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

Ландин.М.О.

Проверил: Бикмуллина И.И.

Казань 2023

**Лабораторная работа № 1. Разработка транслитератора**

**Текст задания:**

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

**Краткое теоретическое обоснование:**

**Общая организация транслитератора**

Транслитератор демонстрационного языка программирования используется для выделения следующих классов отдельных символов:

**Класс букв:** содержит прописные и строчные буквы латинского алфавита, используемые при создании разнообразных конструкций языка. Русские буквы в этот класс не включаются, так как используются только внутри строк и комментариев, допускающих почти все символы.

**Класс десятичных цифр:** объединяет арабские цифры от 0 до 9. Используется при формировании описаний действительных, а также некоторых из целых чисел.

**Класс двоичных цифр:** объединяет цифры 0 и 1. Используется при анализе целых двоичных чисел.

**Класс восьмеричных цифр:** объединяет цифры от 0 до 7. Используется при анализе целых восьмеричных чисел.

**Класс шестнадцатеричных цифр:** включает цифры от 0 до 9, а также прописные и строчные буквы: A, B, C, D, E, F, a, b, c, d, e f.

**Класс пропусков:** состоит из пробела, перевода строки, табуляции, перевода формата (разделяющего текст на отдельные страницы). Символы этого класса используются для разделения различных элементарных конструкций, слитное написание которых привело бы к неправильному восприятию (например, следующие друг за другом число и идентификатор "123E4 asdf" без пробела были бы восприняты как "123E4asdf", что является ошибкой).

**Класс игнорируемых символов:** включает все символы, которые, как предполагается, не отображаются на экране текстового редактора. В используемых кодовых таблицах к ним относятся символы, коды которых меньше кода пробела. Исключение составляют перевод строки, табуляция, перевод формата, уже отнесенные к предыдущему классу. В некоторых текстовых редакторов данные символы отображаются в виде специальных значков. Поэтому, выделение данного класса может являться спорным и зависит от различных факторов.

**Класс прочих символов:** включает все оставшиеся символы. Несмотря на то, что их тоже можно группировать в различные классы, в рассматриваемом языке нам, в большинстве ситуаций, достаточно использовать их непосредственные значения

Следует отметить, что классы символов пресекаются. Однако, вопрос принадлежности нужному классу можно решать, основываясь на текущем контексте. Класс символов можно специально не хранить, а проверять тогда, когда потребуется.

**Код программы:**

**ULex.cs**

using System;

using System.Collections.Generic;

using System.Text;

namespace Tyap

{

public partial class Form1 : Form

{

public enum TState { Start, Continue, Finish }; //типы состояния символов и токенов

public enum TCharType { Letter, Digit, EndRow, EndText, Space, ReservedSymbol, opsko, closko, voskl, zap, tchz, dvach, toch, tir };

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

public class CLex //класс лексический анализатора

{

//Поля для хранения исходного текста, сообщений, текущей позиции, выделенной лексемы и ее атрибутов

private String[] strFSource;

private String[] strFMessage;

public TCharType enumFSelectionCharType;

public char chrFSelection;

private TState enumFState;

private int intFSourceRowSelection;

private int intFSourceColSelection;

private String strFLexicalUnit;

private TToken enumFToken;

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

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

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

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

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

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

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

public void GetSymbol() //метод класса по обработке символа

{

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.opsko;

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

else if (chrFSelection == ':' ) enumFSelectionCharType = TCharType.dvach;

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

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

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

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

else if (chrFSelection == '-') enumFSelectionCharType = TCharType.tir;

else throw new System.Exception("Cимвол вне алфавита");//Распознание типа символа или вызов исключения

enumFState = TState.Continue;

}

intFSourceColSelection++; // продвигаем номер колонки

}

private void TakeSymbol()

{

char[] c = { chrFSelection };

String s = new string(c);

strFLexicalUnit += s;

GetSymbol();

}

public void NextToken()//сдвиг

{

strFLexicalUnit = "";

if (enumFState == TState.Start)

{

intFSourceRowSelection = 0;

intFSourceColSelection = -1;

GetSymbol();

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

}

}

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 Tyap

{

public partial class Form1 : Form

{

public Form1()//инициализация формы и заполнение textbox1 текстовыми данлными

{

InitializeComponent();

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

textBox1.AppendText("1 a");

string s = "" + '\0';

textBox1.AppendText(s);

int n = textBox1.Lines.Length;

}

private void button1\_Click\_1(object sender, EventArgs e)

{

CLex Lex = new CLex();

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.toch: { s1 = "Tochka"; break; }

case TCharType.tchz: { s1 = "TochkaZap"; break; }

case TCharType.opsko: { s1 = "OtkrSkobki"; break; }

case TCharType.closko: { s1 = "ZakrSkobki"; break; }

case TCharType.dvach: { s1 = "Dvoetoch"; break; }

case TCharType.tir: { s1 = "Tire"; break; }

case TCharType.zap: { s1 = "Zapyataya"; break; }

case TCharType.voskl: { s1 = "Vosklic"; 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) // исключение

{

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;

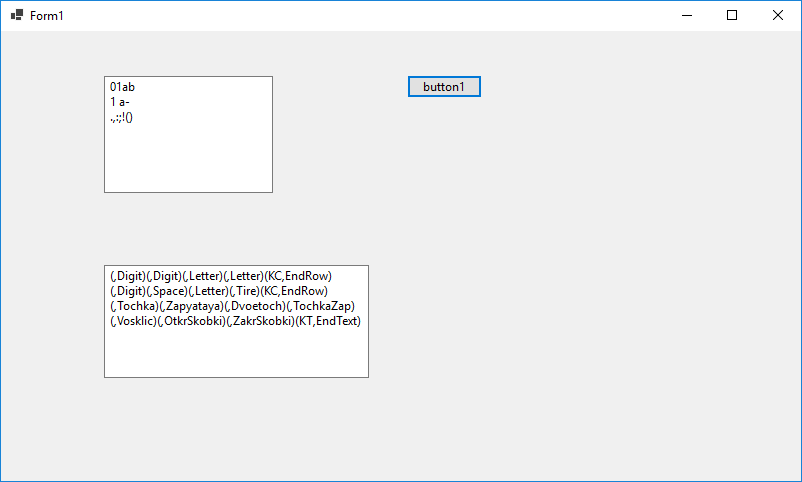
textBox1.SelectionLength = n;

}

}

}

}



**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.

|  |  |  |  |
| --- | --- | --- | --- |
| 115 | (010)\*001(000)\* | (a|b|c|d)+ | Первые два символа всегда cb |

**Первое слово:**

(010)\*001(000)\*

A → 0B

B → 1С

B → 0D

C → 0А

D → 1 | 1E

E → 0F

F → 0G

G → 0 | 0E

**Недетерминированная матрица:**

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

**Детерминированная матрица:**

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

**Второе слово:**

(a|b|c|d)+

Первые два символа всегда cb

A → cB

B → b | bС

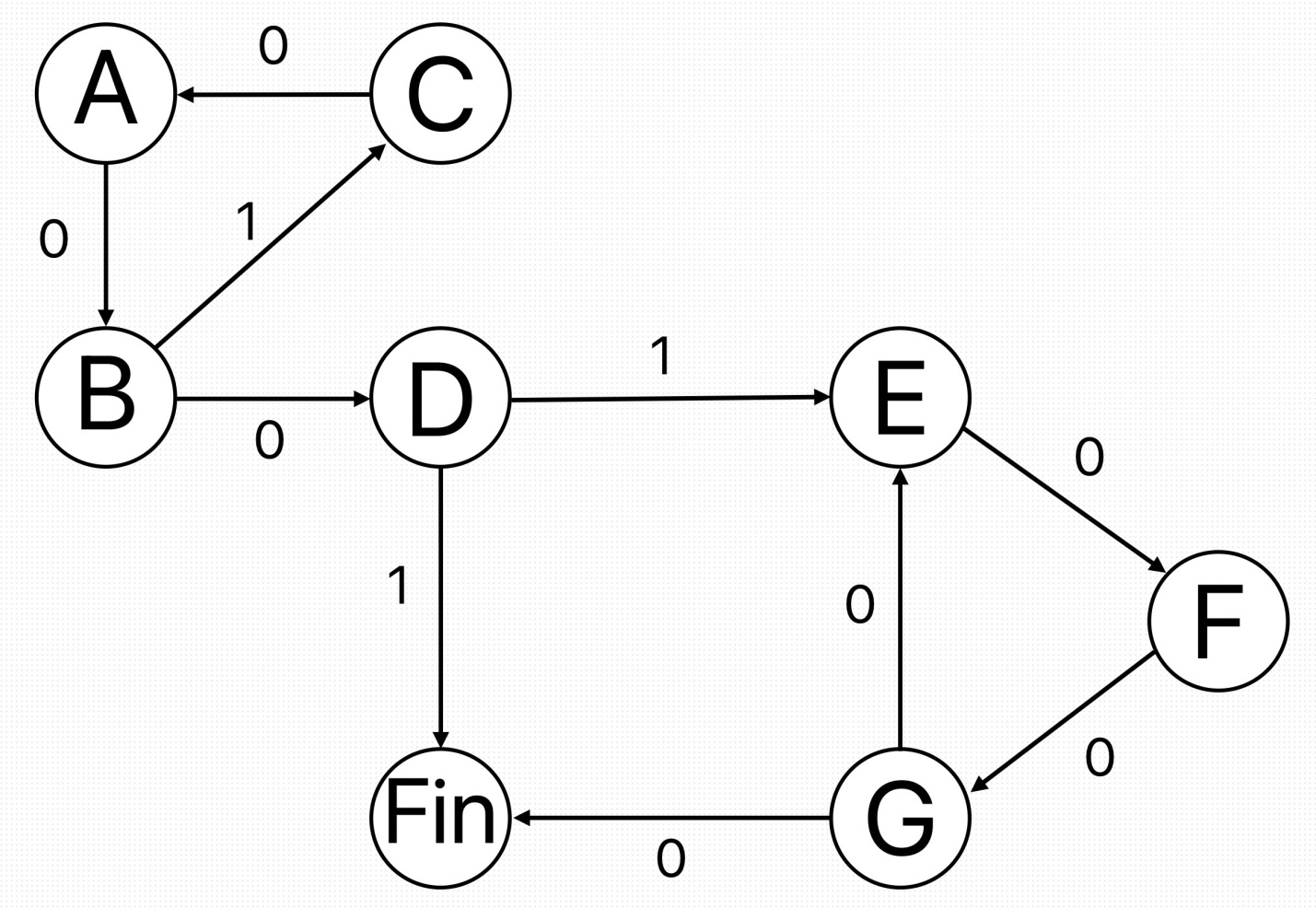
C → a | b | c | d | aC | bC | cC | dC

**Недетерминированная матрица:**

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

**Детерминированная матрица:**

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



**Краткое теоретическое обоснование:**

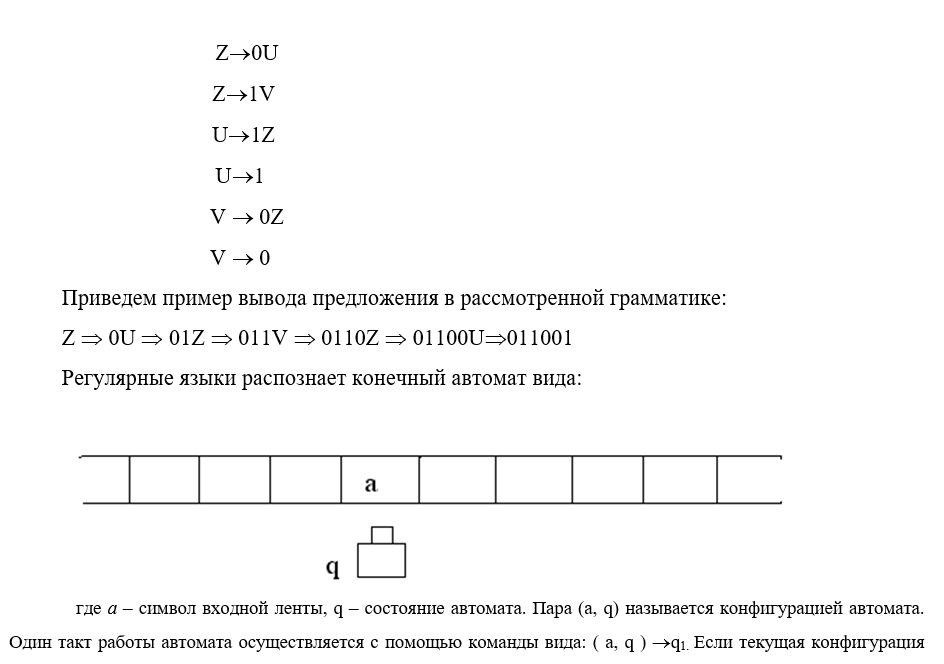
**Лексический анализатор** предназначен для чтения слов в исходном тексте и классификации прочитанных слов.

Основные функции лексического анализатора:

1. Чтение с помощью транслитератора очередного слова в исходном тексте и его классификация;
2. Пропуск пробелов и комментариев;
3. Выдача диагностических сообщений об обнаруженных лексических ошибках.

**Грамматикой Хомского типа 3, или регулярной грамматикой,** называется контекстно-свободная-грамматика, все правила которой односторонне линейны. Это означает, что либо они все праволинейны ,т.е имеют вид **A→ Ba** или **A→ а**; либо леволинейны , т.е имеют вид **А→aВ** или **A→ а**. Здесь **А** и **В** и обозначают нетерминальные символы, а через **а** обозначен терминальный символ.

Пример: язык L= { Rn | n > 0}, где R = {01, 10} задается регулярной грамматикой



где *а* – символ входной ленты, q – состояние автомата. Пара (a, q) называется конфигурацией автомата. Один такт работы автомата осуществляется с помощью команды вида: ( a, q ) →q1. Если текущая конфигурация автомата совпадает с левой частью одной из команд, то команда выполняется: символ входной ленты допускается, головка сдвигается на одну позицию вправо, автомат переходит в состояние q1 (вправо на 1 клетку). Среди команд обязательно должны присутствовать команды, в правой части которых находится финальное состояние. В финальном состоянии автомат может завершить работу с положительным результатом. Возможны также остановы с отрицательным результатом: автомат не может подобрать команду для текущей конфигурации.

**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;

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

namespace Tyap

{

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 Tyap

{

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

public enum TCharType

{

Letter, Digit, EndRow, EndText, Space, ReservedSymbol, opBracket, clBracket, expMark,

comma, semicolon, dobdot, plus, zv, min

}; // тип символа

public enum TToken

{

lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs,

lxmDot, lxmComma, lxmMinus, lxmplus, lxmzv, lxmravn, lxmdobdot, lxmVoskl

};

class CLex

{

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, для считывания очередного символа из входной строки и определения его типа

{

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.zv;

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 == ':') enumFSelectionCharType = TCharType.dobdot;

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

else if (chrFSelection == '-') enumFSelectionCharType = TCharType.min;

//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()// метод для добавления текущего символа

{

char[] c = { chrFSelection };

String s = new string(c);

strFLexicalUnit += s;

GetSymbol();

}

public void NextToken()// основной метод, реализующий лексический анализ. Разбирает входнуую строку на лексемы, определяет тип каждой лексемы и записывает в соответвующее поле класса

{

strFLexicalUnit = "";

if (enumFState == TState.Start)

{

intFSourceRowSelection = 0;

intFSourceColSelection = -1;

GetSymbol();

}

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

{

GetSymbol();

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

// Вариант 15

switch (enumFSelectionCharType)

{

case TCharType.Letter: // тип присваивается если элемент является буквой

{

// a b c d

// A | | | B | |

// B | |CFin| | |

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

A:

{

if (chrFSelection == 'c')

{

TakeSymbol();

goto B;

}

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

}

B:

{

if (chrFSelection == 'b')

{

TakeSymbol();

goto CFin;

}

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

}

CFin:

{

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

{

TakeSymbol();

goto CFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

}

case TCharType.Digit: // тип присваивается если элемент является цифрой

{

// 0 1

// A | B | |

// B | D | C |

// C | A | |

// D | |EFin |

// EFin | F | |

// F | G | |

// G | EFin| |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto BC;

}

else throw new Exception("Ожидался 0");

BC:

if (chrFSelection == '0')

{

TakeSymbol();

goto E;

}

if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

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

D:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидался 0");

E:

if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидалась 1");

FFin:

if (chrFSelection == '0')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидалась 0");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 0");

H:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0");

}

case TCharType.ReservedSymbol: // этот тип присваивается служебным символам например :;!() и тд

{

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

break;

}

case TCharType.EndText: // конец текста

{

enumFToken = TToken.lxmEmpty;

break;

}

case TCharType.Space: // пробел

{

GetSymbol();

break;

}

case TCharType.opBracket: // открывающая скобка

{

enumFToken = TToken.lxmLeftParenth;

GetSymbol();

break;

}

case TCharType.clBracket: // закрывающая скобка

{

enumFToken = TToken.lxmRightParenth;

GetSymbol();

break;

}

case TCharType.expMark: //восклицательный знак

{

enumFToken = TToken.lxmVoskl;

GetSymbol();

break;

}

case TCharType.plus: // плюс

{

enumFToken = TToken.lxmplus;

GetSymbol();

break;

}

case TCharType.dobdot: // двоеточие

{

enumFToken = TToken.lxmdobdot;

GetSymbol();

break;

}

case TCharType.zv: // звездочка \*

{

enumFToken = TToken.lxmzv;

GetSymbol();

break;

}

case TCharType.min: // минус

{

enumFToken = TToken.lxmMinus;

GetSymbol();

break;

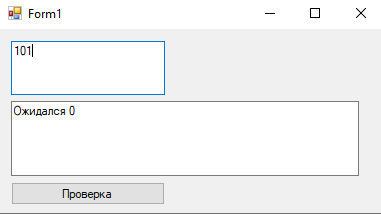
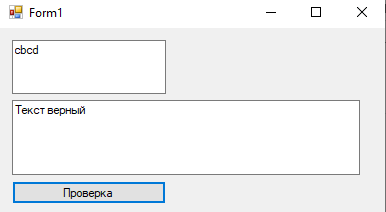
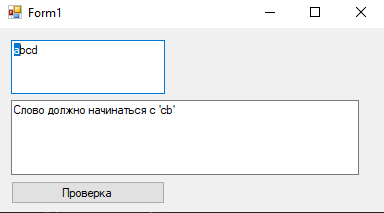
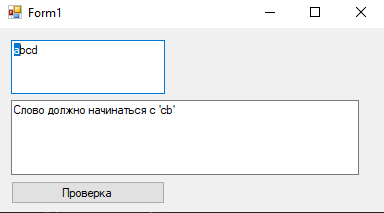
}

}

}

}

}

****

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

**Задание.**

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 или поля класса или метода

класса в синтаксический анализатор.

2. Оформить синтаксический анализатор в виде процедуры или функции или класса, которые при

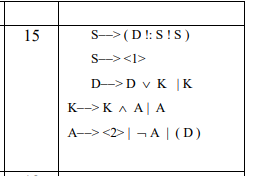
обращении обрабатывают весь исходный текст.

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:**



**Краткое теоретическое обоснование:**

Разработка контекстно-свободного (КС) синтаксического анализатора - это процесс создания программного инструмента, способного анализировать структуру текстового или программного кода на основе контекстно-свободной грамматики. Контекстно-свободная грамматика (КС-грамматика) — это формальное описание языка, которое определяет его синтаксическую структуру в виде набора правил.

Важные компоненты разработки КС-синтаксического анализатора включают в себя следующие этапы и концепции:

**Грамматика:** Разработка КС-синтаксического анализатора начинается с определения КС-грамматики для языка, который вы хотите анализировать. Грамматика включает в себя терминальные и нетерминальные символы, правила продукции, стартовый символ и другие сведения о структуре языка.

**Терминальные и нетерминальные символы:** Терминальные символы представляют лексемы (токены) языка, такие как ключевые слова, операторы и идентификаторы. Нетерминальные символы представляют собой абстрактные конструкции языка, которые могут быть разложены на более простые элементы с помощью правил продукции.

**Правила продукции:** Правила продукции определяют, как нетерминальные символы могут быть заменены другими символами (терминальными или нетерминальными) в контексте грамматики. Эти правила определяют синтаксическую структуру языка.

**Стартовый символ:** Стартовый символ — это нетерминальный символ, с которого начинается синтаксический анализ текста. Обычно это самый верхний уровень абстракции в грамматике.

**Токенизация:** прежде чем начать синтаксический анализ, текст обычно проходит процесс токенизации, в ходе которого он разбивается на отдельные лексемы (токены), например, разделяя ключевые слова и операторы.

**Алгоритмы синтаксического анализа:** существует несколько методов синтаксического анализа для обработки текста на основе КС-грамматики. Наиболее распространенными методами являются LL(к) и LR(к) анализ. Каждый из них имеет свои преимущества и недостатки, и выбор зависит от конкретных требований проекта.

**Генерация синтаксического анализатора:** Синтаксический анализатор может быть создан вручную, но также существуют инструменты, способные автоматически генерировать анализаторы на основе заданной КС-грамматики, такие как YACC, Bison, ANTLR и другие.

**Обработка синтаксического дерева:** после успешного синтаксического анализа текста обычно строится синтаксическое дерево, которое представляет собой структуру данных, отражающую иерархическую структуру текста. Это дерево затем может быть использовано для выполнения дополнительных анализов или для генерации кода.

Разработка КС-синтаксического анализатора является важным этапом при создании компиляторов, интерпретаторов, сред разработки и других инструментов обработки текста. Этот процесс требует хорошего понимания теории формальных языков и компьютерных наук, а также опыта в программировании.

Разработка контекстно-свободного (КС) синтаксического анализатора - это сложный процесс, который включает в себя несколько этапов и требует тщательного планирования и проектирования. Давайте более подробно рассмотрим некоторые из ключевых аспектов этого процесса:

**Определение целей и требований:** прежде чем начать разработку КС-синтаксического анализатора, необходимо четко определить его цели и требования. Это включает в себя понимание языка, который анализатор будет обрабатывать, и конечных целей, таких как компиляция кода, интерпретация или структурный анализ.

