Министерство образования Российской Федерации

Федеральное агентство по образованию

КАЗАНСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. А.Н. Туполева - КАИ

Кафедра АСОИУ

Лабораторная работа №1-8

по дисциплине

«ТЕОРИЯ фОРМАЛЬНЫХ ГРАММАТИК И АВТОМАТОВ»

Выполнила:

Студентка группы 4309

Газимзянова А.Д.

Проверила: Бикмуллина И.И.

Казань 2023

**Лабораторная работа № 1. Разработка транслитератора**

**Текст задания:**

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

**Теория. Основные правила разработки программ**

Разработка программ имеет множество правил и рекомендаций, которые помогают создать эффективный и надежный код. Вот основные правила разработки программ:

1. Используйте понятные и описательные имена переменных, функций и классов. Хорошие имена помогают другим разработчикам легко понять ваш код и улучшают его читаемость. Например, вместо x, y, z предпочтительнее использовать более осмысленные имена, такие как width, height, radius.

2. Разбивайте код на модули или функции с определенной функциональностью. Это улучшает читаемость кода и позволяет повторно использовать функции в разных частях программы.

3. Следуйте принципу единственной ответственности (Single Responsibility Principle). Каждая функция или класс должны быть ответственными только за одну часть функциональности. Это делает код более модульным и легко поддерживаемым.

4. Используйте комментарии для пояснения сложного или непонятного кода. Хорошо написанные комментарии помогут другим разработчикам понять ваш код и его цель. Однако, избегайте комментирования очевидных вещей и старайтесь писать понятный код без необходимости комментариев.

5. Следуйте стандартам и рекомендациям языка программирования, которым вы пользуетесь. Различные языки программирования имеют свои собственные стандарты и рекомендации, которые помогают создавать качественный код.

6. Пишите тесты для своего кода. Тестирование помогает обнаружить ошибки и убедиться в правильной работе программы. Существуют различные подходы к тестированию, такие как модульное тестирование, функциональное тестирование и интеграционное тестирование.

7. Используйте систему контроля версий, такую как Git, для отслеживания изменений в коде и совместной работы с другими разработчиками. Система контроля версий позволяет вам сохранять историю изменений, легко возвращаться к предыдущим версиям и объединять изменения от разных разработчиков.

8. Обрабатывайте ошибки и исключения. Ваш код должен быть устойчивым к возможным ошибкам и исключительным ситуациям. Обработка ошибок помогает предотвратить сбои программы и предоставить пользователю информацию о возникшей проблеме.

9. Оптимизируйте код только по необходимости. Оптимизация должна быть основана на реальных проблемах производительности и должна проводиться только после профилирования и выявления узких мест в коде.

**Код программы:**

**Form1.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp53

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

// Инициализация текстового поля с примером текста.

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

textBox1.AppendText("1 a");

int n = textBox1.Lines.Length;

}

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

{

}

// Обработчик нажатия кнопки "button1".

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

{

// Создание объекта лексического анализатора.

uLex Lex = new uLex();

// Установка исходного текста и текстового поля для вывода сообщений.

Lex.strPSource = textBox1.Lines;

Lex.strPMessage = textBox2.Lines;

int x = textBox1.TextLength;

int y = textBox1.Lines.Length;

textBox2.Text = "";

try

{

// Цикл анализа и классификации символов.

while (Lex.enumPState != TState.Finish)

{

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

Lex.NextToken();

String s = "";

String s1 = "";

// Определение типа символа для вывода в сообщении.

switch (Lex.enumFSelectionCharType)

{

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

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

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

case TCharType.clBracket: { s1 = "ClosedBracket"; break; }

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

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

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

case TCharType.tchkZap: { s1 = "TochkaZapyataya"; break; }

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

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

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

}

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

textBox2.Text += m; // Добавление в строку сообщения.

}

}

catch (Exception exc)

{

textBox2.Text += exc.Message;

textBox1.Select();

textBox1.SelectionStart = 0;

int n = 0;

// Определение положения ошибки в исходном коде.

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 WindowsFormsApp53

{

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

public enum TCharType { Letter, Digit, EndRow, EndText, Space, ReservedSymbol, opBracket, clBracket, expMark, comma, tchkZap, dot }; // Тип символа

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

class uLex

{

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

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

public TCharType enumFSelectionCharType;

public char chrFSelection;

private TState enumFState;

private int intFSourceRowSelection;

private int intFSourceColSelection;

private String strFLexicalUnit;

private TToken enumFToken;

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

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

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

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

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

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

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

public uLex()

{

}

// Метод для получения следующего символа из исходного кода.

public void GetSymbol()

{

// Продвигаем номер колонки.

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

{

intFSourceRowSelection++;

if (intFSourceRowSelection <= strFSource.Length - 1)

{

intFSourceColSelection = -1;

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndRow;

enumFState = TState.Continue;

}

else

{

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndText;

enumFState = TState.Finish;

}

}

else

{

chrFSelection = strFSource[intFSourceRowSelection][intFSourceColSelection];

// Классификация прочитанной литеры.

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

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

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

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

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

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

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

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

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

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

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

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

enumFState = TState.Continue;

}

intFSourceColSelection++;

}

// Приватный метод для добавления символа в текущую лексическую единицу.

private void TakeSymbol()

{

char[] c = { chrFSelection };

String s = new string(c);

strFLexicalUnit += s;

GetSymbol();

}

// Метод для получения следующей лексической единицы (токена).

public void NextToken()

{

strFLexicalUnit = "";

// Если состояние начальное, инициализируем счетчики и получаем первый символ.

if (enumFState == TState.Start)

{

intFSourceRowSelection = 0;

intFSourceColSelection = -1;

GetSymbol();

}

// Если символ '/' - возможно начало комментария.

if (chrFSelection == '/')

{

GetSymbol();

// Если следующий символ тоже '/', это комментарий до конца строки.

if (chrFSelection == '/')

{

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

}

GetSymbol();

