode newn = new TreeNode(child.Text);

int value = -1;

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

{

int newValue = ConvertToBase10(value);

newn.Text = newValue.ToString();

}

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

{

skipToRoot = true;

}

ReNode(newNode, child, ref newn);

}

if (parent != null)

{

if (currentTree != null && skipToRoot)

{

currentTree.Nodes.Add(newNode);

}

else

{

parent.Nodes.Add(newNode);

}

}

}

public static void RefactorTree(TreeView tree)

{

foreach (TreeNode node in tree.Nodes)

{

RefactorTreeNodes(tree, node);

}

}

public static void RefactorTreeNodes(TreeView tree, TreeNode parent)

{

TreeNode nodeToMove = null;

foreach (TreeNode node in parent.Nodes)

{

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

{

nodeToMove = node;

}

RefactorTreeNodes(tree, node);

}

if (nodeToMove != null)

{

parent.Nodes.Remove(nodeToMove);

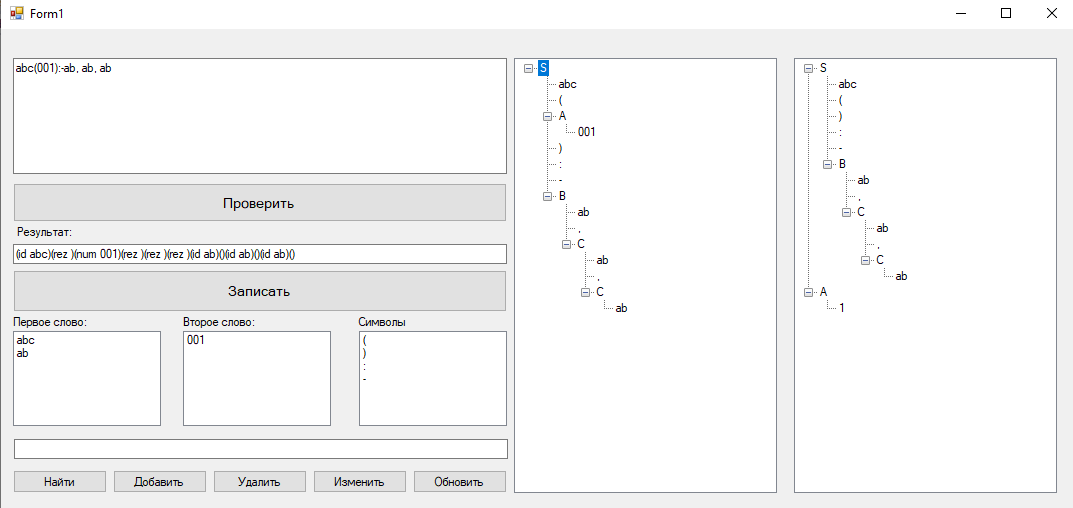
tree.Nodes.Add(nodeToMove);

}

}

}

}



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

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

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

Код программы:  
**Class2.cs**

using System;

using System.Windows.Forms;

namespace WindowsFormsApp57

{

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 derevo;

public TreeView derevo2;

public void S()

{

TreeNode tek = new TreeNode("S");

derevo.Nodes.Add(tek);

if (Lex.enumPToken == TToken.lxmIdentifier)

{

if (!IsIdenUnique(Lex.strPLexicalUnit))

{

throw new Exception("Идентификатор " + Lex.strPLexicalUnit + " повторился!");

}

else

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

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmLeftParenth)

{

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

Lex.NextToken();

A(tek);

if (Lex.enumPToken == TToken.lxmRightParenth)

{

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

Lex.NextToken();

if (Lex.enumPToken == TToken.twopoints)

{

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

Lex.NextToken();

if (Lex.enumPToken == TToken.minus)

{

tek.Nodes.Add(new TreeNode("-"));

B(tek);

}

else throw new Exception("Ожидалась -");

}

else throw new Exception("Ожидалась :");

}

else throw new Exception("Ожидалась )");

}

else throw new Exception("Ожидалась (");

}

else throw new Exception("Неверно. Введите буквы");