**Выбор языка программирования:** Выбор языка программирования для реализации синтаксического анализатора имеет значение. Часто используются языки с богатой поддержкой работы со строками и регулярными выражениями, такие как Python, Java, C++ или другие.

**Определение КС-грамматики:** Создайте КС-грамматику, которая описывает синтаксис целевого языка. Грамматика должна быть формально определена и документирована. Можно использовать формальные спецификации, такие как Backus-Naur Form (BNF) или Extended Backus-Naur Form (EBNF), для представления грамматики.

**Разработка лексического анализатора:** Часто КС-синтаксический анализатор включает в себя и лексический анализатор, который разбивает входной текст на лексемы (токены). Лексический анализатор может быть разработан отдельно или встроен в синтаксический анализатор.

**Выбор метода синтаксического анализа:** Выберите подходящий метод синтаксического анализа, такой как LL(к), LR(к), или другие, в зависимости от сложности грамматики и требований к анализу. Этот выбор может потребовать тщательного анализа грамматики и оценки производительности анализатора.

**Разработка и отладка анализатора:** на этом этапе происходит фактическая реализация синтаксического анализатора с использованием выбранного метода анализа. Здесь также проводятся тесты и отладка для обеспечения корректной работы.

**Обработка ошибок:** Разработайте механизм обработки синтаксических ошибок, чтобы предоставить информативные сообщения об ошибках во входном тексте.

**Интеграция с другими компонентами:** если синтаксический анализатор является частью компилятора или интерпретатора, необходимо интегрировать его с другими компонентами, такими как лексический анализатор, семантический анализ и генератор кода.

**Оптимизация и улучшение производительности:** Проведите профилирование и оптимизацию синтаксического анализатора, чтобы улучшить его производительность и эффективность.

**Документация и тестирование:** Документируйте код синтаксического анализатора и проведите обширное тестирование, включая функциональное тестирование и тестирование производительности.

Разработка КС-синтаксического анализатора — это сложный и важный процесс, который требует глубокого понимания теории формальных языков, алгоритмов анализа и практических навыков в программировании. Успешно разработанный анализатор способен обрабатывать и анализировать текст или код в соответствии с заданной грамматикой и помогать в создании высококачественных инструментов для обработки текста.

**Код программы**

**USyntAnalyzer.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Tyap

{

class uSyntAnalyzer

{

string s;

private String[] strFSource; // хронят исходный код и сообщения о ошибках

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();// содержит экземпляр лексического анализатора

public void S()// входная точка, запускает разбор. Проверяет наличие открывающих скобок, вызывает разбор выражения в D(), проверяет остальные токины

{

if (Lex.enumPToken == TToken.lxmLeftParenth)

{

Lex.NextToken();

s = Lex.enumPToken.ToString();

D();

if (Lex.enumPToken == TToken.lxmVoskl)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmdobdot)

{

Lex.NextToken();

S();

if (Lex.enumPToken == TToken.lxmVoskl)

{

Lex.NextToken();

S();

if (Lex.enumPToken == TToken.lxmRightParenth)

{

Lex.NextToken();

}

else throw new Exception("Ожидалcя )");

}

}

else throw new Exception("Ожидалcя :");

}

else throw new Exception("Ожидалcя !");

}

else

if (Lex.enumPToken == TToken.lxmNumber)

{ Lex.NextToken(); }

else { throw new Exception("Ожидалcя ( или число"); }

}

public void D()// метод разбирает выражение: сначала один элемент в K() , потом возможно сложение элементов в С()

{

K();

if (Lex.enumPToken == TToken.lxmplus)

{

C();

}

}

public void C()//методы для разбора отдельных правил грамматики: сложение, константы/переменные, умножение, атомы. Рекурсивно вызывают друг друга.

{

if (Lex.enumPToken == TToken.lxmplus)

{

Lex.NextToken();

K();

if (Lex.enumPToken == TToken.lxmplus)

{

C();

}

}

}

public void K()//методы для разбора отдельных правил грамматики: сложение, константы/переменные, умножение, атомы. Рекурсивно вызывают друг друга.

{

A();

if (Lex.enumPToken == TToken.lxmzv)

{

P();

}

}

public void P()//методы для разбора отдельных правил грамматики: сложение, константы/переменные, умножение, атомы. Рекурсивно вызывают друг друга.

{

if (Lex.enumPToken == TToken.lxmzv)

{

Lex.NextToken();

A();

if (Lex.enumPToken == TToken.lxmzv)

{

P();

}

}

}

public void A()//методы для разбора отдельных правил грамматики: сложение, константы/переменные, умножение, атомы. Рекурсивно вызывают друг друга.

{

s = Lex.enumPToken.ToString();

if (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

}

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

{

Lex.NextToken();

D();

if (Lex.enumPToken == TToken.lxmRightParenth)

{

Lex.NextToken();

}

else throw new Exception("Ожидалась )");

}

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

{

Lex.NextToken();

A();

}

else

{

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

MessageBox.Show(s);

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

}

}

}

}

**Forn1.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;

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

namespace Tyap

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

textBox1.AppendText("(cba!:001!001)");

}

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

{

textBox2.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

Synt.Lex.strPSource = textBox1.Lines;

Synt.Lex.strPMessage = textBox2.Lines;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.S();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

textBox2.Text += exc.Message;

textBox1.Select();

textBox1.SelectionStart = 0;

int n = 0;

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

n += Synt.Lex.intPSourceColSelection;

textBox1.SelectionLength = n;

}

}

**uLex.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Tyap

{

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

public enum TCharType

{

Letter, Digit, EndRow, EndText, Space, ReservedSymbol, opBracket, clBracket, expMark,

comma, semicolon, dobdot, plus, zv, min

}; // тип символа

public enum TToken

{

lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs,

lxmDot, lxmComma, lxmMinus, lxmplus, lxmzv, lxmravn, lxmdobdot, lxmVoskl

};

class CLex

{

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.zv;

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 == ':') enumFSelectionCharType = TCharType.dobdot;

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

else if (chrFSelection == '-') enumFSelectionCharType = TCharType.min;

//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();

}

// Вариант 15

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// A | | | B | |

// B | |CFin| | |

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

A:

{

if (chrFSelection == 'c')

{

TakeSymbol();

goto B;

}

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

}

B:

{

if (chrFSelection == 'b')

{

TakeSymbol();

goto CFin;

}

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

}

CFin:

{

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

{

TakeSymbol();

goto CFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

}

case TCharType.Digit:

{

// 0 1

// A | B | |

// B | D | C |

// C | A | |

// D | |EFin |

// EFin | F | |

// F | G | |

// G | EFin| |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto BC;

}

else throw new Exception("Ожидался 0");

BC:

if (chrFSelection == '0')

{

TakeSymbol();

goto E;

}

if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

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

D:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидался 0");

E:

if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидалась 1");

FFin:

if (chrFSelection == '0')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидалась 0");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 0");

H:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0");

}

case TCharType.ReservedSymbol:

{

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

case TCharType.Space:

{

GetSymbol();

break;

}

case TCharType.opBracket:

{

enumFToken = TToken.lxmLeftParenth;

GetSymbol();

break;

}

case TCharType.clBracket:

{

enumFToken = TToken.lxmRightParenth;

GetSymbol();

break;

}

case TCharType.expMark:

{

enumFToken = TToken.lxmVoskl;

GetSymbol();

break;

}

case TCharType.plus:

{

enumFToken = TToken.lxmplus;

GetSymbol();

break;

}

case TCharType.dobdot:

{

enumFToken = TToken.lxmdobdot;

GetSymbol();

break;

}

case TCharType.zv:

{

enumFToken = TToken.lxmzv;

GetSymbol();

break;

}

case TCharType.min:

{

enumFToken = TToken.lxmMinus;

GetSymbol();

break;

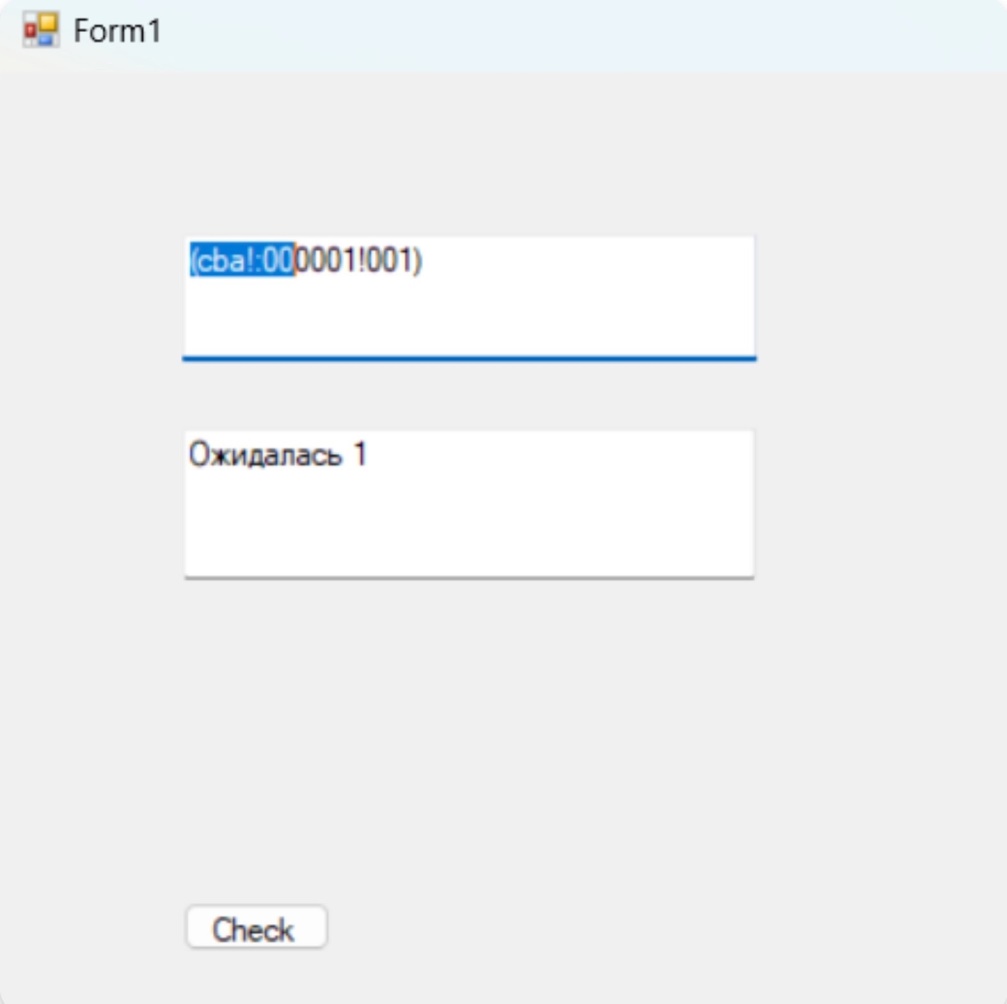
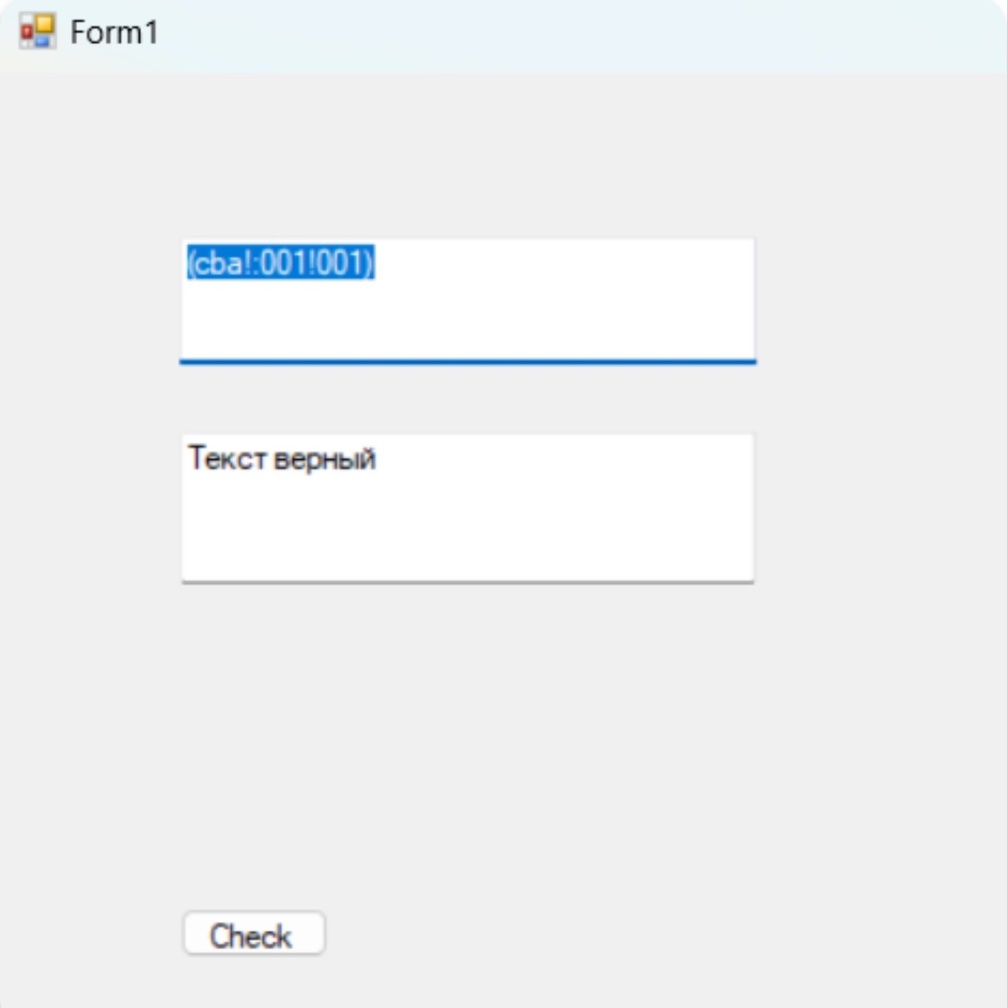
}

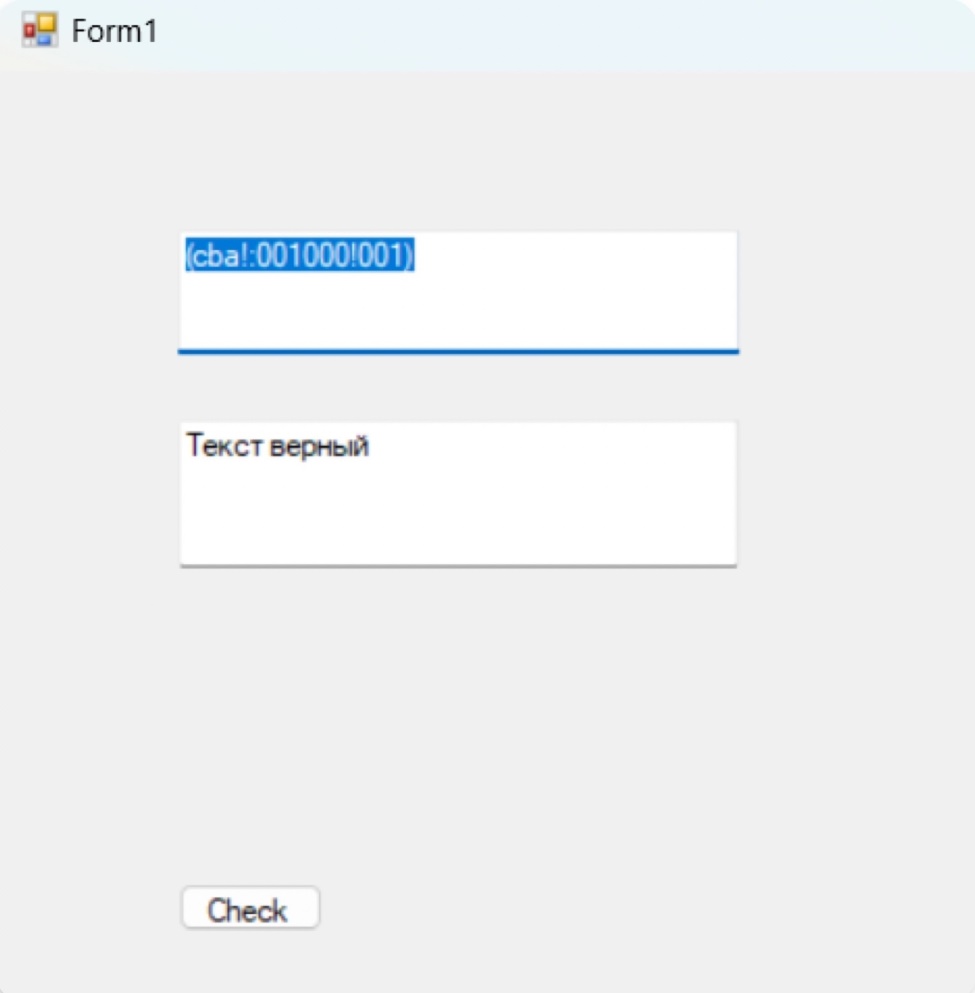
}

}

}

}





**Лабораторная работа №4**

**Введение табличного способа хранения слов**

Таблица символов (слов) – это структура данных элементов с ключами, которая поддерживает две базовые операции: вставку нового элемента в таблицу и поиск заданного элемента в таблице. В области трансляторов часто таблицы слов называют таблицами символов, иногда таблицы символов называют также словарями. Сложно устроенные словари называются тезаурусами. Во многих приложениях данные в таблицах символов хранятся вместе с их ключами. Примером могут служить базы данных.

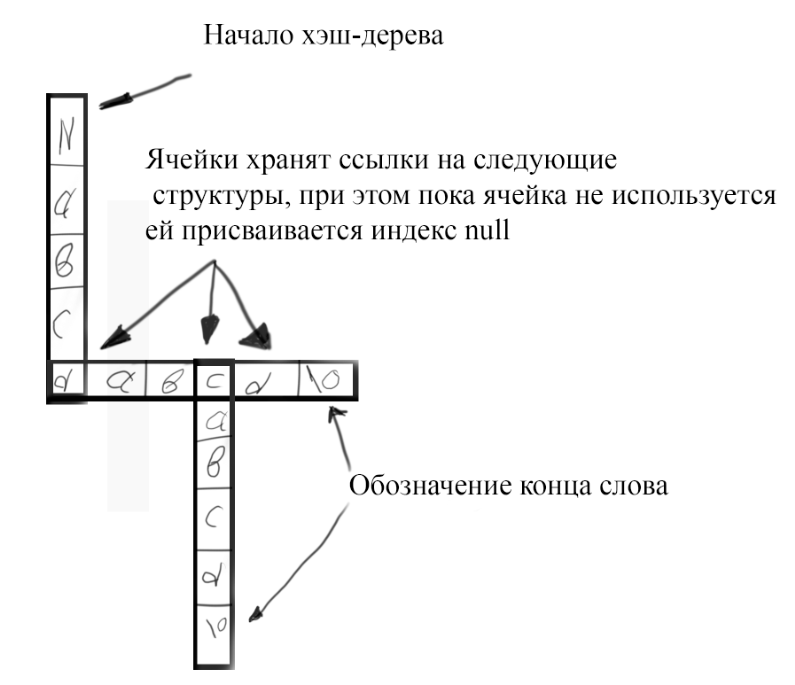
Виды таблиц и их количество в трансляторе определяются программистом исходя из особенностей входного языка и алгоритмов трансляции. В трансляторах популярных языков программирования создается до 30 таблиц слов. Приведем примеры традиционных таблиц транслятора:

* Таблица идентификаторов переменных целого типа;
* Таблица целых констант;
* Таблица служебных слов языка;
* Таблица разделителей;
* Таблица имен функций;
* Таблица встроенных типов данных;
* Таблица арифметических операций и ряд других.

Правило языков программирования «определение идентификатора должно предшествовать использованию идентификатора» появилось из требований технологии трансляции. Обычно операции вставки нового элемента и поиск элемента именуют кратко «поиск слова в таблице», т.к. операции вставки нового слова предшествует операция поиска слова, а сама вставка тривиальна.

Существует три основных алгоритма решения задачи поиска слова в таблице: алгоритм последовательного поиска (алгоритм перебора), алгоритм бинарного поиска и алгоритм хеш-поиска.

В данной работе была использована модифицированная хэш-таблица, которая сочетает в себе хэш-таблицу и списковую структуру-хэш-дерево.