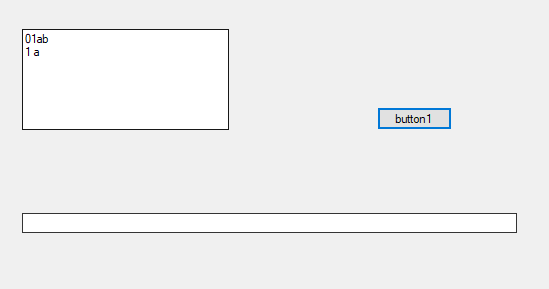
}

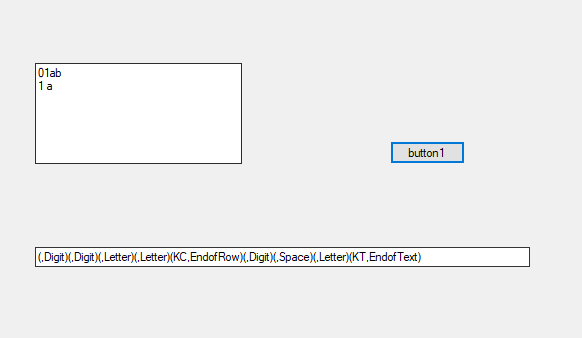
}

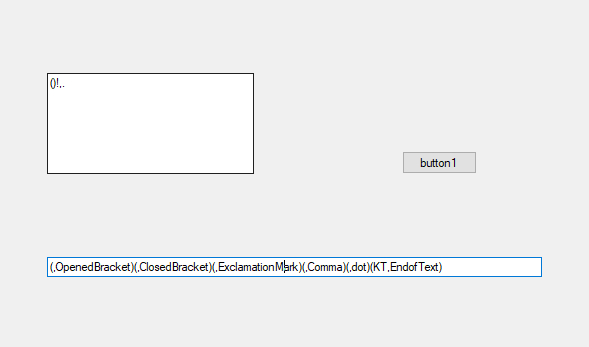
}

}

**Результаты тестирования:**







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

**Теория**

*Лексический анализатор* (или токенизатор, ЛА) — это программа или часть программы, которая выполняет лексический анализ. Лексический анализ — это процесс разбора входной последовательности символов на распознанные группы, называемые лексемами, с целью получить на выходе идентифицированные последовательности, которые называются "токенами".

Лексический анализатор обычно работает в две стадии: сканирование и оценка. На стадии сканирования ЛА обычно реализуется в виде конечного автомата, определяемого регулярными выражениями. Это позволяет кодировать информацию о возможных последовательностях символов, которые могут встречаться в токенах. На стадии оценки, для получения токена со значением, соответствующим типу (например, целое или дробное число), выполняется проход по символам и вычисление значения.

*Детерминированный граф* — граф, в котором для каждой пары вершин существует только одно ребро, связывающее их. Другими словами, для каждой вершины графа определены все ее соседние вершины. Такой граф представляет собой конечное множество вершин, каждая из которых связана с другими вершинами ребрами, и каждое ребро имеет направление или не имеет направления.

*Недетерминированный граф* — граф, в котором для некоторых пар вершин может существовать более одного ребра, связывающего их. Это означает, что для некоторых вершин графа не определены все ее соседние вершины, и между некоторыми парами вершин может быть несколько ребер.

*Матрица* — это двухмерная таблица чисел, которую можно представить как массив массивов. Матрицы обычно обозначаются большими буквами (например, A, B, C). Каждый элемент матрицы можно идентифицировать с помощью двух индексов: одного для строки и одного для столбца, в которых он находится. Например, элемент A[i][j] находится в i-й строке и j-й столбце матрицы A.

*Существует несколько основных операций над матрицами:*

1. Сложение и вычитание матриц: Две матрицы можно сложить или вычесть, если они имеют одинаковые размеры. Результатом будет новая матрица того же размера, каждый элемент которой равен сумме (или разности) соответствующих элементов исходных матриц.
2. Умножение матрицы на число (скаляр): Каждый элемент матрицы умножается на данное число.
3. Умножение матриц: Матрицу A размером m x n можно умножить на матрицу B размером n x p. Результатом будет новая матрица C размером m x p, где каждый элемент C[i][j] вычисляется как сумма произведений соответствующих элементов i-й строки матрицы A и j-го столбца матрицы B.
4. Транспонирование матрицы: Транспонированная матрица получается путем перестановки строк и столбцов исходной матрицы. То есть, если A[i][j] - элемент исходной матрицы, то в транспонированной матрице этот элемент будет находиться в позиции A[j][i].