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

}

public void A(TreeNode papa)

{

TreeNode tek = new TreeNode("A");

papa.Nodes.Add(tek);

if (Lex.enumPToken == TToken.lxmNumber)

{

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

Lex.NextToken();

if (Lex.enumPToken == TToken.space)

{

tek.Nodes.Add(new TreeNode("\_"));

A(tek);

}

else

{

if (Lex.enumPToken == TToken.lxmNumber)

{

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

Lex.NextToken();

}

}

}

else throw new Exception("Неверно. Введите цифры");

}

public void B(TreeNode papa)

{

TreeNode tek = new TreeNode("B");

papa.Nodes.Add(tek);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier)

{

if (!IsIdenUnique(Lex.strPLexicalUnit))

{

throw new Exception("Идентификатор " + Lex.strPLexicalUnit + " повторился!");

}

else

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

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmComma)

{

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

C(tek);

}

else

Lex.NextToken();

}

else throw new Exception("Ожидались буквы");

}

public void C(TreeNode papa)

{

TreeNode tek = new TreeNode("C");

papa.Nodes.Add(tek);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier)

{

if (!IsIdenUnique(Lex.strPLexicalUnit))

{

throw new Exception("Идентификатор " + Lex.strPLexicalUnit + " повторился!");

}

else

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

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmComma)

{

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

//Lex.NextToken();

C(tek);

}

}

else throw new Exception("Ожидались буквы");

}

private bool IsIdenUnique(string Num)

{

foreach (TreeNode node in derevo.Nodes)

{

if (!IsIdenUniqueInNode(Num, node))

{

return false;

}

}

return true;

}

private bool IsIdenUniqueInNode(string Num, TreeNode node)

{

if (node.Text == Num)

{

return false;

}

foreach (TreeNode child in node.Nodes)

{

if (!IsIdenUniqueInNode(Num, child))

{

return false;

}

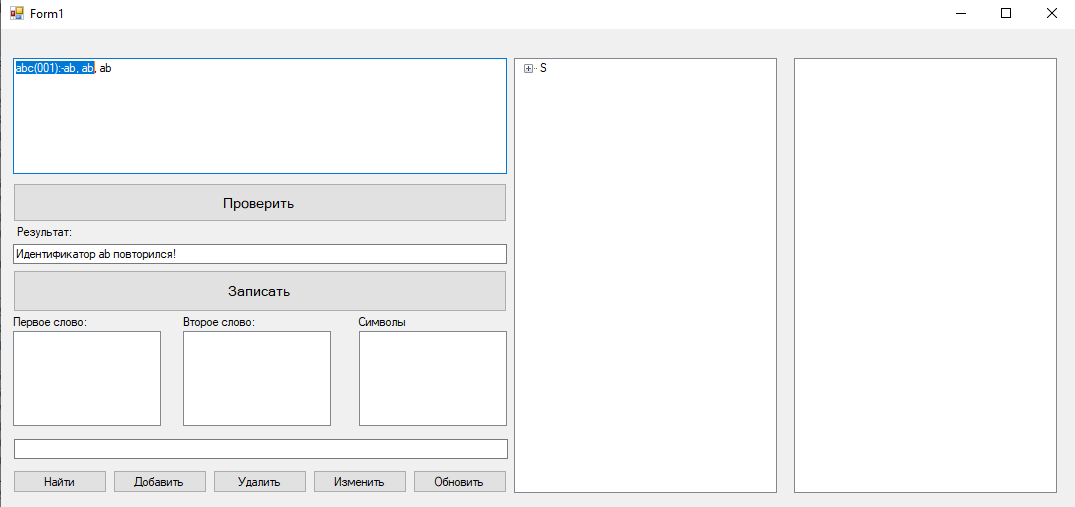
}

return true;

}

}

}



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

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

**Вариант:** После идентификатора aa не может быть идентификатор bb.

Код программы:

**Form1.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Reflection.Emit;

using System.Text;

using System.Windows.Forms;