**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 Tyap

{

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] % 100 + 100);

}

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 + i) % (Int32)(arrFHashTable.Count);

}

} 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)

{

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);

}

}

}

**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;

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

namespace Tyap

{

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();

textBox1.AppendText("(cba!:001!001)");

}

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

{

textBox2.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

Synt.Lex.strPSource = textBox1.Lines;

Synt.Lex.strPMessage = textBox2.Lines;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.S();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

textBox2.Text += exc.Message;

textBox1.Select();

textBox1.SelectionStart = 0;

int n = 0;

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

n += Synt.Lex.intPSourceColSelection;

textBox1.SelectionLength = n;

}

}

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

{

CLex Lex = new CLex();

Lex.strPSource = textBox1.Lines;

Lex.strPMessage = textBox2.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = textBox1.TextLength;

int y = textBox1.Lines.Length;

textBox2.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

s1 = "id " + Lex.strPLexicalUnit;

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 0);

// htl.DeleteLexicalCode(0);

TablesToMemo(this, e);

break;

}

case TToken.lxmNumber:

{

s1 = "num " + Lex.strPLexicalUnit;

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 1);

// htl.DeleteLexicalCode(0);

TablesToMemo(this, e);

break;

}

}

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

textBox2.Text += m;

}

}

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 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)

{

case TToken.lxmIdentifier:

{

int b = 0;

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

{

while (Lex2.enumPState != TState.Finish)

{

Lex2.NextToken();

switch (Lex2.enumPToken)

{

case TToken.lxmIdentifier:

{

int b1 = 0;

if (htl.AddLexicalUnit(Lex2.strPLexicalUnit, 0, ref b1))

{

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 0);

TablesToMemo(this, e);

}

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

break;

}

case TToken.lxmNumber:

{

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

break;

}

}

}

}

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

break;

}

case TToken.lxmNumber:

{

int b = 0;

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

{

while (Lex2.enumPState != TState.Finish)

{

Lex2.NextToken();

switch (Lex2.enumPToken)

{

case TToken.lxmIdentifier:

{

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

break;

}

case TToken.lxmNumber:

{

int b1 = 0;

if (htl.AddLexicalUnit(Lex2.strPLexicalUnit, 1, ref b1))

{

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 1);

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;

textBox1.Select();

textBox1.SelectionStart = 0;

int n1 = 0;

for (int i = 0; i < Lex.intPSourceRowSelection; i++) n1 += textBox1.Lines[i].Length + 2;

n1 += Lex.intPSourceColSelection;

textBox1.SelectionLength = n1;

}

}

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

{

CLex Lex = new CLex();

Lex.strPSource = textBox1.Lines;

Lex.strPMessage = textBox2.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = textBox1.TextLength;

int y = textBox1.Lines.Length;

textBox2.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

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

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

{

TablesToMemo(this, e);

}

break;

}

}

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

textBox2.Text += m;

}

}

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 button3\_Click(object sender, EventArgs e)

{

tbFMessage2.Clear();

CLex Lex = new CLex();

Lex.strPSource = textBox1.Lines;

Lex.strPMessage = textBox2.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = textBox1.TextLength;

int y = textBox1.Lines.Length;

textBox2.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))

{

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))

{

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

}

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

break;

}

}

}

}

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;

}

}

}

}

\

**Form1.Designer.cs**

namespace Tyap

{

partial class Form1

{

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

private System.ComponentModel.IContainer components = null;

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

/// <param name="disposing">истинно, если управляемый ресурс должен быть удален; иначе ложно.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Код, автоматически созданный конструктором форм Windows

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

private void InitializeComponent()

{

this.textBox1 = new System.Windows.Forms.TextBox();

this.textBox2 = new System.Windows.Forms.TextBox();

this.button1 = new System.Windows.Forms.Button();

this.tbFMessage2 = new System.Windows.Forms.TextBox();

this.button2 = new System.Windows.Forms.Button();

this.button3 = new System.Windows.Forms.Button();

this.button4 = new System.Windows.Forms.Button();

this.button5 = new System.Windows.Forms.Button();

this.listBox1 = new System.Windows.Forms.ListBox();

this.listBox2 = new System.Windows.Forms.ListBox();

this.label1 = new System.Windows.Forms.Label();

this.label2 = new System.Windows.Forms.Label();

this.tbFMessage1 = new System.Windows.Forms.TextBox();

this.tbFSource1 = new System.Windows.Forms.TextBox();

this.tbFSource2 = new System.Windows.Forms.TextBox();

this.SuspendLayout();

//

// textBox1

//

this.textBox1.Location = new System.Drawing.Point(69, 61);

this.textBox1.Margin = new System.Windows.Forms.Padding(2);

this.textBox1.Multiline = true;

this.textBox1.Name = "textBox1";

this.textBox1.Size = new System.Drawing.Size(216, 48);

this.textBox1.TabIndex = 0;

//

// textBox2

//

this.textBox2.Location = new System.Drawing.Point(69, 134);

this.textBox2.Margin = new System.Windows.Forms.Padding(2);

this.textBox2.Multiline = true;

this.textBox2.Name = "textBox2";

this.textBox2.Size = new System.Drawing.Size(216, 57);

this.textBox2.TabIndex = 1;

//

// button1

//

this.button1.Location = new System.Drawing.Point(69, 312);

this.button1.Margin = new System.Windows.Forms.Padding(2);

this.button1.Name = "button1";

this.button1.Size = new System.Drawing.Size(56, 19);

this.button1.TabIndex = 2;

this.button1.Text = "Check";

this.button1.UseVisualStyleBackColor = true;

this.button1.Click += new System.EventHandler(this.button1\_Click);

//

// tbFMessage2

//

this.tbFMessage2.Location = new System.Drawing.Point(69, 210);

this.tbFMessage2.Multiline = true;

this.tbFMessage2.Name = "tbFMessage2";

this.tbFMessage2.Size = new System.Drawing.Size(216, 37);

this.tbFMessage2.TabIndex = 3;

//

// button2

//

this.button2.Location = new System.Drawing.Point(48, 377);

this.button2.Name = "button2";

this.button2.Size = new System.Drawing.Size(75, 23);

this.button2.TabIndex = 4;

this.button2.Text = "Dobavit";

this.button2.UseVisualStyleBackColor = true;

this.button2.Click += new System.EventHandler(this.button2\_Click);

//

// button3

//

this.button3.Location = new System.Drawing.Point(129, 377);

this.button3.Name = "button3";

this.button3.Size = new System.Drawing.Size(75, 23);

this.button3.TabIndex = 5;

this.button3.Text = "Naity";

this.button3.UseVisualStyleBackColor = true;

this.button3.Click += new System.EventHandler(this.button3\_Click);

//

// button4

//

this.button4.Location = new System.Drawing.Point(210, 377);

this.button4.Name = "button4";

this.button4.Size = new System.Drawing.Size(75, 23);

this.button4.TabIndex = 6;

this.button4.Text = "Udalit";

this.button4.UseVisualStyleBackColor = true;

this.button4.Click += new System.EventHandler(this.button4\_Click);

//

// button5

//

this.button5.Location = new System.Drawing.Point(441, 70);

this.button5.Name = "button5";

this.button5.Size = new System.Drawing.Size(75, 23);

this.button5.TabIndex = 7;

this.button5.Text = "Zamena";

this.button5.UseVisualStyleBackColor = true;

this.button5.Click += new System.EventHandler(this.button5\_Click);

//

// listBox1

//

this.listBox1.FormattingEnabled = true;

this.listBox1.Location = new System.Drawing.Point(509, 338);

this.listBox1.Name = "listBox1";

this.listBox1.Size = new System.Drawing.Size(120, 212);

this.listBox1.TabIndex = 10;

//

// listBox2

//

this.listBox2.FormattingEnabled = true;

this.listBox2.Location = new System.Drawing.Point(681, 338);

this.listBox2.Name = "listBox2";

this.listBox2.Size = new System.Drawing.Size(120, 212);

this.listBox2.TabIndex = 11;

//

// label1

//

this.label1.AutoSize = true;

this.label1.Location = new System.Drawing.Point(509, 319);

this.label1.Name = "label1";

this.label1.Size = new System.Drawing.Size(36, 13);

this.label1.TabIndex = 12;

this.label1.Text = "word1";

//

// label2

//

this.label2.AutoSize = true;

this.label2.Location = new System.Drawing.Point(681, 318);

this.label2.Name = "label2";

this.label2.Size = new System.Drawing.Size(36, 13);

this.label2.TabIndex = 13;

this.label2.Text = "word2";

//

// tbFMessage1

//

this.tbFMessage1.Location = new System.Drawing.Point(416, 134);

this.tbFMessage1.Multiline = true;

this.tbFMessage1.Name = "tbFMessage1";

this.tbFMessage1.Size = new System.Drawing.Size(144, 77);

this.tbFMessage1.TabIndex = 14;

//

// tbFSource1

//

this.tbFSource1.Location = new System.Drawing.Point(681, 109);

this.tbFSource1.Multiline = true;

this.tbFSource1.Name = "tbFSource1";

this.tbFSource1.Size = new System.Drawing.Size(137, 20);

this.tbFSource1.TabIndex = 15;

//

// tbFSource2

//

this.tbFSource2.Location = new System.Drawing.Point(681, 170);

this.tbFSource2.Multiline = true;

this.tbFSource2.Name = "tbFSource2";

this.tbFSource2.Size = new System.Drawing.Size(137, 20);

this.tbFSource2.TabIndex = 16;

//

// Form1

//

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.ClientSize = new System.Drawing.Size(844, 573);

this.Controls.Add(this.tbFSource2);

this.Controls.Add(this.tbFSource1);

this.Controls.Add(this.tbFMessage1);

this.Controls.Add(this.label2);

this.Controls.Add(this.label1);

this.Controls.Add(this.listBox2);

this.Controls.Add(this.listBox1);

this.Controls.Add(this.button5);

this.Controls.Add(this.button4);

this.Controls.Add(this.button3);

this.Controls.Add(this.button2);

this.Controls.Add(this.tbFMessage2);

this.Controls.Add(this.button1);

this.Controls.Add(this.textBox2);

this.Controls.Add(this.textBox1);

this.Margin = new System.Windows.Forms.Padding(2);

this.Name = "Form1";

this.Text = "Form1";

this.ResumeLayout(false);

this.PerformLayout();

}

#endregion

private System.Windows.Forms.TextBox textBox1;

private System.Windows.Forms.TextBox textBox2;

private System.Windows.Forms.Button button1;

private System.Windows.Forms.TextBox tbFMessage2;

private System.Windows.Forms.Button button2;

private System.Windows.Forms.Button button3;

private System.Windows.Forms.Button button4;

private System.Windows.Forms.Button button5;

private System.Windows.Forms.ListBox listBox1;

private System.Windows.Forms.ListBox listBox2;

private System.Windows.Forms.Label label1;

private System.Windows.Forms.Label label2;

private System.Windows.Forms.TextBox tbFMessage1;

private System.Windows.Forms.TextBox tbFSource1;

private System.Windows.Forms.TextBox tbFSource2;

}

}

**uLex.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Tyap

{

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

public enum TCharType

{

Letter, Digit, EndRow, EndText, Space, ReservedSymbol, opBracket, clBracket, expMark,

comma, semicolon, dobdot, plus, zv, min

}; // тип символа

public enum TToken

{

lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs,

lxmDot, lxmComma, lxmMinus, lxmplus, lxmzv, lxmravn, lxmdobdot, lxmVoskl

};

class CLex

{

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.zv;

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 == ':') enumFSelectionCharType = TCharType.dobdot;

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

else if (chrFSelection == '-') enumFSelectionCharType = TCharType.min;

//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();

}

// Вариант 15

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// A | | | B | |

// B | |CFin| | |

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

A:

{

if (chrFSelection == 'c')

{

TakeSymbol();

goto B;

}

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

}

B:

{

if (chrFSelection == 'b')

{

TakeSymbol();

goto CFin;

}

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

}

CFin:

{

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

{

TakeSymbol();

goto CFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

}

case TCharType.Digit:

{

// 0 1

// A | B | |

// B | D | C |

// C | A | |

// D | |EFin |

// EFin | F | |

// F | G | |

// G | EFin| |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto BC;

}

else throw new Exception("Ожидался 0");

BC:

if (chrFSelection == '0')

{

TakeSymbol();

goto E;

}

if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

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

D:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидался 0");

E:

if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидалась 1");

FFin:

if (chrFSelection == '0')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидалась 0");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 0");

H:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0");

}

case TCharType.ReservedSymbol:

{

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

case TCharType.Space:

{

GetSymbol();

break;

}

case TCharType.opBracket:

{

enumFToken = TToken.lxmLeftParenth;

GetSymbol();

break;

}

case TCharType.clBracket:

{

enumFToken = TToken.lxmRightParenth;

GetSymbol();

break;

}

case TCharType.expMark:

{

enumFToken = TToken.lxmVoskl;

GetSymbol();

break;

}

case TCharType.plus:

{

enumFToken = TToken.lxmplus;

GetSymbol();

break;

}

case TCharType.dobdot:

{

enumFToken = TToken.lxmdobdot;

GetSymbol();

break;

}

case TCharType.zv:

{

enumFToken = TToken.lxmzv;

GetSymbol();

break;

}

case TCharType.min:

{

enumFToken = TToken.lxmMinus;

GetSymbol();

break;

}

}

}

}

}

**uSyntAnalyzer.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Tyap

{

class uSyntAnalyzer

{

string s;

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()

{

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

//MessageBox.Show(s);

//Lex.NextToken();

// Lex.NextToken();

if (Lex.enumPToken == TToken.lxmLeftParenth)

{

Lex.NextToken();

s = Lex.enumPToken.ToString();

//MessageBox.Show(s);

//MessageBox.Show("\_\_\_\_\_");

D();

if (Lex.enumPToken == TToken.lxmVoskl)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmdobdot)

{

Lex.NextToken();

S();

if (Lex.enumPToken == TToken.lxmVoskl)

{

Lex.NextToken();

S();

if (Lex.enumPToken == TToken.lxmRightParenth)

{

Lex.NextToken();

}

else throw new Exception("Ожидалcя )");

}

}

else throw new Exception("Ожидалcя :");

}

else throw new Exception("Ожидалcя !");

}

else

if (Lex.enumPToken == TToken.lxmNumber)

{ Lex.NextToken(); }

else { throw new Exception("Ожидалcя ( или число"); }

}

public void D()

{

K();

if (Lex.enumPToken == TToken.lxmplus)

{

//Lex.NextToken();

C();

}

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

}

public void C()

{

if (Lex.enumPToken == TToken.lxmplus)

{

Lex.NextToken();

K();

if (Lex.enumPToken == TToken.lxmplus)

{

//Lex.NextToken();

C();

}

}

}

public void K()

{

A();

//Lex.NextToken();

if (Lex.enumPToken == TToken.lxmzv)

{

P();

}

}

public void P()

{

if (Lex.enumPToken == TToken.lxmzv)

{

Lex.NextToken();

A();

//Lex.NextToken();

if (Lex.enumPToken == TToken.lxmzv)

{

P();

}

}

}

public void A()

{

s = Lex.enumPToken.ToString();

//MessageBox.Show(s);

//MessageBox.Show("AAAAAAAAAAA");

if (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

}

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

{

Lex.NextToken();

D();

//Lex.NextToken();

if (Lex.enumPToken == TToken.lxmRightParenth)

{

Lex.NextToken();

}

else throw new Exception("Ожидалась )");

}

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

{

Lex.NextToken();

A();

}

else

{

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

MessageBox.Show(s);

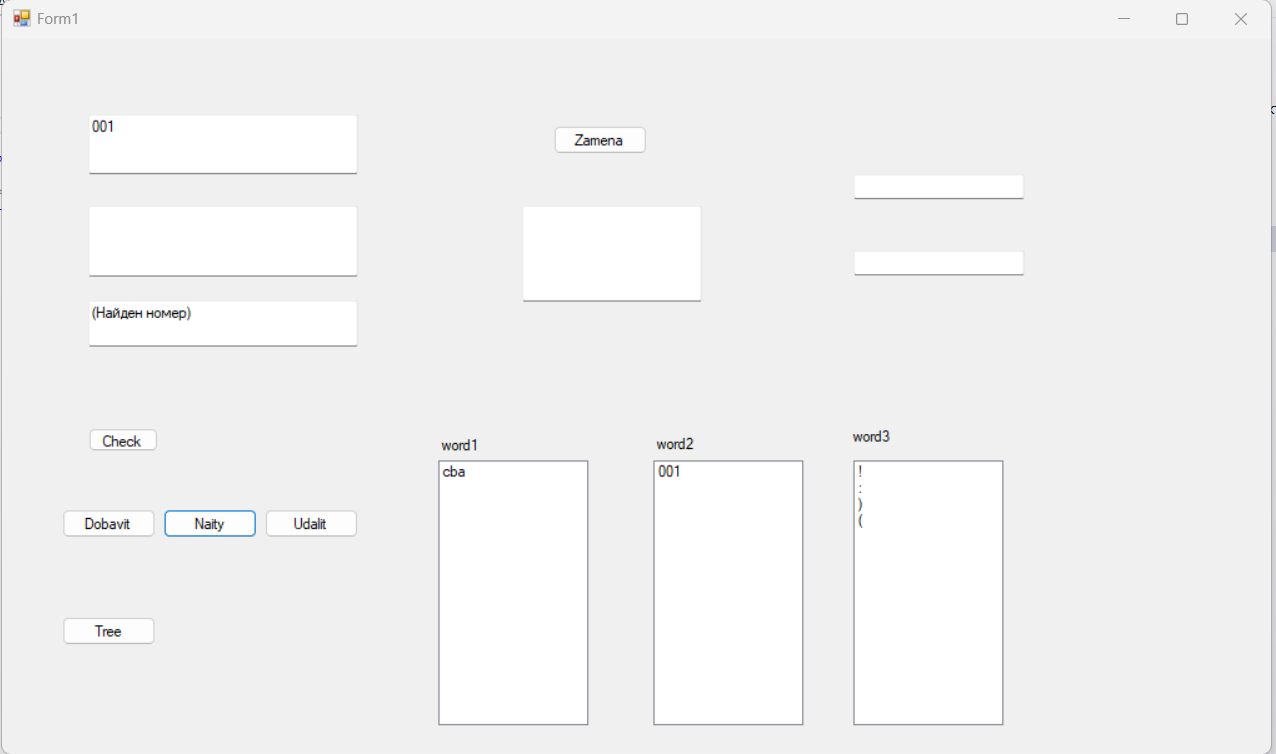
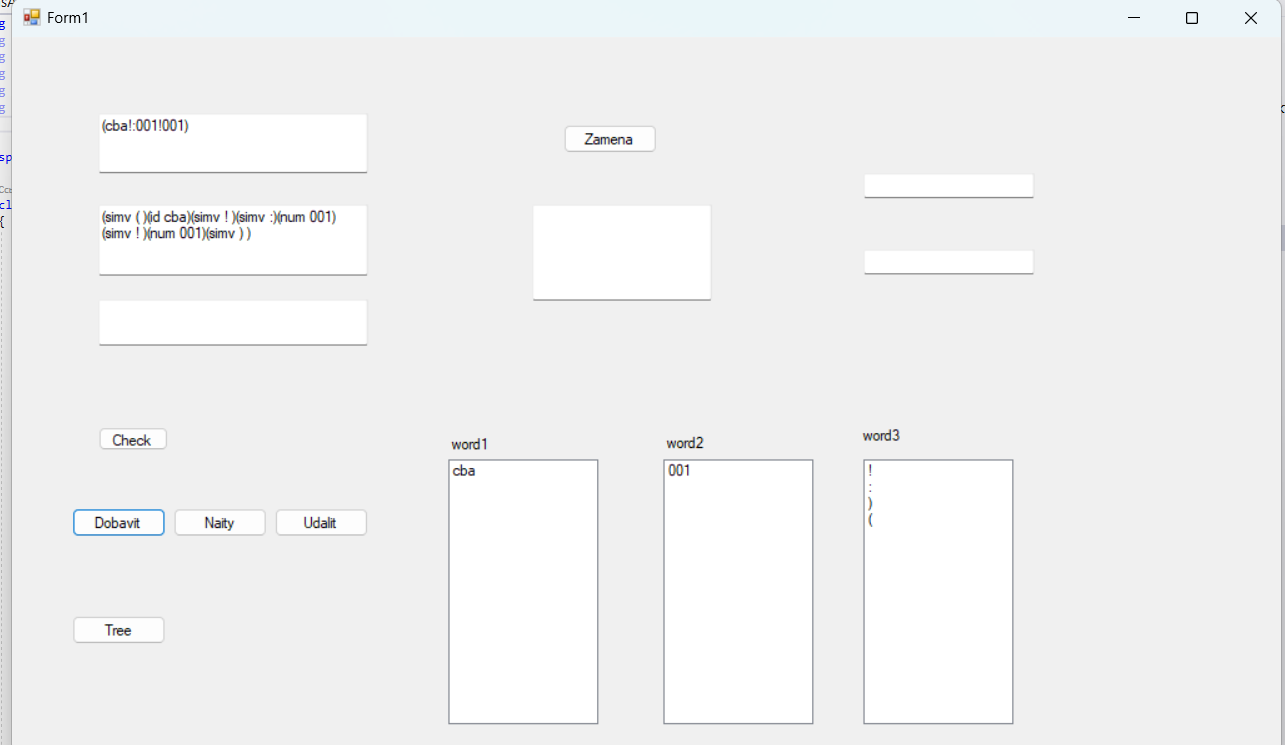
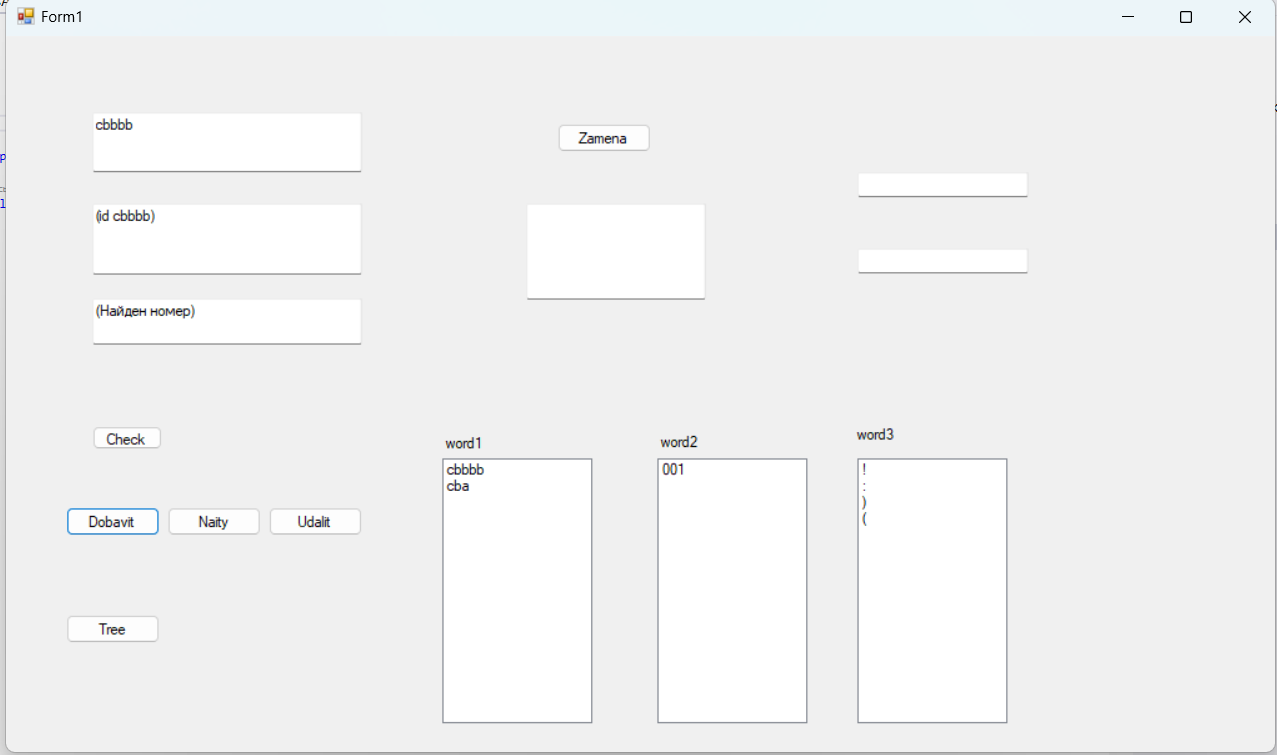
throw new Exception("Ожидались - или ( или слово");