**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.

**Вариант задания:**



**Первое слово:**

(000)\*010(100)\*

A → 0B

B → 0C|1D

C → 0A

D → 0|0F

F → 1G

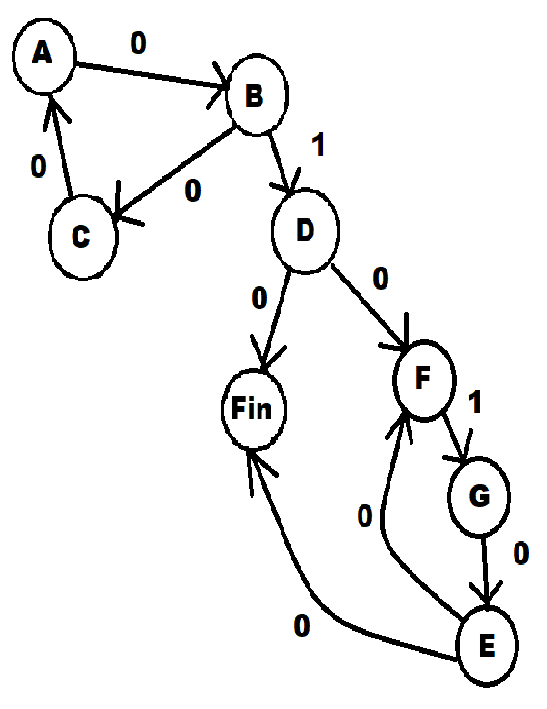
G → 0E

E → 0|0F

**1. Недетерминированная матрица:**

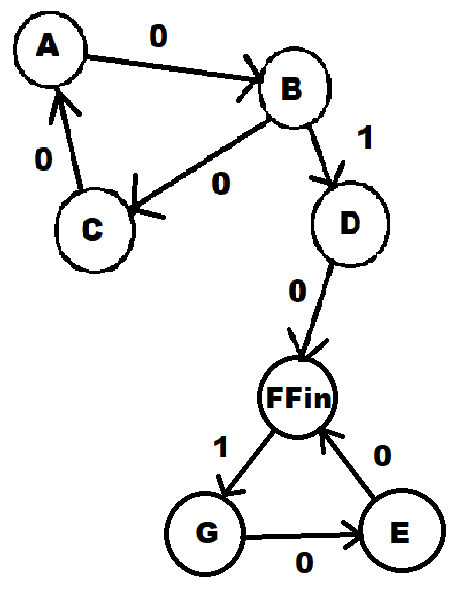
|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| A | B |  |
| B | C | D |
| C | A |  |
| D | F,Fin |  |
| E | F,Fin |  |
| F |  | G |
| G | E |  |

**Граф:**



**2. Детерминированная матрица: Граф:**

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| A | B |  |
| B | C | D |
| C | A |  |
| D | FFin |  |
| E | FFin |  |
| FFin |  | G |
| G | E |  |

****

**Второе слово:**

(a|b|c|d)+

Не должно начинаться с ab

A → a|b|c|d| aB|bC|cC|dC

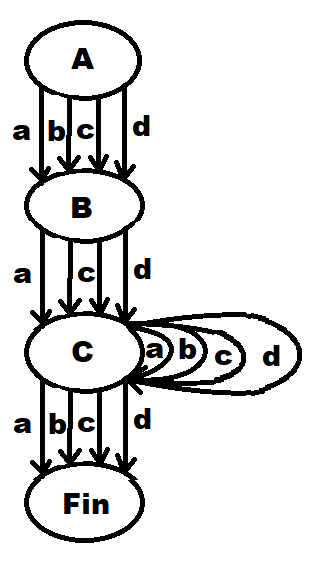
B → a|c|d| aC|cC|dC

C → a|b|c|d| aC|bC|cC|dC

**1. Недетерминированная матрица:**

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

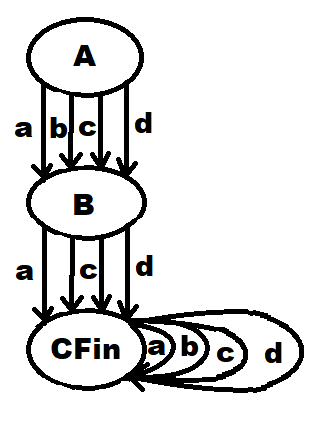
**Граф:**



**2. Детерминированная матрица:**

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

**Граф:**



**Код программы:**

**Form1.cs**

using System;

using System.Windows.Forms;

namespace WindowsFormsApp46

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

textBox1.AppendText("acd\r\n");

textBox1.AppendText("000010100");

}

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

{

}

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

{

uLex Lex = new uLex();

Lex.strPSource = textBox1.Lines;

Lex.strPMessage = textBox2.Lines;

Lex.enumPState = TState.Start;

textBox2.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

}

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;

namespace WindowsFormsApp46

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

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

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

public class uLex

{

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

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

public TCharType enumFSelectionCharType;

public char chrFSelection;

private TState enumFState;

private int intFSourceRowSelection;

private int intFSourceColSelection;

private String strFLexicalUnit;

private TToken enumFToken;

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

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

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

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

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

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

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

public uLex()

{

}

public void GetSymbol() //метод класса лексический анализатор

{

// продвигаем номер колонки

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

{

intFSourceRowSelection++;

if (intFSourceRowSelection <= strFSource.Length - 1)

{

intFSourceColSelection = -1;

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndRow;

enumFState = TState.Continue;

}

Else

{

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndText;

enumFState = TState.Finish;

}

}

Else

{

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

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

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

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

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

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

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

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

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

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

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

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

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 4

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// A | CFin |CFin|CFin| CFin |

// B | CFin | |CFin| CFin |

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

A:

{

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

{

TakeSymbol();

goto B;

}

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

}

B:

{

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

{

TakeSymbol();

goto CFin;

}

else throw new Exception("Слово не должно начинаться с ab");

}

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 | C | D |

// C | A | |

// D |FFin | |

// E |FFin | |

// FFin | | G |

// G | E | |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto B;

}

else throw new Exception("Ожидался 0");

B:

if (chrFSelection == '0')

{

TakeSymbol();

goto C;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

else throw new Exception("Ожидался 0");

C:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидался 0");

D:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0");

FFin:

if (chrFSelection == '1')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидался 1");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto E;

}

else throw new Exception("Ожидался 0");

E:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

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

else throw new Exception("Ожидался 0");

}

case TCharType.ReservedSymbol:

{

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

{

while (enumFSelectionCharType != TCharType.EndRow)

GetSymbol();

}

GetSymbol();

}

if (chrFSelection == '(')

{

enumFToken = TToken.lxmLeftParenth;

GetSymbol();

return;

}

if (chrFSelection == ')')

{

enumFToken = TToken.lxmRightParenth;