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

namespace WindowsFormsApp57

{

public partial class Form1 : Form

{

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

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

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

public Class3 hashFunction = new Class3();

public CHashTableList htl = new CHashTableList(2);

public Form1()

{

InitializeComponent();

tbFSource.AppendText("abc(001):-ab, ab, ab" + "\r\n");

int n = tbFSource.Lines.Length;

}

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

{

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

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

htl.TableToStringList(0, listTable);

foreach (var entry in hashTableIdentifier)

{

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

}

listTable.Clear();

htl.TableToStringList(1, listTable);

foreach (var entry in hashTableDigital)

{

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

}

listTable.Clear();

foreach (var entry in hashTableRezerv)

{

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

}

listTable.Clear();

}

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

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

treeView1.Nodes.Clear();

Synt.derevo = treeView1;

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.Lines;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.S();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Synt.Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

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

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

Generator gen = new Generator();

gen.Restruct(treeView1, treeView2);

Generator.RefactorTree(treeView2);

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmLeftParenth:

{

hashFunction.AddWord(hashTableRezerv, "(");

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmRightParenth:

{

hashFunction.AddWord(hashTableRezerv, ")");

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.twopoints:

{

hashFunction.AddWord(hashTableRezerv, ":");

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.minus:

{

hashFunction.AddWord(hashTableRezerv, "-");

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

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

{

TablesToMemo(this, e);

}

break;

}

}

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

tbFMessage.Text += m;

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

private void button3\_Click(object sender, EventArgs e)

{

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

{

MessageBox.Show("Успешно", "Поиск", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show("Ошибка", "Поиск", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

private void button4\_Click(object sender, EventArgs e)

{

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

MessageBox.Show("Успешно", "Добавление", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

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

{

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

{

MessageBox.Show("Успешно", "Удаление", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show("Ошибка", "Удаление", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

private void button6\_Click(object sender, EventArgs e)

{

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

{

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

MessageBox.Show("Успешно", "Изменение", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show("Ошибка", "Изменение", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

private void button7\_Click(object sender, EventArgs e)

{

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

foreach (var entry in hashTableIdentifier)

{

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

}

foreach (var entry in hashTableDigital)

{

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

}

foreach (var entry in hashTableRezerv)

{

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

}

}

}

}

**Class1.cs**

using System;

using System.Collections.Generic;

using System.Text;

namespace WindowsFormsApp57

{

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

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

public enum TToken { lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma, twopoints, minus, lxmls, lxmrs, lxmr, lxpoint, lxvz, lxdorar, lxpz, space };

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 == '\*') enumFSelectionCharType = TCharType.ReservedSymbol;

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

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

enumFState = TState.Continue;

}

}

private void TakeSymbol()

{

char[] c = { chrFSelection };

String s = new string(c);

strFLexicalUnit += s;

GetSymbol();

}

public void NextToken()

{

strFLexicalUnit = "";

if (enumFState == TState.Start)

{

intFSourceRowSelection = 0;

intFSourceColSelection = -1;

GetSymbol();

}

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

{

GetSymbol();

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

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

// BFin | |BFin | CFin | DFin |

// CFin | | | CFin | DFin |

// DFin | | | | DFin |

AFin:

{

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;

}

}

BFin:

{

if (chrFSelection == 'a')

{

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

}

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 == 'a')

{

throw new Exception("Ожидался c или d");

}

else if (chrFSelection == 'b')

{

throw new Exception("Ожидался c или d");

}

if (chrFSelection == 'c')

{

TakeSymbol();

goto CFin;

}

else if (chrFSelection == 'd')

{

TakeSymbol();

goto DFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

DFin:

{

if (chrFSelection == 'a')

{

throw new Exception("Ожидался d");

}

else if (chrFSelection == 'b')

{

throw new Exception("Ожидался d");

}

else if (chrFSelection == 'c')

{

throw new Exception("Ожидался d");

}

if (chrFSelection == 'd')

{

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 | BD | |

// BD | CE | |

// CE | A | FFin |

// FFin | G | |

// G | | H |

// H | FFin | |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto BD;