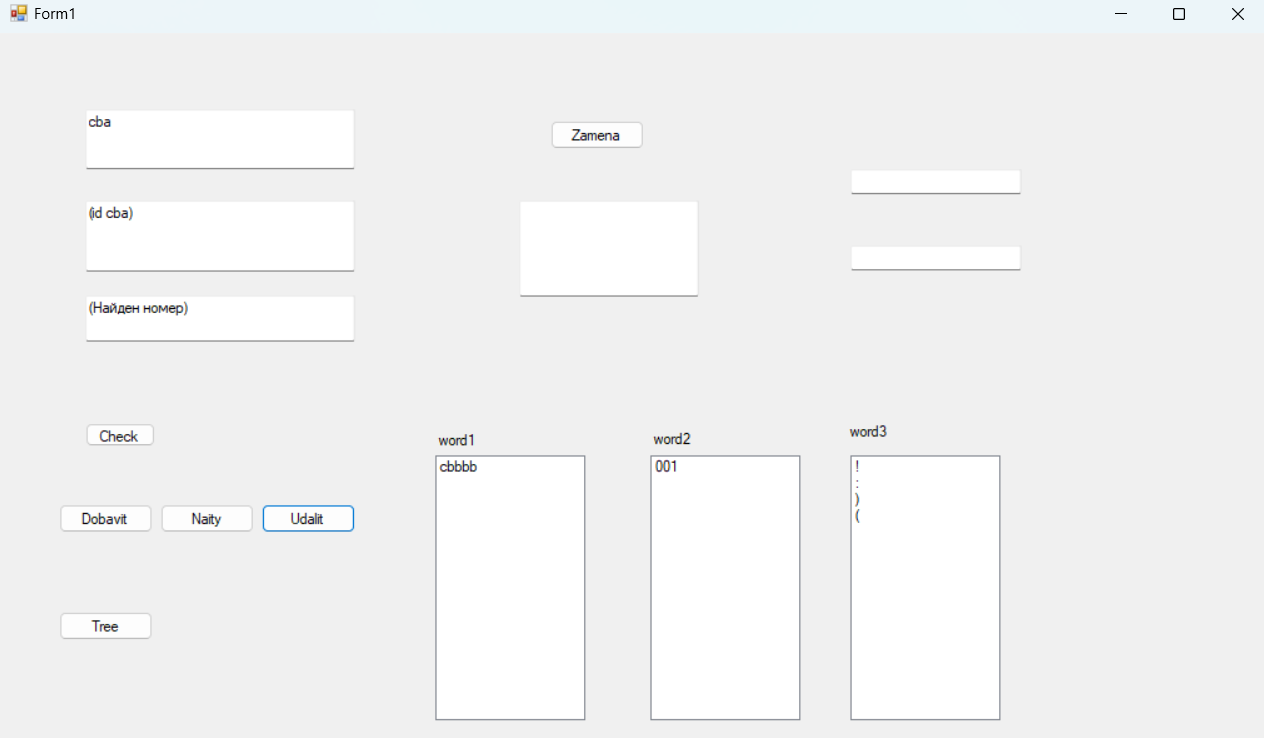
}

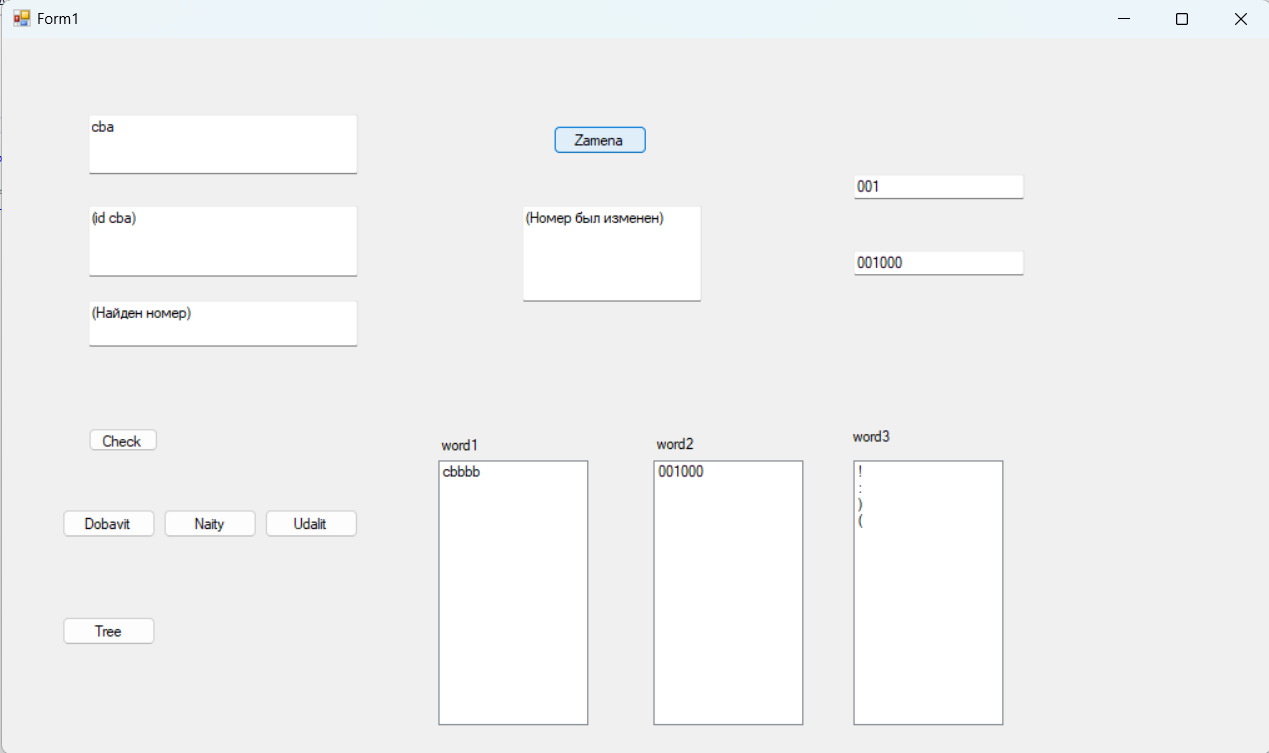
}

}

}





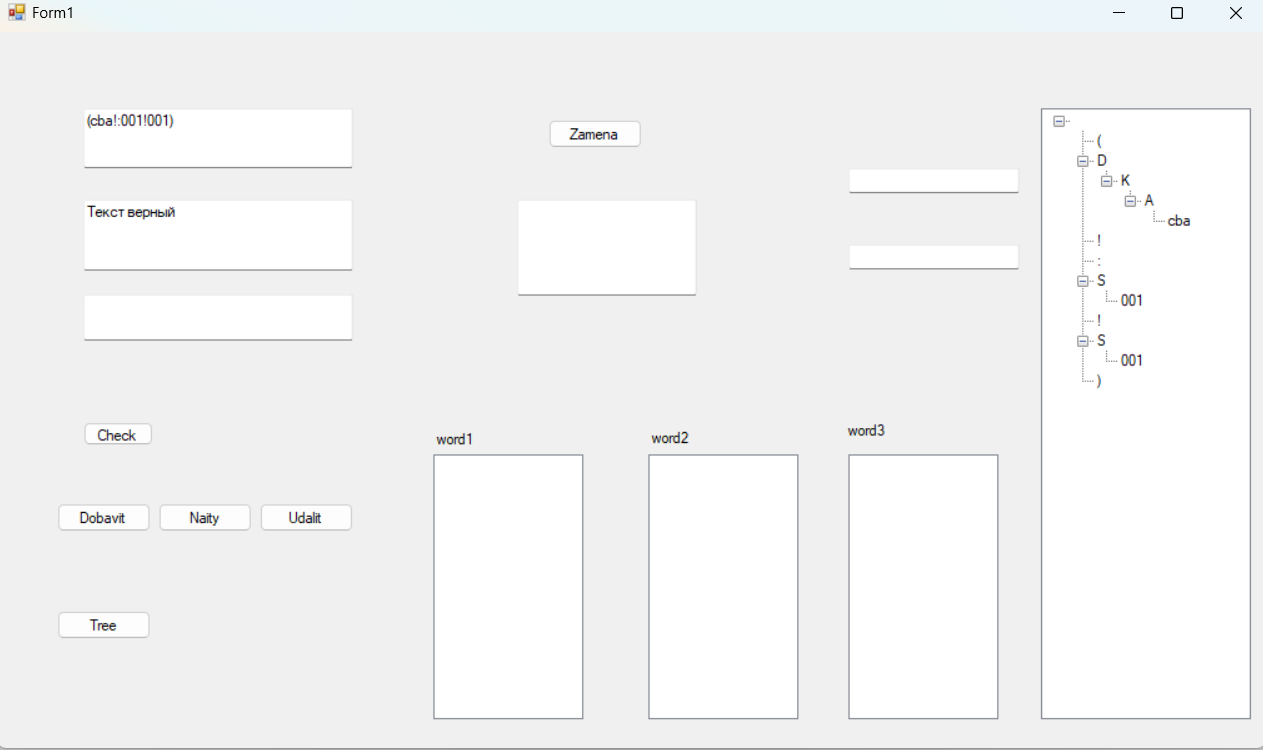
**Лабораторная работа № 5. Построение синтаксического дерева**

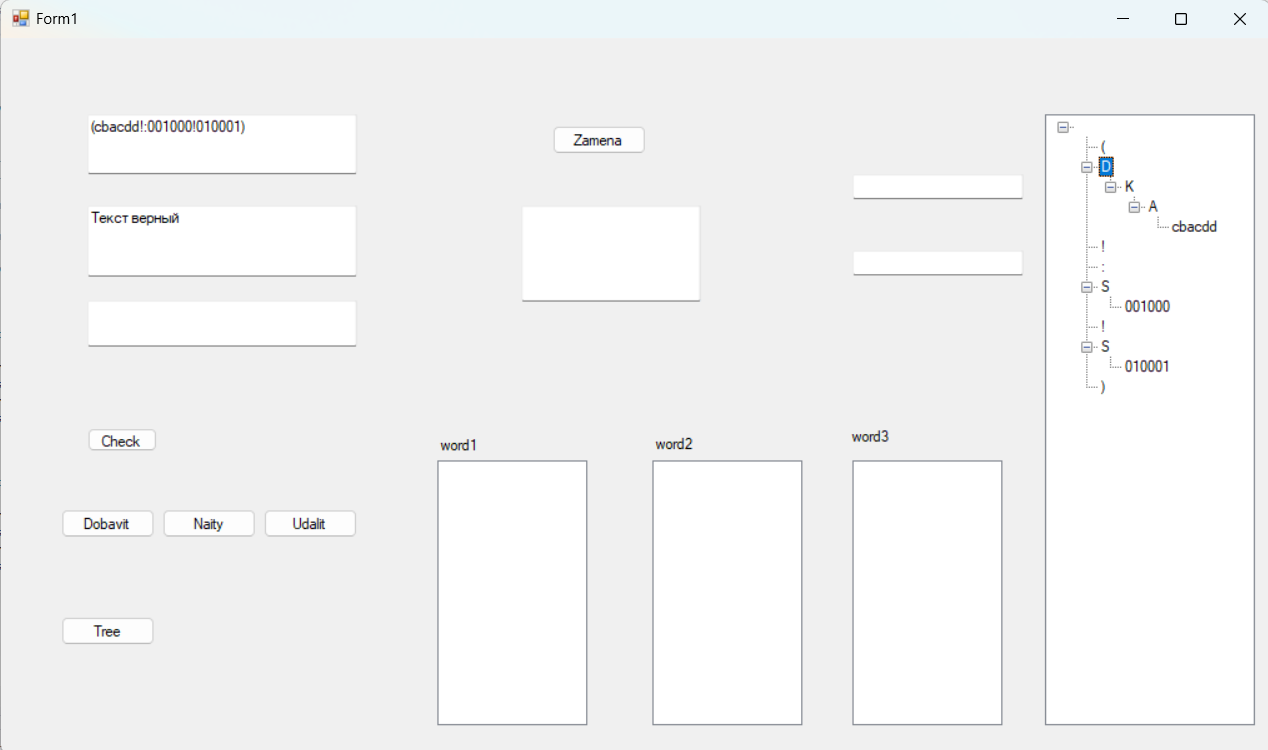
Теория:

В практике трансляции часто возникает необходимость не просто выяснить грамматическую правильность текста, а осуществляя разбор, извлечь какие-то данные или произвести какие-то действия. При этом предполагается, что данные и действия связаны с получением информации, природа которых отличается от определяемых грамматикой (тексты). Например, далее будет рассмотрен пример грамматики двоичных вещественных чисел, в котором ставится дополнительная задача вычисления десятичного значения числа, заданного исходным текстом. Напомним, что грамматики в чистом виде предназначены для обработки текстовой информации, а десятичные числа не являются текстовыми данными. Часто такие, отличающиеся от текстовых, данные и действия называют семантическими. Одним из решений поставленной задачи является получение дерева разбора исходного текста, на котором в процессе обхода можно выполнить семантические действия. Идеальным решением в этом случае будет совмещение грамматического разбора с выполнением семантических действий.

Текст задания:

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.





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;

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

namespace Tyap

{

public partial class Form1 : Form

{

int kolsym = 0;

derevo derev = new derevo();

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();

treeView1.Visible = false;

textBox1.AppendText("(cba!:001!001)");

}

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

{

textBox2.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer(derev);

Synt.Lex.strPSource = textBox1.Lines;

Synt.Lex.strPMessage = textBox2.Lines;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.S(derev);

throw new Exception("Текст верный");

}

catch (Exception exc)

{

textBox2.Text += exc.Message;

textBox1.Select();

textBox1.SelectionStart = 0;

int n = 0;

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

n += Synt.Lex.intPSourceColSelection;

textBox1.SelectionLength = n;

}

}

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

{

CLex Lex = new CLex();

Lex.strPSource = textBox1.Lines;

Lex.strPMessage = textBox2.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = textBox1.TextLength;

int y = textBox1.Lines.Length;

textBox2.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

s1 = "id " + Lex.strPLexicalUnit;

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 0);

// htl.DeleteLexicalCode(0);

TablesToMemo(this, e);

break;

}

case TToken.lxmNumber:

{

s1 = "num " + Lex.strPLexicalUnit;

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 1);

// htl.DeleteLexicalCode(0);

TablesToMemo(this, e);

break;

}

case TToken.lxmdobdot:

{

string t = ":";

s1 = "sim " + t;

htl.DeleteLexicalUnit(t, 2);

TablesToMemo(this, e);

break;

}

case TToken.lxmVoskl:

{

string t = "!";

s1 = "sim " + t;

htl.DeleteLexicalUnit(t, 2);

TablesToMemo(this, e);

break;

}

case TToken.lxmMinus:

{

string t = "-";

s1 = "sim " + t;

htl.DeleteLexicalUnit(t, 2);

TablesToMemo(this, e);

break;

}

case TToken.lxmravn:

{

string t = "=";

s1 = "sim " + t;

htl.DeleteLexicalUnit(t, 2);

TablesToMemo(this, e);

break;

}

case TToken.lxmzv:

{

string t = "\*";

s1 = "sim " + t;

htl.DeleteLexicalUnit(t, 2);

TablesToMemo(this, e);

break;

}

case TToken.lxmRightParenth:

{

string t = ")";

s1 = "sim " + t;

htl.DeleteLexicalUnit(t, 2);

TablesToMemo(this, e);

break;

}

case TToken.lxmDot:

{

string t = ".";

s1 = "sim " + t;

htl.DeleteLexicalUnit(t, 2);

TablesToMemo(this, e);

break;

}

case TToken.lxmLeftParenth:

{

string t = "(";

s1 = "sim " + t;

htl.DeleteLexicalUnit(t, 2);

TablesToMemo(this, e);

break;

}

}

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

textBox2.Text += m;

}

}

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 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();

//MessageBox.Show(Lex.strPLexicalUnit + " " + Lex.enumPToken);

if (Lex.enumPToken == TToken.lxmIdentifier)

{

int b = 0;

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

{

while (Lex2.enumPState != TState.Finish)

{

Lex2.NextToken();

switch (Lex2.enumPToken)

{

case TToken.lxmIdentifier:

{

int b1 = 0;

if (htl.AddLexicalUnit(Lex2.strPLexicalUnit, 0, ref b1))

{

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 0);

TablesToMemo(this, e);

}

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

break;

}

case TToken.lxmNumber:

{

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

break;

}

}

}

}

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

break;

}

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

{

int b = 0;

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

{

while (Lex2.enumPState != TState.Finish)

{

Lex2.NextToken();

switch (Lex2.enumPToken)

{

case TToken.lxmIdentifier:

{

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

break;

}

case TToken.lxmNumber:

{

int b1 = 0;

if (htl.AddLexicalUnit(Lex2.strPLexicalUnit, 1, ref b1))

{

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 1);

TablesToMemo(this, e);

}

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

break;

}

}

}

}

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

break;

}

else

{

int b = 0;

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

{

while (Lex2.enumPState != TState.Finish)

{

Lex2.NextToken();

if (Lex2.enumPToken == TToken.lxmIdentifier)

{

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

break;

}

else if (Lex2.enumPToken == TToken.lxmNumber)

{

tbFMessage1.AppendText("(Нельзя заменить символ на число)");

break;

}

else

{

int b2 = 0;

if (htl.AddLexicalUnit(Lex2.strPLexicalUnit, 2, ref b2))

{

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 2);

TablesToMemo(this, e);

}

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

TablesToMemo(this, e);

}

}

}

else { tbFMessage1.AppendText("(Символ для удаления не найден)"); }

break;

}

}

TablesToMemo(this, e);

// TablesToMemo(this, e);

}

catch (Exception exc)

{

tbFMessage1.Text += exc.Message;

textBox1.Select();

textBox1.SelectionStart = 0;

int n1 = 0;

for (int i = 0; i < Lex.intPSourceRowSelection; i++) n1 += textBox1.Lines[i].Length + 2;

n1 += Lex.intPSourceColSelection;

textBox1.SelectionLength = n1;

}

}

private bool sersym(int[] r, int w)

{

int l = r.Length;

bool tr = true;

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

{

if (w == r[i])

{

tr = false;

}

}

return tr;

}

private void adsym(int[] r,char c)

{

r[kolsym] = c;

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

listBox3.Items.Add(r[i] +"");

}

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

{

int v = 0;

int[] r = new int[7];

CLex Lex = new CLex();

Lex.strPSource = textBox1.Lines;

Lex.strPMessage = textBox2.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = textBox1.TextLength;

int y = textBox1.Lines.Length;

textBox2.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmdobdot:

{

s1 = "simv " + ":";

if (htl.AddLexicalUnit(":", 2, ref v))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmVoskl:

{

//string s5 = Lex.enumPToken+"";

//MessageBox.Show(s5);

s1 = "simv " + "! ";

if (sersym(r, '!'))

adsym(r,'!');

if (htl.AddLexicalUnit("!", 2, ref v))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmMinus:

{

s1 = "simv " + "- ";

char er= '-';

if (sersym(r, er))

adsym(r, er);

if (htl.AddLexicalUnit("-", 2, ref v))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmravn:

{

s1 = "simv " + "= ";

char er = '=';

if (sersym(r, er))

adsym(r, er);

if (htl.AddLexicalUnit("=", 2, ref v))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmzv:

{

s1 = "simv " + "\* ";

char er = '\*';

if (sersym(r, er))

adsym(r, er);

if (htl.AddLexicalUnit("\*", 2, ref v))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmRightParenth:

{

s1 = "simv " + ") ";

char er = ')';

if (sersym(r, er))

adsym(r, er);

//MessageBox.Show(")))))");

if (htl.AddLexicalUnit(")", 2, ref v))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmDot:

{

s1 = "simv " + ".";

char er = '.';

if (sersym(r, er))

adsym(r, er);

if (htl.AddLexicalUnit(".", 2, ref v))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmLeftParenth:

{

s1 = "simv " + "( ";

char er = '(';

if (sersym(r, er))

adsym(r, er);

if (htl.AddLexicalUnit("(", 2, ref v))

{

TablesToMemo(this, e);

}

break;

}

}

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

textBox2.Text += m;

TablesToMemo(this, e);

}

}

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 button3\_Click(object sender, EventArgs e)

{

tbFMessage2.Clear();

CLex Lex = new CLex();

Lex.strPSource = textBox1.Lines;

Lex.strPMessage = textBox2.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = textBox1.TextLength;

int y = textBox1.Lines.Length;

textBox2.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))

{

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))

{

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

}

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

break;

}

case TToken.lxmdobdot:

{

int b = 0;

if (htl.SearchLexicalUnit(":", 2, ref b))

{

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

}

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

break;

}

case TToken.lxmVoskl:

{

int b = 0;

if (htl.SearchLexicalUnit("!", 2, ref b))

{

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

}

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

break;

}

case TToken.lxmMinus:

{

int b = 0;

if (htl.SearchLexicalUnit("-", 2, ref b))

{

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

}

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

break;

}

case TToken.lxmravn:

{

int b = 0;

if (htl.SearchLexicalUnit("=", 2, ref b))

{

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

}

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

break;

}

case TToken.lxmzv:

{

int b = 0;

if (htl.SearchLexicalUnit("\*", 2, ref b))