GetSymbol();

return;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

}

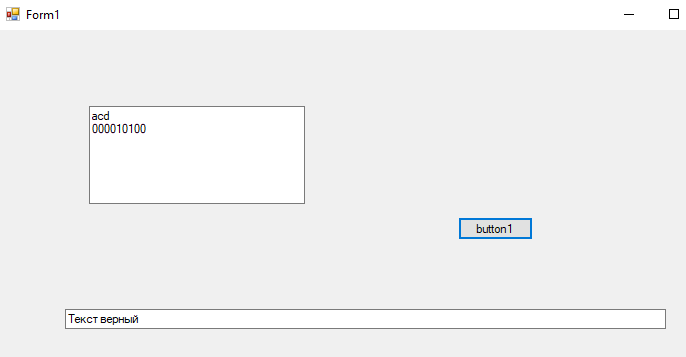
}

}

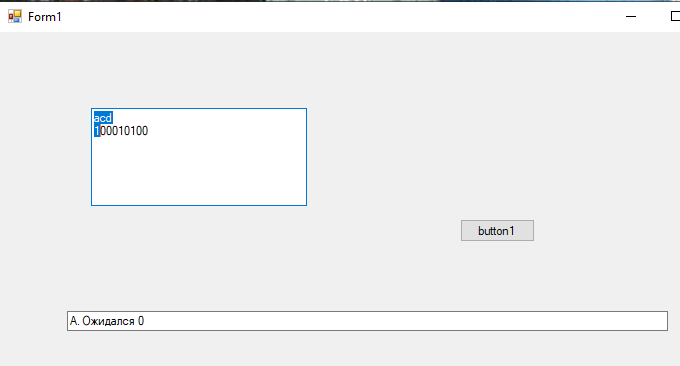
}

**Результат работы программы:**

Верно:



С ошибкой:



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

**Теория**

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

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

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

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

A -> Aα | β

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

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

A -> βA'

A' -> αA' | ε

где ε обозначает пустую строку.

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

**Текст задания:**

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

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

Указания:

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

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

3. Если грамматика леворекурсивная, то устранить левую рекурсию.

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

**Грамматика (вариант 4):**

S → A :− B .

A → <2> (<1>)

B → B , A

B → A

B → B, A — левая рекурсия

**Избавление от левой рекурсии:**

S → A :− B .

A → <2> (<1>)

B → A | AC

C → , A | , AC

1. S → A :− B

**B** → <2> (<1>) :− A → **<2> (<1>) :− <2> (<1>)**

2. S → A :− B

**A** → A :− B , A → **<2> (<1>) :− A , <2> (<1>)** →

→ **<2> (<1>) :− <2> (<1>) , <2> (<1>)**

**Код программы:**

**Form1.cs**

using System;

using System.Windows.Forms;

using Lex;

using Synt;

namespace WindowsFormsApp2

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

int n = textBox1.Lines.Length;

textBox1.Text = "acd(000010100):-acd(000010100)";

}

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;

namespace Lex

{

public enum TState { Start, Continue, Finish };

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

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

public class uLex

{

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

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

public TCharType enumFSelectionCharType;

private 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; } }

public int intPSourceColSelection { get { return intFSourceColSelection; } }

public uLex(){}

public void GetSymbol(bool check = false) //метод класса лексический анализатор

{

intFSourceColSelection++; // продвигаем номер колонки

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

{

intFSourceRowSelection++;

if (intFSourceRowSelection <= strFSource.Length - 1)

{

intFSourceColSelection = -1;

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndRow;

enumFState = TState.Continue;

}

else

{

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndText;

enumFState = TState.Finish;

}

}

else

{

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

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

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

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

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

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

else if (chrFSelection == '(' || chrFSelection == ')' || chrFSelection == ',' || chrFSelection == '.' || chrFSelection == '/') 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(true);

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol(true);

}

GetSymbol(true);

}

// Вариант 4

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// A | CFin |CFin|CFin| CFin |

// B | CFin | |CFin| CFin |

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

A:

{

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

{

TakeSymbol();

goto B;

}

else throw new Exception("Слово не должно начинаться с ab");

}

B:

{

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

{

TakeSymbol();

goto CFin;

}

else throw new Exception("Слово не должно начинаться с ab");

}

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 | C | D |

// C | A | |

// D |FFin | |

// E |FFin | |

// G | E | |

// FFin | | G |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto B;

}

else throw new Exception("Ожидался 0");

B:

if (chrFSelection == '0')

{

TakeSymbol();

goto C;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

else throw new Exception("Ожидался 0");

C:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидался 0");

D:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0");

FFin:

if (chrFSelection == '1')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидалась 1");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto E;

}

else throw new Exception("Ожидался 0");

E:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

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

else throw new Exception("Ожидался 0");

}

case TCharType.ReservedSymbol:

{

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

{

while (enumFSelectionCharType != TCharType.EndRow)

GetSymbol();

}

GetSymbol();

}

if (chrFSelection == '(')

{

enumFToken = TToken.lxmLeftParenth;

GetSymbol();

return;

}

if (chrFSelection == ')')

{

enumFToken = TToken.lxmRightParenth;

GetSymbol();

return;

}

if (chrFSelection == '[')

{

enumFToken = TToken.lxmIs;

GetSymbol();

return;

}

if (chrFSelection == ']')

{

enumFToken = TToken.lxmrs;

GetSymbol();

return;

}

if (chrFSelection == ',')

{

enumFToken = TToken.lxmComma;

GetSymbol();

return;

}

if (chrFSelection == ':')

{

enumFToken = TToken.lxmdt;

GetSymbol();

return;

}

if (chrFSelection == '-')

{

enumFToken = TToken.lxmMinus;

GetSymbol();

return;

}

if (chrFSelection == '.')

{

enumFToken = TToken.lxmDot;

GetSymbol();

return;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

}

}

}

}

**uSyntAnalyzer.cs**

using System;

using Lex;

namespace Synt

{

class uSyntAnalyzer

{

private String[] strFSource;

private String[] strFMessage;

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

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

public uLex Lex = new uLex();

public void S()

{

A();

if (Lex.enumPToken == TToken.lxmdt)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmMinus)

{

Lex.NextToken();

B();

if (Lex.enumPToken == TToken.lxmDot)

{

Lex.NextToken();

}

}

else throw new Exception("Ожидался минус");

}

else throw new Exception("Ожидалось двоеточие");

}

public void A()

{

if (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmLeftParenth)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmRightParenth)

{

Lex.NextToken();

}

else throw new Exception("Ожидалась )");

}

else throw new Exception("Ожидался числовой идентификатор");

}

else throw new Exception("Ожидалась (");

}

else throw new Exception("Ожидался буквенный идентификатор");

}

public void B()

{

A();

if (Lex.enumPToken == TToken.lxmComma)

C();

}

public void C()

{

if (Lex.enumPToken == TToken.lxmComma)