}

else throw new Exception("Ожидался 0");

BD:

if (chrFSelection == '0')

{

TakeSymbol();

goto CE;

}

else throw new Exception("Ожидался 0");

CE:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

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

FFin:

if (chrFSelection == '0')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидался 0");

G:

if (chrFSelection == '1')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидалась 1");

H:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидалась 1");

}

case TCharType.ReservedSymbol:

{

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

{

while (enumFSelectionCharType != TCharType.EndRow)

GetSymbol();

}

GetSymbol();

}

if (chrFSelection == '(')

{

enumFToken = TToken.lxmLeftParenth;

GetSymbol();

return;

}

if (chrFSelection == ')')

{

enumFToken = TToken.lxmRightParenth;

GetSymbol();

return;

}

if (chrFSelection == ':')

{

enumFToken = TToken.twopoints;

GetSymbol();

return;

}

if (chrFSelection == '-')

{

enumFToken = TToken.minus;

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.lxmr;

GetSymbol();

return;

}

if (chrFSelection == '.')

{

enumFToken = TToken.lxpoint;

GetSymbol();

return;

}

if (chrFSelection == '!')

{

enumFToken = TToken.lxvz;

GetSymbol();

return;

}

if (chrFSelection == '$')

{

enumFToken = TToken.lxdorar;

GetSymbol();

return;

}

if (chrFSelection == ';')

{

enumFToken = TToken.lxpz;

GetSymbol();

return;

}

if (chrFSelection == ' ')

{

enumFToken = TToken.space;

GetSymbol();

return;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

}

}

}

}

**Class2.cs**

using System;

using System.Windows.Forms;

namespace WindowsFormsApp57

{

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 derevo;

public TreeView derevo2;

public void S()

{

TreeNode tek = new TreeNode("S");

derevo.Nodes.Add(tek);

if (Lex.enumPToken == TToken.lxmIdentifier)

{

if (!IsIdenUnique(Lex.strPLexicalUnit))

{

throw new Exception("Идентификатор " + Lex.strPLexicalUnit + " повторился!");

}

else

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

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmLeftParenth)

{

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

Lex.NextToken();

A(tek);

if (Lex.enumPToken == TToken.lxmRightParenth)

{

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

Lex.NextToken();

if (Lex.enumPToken == TToken.twopoints)

{

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

Lex.NextToken();

if (Lex.enumPToken == TToken.minus)

{

tek.Nodes.Add(new TreeNode("-"));

B(tek);

}

else throw new Exception("Ожидалась -");

}

else throw new Exception("Ожидалась :");

}

else throw new Exception("Ожидалась )");

}

else throw new Exception("Ожидалась (");

}

else throw new Exception("Неверно. Введите буквы");

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

}

public void A(TreeNode papa)

{

TreeNode tek = new TreeNode("A");

papa.Nodes.Add(tek);

if (Lex.enumPToken == TToken.lxmNumber)

{

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

Lex.NextToken();

if (Lex.enumPToken == TToken.space)

{

tek.Nodes.Add(new TreeNode("\_"));

A(tek);

}

else

{

if (Lex.enumPToken == TToken.lxmNumber)

{

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

Lex.NextToken();

}

}

}

else throw new Exception("Неверно. Введите цифры");

}

public void B(TreeNode papa)

{

TreeNode tek = new TreeNode("B");

papa.Nodes.Add(tek);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier)

{

if (!IsIdenUnique(Lex.strPLexicalUnit))

{

throw new Exception("Идентификатор " + Lex.strPLexicalUnit + " повторился!");

}

else

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

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmComma)

{

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

C(tek);

}

else

Lex.NextToken();

}

else throw new Exception("Ожидались буквы");

}

public void C(TreeNode papa)

{

TreeNode tek = new TreeNode("C");

papa.Nodes.Add(tek);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier)

{

if (!IsIdenUnique(Lex.strPLexicalUnit))

{

throw new Exception("Идентификатор " + Lex.strPLexicalUnit + " повторился!");

}

else

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

Lex.NextToken();