{

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

}

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

break;

}

case TToken.lxmRightParenth:

{

int b = 0;

if (htl.SearchLexicalUnit(")", 2, ref b))

{

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

}

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

break;

}

case TToken.lxmDot:

{

int b = 0;

if (htl.SearchLexicalUnit(".", 2, ref b))

{

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

}

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

break;

}

case TToken.lxmLeftParenth:

{

int b = 0;

if (htl.SearchLexicalUnit("(", 2, ref b))

{

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

}

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

break;

}

}

}

}

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 button6\_Click(object sender, EventArgs e)

{

textBox2.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer(derev);

Synt.Lex.strPSource = textBox1.Lines;

Synt.Lex.strPMessage = textBox2.Lines;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.S(derev);

der();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

textBox2.Text += exc.Message;

textBox1.Select();

textBox1.SelectionStart = 0;

int n = 0;

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

n += Synt.Lex.intPSourceColSelection;

textBox1.SelectionLength = n;

}

}

public void der()

{

treeView1.BeginUpdate();

Star(derev);

treeView1.EndUpdate();

treeView1.Visible = true;

}

public void son(derevo q, TreeNode w)

{

//MessageBox.Show("Sooon");

if (q.son != null)

{

w.Nodes.Add(q.son.a + "");

son(q.son, w.Nodes[0]);

if (q.son.Brat != null)

{

int r = 1;

derevo e = q.son.Brat;

TreeNode d = w;

while (e != null)

{

d.Nodes.Add(e.a + "");

son(e, d.Nodes[r]);

r++;

e = e.Brat;

}

/\*w.Nodes.Add(q.son.Brat.a + "");

son(q.son.Brat, w.Nodes[1]);\*/

}

}

}

public void Star(derevo q)

{

treeView1.Nodes.Clear();

if (q.son != null)

{

treeView1.Nodes.Add(q.a + "");

treeView1.Nodes[0].Nodes.Add(q.son.a + "");

son(q.son, treeView1.Nodes[0].Nodes[0]);

if (q.son.Brat != null)

{

int r = 1;

derevo e = q.son.Brat;

TreeNode d = treeView1.Nodes[0];

while (e != null)

{

d.Nodes.Add(e.a + "");

son(e, d.Nodes[r]);

r++;

e = e.Brat;

}

}

}

}

private void label3\_Click(object sender, EventArgs e)

{

}

}

}

derevo.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Tyap

{

public class derevo

{

public string a;

public derevo son = null;

public derevo Brat = null;

public void adbra(derevo f, string t)

{

f.Brat = new derevo();

f.Brat.a = t;

}

public void adson(derevo f, string t)

{

//MessageBox.Show("Ads");

f.son = new derevo();

f.son.a = t;

}

}

}

Form1.Designer.cs

namespace Tyap

{

partial class Form1

{

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

private System.ComponentModel.IContainer components = null;

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

/// <param name="disposing">истинно, если управляемый ресурс должен быть удален; иначе ложно.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Код, автоматически созданный конструктором форм Windows

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

private void InitializeComponent()

{

this.textBox1 = new System.Windows.Forms.TextBox();

this.textBox2 = new System.Windows.Forms.TextBox();

this.button1 = new System.Windows.Forms.Button();

this.tbFMessage2 = new System.Windows.Forms.TextBox();

this.button2 = new System.Windows.Forms.Button();

this.button3 = new System.Windows.Forms.Button();

this.button4 = new System.Windows.Forms.Button();

this.button5 = new System.Windows.Forms.Button();

this.listBox1 = new System.Windows.Forms.ListBox();

this.listBox2 = new System.Windows.Forms.ListBox();

this.label1 = new System.Windows.Forms.Label();

this.label2 = new System.Windows.Forms.Label();

this.tbFMessage1 = new System.Windows.Forms.TextBox();

this.tbFSource1 = new System.Windows.Forms.TextBox();

this.tbFSource2 = new System.Windows.Forms.TextBox();

this.treeView1 = new System.Windows.Forms.TreeView();

this.button6 = new System.Windows.Forms.Button();

this.listBox3 = new System.Windows.Forms.ListBox();

this.label3 = new System.Windows.Forms.Label();

this.SuspendLayout();

//

// textBox1

//

this.textBox1.Location = new System.Drawing.Point(69, 61);

this.textBox1.Margin = new System.Windows.Forms.Padding(2);

this.textBox1.Multiline = true;

this.textBox1.Name = "textBox1";

this.textBox1.Size = new System.Drawing.Size(216, 48);

this.textBox1.TabIndex = 0;

//

// textBox2

//

this.textBox2.Location = new System.Drawing.Point(69, 134);

this.textBox2.Margin = new System.Windows.Forms.Padding(2);

this.textBox2.Multiline = true;

this.textBox2.Name = "textBox2";

this.textBox2.Size = new System.Drawing.Size(216, 57);

this.textBox2.TabIndex = 1;

//

// button1

//

this.button1.Location = new System.Drawing.Point(69, 312);

this.button1.Margin = new System.Windows.Forms.Padding(2);

this.button1.Name = "button1";

this.button1.Size = new System.Drawing.Size(56, 19);

this.button1.TabIndex = 2;

this.button1.Text = "Check";

this.button1.UseVisualStyleBackColor = true;

this.button1.Click += new System.EventHandler(this.button1\_Click);

//

// tbFMessage2

//

this.tbFMessage2.Location = new System.Drawing.Point(69, 210);

this.tbFMessage2.Multiline = true;

this.tbFMessage2.Name = "tbFMessage2";

this.tbFMessage2.Size = new System.Drawing.Size(216, 37);

this.tbFMessage2.TabIndex = 3;

//

// button2

//

this.button2.Location = new System.Drawing.Point(48, 377);

this.button2.Name = "button2";

this.button2.Size = new System.Drawing.Size(75, 23);

this.button2.TabIndex = 4;

this.button2.Text = "Dobavit";

this.button2.UseVisualStyleBackColor = true;

this.button2.Click += new System.EventHandler(this.button2\_Click);

//

// button3

//

this.button3.Location = new System.Drawing.Point(129, 377);

this.button3.Name = "button3";

this.button3.Size = new System.Drawing.Size(75, 23);

this.button3.TabIndex = 5;

this.button3.Text = "Naity";

this.button3.UseVisualStyleBackColor = true;

this.button3.Click += new System.EventHandler(this.button3\_Click);

//

// button4

//

this.button4.Location = new System.Drawing.Point(210, 377);

this.button4.Name = "button4";

this.button4.Size = new System.Drawing.Size(75, 23);

this.button4.TabIndex = 6;

this.button4.Text = "Udalit";

this.button4.UseVisualStyleBackColor = true;

this.button4.Click += new System.EventHandler(this.button4\_Click);

//

// button5

//

this.button5.Location = new System.Drawing.Point(441, 70);

this.button5.Name = "button5";

this.button5.Size = new System.Drawing.Size(75, 23);

this.button5.TabIndex = 7;

this.button5.Text = "Zamena";

this.button5.UseVisualStyleBackColor = true;

this.button5.Click += new System.EventHandler(this.button5\_Click);

//

// listBox1

//

this.listBox1.FormattingEnabled = true;

this.listBox1.Location = new System.Drawing.Point(349, 338);

this.listBox1.Name = "listBox1";

this.listBox1.Size = new System.Drawing.Size(120, 212);

this.listBox1.TabIndex = 10;

//

// listBox2

//

this.listBox2.FormattingEnabled = true;

this.listBox2.Location = new System.Drawing.Point(521, 338);

this.listBox2.Name = "listBox2";

this.listBox2.Size = new System.Drawing.Size(120, 212);

this.listBox2.TabIndex = 11;

//

// label1

//

this.label1.AutoSize = true;

this.label1.Location = new System.Drawing.Point(349, 319);

this.label1.Name = "label1";

this.label1.Size = new System.Drawing.Size(36, 13);

this.label1.TabIndex = 12;

this.label1.Text = "word1";

//

// label2

//

this.label2.AutoSize = true;

this.label2.Location = new System.Drawing.Point(521, 318);

this.label2.Name = "label2";

this.label2.Size = new System.Drawing.Size(36, 13);

this.label2.TabIndex = 13;

this.label2.Text = "word2";

//

// tbFMessage1

//

this.tbFMessage1.Location = new System.Drawing.Point(416, 134);

this.tbFMessage1.Multiline = true;

this.tbFMessage1.Name = "tbFMessage1";

this.tbFMessage1.Size = new System.Drawing.Size(144, 77);

this.tbFMessage1.TabIndex = 14;

//

// tbFSource1

//

this.tbFSource1.Location = new System.Drawing.Point(681, 109);

this.tbFSource1.Multiline = true;

this.tbFSource1.Name = "tbFSource1";

this.tbFSource1.Size = new System.Drawing.Size(137, 20);

this.tbFSource1.TabIndex = 15;

//

// tbFSource2

//

this.tbFSource2.Location = new System.Drawing.Point(681, 170);

this.tbFSource2.Multiline = true;

this.tbFSource2.Name = "tbFSource2";

this.tbFSource2.Size = new System.Drawing.Size(137, 20);

this.tbFSource2.TabIndex = 16;

//

// treeView1

//

this.treeView1.Location = new System.Drawing.Point(835, 61);

this.treeView1.Name = "treeView1";

this.treeView1.Size = new System.Drawing.Size(168, 489);

this.treeView1.TabIndex = 17;

//

// button6

//

this.button6.Location = new System.Drawing.Point(48, 463);

this.button6.Name = "button6";

this.button6.Size = new System.Drawing.Size(75, 23);

this.button6.TabIndex = 18;

this.button6.Text = "Tree";

this.button6.UseVisualStyleBackColor = true;

this.button6.Click += new System.EventHandler(this.button6\_Click);

//

// listBox3

//

this.listBox3.FormattingEnabled = true;

this.listBox3.Location = new System.Drawing.Point(681, 338);

this.listBox3.Name = "listBox3";

this.listBox3.Size = new System.Drawing.Size(120, 212);

this.listBox3.TabIndex = 19;

//

// label3

//

this.label3.AutoSize = true;

this.label3.Location = new System.Drawing.Point(678, 312);

this.label3.Name = "label3";

this.label3.Size = new System.Drawing.Size(36, 13);

this.label3.TabIndex = 20;

this.label3.Text = "word3";

this.label3.Click += new System.EventHandler(this.label3\_Click);

//

// Form1

//

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.ClientSize = new System.Drawing.Size(1015, 573);

this.Controls.Add(this.label3);

this.Controls.Add(this.listBox3);

this.Controls.Add(this.button6);

this.Controls.Add(this.treeView1);

this.Controls.Add(this.tbFSource2);

this.Controls.Add(this.tbFSource1);

this.Controls.Add(this.tbFMessage1);

this.Controls.Add(this.label2);

this.Controls.Add(this.label1);

this.Controls.Add(this.listBox2);

this.Controls.Add(this.listBox1);

this.Controls.Add(this.button5);

this.Controls.Add(this.button4);

this.Controls.Add(this.button3);

this.Controls.Add(this.button2);

this.Controls.Add(this.tbFMessage2);

this.Controls.Add(this.button1);

this.Controls.Add(this.textBox2);

this.Controls.Add(this.textBox1);

this.Margin = new System.Windows.Forms.Padding(2);

this.Name = "Form1";

this.Text = "Form1";

this.ResumeLayout(false);

this.PerformLayout();

}

#endregion

private System.Windows.Forms.TextBox textBox1;

private System.Windows.Forms.TextBox textBox2;

private System.Windows.Forms.Button button1;

private System.Windows.Forms.TextBox tbFMessage2;

private System.Windows.Forms.Button button2;

private System.Windows.Forms.Button button3;

private System.Windows.Forms.Button button4;

private System.Windows.Forms.Button button5;

private System.Windows.Forms.ListBox listBox1;

private System.Windows.Forms.ListBox listBox2;

private System.Windows.Forms.Label label1;

private System.Windows.Forms.Label label2;

private System.Windows.Forms.TextBox tbFMessage1;

private System.Windows.Forms.TextBox tbFSource1;

private System.Windows.Forms.TextBox tbFSource2;

private System.Windows.Forms.TreeView treeView1;

private System.Windows.Forms.Button button6;

private System.Windows.Forms.ListBox listBox3;

private System.Windows.Forms.Label label3;

}

}

Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Tyap

{

internal static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

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 Tyap

{

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] % 100 + 100);

}

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 + i) % (Int32)(arrFHashTable.Count);

}

} while (!boolVFinish);

return hi;

}

public void HashIndex(string strALexicalUnit)

{

int h;

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

if (h == 0) { h=arrFHashTable.Count-1; }

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)

{

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

{

//MessageBox.Show(strALexicalUnit);

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.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Tyap

{

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

public enum TCharType

{

Letter, Digit, EndRow, EndText, Space, ReservedSymbol, opBracket, clBracket, expMark,

comma, semicolon, dobdot, plus, zv, min

}; // тип символа

public enum TToken

{

lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs,

lxmDot, lxmComma, lxmMinus, lxmplus, lxmzv, lxmravn, lxmdobdot, lxmVoskl

};

class CLex

{

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.zv;

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 == ':') enumFSelectionCharType = TCharType.dobdot;

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

else if (chrFSelection == '-') enumFSelectionCharType = TCharType.min;

//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();

}

// Вариант 15

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// A | | | B | |

// B | |CFin| | |

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

A:

{

if (chrFSelection == 'c')

{

TakeSymbol();

goto B;

}

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

}

B:

{

if (chrFSelection == 'b')

{

TakeSymbol();

goto CFin;

}

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

}

CFin:

{

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

{

TakeSymbol();

goto CFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

}

case TCharType.Digit:

{

// 0 1

// A | B | |

// B | D | C |

// C | A | |

// D | |EFin |

// EFin | F | |

// F | G | |

// G | EFin| |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto BC;

}

else throw new Exception("Ожидался 0");

BC:

if (chrFSelection == '0')

{

TakeSymbol();

goto E;

}

if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

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

D:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидался 0");

E:

if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидалась 1");

FFin:

if (chrFSelection == '0')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидалась 0");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 0");

H:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0");

}

case TCharType.ReservedSymbol:

{

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

case TCharType.Space:

{

GetSymbol();

break;

}

case TCharType.opBracket:

{

enumFToken = TToken.lxmLeftParenth;

strPLexicalUnit = "(";

GetSymbol();

break;

}

case TCharType.clBracket:

{

enumFToken = TToken.lxmRightParenth;

GetSymbol();

strPLexicalUnit = ")";

break;

}

case TCharType.expMark:

{

enumFToken = TToken.lxmVoskl;

strPLexicalUnit = "!";

GetSymbol();

break;

}

case TCharType.plus:

{

enumFToken = TToken.lxmplus;

strPLexicalUnit = "+";

GetSymbol();

break;

}

case TCharType.dobdot:

{

enumFToken = TToken.lxmdobdot;

strPLexicalUnit = ":";

GetSymbol();

break;

}

case TCharType.zv:

{

enumFToken = TToken.lxmzv;

strPLexicalUnit = "\*";

GetSymbol();

break;

}

case TCharType.min:

{

enumFToken = TToken.lxmMinus;

strPLexicalUnit = "-";

GetSymbol();

break;

}

}

}

}

}

uSyntAnalyzer.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Tyap

{

class uSyntAnalyzer

{

string s;

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 derevo w;

public uSyntAnalyzer()

{

strFSource = new String[0];

strFMessage = new String[0];

strPSource = new String[0];

strPMessage = new String[0];

w = new derevo();

}

public uSyntAnalyzer(derevo q)

{

strFSource = new String[0];

strFMessage = new String[0];

strPSource = new String[0];

strPMessage = new String[0];

w = q;

}

public void S(derevo q)

{

if (Lex.enumPToken == TToken.lxmLeftParenth)

{

q.adson(q, "(");

Lex.NextToken();

q.son.adbra(q.son, "D");

D(q.son.Brat);

if (Lex.enumPToken == TToken.lxmVoskl)

{

q.son.Brat.adbra(q.son.Brat, "!");

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmdobdot)

{

q.son.Brat.Brat.adbra(q.son.Brat.Brat, ":");

Lex.NextToken();

q.son.Brat.Brat.Brat.adbra(q.son.Brat.Brat.Brat, "S");

S(q.son.Brat.Brat.Brat.Brat);

if (Lex.enumPToken == TToken.lxmVoskl)

{

q.son.Brat.Brat.Brat.Brat.adbra(q.son.Brat.Brat.Brat.Brat, "!");

Lex.NextToken();

q.son.Brat.Brat.Brat.Brat.Brat.adbra(q.son.Brat.Brat.Brat.Brat.Brat, "S");

S(q.son.Brat.Brat.Brat.Brat.Brat.Brat);

if (Lex.enumPToken == TToken.lxmRightParenth)

{

q.son.Brat.Brat.Brat.Brat.Brat.Brat.adbra(q.son.Brat.Brat.Brat.Brat.Brat.Brat, ")");

Lex.NextToken();

}

else throw new Exception("Ожидалcя )");

}

}

else throw new Exception("Ожидалcя :");

}

else throw new Exception("Ожидалcя !");

}

else

if (Lex.enumPToken == TToken.lxmNumber)

{

q.adson(q, Lex.strPLexicalUnit);

Lex.NextToken();

}

else { throw new Exception("Ожидалcя ( или число"); }

}

public void D(derevo q)

{

q.adson(q, "K");

K(q.son);

if (Lex.enumPToken == TToken.lxmplus)

{

q.son.adbra(q.son, "+");

q.son.Brat.adbra(q.son.Brat, "C");

C(q.son.Brat.Brat);

}

}

public void C(derevo q)

{

if (Lex.enumPToken == TToken.lxmplus)

{

Lex.NextToken();

q.adson(q, "K");

K(q.son);

if (Lex.enumPToken == TToken.lxmplus)

{

q.son.adbra(q.son, "+");

C(q.son.Brat);

}

}

}

public void K(derevo q)

{

q.adson(q, "A");

A(q.son);

if (Lex.enumPToken == TToken.lxmzv)

{

q.son.adbra(q.son, "\*");

q.son.Brat.adbra(q.son.Brat, "P");

P(q.son.Brat.Brat);

}

}

public void P(derevo q)

{

if (Lex.enumPToken == TToken.lxmzv)

{

Lex.NextToken();

q.adson(q, "A");

A(q.son);

if (Lex.enumPToken == TToken.lxmzv)

{

q.son.adbra(q.son, "\*");

q.son.Brat.adbra( q.son.Brat, "P");

P(q.son.Brat.Brat);

}

}

}

public void A(derevo q)

{

if (Lex.enumPToken == TToken.lxmIdentifier)

{

q.adson(q, Lex.strPLexicalUnit);

Lex.NextToken();

}

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

{

q.adson(q, "(");

Lex.NextToken();

q.son.adbra(q.son, "D");

D(q.son.Brat);

if (Lex.enumPToken == TToken.lxmRightParenth)

{

q.son.Brat.adbra(q.son.Brat, ")");

Lex.NextToken();

}

else throw new Exception("Ожидалась )");

}

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

{

q.adson(q, "-");

Lex.NextToken();

A(q.son);

}

else

{

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

}

}

}

}

**Лабораторная работа № 6. Разработка генератора**

Теория:

В модели анализа – синтеза компилятора на начальной стадии анализируется исходная программа и создается промежуточное представление, из которого на заключительной стадии генерируется целевой код. Основная задача генератора кода – построение машинной программы, эквивалентной по смыслу программе на входном языке.

На рис. 1 начальная стадия представлена блоками «Анализ» и «Генератор промежуточного кода», заключительная стадия – блоком «Генератор кода». В идеальном случае детали исходного языка ограничены начальной стадией (анализатор), а детали целевой машины (исполнителя) – заключительной стадией компилятора (генератор кода).