{

Lex.NextToken();

A();

if (Lex.enumPToken == TToken.lxmComma)

C();

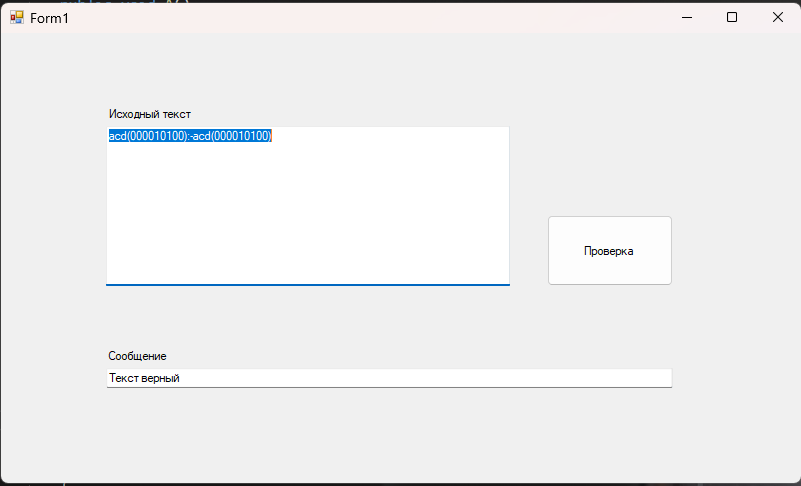
}

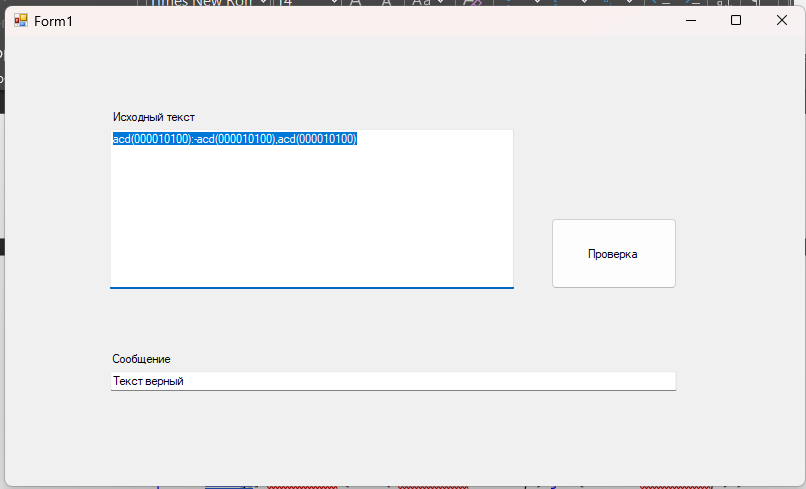
}

}

}

**Результаты работы программы:**





**Лабораторная работа № 4.**

**Введение табличного способа хранения слов**

**Теория**

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

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

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

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

**Текст задания:**

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

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

План работы:

1. Подключить класс «Массив хеш-таблиц» к программе;

2. Завести три таблицы для хранения слов первого типа, слов второго типа и служебных слов (многосимвольных);

3. Отладить программу до рабочего состояния.

**Код программы:**

**Form1.cs**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

public partial class Form1 : Form

{

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

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

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

public MyHashFunction hashFunction = new MyHashFunction();

public CHashTableList htl = new CHashTableList(2);

public Form1()

{

InitializeComponent();

tbFSource.AppendText("acd(000010100):-acd(000010100)");

int n = tbFSource.Lines.Length;

}

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

{

}

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

{

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

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

//htl.

htl.TableToStringList(0, listTable);

foreach (var entry in hashTableIdentifier)

{

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

}

listTable.Clear();

htl.TableToStringList(1, listTable);

foreach (var entry in hashTableDigital)

{

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

}

listTable.Clear();

foreach (var entry in hashTableRezerv)

{

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

}

listTable.Clear();

}

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

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.Lines;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.S();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Synt.Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

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

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmdt):

{

hashFunction.AddWord(hashTableRezerv, ":");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmComma):

{

hashFunction.AddWord(hashTableRezerv, ",");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmMinus):

{

hashFunction.AddWord(hashTableRezerv, "-");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmLeftParenth):

{

hashFunction.AddWord(hashTableRezerv, "(");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmRightParenth):

{

hashFunction.AddWord(hashTableRezerv, ")");

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

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

{

TablesToMemo(this, e);

}

break;

}

}

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

tbFMessage.Text += m;

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

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

{

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

{

searchBtn.BackColor = Color.Green;

}

else

{

searchBtn.BackColor = Color.Red;

}

}

private void resetBtnsColor()

{

searchBtn.ResetBackColor();

searchBtn.UseVisualStyleBackColor = true;

addBtn.ResetBackColor();

addBtn.UseVisualStyleBackColor = true;

deleteBtn.ResetBackColor();

deleteBtn.UseVisualStyleBackColor = true;

changeBtn.ResetBackColor();

changeBtn.UseVisualStyleBackColor = true;

}

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

{

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

resetBtnsColor();

foreach (var entry in hashTableIdentifier)

{

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

}

foreach (var entry in hashTableDigital)

{

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

}

foreach (var entry in hashTableRezerv)

{

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

}

}

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

{

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

addBtn.BackColor = Color.Green;

}

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

{

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

{

deleteBtn.BackColor = Color.Green;

}

else

{

deleteBtn.BackColor= Color.Red;

}

}

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

{

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

{

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

changeBtn.BackColor = Color.Green;

}

else

{

changeBtn.BackColor= Color.Red;

}

}

}

}

**CLex.cs**

using System;

namespace WindowsFormsApp4

{

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

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

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

public class CLex //класс лексический анализатор

{

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

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

private String strFLexicalUnit;

public char chrFSelection;

private int intFSourceRowSelection;

private int intFSourceColSelection = -1;

public TCharType enumFSelectionCharType;

private TState enumFState;

private TToken enumFToken;

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

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

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

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

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(bool check = false) //метод класса лексический анализатор