Class4 class4 = new Class4(derevo);

if (Lex.enumPToken == TToken.lxmComma)

{

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

//Lex.NextToken();

C(tek);

}

}

else throw new Exception("Ожидались буквы");

}

private bool IsIdenUnique(string Num)

{

foreach (TreeNode node in derevo.Nodes)

{

if (!IsIdenUniqueInNode(Num, node))

{

return false;

}

}

return true;

}

private bool IsIdenUniqueInNode(string Num, TreeNode node)

{

if (node.Text == Num)

{

return false;

}

foreach (TreeNode child in node.Nodes)

{

if (!IsIdenUniqueInNode(Num, child))

{

return false;

}

}

return true;

}

}

}

**Class3.cs**

using System;

using System.Collections.Generic;

namespace WindowsFormsApp57

{

public class Class3

{

public int HashFunction(string word)

{

int hashValue = 0;

foreach (char c in word)

{

hashValue += (int)c;

}

return hashValue;

}

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

{

int hashValue = HashFunction(word);

if (!hashTable.ContainsKey(hashValue))

{

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

}

else

{

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

{

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

return;

}

}

hashTable[hashValue].Add(word);

}

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

{

int hashValue = HashFunction(word);

if (hashTable.ContainsKey(hashValue))

{

return 1;

}

else

{

return 0;

}

}

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

{

int hashValue = HashFunction(word);

if (hashTable.ContainsKey(hashValue))

{

List<string> words = hashTable[hashValue];

if (words.Contains(word))

{

words.Remove(word);

if (words.Count == 0)

{

hashTable.Remove(hashValue);

}

return true;

}

}

return false;

}

}

}

**Class4.cs**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp57

{

internal class Class4

{

public int i = 0;

public string strIndentifier1;

public string strIndentifier2;

private TreeView tree;

public Class4(TreeView treeView)

{

tree = treeView;

TreeController(tree);

}

public void TreeController(TreeView tree)

{

foreach (TreeNode node in tree.Nodes)

{

TreeController(node);

}

}

public void TreeController(TreeNode node)

{

if (node.Text == "B")

{

strIndentifier1 = node.Nodes[0].Text.ToString();

strIndentifier2 = node.Nodes[2].Nodes[0].Text.ToString();

Check(strIndentifier1, strIndentifier2, node);

}

foreach (TreeNode childNode in node.Nodes)

{

TreeController(childNode);

}

}

private void Check(string ident1, string ident2, TreeNode node)

{

if (ident1 == "aa" && ident2 == "bb")

{

tree.SelectedNode = node;

tree.SelectedNode.BackColor = Color.Red;

MessageBox.Show("Ошибка", "Условие не выполнено", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

Generator.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp57

{

public class Generator

{

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

{

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

foreach (TreeNode node in tree.Nodes)

{

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

}

return treeContent;

}

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

{

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

foreach (TreeNode child in node.Nodes)

{

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

}

if (edit)

{

int value = -1;

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

{

int newValue = ConvertToBase10(value);

node.Text = newValue.ToString();

}

}

nodeContent.Add(node.Text);

return nodeContent;

}

public int ConvertToBase10(int num)

{

int k = 0;

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

{

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

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

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

}

return k;

}

public TreeView currentTree;

public int depth = 0;

public void Restruct(TreeView box, TreeView tree)

{

currentTree = tree;

tree.Nodes.Clear();

TreeNode sNode = null;

foreach (TreeNode node in box.Nodes)

{

TreeNode newNode = new TreeNode(node.Text);

ReNode(null, node, ref newNode);

tree.Nodes.Add(newNode);

}

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

foreach (TreeNode node in tree.Nodes)

{

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

{

remove.Add(node);

}

}

tree.ExpandAll();

}

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

{

bool skipToRoot = false;

foreach (TreeNode child in node.Nodes)

{

TreeNode newn = new TreeNode(child.Text);

int value = -1;

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

{

int newValue = ConvertToBase10(value);

newn.Text = newValue.ToString();

}

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

{

skipToRoot = true;