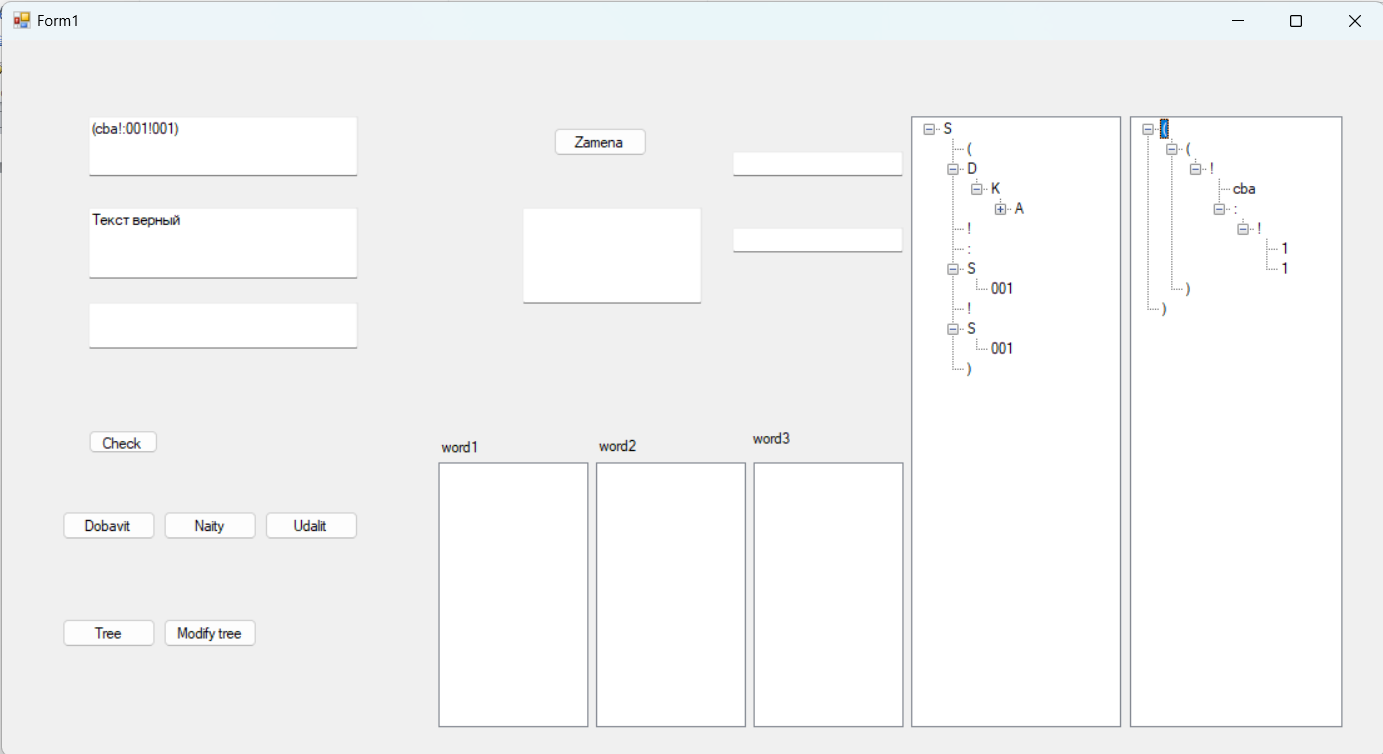


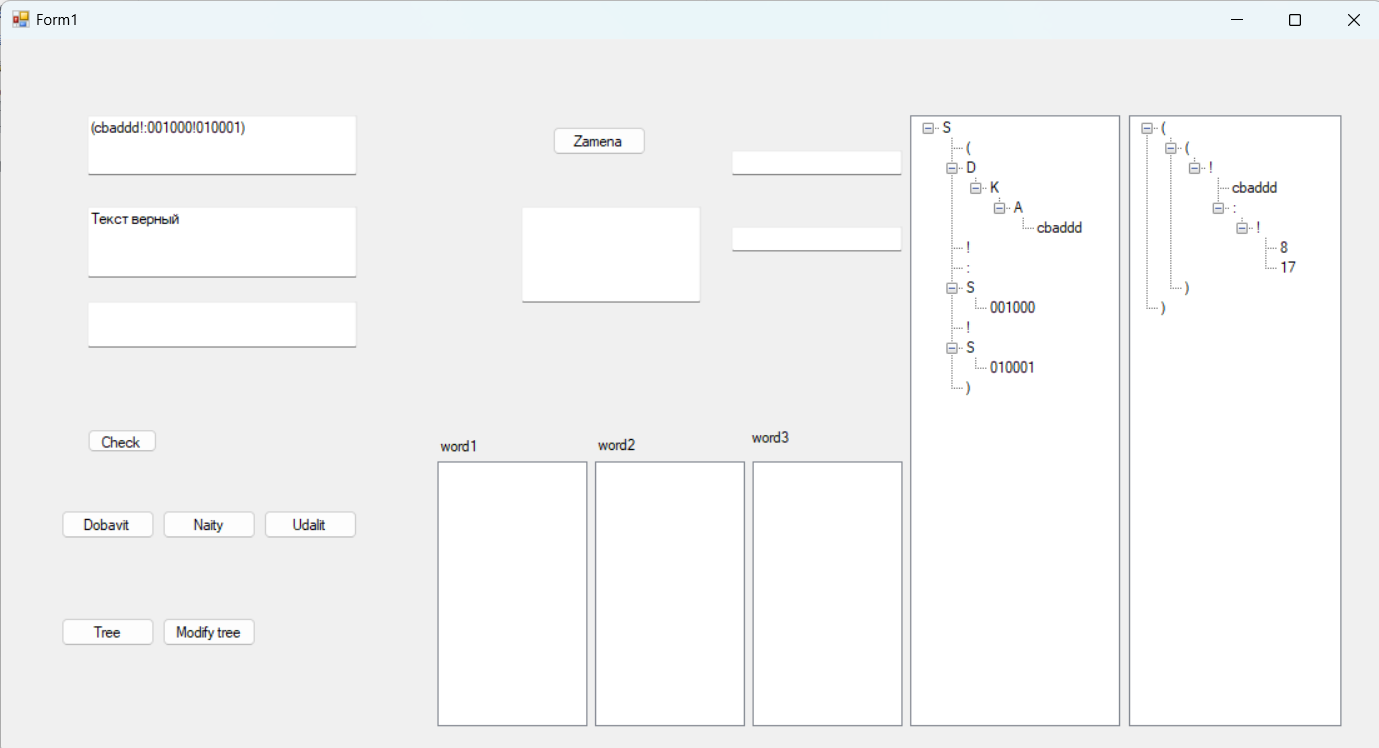
Рис 1. Логическая структура компилятора

Генерация кода состоит из ряда специфических, относительно независимых подзадач: распределение памяти (оперативной и регистровой), подбор команд компьютера, генерация объектного (или загрузочного) модуля. Конечно, независимость этих подзадач относительна: например, при выборе команд нельзя не учитывать схему распределения памяти, или, наоборот, схема распределения памяти необходимо ведет к генерации той или иной последовательности команд. Однако удобно и практично эти задачи все же разделять и при этом особо обращать внимание на их взаимодействие. В значительной мере схема генератора кода зависит от формы промежуточного представления исходной программы. Ясно, что генерация кода из вычислительной модели в виде дерева отличается от генерации кода из троек, а генерация кода из постфиксной записи отличается от генерации кода из модели ориентированного графа. В то же время, все генераторы кода имеют и много общего, и основные применяемые алгоритмы отличаются, как правило, только в деталях, связанных с используемым промежуточным представлением. Такое разделение генератора на генератор промежуточного представления и генератор кода способствует созданию типовых решений на уровне промежуточного кода за счет абстрагирования от архитектуры конкретного компьютера.

Текст задания:

1. Перевести все числа в десятичное представление.

2. Выполнить вывод исходного текста в структурированном виде.



**Изменения в коде в Form1.cs, было добавлено несколько функций: step, seder, prodvder, maxeva, otri, pois.**

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

{

treeView2.Visible = true;

seder(derev);

prodvder();

}

int step(int a, int b) // определяет систему координат, считает вес разряда

{

int s = 1;

for (int i = 0; i < a; i++) { s = s \* b; }

return s;

}

public void seder(derevo q) // обшая функция. пероеводит из дерева все числа в десятичные и сохраняет в список tex

{

if (q != null)

{

if ((!zagl.Contains(q.a)) && (q.a != null))

{

if (q.a[0] == '0' || q.a[0] == '1')

{

int r = Convert.ToInt32(q.a);

int y = q.a.Length;

int re = 0;

for (int i = 0; i < y; i++) // перевод чисел в десятичную систему

{

if (r % 10 == 1)

re = re + step(i, 2);

r = r / 10;

}

tex.Add(re.ToString());

}

else

{ tex.Add(q.a); }

siz++;

}

seder(q.son); // рекурсивно обрабатывает каждую ветку

seder(q.Brat);

}

}

public void prodvder() // анализирует список tex и присваивает каждому элементу коэффициент глубины

{

koef = new int[tex.Count]; // для хронения коэффицентов

int obkoef = 0;

int[] bloc = new int[tex.Count];

int kolblo = 0;

if (tex.Count > 0)

{

if (tex.Contains("(")) // проверка на одинаковое количсвтво скобок ) м ( , и присваиваниеи номеров им

{

int kolsko = 0;

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

{

if (tex[i] == "(") { kolsko++; koef[i]--; }

koef[i] = koef[i] + 10 \* kolsko;

if (tex[i] == ")") { kolsko--; koef[i]--; }

}

}

if (tex.Contains(":")) // счет :

{

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

{

if (tex[i] == ":")

{

int beg = i + 1;

// koef[beg]++;

while ((beg < tex.Count) && (koef[beg] >= koef[i]))

{

koef[beg]++;

beg++;

}

}

}

}

if (tex.Contains("!"))

{

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

{

if (tex[i] == "!")

{

int beg = i;

beg--;

while (beg >= 0 && (koef[beg] >= koef[i]))

{

koef[beg]++;

beg--;

}

beg = i + 1;

while ((beg < tex.Count) && (koef[beg] >= koef[i]))

{

koef[beg]++;

beg++;

}

}

}

}

}

treeView2.BeginUpdate();

treeView2.Nodes.Clear(); // отчишает вск ветки

int pole = 0;

pole = maxeva(koef);

treeView2.Nodes.Add(tex[pole]);

otri(treeView2.Nodes[0], koef, pole);

treeView2.EndUpdate();

}

public int maxeva(int[] t) // находит минимальный коэффицтент глубины, корень дерева

{

int ma = 0;

for (int i = 0; i < t.Length; i++) { ma = ((t[i] < t[ma])) ? i : ma; }

return ma;

}

public void otri(TreeNode ve, int[] koef, int cur) // главная, строит дерево с корня

{

//MessageBox.Show(tex[cur]);

if (oper.Contains(tex[cur])) // содержит ли этот элимет

{

if (tex[cur] != "(" && tex[cur] != ")") // проверка с другими

{

if (tex[cur] != ":")

{

ve.Nodes.Add(tex[pois(koef, cur, -1)]); // прикручиваем -1 ветку

otri(ve.Nodes[0], koef, pois(koef, cur, -1)); // вызов рекурсивной

ve.Nodes.Add(tex[pois(koef, cur, 1)]); // прикручиваем 1 ветку

otri(ve.Nodes[ve.Nodes.Count - 1], koef, pois(koef, cur, 1));

}

else

{

ve.Nodes.Add(tex[pois(koef, cur, 1)]);

otri(ve.Nodes[ve.Nodes.Count - 1], koef, pois(koef, cur, 1));

}

}

else

{

if (tex[cur] == "(")

{

if (ve.Parent != null)

{ ve.Parent.Nodes.Add(")"); }

else { treeView2.Nodes.Add(")"); }

ve.Nodes.Add(tex[pois(koef, cur, 1)]);

otri(ve.Nodes[0], koef, pois(koef, cur, 1));

}

}

}

}

public int pois(int[] t, int cur, int ti) // нахождение с право и слева элементов для узла ( yzli, связи между элементами )

{

int ma = cur + ti;

int beg = cur + ti;

while ((beg < tex.Count) && (beg >= 0) && (t[beg] >= t[cur]))

{

if ((t[beg] <= t[ma]) && (tex[beg] != ")"))

{

ma = beg;

}

beg = beg + ti;

}

return ma;

}

**Лабораторная работа № 7. ТЕОРИЯ ФОРМАЛЬНЫХ ГРАММАТИК И**

**АВТОМАТОВ**

Теория:

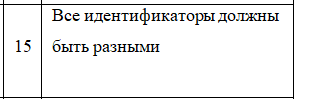
Контекстуальность в языкознании представляет собой условие осмысленного употребления той или иной конкретной языковой единицы в речи (письменной или устной), с учётом её языкового окружения и ситуации речевого общения.

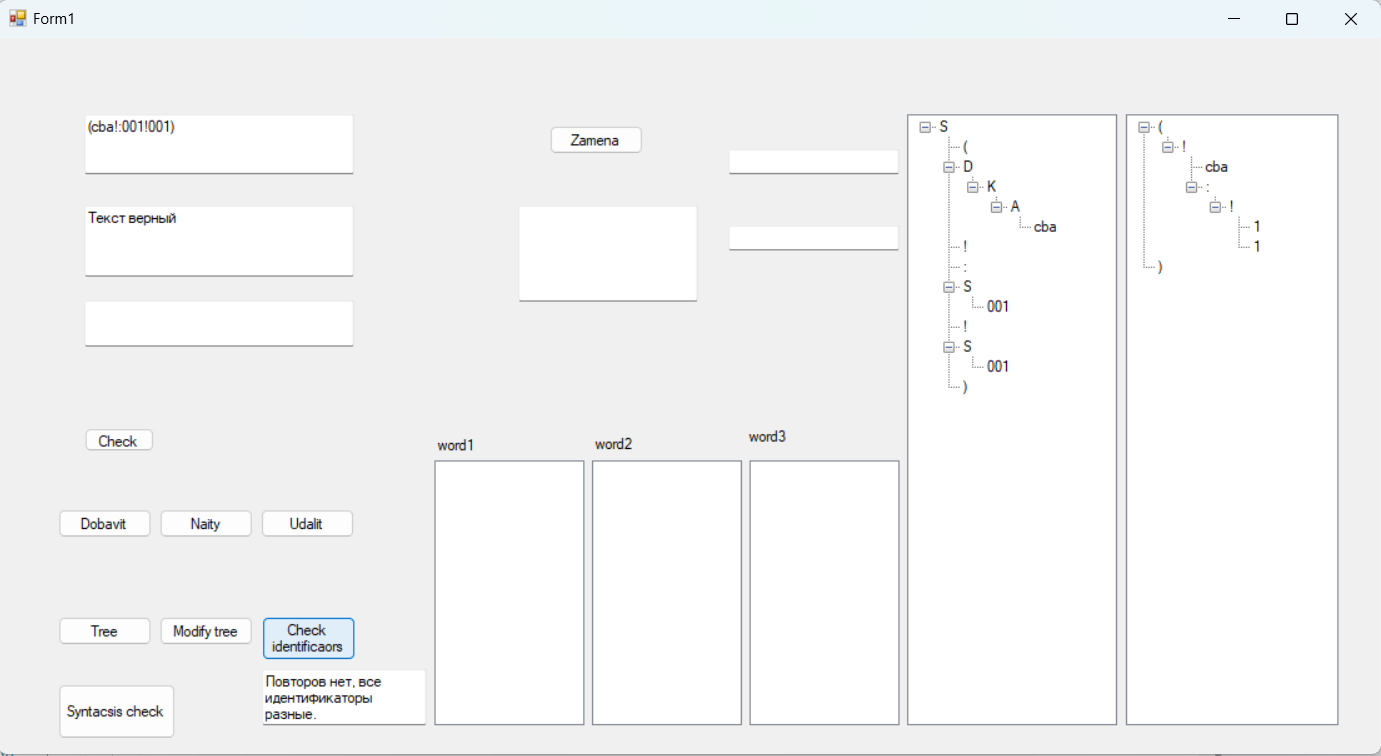
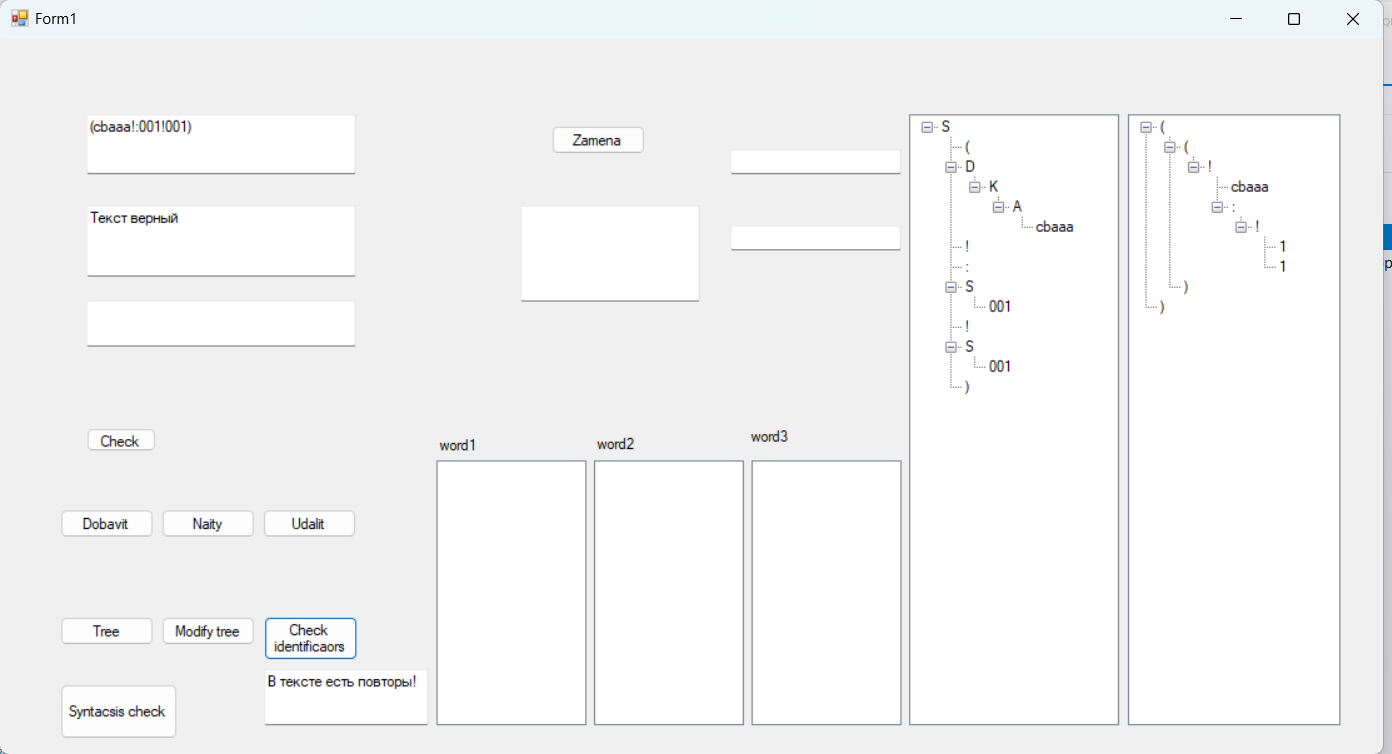
Задачей контекстного анализа языков программирования является установление правильности совместного использования синтаксических единиц в тексте исходной программы. В языках программирования контекст – это текстовое окружение некоторой фразы языка программирования, некоторого фрагмента текста. Правила согласованного использования отдельных фрагментов текста программы называются контекстными условиями.

Контекстными условиями будем называть такие правила построения определяемого языка программирования, которые нецелесообразно или невозможно изобразить с помощью контекстно-свободной грамматики.

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.

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.



**Изменения в коде в Form1.cs, было добавлен код**

bool findpov()

{

bool rez = true;

List<string> prov = new List<string>(); // пустой список

int len = tex.Count;

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

{

if (tex[i][0] == 'a' || tex[i][0] == 'b' || tex[i][0] == 'c' || tex[i][0] == 'd') // строка и первый символ

{

if (prov.Contains(tex[i]))

{

rez = false;

}

else { prov.Add(tex[i]); } // список заполняется данными из дерева из **функции seder**

}

}

return rez;

}

private void button8\_Click(object sender, EventArgs e)

{

string sz;

if (findpov())

{ sz = "Повторов нет, все идентификаторы разные."; }

else { sz = "В тексте есть повторы!"; }

textBox3.Text = sz;

}

**Лабораторная работа № 8. Семантический анализатор**

Теория:

Семантический анализатор (или проверщик семантики) в

компиляторах и языках программирования отвечает за анализ семантической корректности программы. Это означает, что семантический анализатор проверяет, соответствует ли программа правилам семантики языка программирования.

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.

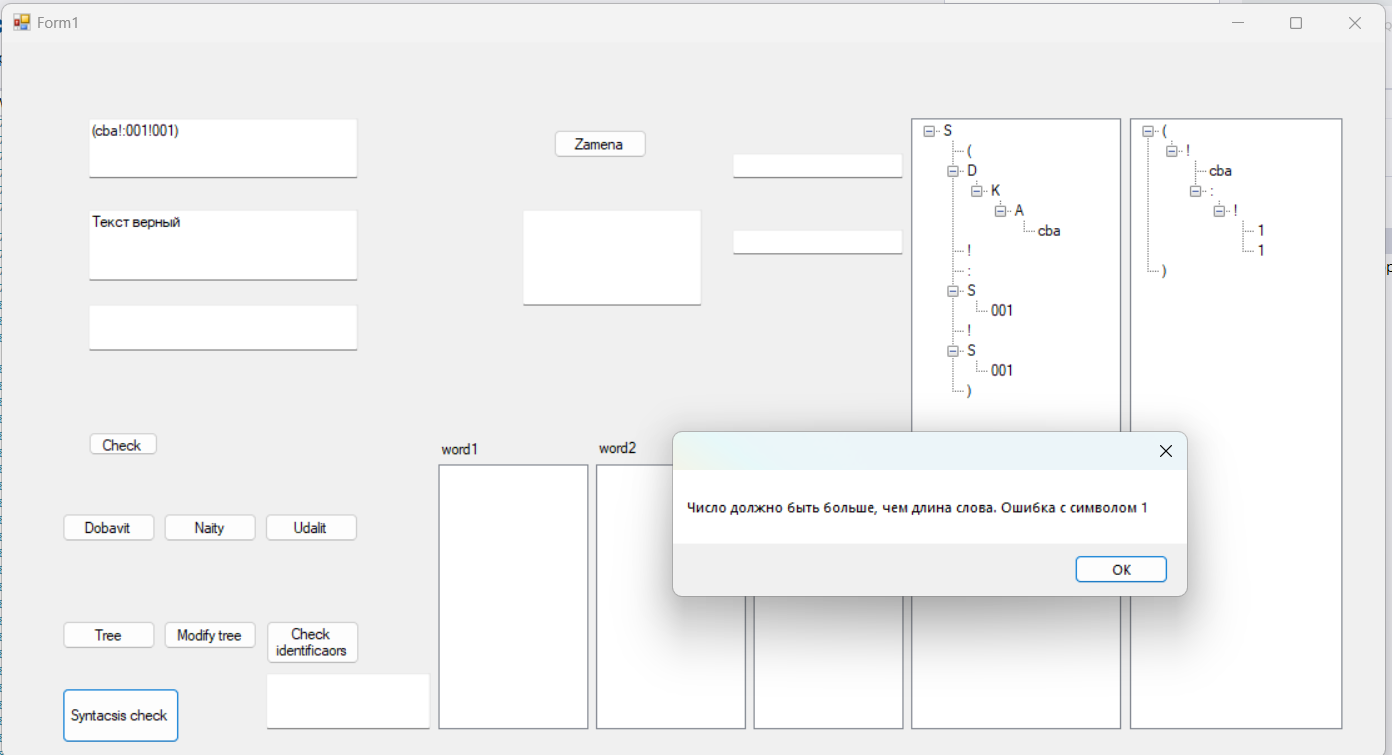
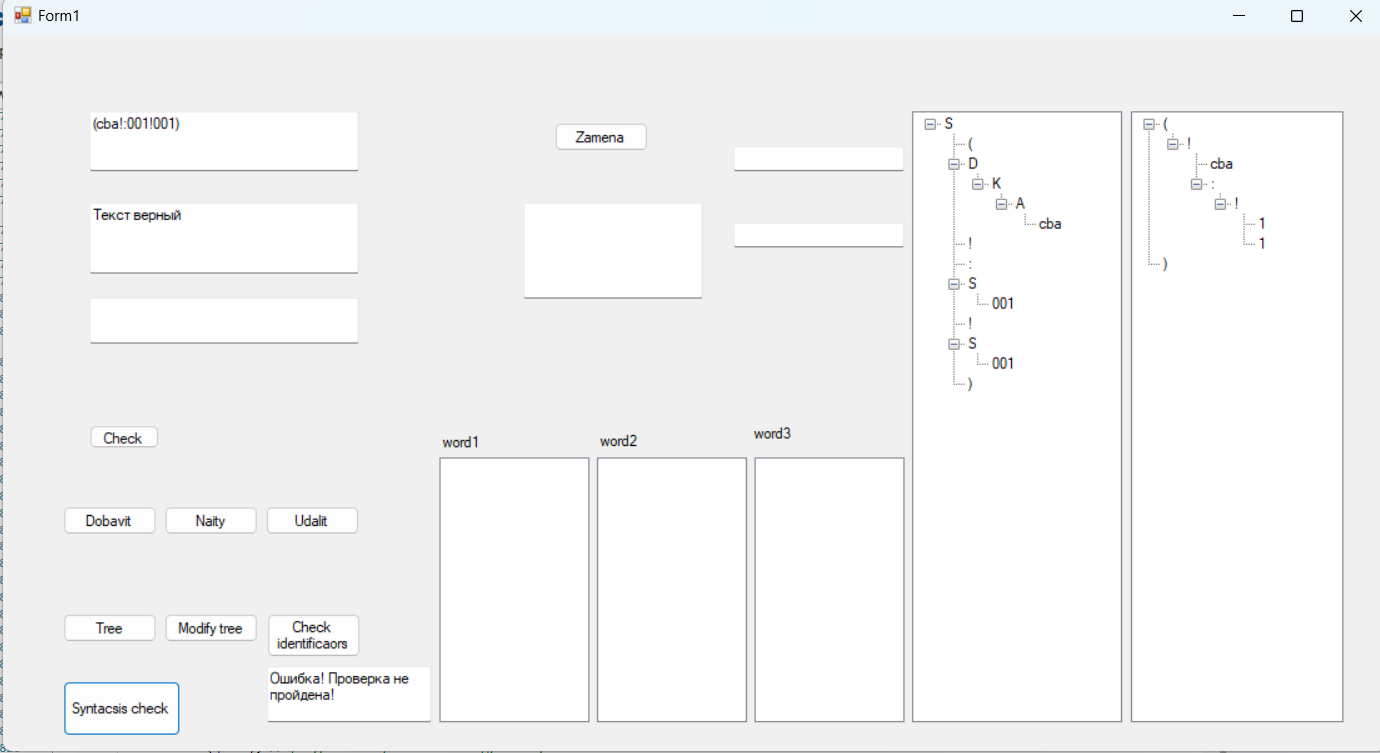
Важным аспектом семантического анализа является проверка типов, когда компилятор проверяет, что каждый оператор имеет операнды допустимого спецификациями языка типа. Например, определение многих языков программирования требует, чтобы при использовании действительного числа в качестве индекса массива генерировалось сообщение об ошибке. В то же время спецификация языка может позволить определенное преобразование типов, например, когда бинарный арифметический оператор применяется к операндам целого и действительного типов. В этом случае компилятору может потребоваться преобразование целого числа в действительное.