{

intFSourceColSelection++; // продвигаем номер колонки

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

{

intFSourceRowSelection++;

if (intFSourceRowSelection <= strFSource.Length - 1)

{

intFSourceColSelection = -1;

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndRow;

enumFState = TState.Continue;

}

else

{

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndText;

enumFState = TState.Finish;

}

}

else

{

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

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

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

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

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

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

else if (chrFSelection == '(' || chrFSelection == ')' || chrFSelection == ',' || chrFSelection == '.' || chrFSelection == '/') 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(true);

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol(true);

}

GetSymbol(true);

}

// Вариант 4

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// A | CFin |CFin|CFin| CFin |

// B | CFin | |CFin| CFin |

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

A:

{

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

{

TakeSymbol();

goto B;

}

else throw new Exception("Слово не должно начинаться с ab");

}

B:

{

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

{

TakeSymbol();

goto CFin;

}

else throw new Exception("Слово не должно начинаться с ab");

}

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 | C | D |

// C | A | |

// D |FFin | |

// E |FFin | |

// G | E | |

// FFin | | G |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto B;

}

else throw new Exception("Ожидался 0");

B:

if (chrFSelection == '0')

{

TakeSymbol();

goto C;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

else throw new Exception("Ожидался 0");

C:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидался 0");

D:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0");

FFin:

if (chrFSelection == '1')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидалась 1");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto E;

}

else throw new Exception("Ожидался 0");

E:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

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

else throw new Exception("Ожидался 0");

}

case TCharType.ReservedSymbol:

{

if (chrFSelection == '/')

{

TakeSymbol();

if (chrFSelection == '/')

{

while (enumFSelectionCharType != TCharType.EndRow)

TakeSymbol();

}

TakeSymbol();

}

if (chrFSelection == '(')

{

enumFToken = TToken.lxmLeftParenth;

TakeSymbol();

return;

}

if (chrFSelection == ')')

{

enumFToken = TToken.lxmRightParenth;

TakeSymbol();

return;

}

if (chrFSelection == '[')

{

enumFToken = TToken.lxmIs;

TakeSymbol();

return;

}

if (chrFSelection == ']')

{

enumFToken = TToken.lxmrs;

TakeSymbol();

return;

}

if (chrFSelection == ',')

{

enumFToken = TToken.lxmComma;

TakeSymbol();

return;

}

if (chrFSelection == ':')

{

enumFToken = TToken.lxmdt;

TakeSymbol();

return;

}

if (chrFSelection == '-')

{

enumFToken = TToken.lxmMinus;

TakeSymbol();

return;

}

if (chrFSelection == '.')

{

enumFToken = TToken.lxmDot;

TakeSymbol();

return;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

}

}

}

}

**uSyntAnalyzer.cs**

using System;

namespace WindowsFormsApp4

{

class uSyntAnalyzer

{

private String[] strFSource;

private String[] strFMessage;

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

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

public CLex Lex = new CLex();

public void S()

{

A();

if (Lex.enumPToken == TToken.lxmdt)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmMinus)

{

Lex.NextToken();

B();

if (Lex.enumPToken == TToken.lxmDot)

{

Lex.NextToken();

}

}

else throw new Exception("Ожидался минус");

}

else throw new Exception("Ожидалось двоеточие");

}

public void A()

{

if (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmLeftParenth)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmRightParenth)

{

Lex.NextToken();

}

else throw new Exception("Ожидалась )");

}

else throw new Exception("Ожидался числовой идентификатор");

}

else throw new Exception("Ожидалась (");

}

else throw new Exception("Ожидался буквенный идентификатор");

}

public void B()

{

A();

if (Lex.enumPToken == TToken.lxmComma)

C();

}

public void C()

{

if (Lex.enumPToken == TToken.lxmComma)

{

Lex.NextToken();

A();

if (Lex.enumPToken == TToken.lxmComma)

C();

}

}

}

}

**Hash.Tables.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

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;

}

UInt32 HashFunction\_Wainberger(string strALexicalUnit)

{

UInt32 h = 0, g;

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

{

h = (h << 4) + strALexicalUnit[i];

g = h & 0xF0000000;

if (h != 0)

{

h ^= g >> 24 ^ g;

}

}

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 + 1) % (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);

}

}

}

**HashTablesList.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

public class CHashTableList

{

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

private 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 CHashTableList(string strAFileName)

{

try

{

boolFIsLoaded = Load(strAFileName);

}

catch (InvalidCastException)

{

MessageBox.Show("Ошибка при восстановлении хеш-таблиц из файла!");

boolFIsLoaded = false;

}

}

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

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; }

}

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

public bool Load(string strAFileName)

{

boolFIsLoaded = false;

try

{

StreamReader sr = new StreamReader(strAFileName);

byteFTablesSize = Convert.ToByte(sr.ReadLine());

if (byteFTablesSize < 1 || byteFTablesSize > 16)

{

MessageBox.Show("Unbelivable count of tables: " + byteFTablesSize.ToString());

return boolFIsLoaded;

}

arrFHashTableList.Clear();

Resize(arrFHashTableList, byteFTablesSize/\*+1\*/);

string line = sr.ReadLine();

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

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

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

{

arrFHashTableList[i].Init(Convert.ToInt32(counts[i]));

}

objFHeap.Load(ref sr);

sr.Close();

int n = objFHeap.arrFHeapTable.Count;

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

{

THeapItem Item = objFHeap.arrFHeapTable[i];

if (Item.strFLexicalUnit.Length == 0)

break;

arrFHashTableList[Item.byteFHashTable].arrFHashTable[Item.intFHashIndex] = i;

}

boolFIsLoaded = true;

}

catch (InvalidDataException)

{

MessageBox.Show("Ошибка при восстановлении из файла хеш-таблиц!"); boolFIsLoaded = false;

}

return boolFIsLoaded;

}

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

// 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;

}

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

}

}

**THeap.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

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);

}

}

}

**MyHashFunction.cs**

using System;

using System.Collections.Generic;

namespace WindowsFormsApp4

{

public class MyHashFunction

{

public int HashFunction(string word)