}

ReNode(newNode, child, ref newn);

}

if (parent != null)

{

if (currentTree != null && skipToRoot)

{

currentTree.Nodes.Add(newNode);

}

else

{

parent.Nodes.Add(newNode);

}

}

}

public static void RefactorTree(TreeView tree)

{

foreach (TreeNode node in tree.Nodes)

{

RefactorTreeNodes(tree, node);

}

}

public static void RefactorTreeNodes(TreeView tree, TreeNode parent)

{

TreeNode nodeToMove = null;

foreach (TreeNode node in parent.Nodes)

{

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

{

nodeToMove = node;

}

RefactorTreeNodes(tree, node);

}

if (nodeToMove != null)

{

parent.Nodes.Remove(nodeToMove);

tree.Nodes.Add(nodeToMove);

}

}

}

}

**Hash.Tables.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 WindowsFormsApp57

{

public class TableItem

{

public int value;

public string lex;

public TableItem next;

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

{

value = v;

lex = s;

next = t;

}

}

public class THashTable

{

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

public int intFHashIndex;

static int tableSize = 20;

public THashTable()

{

Init(tableSize);

}

public void Init(int count)

{

arrFHashTable.Clear();

Resize(arrFHashTable, count);

}

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

{

if (size > list.Count)

while (size > list.Count)

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

else if (size < list.Count)

while (list.Count - size > 0)

list.RemoveAt(list.Count - 1);

}

int HashFunction(string strALexicalUnit)

{

int h = 0;

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

{

h += strALexicalUnit[i];

}

return h % tableSize;

}

public void HashIndex(string strALexicalUnit)

{

int h = HashFunction(strALexicalUnit);

intFHashIndex = h;

}

public bool SearchLexicalUnit(string strAlexicalUnit, ref int intALexicalCode)

{

HashIndex(strAlexicalUnit);

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

else

{

intALexicalCode = arrFHashTable[intFHashIndex].value;

return true;

}

}

public bool AddLexicalUnit(string strALexicalUnit, ref int intALexicalCode)

{

HashIndex(strALexicalUnit);

intALexicalCode = intFHashIndex;

TableItem item = arrFHashTable[intFHashIndex];

TableItem prev = arrFHashTable[intFHashIndex];

bool exist = false;

if (prev.lex == strALexicalUnit)

{

exist = true;

}

while (prev.next != null)

{

prev = prev.next;

if (prev.lex == strALexicalUnit)

{

exist = true;

}

}

if (!exist)

{

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

{

item.value = intALexicalCode;

item.lex = strALexicalUnit;

}

else

{

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

prev.next = newItem;

}

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

return true;

}

return false;

}

public bool DeleteLexicalUnit(string strALexicalUnit)

{

HashIndex(strALexicalUnit);

int indx = intFHashIndex;

if (arrFHashTable[indx] != null)

{

TableItem item = arrFHashTable[indx];

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

{

item = item.next;

}

if (item.lex == strALexicalUnit)

{

if (item.next == null)

{

TableItem prev = arrFHashTable[indx];

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

{

prev = prev.next;

}

prev.next = null;

item.value = 0;

item.lex = "";

}

else

{

TableItem prev = arrFHashTable[indx];

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

{

prev = prev.next;

}

item.value = 0;

item.lex = "";

prev.next = item.next;

}

return true;

}

}

return false;

}

public void GetLexicalUnitList(ref List<string> sList)

{

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

{

TableItem item = arrFHashTable[i];

while (item != null)

{

if (item.lex != "")

{

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

}

item = item.next;

}

}

}

}

}

**HashTablesList.cs**

using System.Collections.Generic;

using System.Windows.Forms;

namespace WindowsFormsApp57

{

public class CHashTableList

{

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

private byte byteFTablesSize;

public CHashTableList(byte byteATableCount)

{

this.byteFTablesSize = byteATableCount;

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

{

arrFHashTableList.Add(new THashTable());

}

}

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

{

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

}

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

{

arrFHashTableList[byteATable].GetLexicalUnitList(ref sList);

}

}

}