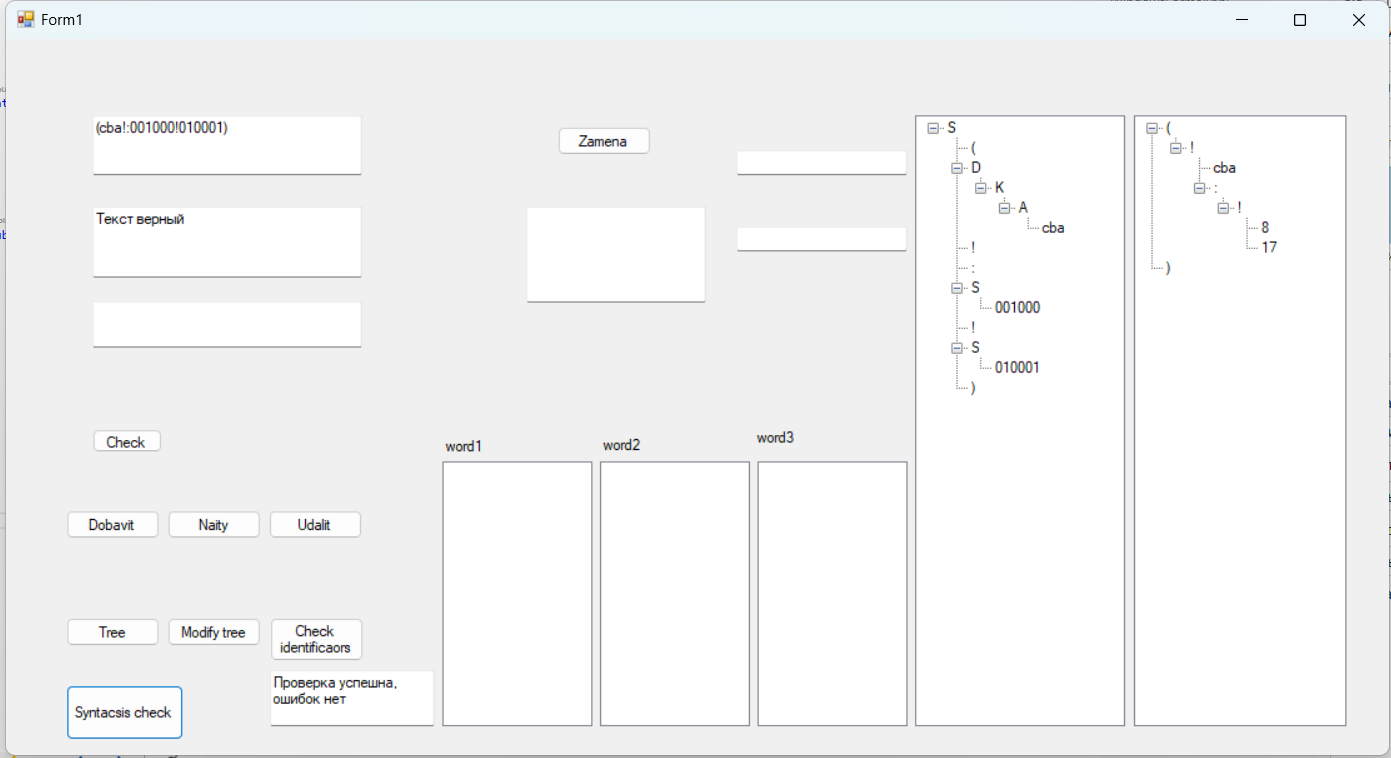
Задание: задать правило семантического анализатора с помощью одного из методов построения семантического анализатора.

Правило:

Все числа должны быть больше, чем длина слова.

Для этого были добавлены методы проверки на то, является ли числом содержимое строки и метод анализа.



public bool provsem()

{

int cur = 0;

string se;

int chi = 0;

bool rez = true;

int lenmin = tex[cur].Length;

int koe = koef[cur];

int caret = cur + 1;

while (cur < tex.Count)

{

caret = cur + 1;

if (tex[cur].Contains('a') || tex[cur].Contains('b') || tex[cur].Contains('c') || tex[cur].Contains('d'))

{

lenmin = tex[cur].Length; // длина

while ((caret < koef.Length) && (koef[caret] <= koef[cur] + 5))

{

if (ischisl(tex[caret]))

{

chi = Convert.ToInt32(tex[caret]);

if (!(lenmin <= chi))

{

string s = "Число должно быть больше, чем длина слова. Ошибка с символом " + tex[caret];

MessageBox.Show(s);

rez = false;

}

}

caret++;

}

}

cur++;

}

return rez;

}

private void button9\_Click(object sender, EventArgs e)

{

if (provsem())

{

textBox3.Text = "Проверка успешна, ошибок нет";

}

else

{

textBox3.Text = "Ошибка! Проверка не пройдена!";

}

}

**Код всей программы**

Form1.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Security;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using static System.Net.Mime.MediaTypeNames;

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

namespace Tyap

{

public partial class Form1 : Form

{

int kolsym = 0;

derevo derev = new derevo();

public CHashTableList htl = new CHashTableList(3);

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

int siz = 0;

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

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

int[] koef;

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();

treeView1.Visible = false;

zagl.Clear();

zagl.Add("A");

zagl.Add("B");

zagl.Add("S");

zagl.Add("C");

zagl.Add("D");

zagl.Add("K");

zagl.Add("O");

oper.Clear();

oper.Add(":");

oper.Add("(");

oper.Add(")");

oper.Add("-");

oper.Add("!");

oper.Add("=");

textBox1.AppendText("(cba!:001!001)");

}

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

{

textBox2.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer(derev);

Synt.Lex.strPSource = textBox1.Lines;

Synt.Lex.strPMessage = textBox2.Lines;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.S(derev);

throw new Exception("Текст верный");

}

catch (Exception exc)

{

textBox2.Text += exc.Message;

textBox1.Select();

textBox1.SelectionStart = 0;

int n = 0;

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

n += Synt.Lex.intPSourceColSelection;

textBox1.SelectionLength = n;

}

}

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

{

CLex Lex = new CLex();

Lex.strPSource = textBox1.Lines;

Lex.strPMessage = textBox2.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = textBox1.TextLength;

int y = textBox1.Lines.Length;

textBox2.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

s1 = "id " + Lex.strPLexicalUnit;

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 0);

// htl.DeleteLexicalCode(0);

TablesToMemo(this, e);

break;

}

case TToken.lxmNumber:

{

s1 = "num " + Lex.strPLexicalUnit;

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 1);

// htl.DeleteLexicalCode(0);

TablesToMemo(this, e);

break;

}

case TToken.lxmdobdot:

{

string t = ":";

s1 = "sim " + t;

htl.DeleteLexicalUnit(t, 2);

TablesToMemo(this, e);

break;

}

case TToken.lxmVoskl:

{

string t = "!";

s1 = "sim " + t;

htl.DeleteLexicalUnit(t, 2);

TablesToMemo(this, e);

break;

}

case TToken.lxmMinus:

{

string t = "-";

s1 = "sim " + t;

htl.DeleteLexicalUnit(t, 2);

TablesToMemo(this, e);

break;

}

case TToken.lxmravn:

{

string t = "=";

s1 = "sim " + t;

htl.DeleteLexicalUnit(t, 2);

TablesToMemo(this, e);

break;

}

case TToken.lxmzv:

{

string t = "\*";

s1 = "sim " + t;

htl.DeleteLexicalUnit(t, 2);

TablesToMemo(this, e);

break;

}

case TToken.lxmRightParenth:

{

string t = ")";

s1 = "sim " + t;

htl.DeleteLexicalUnit(t, 2);

TablesToMemo(this, e);

break;

}

case TToken.lxmDot:

{

string t = ".";

s1 = "sim " + t;

htl.DeleteLexicalUnit(t, 2);

TablesToMemo(this, e);

break;

}

case TToken.lxmLeftParenth:

{

string t = "(";

s1 = "sim " + t;

htl.DeleteLexicalUnit(t, 2);

TablesToMemo(this, e);

break;

}

}

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

textBox2.Text += m;

}

}

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 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();

//MessageBox.Show(Lex.strPLexicalUnit + " " + Lex.enumPToken);

if (Lex.enumPToken == TToken.lxmIdentifier)

{

int b = 0;

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

{

while (Lex2.enumPState != TState.Finish)

{

Lex2.NextToken();

switch (Lex2.enumPToken)

{

case TToken.lxmIdentifier:

{

int b1 = 0;

if (htl.AddLexicalUnit(Lex2.strPLexicalUnit, 0, ref b1))

{

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 0);

TablesToMemo(this, e);

}

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

break;

}

case TToken.lxmNumber:

{

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

break;

}

}

}

}

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

break;

}

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

{

int b = 0;

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

{

while (Lex2.enumPState != TState.Finish)

{

Lex2.NextToken();

switch (Lex2.enumPToken)

{

case TToken.lxmIdentifier:

{

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

break;

}

case TToken.lxmNumber:

{

int b1 = 0;

if (htl.AddLexicalUnit(Lex2.strPLexicalUnit, 1, ref b1))

{

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 1);

TablesToMemo(this, e);

}

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

break;

}

}

}

}

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

break;

}

else

{

int b = 0;

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

{

while (Lex2.enumPState != TState.Finish)

{

Lex2.NextToken();

if (Lex2.enumPToken == TToken.lxmIdentifier)

{

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

break;

}

else if (Lex2.enumPToken == TToken.lxmNumber)

{

tbFMessage1.AppendText("(Нельзя заменить символ на число)");

break;

}

else

{

int b2 = 0;

if (htl.AddLexicalUnit(Lex2.strPLexicalUnit, 2, ref b2))

{

htl.DeleteLexicalUnit(Lex.strPLexicalUnit, 2);

TablesToMemo(this, e);

}

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

TablesToMemo(this, e);

}

}

}

else { tbFMessage1.AppendText("(Символ для удаления не найден)"); }

break;

}

}

TablesToMemo(this, e);

// TablesToMemo(this, e);

}

catch (Exception exc)

{

tbFMessage1.Text += exc.Message;

textBox1.Select();

textBox1.SelectionStart = 0;

int n1 = 0;

for (int i = 0; i < Lex.intPSourceRowSelection; i++) n1 += textBox1.Lines[i].Length + 2;

n1 += Lex.intPSourceColSelection;

textBox1.SelectionLength = n1;

}

}

private bool sersym(int[] r, int w)

{

int l = r.Length;

bool tr = true;

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

{

if (w == r[i])

{

tr = false;

}

}

return tr;

}

private void adsym(int[] r,char c)

{

r[kolsym] = c;

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

listBox3.Items.Add(r[i] +"");

}

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

{

int v = 0;

int[] r = new int[7];

CLex Lex = new CLex();

Lex.strPSource = textBox1.Lines;

Lex.strPMessage = textBox2.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = textBox1.TextLength;

int y = textBox1.Lines.Length;

textBox2.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmdobdot:

{

s1 = "simv " + ":";

if (htl.AddLexicalUnit(":", 2, ref v))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmVoskl:

{

//string s5 = Lex.enumPToken+"";

//MessageBox.Show(s5);

s1 = "simv " + "! ";

if (sersym(r, '!'))

adsym(r,'!');

if (htl.AddLexicalUnit("!", 2, ref v))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmMinus:

{

s1 = "simv " + "- ";

char er= '-';

if (sersym(r, er))

adsym(r, er);

if (htl.AddLexicalUnit("-", 2, ref v))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmravn:

{

s1 = "simv " + "= ";

char er = '=';

if (sersym(r, er))

adsym(r, er);

if (htl.AddLexicalUnit("=", 2, ref v))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmzv:

{

s1 = "simv " + "\* ";

char er = '\*';

if (sersym(r, er))

adsym(r, er);

if (htl.AddLexicalUnit("\*", 2, ref v))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmRightParenth:

{

s1 = "simv " + ") ";

char er = ')';

if (sersym(r, er))

adsym(r, er);

//MessageBox.Show(")))))");

if (htl.AddLexicalUnit(")", 2, ref v))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmDot:

{

s1 = "simv " + ".";

char er = '.';

if (sersym(r, er))

adsym(r, er);

if (htl.AddLexicalUnit(".", 2, ref v))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmLeftParenth:

{

s1 = "simv " + "( ";

char er = '(';

if (sersym(r, er))

adsym(r, er);

if (htl.AddLexicalUnit("(", 2, ref v))

{

TablesToMemo(this, e);

}

break;

}

}

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

textBox2.Text += m;

TablesToMemo(this, e);

}

}

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 button3\_Click(object sender, EventArgs e)

{

tbFMessage2.Clear();

CLex Lex = new CLex();

Lex.strPSource = textBox1.Lines;

Lex.strPMessage = textBox2.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = textBox1.TextLength;

int y = textBox1.Lines.Length;

textBox2.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))

{

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))

{

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

}

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

break;

}

case TToken.lxmdobdot:

{

int b = 0;

if (htl.SearchLexicalUnit(":", 2, ref b))

{

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

}

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

break;

}

case TToken.lxmVoskl:

{

int b = 0;

if (htl.SearchLexicalUnit("!", 2, ref b))

{

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

}

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

break;

}

case TToken.lxmMinus:

{

int b = 0;

if (htl.SearchLexicalUnit("-", 2, ref b))

{

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

}

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

break;

}

case TToken.lxmravn:

{

int b = 0;

if (htl.SearchLexicalUnit("=", 2, ref b))

{

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

}

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

break;

}

case TToken.lxmzv:

{

int b = 0;

if (htl.SearchLexicalUnit("\*", 2, ref b))

{

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

}

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

break;

}

case TToken.lxmRightParenth:

{

int b = 0;

if (htl.SearchLexicalUnit(")", 2, ref b))

{

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

}

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

break;

}

case TToken.lxmDot:

{

int b = 0;

if (htl.SearchLexicalUnit(".", 2, ref b))

{

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

}

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

break;

}

case TToken.lxmLeftParenth:

{

int b = 0;

if (htl.SearchLexicalUnit("(", 2, ref b))

{

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

}

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

break;

}

}

}

}

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 button6\_Click(object sender, EventArgs e)

{

textBox2.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer(derev);

Synt.Lex.strPSource = textBox1.Lines;

Synt.Lex.strPMessage = textBox2.Lines;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.S(derev);

der();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

textBox2.Text += exc.Message;

textBox1.Select();

textBox1.SelectionStart = 0;

int n = 0;

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

n += Synt.Lex.intPSourceColSelection;

textBox1.SelectionLength = n;

}

}

public void der()

{

treeView1.BeginUpdate();

Star(derev);

treeView1.EndUpdate();

treeView1.Visible = true;

}

public void son(derevo q, TreeNode w)

{

//MessageBox.Show("Sooon");

if (q.son != null)

{

w.Nodes.Add(q.son.a + "");

son(q.son, w.Nodes[0]);

if (q.son.Brat != null)

{

int r = 1;

derevo e = q.son.Brat;

TreeNode d = w;

while (e != null)

{

d.Nodes.Add(e.a + "");

son(e, d.Nodes[r]);

r++;

e = e.Brat;

}

/\*w.Nodes.Add(q.son.Brat.a + "");

son(q.son.Brat, w.Nodes[1]);\*/

}

}

}

public void Star(derevo q)

{

treeView1.Nodes.Clear();

if (q.son != null)

{

treeView1.Nodes.Add("S");

treeView1.Nodes[0].Nodes.Add(q.son.a + "");

son(q.son, treeView1.Nodes[0].Nodes[0]);

if (q.son.Brat != null)

{

int r = 1;

derevo e = q.son.Brat;

TreeNode d = treeView1.Nodes[0];

while (e != null)

{

d.Nodes.Add(e.a + "");

son(e, d.Nodes[r]);

r++;

e = e.Brat;

}

}

}

}

private void label3\_Click(object sender, EventArgs e)

{

}

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

{

seder(derev);

vivtre2();

//MessageBox.Show(tex.Count.ToString());

}

int step(int a,int b)

{

int s=1;

for (int i = 0; i < a; i++) { s = s \* b; }

return s;

}

public void seder(derevo q)

{

if (q != null)

{

if ((!zagl.Contains(q.a)) && (q.a != null))

{

if (q.a[0] == '0'|| q.a[0] == '1')

{

int r = Convert.ToInt32(q.a);

int y = q.a.Length;

int re = 0;

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

{

if (r%10==1)

re = re+step(i,2);

r = r / 10;

}

tex.Add(re.ToString());

//MessageBox.Show(re.ToString());

//MessageBox.Show("Chisl");

}

else

{ tex.Add(q.a); }

siz++;

//MessageBox.Show(q.a);

}

seder(q.son);

seder(q.Brat);

}

}

public void vivtre2()

{

prodvder();

}

public void prodvder()

{

koef = new int[tex.Count];

int obkoef = 0;

int[] bloc = new int[tex.Count];

int kolblo = 0;

if (tex.Count > 0)

{

if (tex.Contains("("))

{

int kolsko = 0;

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

{

if (tex[i] == "(") { kolsko++; koef[i]--; }

koef[i] = koef[i] + 10 \* kolsko;

if (tex[i] == ")") { kolsko--; koef[i]--; }

}

}

if (tex.Contains(":"))

{

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

{

if (tex[i] == ":")

{

int beg = i + 1;

// koef[beg]++;

while ((beg < tex.Count) && (koef[beg] >= koef[i]))

{

koef[beg]++;

beg++;

}

}

}

}

if (tex.Contains("!"))

{

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

{

if (tex[i] == "!")