{

int hashValue = 0;

foreach (char c in word)

{

hashValue += (int)c;

}

return hashValue;

}

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

{

int hashValue = HashFunction(word);

if (!hashTable.ContainsKey(hashValue))

{

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

}

else

{

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

{

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

return;

}

}

hashTable[hashValue].Add(word);

}

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

{

int hashValue = HashFunction(word);

if (hashTable.ContainsKey(hashValue))

{

return 1;

}

else

{

return 0;

}

}

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

{

int hashValue = HashFunction(word);

if (hashTable.ContainsKey(hashValue))

{

List<string> words = hashTable[hashValue];

if (words.Contains(word))

{

words.Remove(word);

if (words.Count == 0)

{

hashTable.Remove(hashValue);

}

return true;

}

}

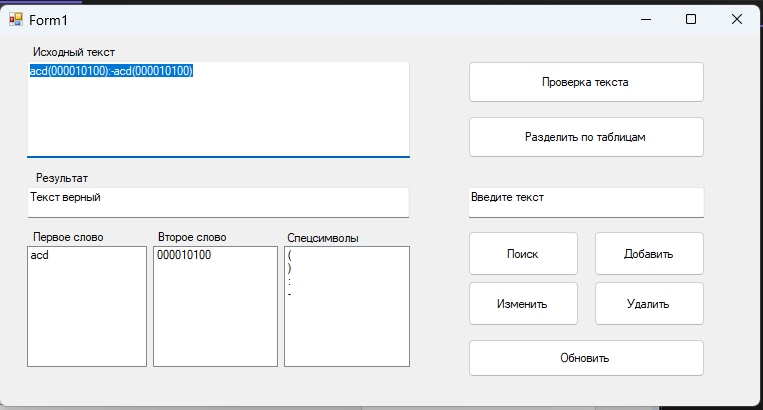
return false;

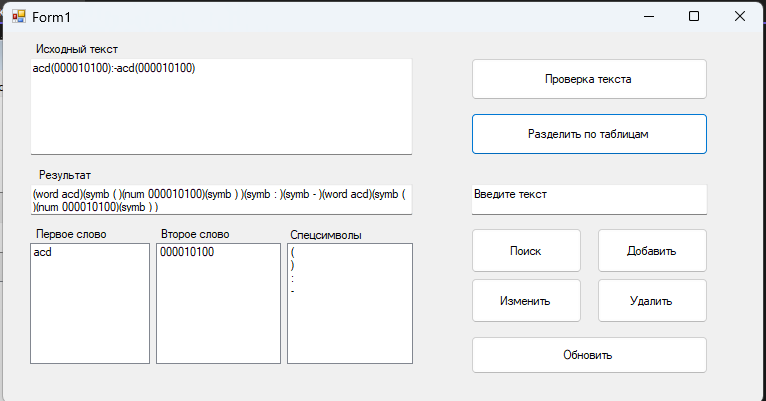
}

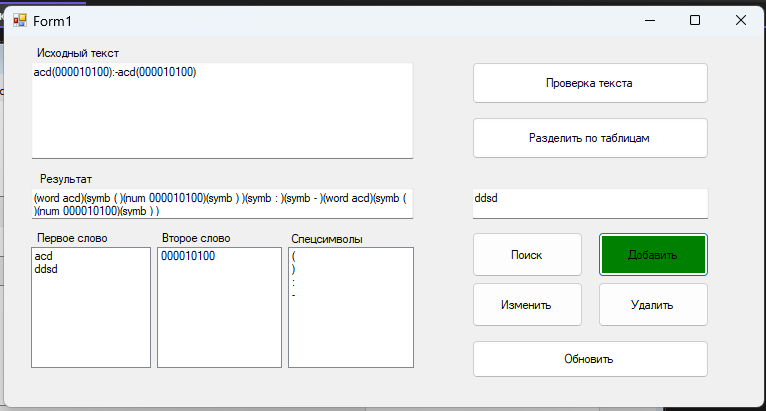
}

}

**Результаты работы программы:**







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

**Задание:**

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

**Теория:**

*Синтаксическое дерево* (или абстрактное синтаксическое дерево, AST) — это конечное, помеченное и ориентированное дерево, где внутренние вершины сопоставлены с операторами языка программирования, а листья — с соответствующими операндами. То есть листья представляют переменные и константы, а внутренние узлы — операторы.

Синтаксические деревья используются в синтаксических анализаторах для промежуточного представления программы между деревом разбора и структурой данных, которая затем используется в качестве внутреннего представления в компиляторе или интерпретаторе программы для оптимизации и генерации кода.

Формирование синтаксического дерева представляет собой ключевой этап синтаксического анализа, который трансформирует текст программы или входные данные в структуру данных, абстрагирующую синтаксическую организацию данного текста. Это структурированное дерево обеспечивает удобный анализ и обработку синтаксической структуры программы, а также возможность проведения различных видов статического анализа или генерации кода.

Основные этапы построения синтаксического дерева:

1. Лексический анализ (токенизация):

- Производится лексический анализ входного текста или кода, разбивая его на лексемы, представляющие собой минимальные логические единицы, такие как идентификаторы, ключевые слова, операторы и числа.

2. Синтаксический анализ:

- Применяется синтаксический анализатор для создания структуры синтаксического дерева на основе лексем.

- Синтаксический анализатор использует контекстно-свободную грамматику, определенную для языка, для определения структуры программы и построения соответствующего синтаксического дерева.

- Обработка грамматических правил, определенных в грамматике языка, результатом которой является дерево, где узлы представляют синтаксические конструкции, а дуги отражают связи между ними.

3. Построение синтаксического дерева:

- В ходе синтаксического анализа происходит формирование синтаксического дерева.

- Корень дерева обычно представляет программу в целом, а узлы – конкретные синтаксические конструкции (выражения, операторы, условия и т. д.).

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

4. Оптимизация дерева (по необходимости):

- В некоторых случаях синтаксическое дерево может быть упрощено или оптимизировано путем удаления ненужных узлов, представляющих промежуточные вычисления.