{

int beg = i;

beg--;

while (beg >= 0 && (koef[beg] >= koef[i]))

{

koef[beg]++;

beg--;

}

beg = i + 1;

while ((beg < tex.Count) && (koef[beg] >= koef[i]))

{

koef[beg]++;

beg++;

}

}

}

}

}

// Viviod

treeView2.BeginUpdate();

treeView2.Nodes.Clear();

int pole = 0;

pole = maxeva(koef);

treeView2.Nodes.Add(tex[pole]);

otri(treeView2.Nodes[0], koef, pole);

treeView2.EndUpdate();

}

public int maxeva(int[] t)

{

int ma = 0;

for (int i = 0; i < t.Length; i++) { ma = ((t[i] < t[ma])) ? i : ma; }

return ma;

}

public void otri(TreeNode ve, int[] koef, int cur)

{

//MessageBox.Show(tex[cur]);

if (oper.Contains(tex[cur]))

{

if (tex[cur] != "(" && tex[cur] != ")")

{

if (tex[cur] != ":")

{

ve.Nodes.Add(tex[pois(koef, cur, -1)]);

otri(ve.Nodes[0], koef, pois(koef, cur, -1));

ve.Nodes.Add(tex[pois(koef, cur, 1)]);

otri(ve.Nodes[ve.Nodes.Count - 1], koef, pois(koef, cur, 1));

}

else

{

ve.Nodes.Add(tex[pois(koef, cur, 1)]);

otri(ve.Nodes[ve.Nodes.Count - 1], koef, pois(koef, cur, 1));

}

}

else

{

if (tex[cur] == "(")

{

if (ve.Parent != null)

{ ve.Parent.Nodes.Add(")"); }

else { treeView2.Nodes.Add(")"); }

ve.Nodes.Add(tex[pois(koef, cur, 1)]);

otri(ve.Nodes[0], koef, pois(koef, cur, 1));

}

else

{

}

}

}

}

public int pois(int[] t, int cur, int ti)

{

int ma = cur + ti;

int beg = cur + ti;

while ((beg < tex.Count) && (beg >= 0) && (t[beg] >= t[cur]))

{

if ((t[beg] <= t[ma])/\*&& (t[beg] != t[cur])\*/&& (tex[beg] != ")"))

{

ma = beg;

}

beg = beg + ti;

}

return ma;

}

bool findpov()

{

bool rez = true;

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

int len=tex.Count;

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

{

if (tex[i][0] == 'a' || tex[i][0] == 'b' || tex[i][0] == 'c' || tex[i][0] == 'd')

{

if (prov.Contains(tex[i]))

{

rez = false;

}

else { prov.Add(tex[i]); }

}

}

return rez;

}

private void button8\_Click(object sender, EventArgs e)

{

string sz;

if(findpov())

{ sz = "Повторов нет, все идентификаторы разные."; }

else { sz = "В тексте есть повторы!"; }

textBox3.Text = sz;

}

public bool ischisl(string a)

{

bool rez = true;

try

{

string s = (Convert.ToInt32(a)).ToString();

}

catch

{

rez = false;

}

return rez;

}

public bool provsem()

{

int cur = 0;

string se;

int chi = 0;

bool rez = true;

int lenmin = tex[cur].Length;

int koe = koef[cur];

int caret = cur + 1;

while (cur < tex.Count)

{

caret = cur + 1;

if (tex[cur].Contains('a') || tex[cur].Contains('b') || tex[cur].Contains('c') || tex[cur].Contains('d'))

{

lenmin = tex[cur].Length;

while ((caret < koef.Length) && (koef[caret] <= koef[cur]+5))

{

if (ischisl(tex[caret]))

{

chi = Convert.ToInt32(tex[caret]);

if (!(lenmin <= chi))

{

string s = "Число должно быть меньше, чем длина слова. Ошибка на позиции: " + caret + " с символом " + tex[caret];

MessageBox.Show(s);

rez = false;

}

}

caret++;

}

}

cur++;

}

return rez;

}

private void button9\_Click(object sender, EventArgs e)

{

if (provsem())

{

textBox3.Text = "Проверка успешна, ошибок нет";

}

else

{

textBox3.Text = "Ошибка! Проверка не пройдена!";

}

}

}

}

Form1.Designer.cs

namespace Tyap

{

partial class Form1

{

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

private System.ComponentModel.IContainer components = null;

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

/// <param name="disposing">истинно, если управляемый ресурс должен быть удален; иначе ложно.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Код, автоматически созданный конструктором форм Windows

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

private void InitializeComponent()

{

this.textBox1 = new System.Windows.Forms.TextBox();

this.textBox2 = new System.Windows.Forms.TextBox();

this.button1 = new System.Windows.Forms.Button();

this.tbFMessage2 = new System.Windows.Forms.TextBox();

this.button2 = new System.Windows.Forms.Button();

this.button3 = new System.Windows.Forms.Button();

this.button4 = new System.Windows.Forms.Button();

this.button5 = new System.Windows.Forms.Button();

this.listBox1 = new System.Windows.Forms.ListBox();

this.listBox2 = new System.Windows.Forms.ListBox();

this.label1 = new System.Windows.Forms.Label();

this.label2 = new System.Windows.Forms.Label();

this.tbFMessage1 = new System.Windows.Forms.TextBox();

this.tbFSource1 = new System.Windows.Forms.TextBox();

this.tbFSource2 = new System.Windows.Forms.TextBox();

this.treeView1 = new System.Windows.Forms.TreeView();

this.button6 = new System.Windows.Forms.Button();

this.listBox3 = new System.Windows.Forms.ListBox();

this.label3 = new System.Windows.Forms.Label();

this.treeView2 = new System.Windows.Forms.TreeView();

this.button7 = new System.Windows.Forms.Button();

this.button8 = new System.Windows.Forms.Button();

this.textBox3 = new System.Windows.Forms.TextBox();

this.button9 = new System.Windows.Forms.Button();

this.SuspendLayout();

//

// textBox1

//

this.textBox1.Location = new System.Drawing.Point(69, 61);

this.textBox1.Margin = new System.Windows.Forms.Padding(2);

this.textBox1.Multiline = true;

this.textBox1.Name = "textBox1";

this.textBox1.Size = new System.Drawing.Size(216, 48);

this.textBox1.TabIndex = 0;

//

// textBox2

//

this.textBox2.Location = new System.Drawing.Point(69, 134);

this.textBox2.Margin = new System.Windows.Forms.Padding(2);

this.textBox2.Multiline = true;

this.textBox2.Name = "textBox2";

this.textBox2.Size = new System.Drawing.Size(216, 57);

this.textBox2.TabIndex = 1;

//

// button1

//

this.button1.Location = new System.Drawing.Point(69, 312);

this.button1.Margin = new System.Windows.Forms.Padding(2);

this.button1.Name = "button1";

this.button1.Size = new System.Drawing.Size(56, 19);

this.button1.TabIndex = 2;

this.button1.Text = "Check";

this.button1.UseVisualStyleBackColor = true;

this.button1.Click += new System.EventHandler(this.button1\_Click);

//

// tbFMessage2

//

this.tbFMessage2.Location = new System.Drawing.Point(69, 210);

this.tbFMessage2.Multiline = true;

this.tbFMessage2.Name = "tbFMessage2";

this.tbFMessage2.Size = new System.Drawing.Size(216, 37);

this.tbFMessage2.TabIndex = 3;

//

// button2

//

this.button2.Location = new System.Drawing.Point(48, 377);

this.button2.Name = "button2";

this.button2.Size = new System.Drawing.Size(75, 23);

this.button2.TabIndex = 4;

this.button2.Text = "Dobavit";

this.button2.UseVisualStyleBackColor = true;

this.button2.Click += new System.EventHandler(this.button2\_Click);

//

// button3

//

this.button3.Location = new System.Drawing.Point(129, 377);

this.button3.Name = "button3";

this.button3.Size = new System.Drawing.Size(75, 23);

this.button3.TabIndex = 5;

this.button3.Text = "Naity";

this.button3.UseVisualStyleBackColor = true;

this.button3.Click += new System.EventHandler(this.button3\_Click);

//

// button4

//

this.button4.Location = new System.Drawing.Point(210, 377);

this.button4.Name = "button4";

this.button4.Size = new System.Drawing.Size(75, 23);

this.button4.TabIndex = 6;

this.button4.Text = "Udalit";

this.button4.UseVisualStyleBackColor = true;

this.button4.Click += new System.EventHandler(this.button4\_Click);

//

// button5

//

this.button5.Location = new System.Drawing.Point(441, 70);

this.button5.Name = "button5";

this.button5.Size = new System.Drawing.Size(75, 23);

this.button5.TabIndex = 7;

this.button5.Text = "Zamena";

this.button5.UseVisualStyleBackColor = true;

this.button5.Click += new System.EventHandler(this.button5\_Click);

//

// listBox1

//

this.listBox1.FormattingEnabled = true;

this.listBox1.Location = new System.Drawing.Point(349, 338);

this.listBox1.Name = "listBox1";

this.listBox1.Size = new System.Drawing.Size(120, 212);

this.listBox1.TabIndex = 10;

//

// listBox2

//

this.listBox2.FormattingEnabled = true;

this.listBox2.Location = new System.Drawing.Point(475, 338);

this.listBox2.Name = "listBox2";

this.listBox2.Size = new System.Drawing.Size(120, 212);

this.listBox2.TabIndex = 11;

//

// label1

//

this.label1.AutoSize = true;

this.label1.Location = new System.Drawing.Point(349, 319);

this.label1.Name = "label1";

this.label1.Size = new System.Drawing.Size(36, 13);

this.label1.TabIndex = 12;

this.label1.Text = "word1";

//

// label2

//

this.label2.AutoSize = true;

this.label2.Location = new System.Drawing.Point(475, 318);

this.label2.Name = "label2";

this.label2.Size = new System.Drawing.Size(36, 13);

this.label2.TabIndex = 13;

this.label2.Text = "word2";

//

// tbFMessage1

//

this.tbFMessage1.Location = new System.Drawing.Point(416, 134);

this.tbFMessage1.Multiline = true;

this.tbFMessage1.Name = "tbFMessage1";

this.tbFMessage1.Size = new System.Drawing.Size(144, 77);

this.tbFMessage1.TabIndex = 14;

//

// tbFSource1

//

this.tbFSource1.Location = new System.Drawing.Point(584, 89);

this.tbFSource1.Multiline = true;

this.tbFSource1.Name = "tbFSource1";

this.tbFSource1.Size = new System.Drawing.Size(137, 20);

this.tbFSource1.TabIndex = 15;

//

// tbFSource2

//

this.tbFSource2.Location = new System.Drawing.Point(584, 150);

this.tbFSource2.Multiline = true;

this.tbFSource2.Name = "tbFSource2";

this.tbFSource2.Size = new System.Drawing.Size(137, 20);

this.tbFSource2.TabIndex = 16;

//

// treeView1

//

this.treeView1.Location = new System.Drawing.Point(727, 61);

this.treeView1.Name = "treeView1";

this.treeView1.Size = new System.Drawing.Size(168, 489);

this.treeView1.TabIndex = 17;

//

// button6

//

this.button6.Location = new System.Drawing.Point(48, 463);

this.button6.Name = "button6";

this.button6.Size = new System.Drawing.Size(75, 23);

this.button6.TabIndex = 18;

this.button6.Text = "Tree";

this.button6.UseVisualStyleBackColor = true;

this.button6.Click += new System.EventHandler(this.button6\_Click);

//

// listBox3

//

this.listBox3.FormattingEnabled = true;

this.listBox3.Location = new System.Drawing.Point(601, 338);

this.listBox3.Name = "listBox3";

this.listBox3.Size = new System.Drawing.Size(120, 212);

this.listBox3.TabIndex = 19;

//

// label3

//

this.label3.AutoSize = true;

this.label3.Location = new System.Drawing.Point(598, 312);

this.label3.Name = "label3";

this.label3.Size = new System.Drawing.Size(36, 13);

this.label3.TabIndex = 20;

this.label3.Text = "word3";

this.label3.Click += new System.EventHandler(this.label3\_Click);

//

// treeView2

//

this.treeView2.Location = new System.Drawing.Point(902, 61);

this.treeView2.Name = "treeView2";

this.treeView2.Size = new System.Drawing.Size(170, 489);

this.treeView2.TabIndex = 21;

//

// button7

//

this.button7.Location = new System.Drawing.Point(129, 463);

this.button7.Name = "button7";

this.button7.Size = new System.Drawing.Size(75, 23);

this.button7.TabIndex = 22;

this.button7.Text = "Modify tree";

this.button7.UseVisualStyleBackColor = true;

this.button7.Click += new System.EventHandler(this.button7\_Click);

//

// button8

//

this.button8.Location = new System.Drawing.Point(211, 463);

this.button8.Name = "button8";

this.button8.Size = new System.Drawing.Size(75, 35);

this.button8.TabIndex = 23;

this.button8.Text = "Check identificaors";

this.button8.UseVisualStyleBackColor = true;

this.button8.Click += new System.EventHandler(this.button8\_Click);

//

// textBox3

//

this.textBox3.Location = new System.Drawing.Point(211, 505);

this.textBox3.Multiline = true;

this.textBox3.Name = "textBox3";

this.textBox3.Size = new System.Drawing.Size(132, 45);

this.textBox3.TabIndex = 24;

//

// button9

//

this.button9.Location = new System.Drawing.Point(48, 517);

this.button9.Name = "button9";

this.button9.Size = new System.Drawing.Size(94, 44);

this.button9.TabIndex = 25;

this.button9.Text = "Syntacsis check";

this.button9.UseVisualStyleBackColor = true;

this.button9.Click += new System.EventHandler(this.button9\_Click);

//

// Form1

//

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.ClientSize = new System.Drawing.Size(1106, 573);

this.Controls.Add(this.button9);

this.Controls.Add(this.textBox3);

this.Controls.Add(this.button8);

this.Controls.Add(this.button7);

this.Controls.Add(this.treeView2);

this.Controls.Add(this.label3);

this.Controls.Add(this.listBox3);

this.Controls.Add(this.button6);

this.Controls.Add(this.treeView1);

this.Controls.Add(this.tbFSource2);

this.Controls.Add(this.tbFSource1);

this.Controls.Add(this.tbFMessage1);

this.Controls.Add(this.label2);

this.Controls.Add(this.label1);

this.Controls.Add(this.listBox2);

this.Controls.Add(this.listBox1);

this.Controls.Add(this.button5);

this.Controls.Add(this.button4);

this.Controls.Add(this.button3);

this.Controls.Add(this.button2);

this.Controls.Add(this.tbFMessage2);

this.Controls.Add(this.button1);

this.Controls.Add(this.textBox2);

this.Controls.Add(this.textBox1);

this.Margin = new System.Windows.Forms.Padding(2);

this.Name = "Form1";

this.Text = "Form1";

this.ResumeLayout(false);

this.PerformLayout();

}

#endregion

private System.Windows.Forms.TextBox textBox1;

private System.Windows.Forms.TextBox textBox2;

private System.Windows.Forms.Button button1;

private System.Windows.Forms.TextBox tbFMessage2;

private System.Windows.Forms.Button button2;

private System.Windows.Forms.Button button3;

private System.Windows.Forms.Button button4;

private System.Windows.Forms.Button button5;

private System.Windows.Forms.ListBox listBox1;

private System.Windows.Forms.ListBox listBox2;

private System.Windows.Forms.Label label1;

private System.Windows.Forms.Label label2;

private System.Windows.Forms.TextBox tbFMessage1;

private System.Windows.Forms.TextBox tbFSource1;

private System.Windows.Forms.TextBox tbFSource2;

private System.Windows.Forms.TreeView treeView1;

private System.Windows.Forms.Button button6;

private System.Windows.Forms.ListBox listBox3;

private System.Windows.Forms.Label label3;

private System.Windows.Forms.TreeView treeView2;

private System.Windows.Forms.Button button7;

private System.Windows.Forms.Button button8;

private System.Windows.Forms.TextBox textBox3;

private System.Windows.Forms.Button button9;

}

}

derevo.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Tyap

{

public class derevo

{

public string a;

public derevo son = null;

public derevo Brat = null;

public void adbra(derevo f, string t)

{

f.Brat = new derevo();

f.Brat.a = t;

}

public void adson(derevo f, string t)

{

//MessageBox.Show("Ads");

f.son = new derevo();

f.son.a = t;

}

}

}

Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Tyap

{

internal static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

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 Tyap

{

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] % 100 + 100);

}

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 + i) % (Int32)(arrFHashTable.Count);

}

} while (!boolVFinish);

return hi;

}

public void HashIndex(string strALexicalUnit)

{

int h;

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

if (h == 0) { h=arrFHashTable.Count-1; }

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)

{

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

{

//MessageBox.Show(strALexicalUnit);

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.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Tyap

{

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

public enum TCharType

{

Letter, Digit, EndRow, EndText, Space, ReservedSymbol, opBracket, clBracket, expMark,

comma, semicolon, dobdot, plus, zv, min

}; // тип символа

public enum TToken

{

lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs,

lxmDot, lxmComma, lxmMinus, lxmplus, lxmzv, lxmravn, lxmdobdot, lxmVoskl

};

class CLex

{

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.zv;

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 == ':') enumFSelectionCharType = TCharType.dobdot;

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

else if (chrFSelection == '-') enumFSelectionCharType = TCharType.min;

//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();

}

// Вариант 15

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// A | | | B | |

// B | |CFin| | |

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

A:

{

if (chrFSelection == 'c')

{

TakeSymbol();

goto B;

}

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

}

B:

{

if (chrFSelection == 'b')

{

TakeSymbol();

goto CFin;

}

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

}

CFin:

{

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

{

TakeSymbol();

goto CFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

}

case TCharType.Digit:

{

// 0 1

// A | B | |

// B | D | C |

// C | A | |

// D | |EFin |

// EFin | F | |

// F | G | |

// G | EFin| |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto BC;

}

else throw new Exception("Ожидался 0");

BC:

if (chrFSelection == '0')

{

TakeSymbol();

goto E;

}

if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

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

D:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидался 0");

E:

if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидалась 1");

FFin:

if (chrFSelection == '0')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидалась 0");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 0");

H:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0");

}

case TCharType.ReservedSymbol:

{

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

case TCharType.Space:

{

GetSymbol();

break;

}

case TCharType.opBracket:

{

enumFToken = TToken.lxmLeftParenth;

strPLexicalUnit = "(";

GetSymbol();

break;

}

case TCharType.clBracket:

{

enumFToken = TToken.lxmRightParenth;

GetSymbol();

strPLexicalUnit = ")";

break;

}

case TCharType.expMark:

{

enumFToken = TToken.lxmVoskl;

strPLexicalUnit = "!";

GetSymbol();

break;

}

case TCharType.plus:

{

enumFToken = TToken.lxmplus;

strPLexicalUnit = "+";

GetSymbol();

break;

}

case TCharType.dobdot:

{

enumFToken = TToken.lxmdobdot;

strPLexicalUnit = ":";

GetSymbol();

break;

}

case TCharType.zv:

{

enumFToken = TToken.lxmzv;

strPLexicalUnit = "\*";

GetSymbol();

break;

}

case TCharType.min:

{

enumFToken = TToken.lxmMinus;

strPLexicalUnit = "-";

GetSymbol();

break;

}

}

}

}

}

uSyntAnalyzer.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Tyap

{

class uSyntAnalyzer

{

string s;

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 derevo w;

public uSyntAnalyzer()

{

strFSource = new String[0];

strFMessage = new String[0];

strPSource = new String[0];

strPMessage = new String[0];

w = new derevo();

}

public uSyntAnalyzer(derevo q)

{

strFSource = new String[0];

strFMessage = new String[0];

strPSource = new String[0];

strPMessage = new String[0];

w = q;

}

public void S(derevo q)

{

if (Lex.enumPToken == TToken.lxmLeftParenth)

{

q.adson(q, "(");

Lex.NextToken();

q.son.adbra(q.son, "D");

D(q.son.Brat);

if (Lex.enumPToken == TToken.lxmVoskl)

{

q.son.Brat.adbra(q.son.Brat, "!");

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmdobdot)

{

q.son.Brat.Brat.adbra(q.son.Brat.Brat, ":");

Lex.NextToken();

q.son.Brat.Brat.Brat.adbra(q.son.Brat.Brat.Brat, "S");

S(q.son.Brat.Brat.Brat.Brat);

if (Lex.enumPToken == TToken.lxmVoskl)

{

q.son.Brat.Brat.Brat.Brat.adbra(q.son.Brat.Brat.Brat.Brat, "!");

Lex.NextToken();

q.son.Brat.Brat.Brat.Brat.Brat.adbra(q.son.Brat.Brat.Brat.Brat.Brat, "S");

S(q.son.Brat.Brat.Brat.Brat.Brat.Brat);

if (Lex.enumPToken == TToken.lxmRightParenth)

{

q.son.Brat.Brat.Brat.Brat.Brat.Brat.adbra(q.son.Brat.Brat.Brat.Brat.Brat.Brat, ")");

Lex.NextToken();

}

else throw new Exception("Ожидалcя )");

}

}

else throw new Exception("Ожидалcя :");

}

else throw new Exception("Ожидалcя !");

}

else

if (Lex.enumPToken == TToken.lxmNumber)

{

q.adson(q, Lex.strPLexicalUnit);

Lex.NextToken();

}

else { throw new Exception("Ожидалcя ( или число"); }

}

public void D(derevo q)

{

q.adson(q, "K");

K(q.son);

if (Lex.enumPToken == TToken.lxmplus)

{

q.son.adbra(q.son, "+");

q.son.Brat.adbra(q.son.Brat, "C");

C(q.son.Brat.Brat);

}

}

public void C(derevo q)

{

if (Lex.enumPToken == TToken.lxmplus)

{

Lex.NextToken();

q.adson(q, "K");

K(q.son);

if (Lex.enumPToken == TToken.lxmplus)

{

q.son.adbra(q.son, "+");

C(q.son.Brat);

}

}

}

public void K(derevo q)

{

q.adson(q, "A");

A(q.son);

if (Lex.enumPToken == TToken.lxmzv)

{

q.son.adbra(q.son, "\*");

q.son.Brat.adbra(q.son.Brat, "P");

P(q.son.Brat.Brat);

}

}

public void P(derevo q)

{

if (Lex.enumPToken == TToken.lxmzv)

{

Lex.NextToken();

q.adson(q, "A");

A(q.son);

if (Lex.enumPToken == TToken.lxmzv)

{

q.son.adbra(q.son, "\*");

q.son.Brat.adbra( q.son.Brat, "P");

P(q.son.Brat.Brat);

}

}

}

public void A(derevo q)

{

if (Lex.enumPToken == TToken.lxmIdentifier)

{

q.adson(q, Lex.strPLexicalUnit);

Lex.NextToken();

}

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

{

q.adson(q, "(");

Lex.NextToken();

q.son.adbra(q.son, "D");

D(q.son.Brat);

if (Lex.enumPToken == TToken.lxmRightParenth)

{

q.son.Brat.adbra(q.son.Brat, ")");

Lex.NextToken();

}

else throw new Exception("Ожидалась )");

}

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

{

q.adson(q, "-");

Lex.NextToken();

A(q.son);

}

else

{

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

}

}

}

}