5. Использование синтаксического дерева:

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

- Синтаксическое дерево также может служить основой для создания абстрактных семантических деревьев (ASD), представляющих более высокоуровневое представление смысла программы.

*TreeView* – это инструментарий пользовательского интерфейса в графических окружениях, предназначенный для отображения и эффективного управления иерархическими данными в виде древовидной структуры. В его структурированном варианте узлы дерева грамотно организованы и устанавливают иерархические связи между собой.

Основные компоненты структурированного представления TreeView:

1. Узлы: Каждый узел представляет собой ключевой элемент в древовидной структуре и может включать в себя как данные, так и другие дочерние узлы.

2. Дерево: Объединение всех узлов и их взаимосвязи формируют древовидную структуру, которая отображается в виде иерархической композиции.

3. Разделители и иконки: TreeView применяет разделители для наглядного отображения уровней вложенности и использует иконки для ясной идентификации типа узлов, например, папок или файлов.

Структурированный TreeView эффективно применяется для визуализации разнообразных данных, включая файловые системы, структуры документов, иерархии каталогов и прочее. Пользователи имеют возможность разворачивать и сворачивать ветви дерева, обеспечивая удобный просмотр и управление информацией.

**Код программы:**

**Form1:**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

public partial class Form1 : Form

{

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

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

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

public MyHashFunction hashFunction = new MyHashFunction();

public CHashTableList htl = new CHashTableList(2);

public CTree ctTree;

public Form1()

{

InitializeComponent();

tbFSource.AppendText("acd(000010100):-acd(000010100)");

int n = tbFSource.Lines.Length;

}

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

{

}

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

{

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

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

//htl.

htl.TableToStringList(0, listTable);

foreach (var entry in hashTableIdentifier)

{

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

}

listTable.Clear();

htl.TableToStringList(1, listTable);

foreach (var entry in hashTableDigital)

{

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

}

listTable.Clear();

foreach (var entry in hashTableRezerv)

{

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

}

listTable.Clear();

}

private void resetBtnsColor()

{

searchBtn.ResetBackColor();

searchBtn.UseVisualStyleBackColor = true;

addBtn.ResetBackColor();

addBtn.UseVisualStyleBackColor = true;

deleteBtn.ResetBackColor();

deleteBtn.UseVisualStyleBackColor = true;

changeBtn.ResetBackColor();

changeBtn.UseVisualStyleBackColor = true;

}

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

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.Lines;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.Parser();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Synt.Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

treeView1.Nodes.Clear();

treeView1.Show();

Synt.ctTree.ShowTree(Synt.ctTree, treeView1);

}

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

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmdt):

{

hashFunction.AddWord(hashTableRezerv, ":");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmComma):

{

hashFunction.AddWord(hashTableRezerv, ",");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmMinus):

{

hashFunction.AddWord(hashTableRezerv, "-");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmLeftParenth):

{

hashFunction.AddWord(hashTableRezerv, "(");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmRightParenth):

{

hashFunction.AddWord(hashTableRezerv, ")");

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

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

{

TablesToMemo(this, e);

}

break;

}

}

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

tbFMessage.Text += m;

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

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

{

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

{

searchBtn.BackColor = Color.Green;

}

else

{

searchBtn.BackColor = Color.Red;

}

}

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

{

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

addBtn.BackColor = Color.Green;

}

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

{

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

{

deleteBtn.BackColor = Color.Green;

}

else

{

deleteBtn.BackColor = Color.Red;

}

}

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

{

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

{

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

changeBtn.BackColor = Color.Green;

}

else

{

changeBtn.BackColor = Color.Red;

}

}

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

{

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

resetBtnsColor();

foreach (var entry in hashTableIdentifier)

{

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

}

foreach (var entry in hashTableDigital)

{

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

}

foreach (var entry in hashTableRezerv)

{

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

}

}

}

}

**uSyntAnalyser.cs:**

using System;

namespace WindowsFormsApp4

{

class uSyntAnalyzer

{

private String[] strFSource;

private String[] strFMessage;

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

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

public CLex Lex = new CLex();

public CTree ctTree;

public void Parser()

{

ctTree = new CTree();

ctTree.strFLabel = "S";

AdditionS(ctTree);

}

void AdditionS(CTree ctANode)

{

ctANode.AddSon(ctANode, "A");

AdditionA(ctANode.ctFSon);

if (Lex.enumPToken == TToken.lxmdt)

{

ctANode.ctFSon.AddBrother(ctANode.ctFSon, ":");

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmMinus)

{

ctANode.ctFSon.ctFBrother.AddBrother(ctANode.ctFSon.ctFBrother, "-");

Lex.NextToken();

ctANode.ctFSon.ctFBrother.ctFBrother.AddBrother(ctANode.ctFSon.ctFBrother.ctFBrother, "B");

AdditionB(ctANode.ctFSon.ctFBrother.ctFBrother.ctFBrother);

if (Lex.enumPToken == TToken.lxmDot)

{

ctANode.ctFSon.ctFBrother.ctFBrother.ctFBrother.AddBrother(ctANode.ctFSon.ctFBrother.ctFBrother.ctFBrother, ".");

Lex.NextToken();

}

}

else throw new Exception("Ожидался минус");

}

else throw new Exception("Ожидалось двоеточие");

}

void AdditionA(CTree ctANode)

{

if (Lex.enumPToken == TToken.lxmIdentifier)

{

ctANode.AddSon(ctANode, Lex.strPLexicalUnit);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmLeftParenth)

{

ctANode.ctFSon.AddBrother(ctANode.ctFSon, "(");

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

ctANode.ctFSon.ctFBrother.AddBrother(ctANode.ctFSon.ctFBrother, Lex.strPLexicalUnit);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmRightParenth)

{

ctANode.ctFSon.ctFBrother.ctFBrother.AddBrother(ctANode.ctFSon.ctFBrother.ctFBrother, ")");

Lex.NextToken();

}

else throw new Exception("Ожидалась )");

}

else throw new Exception("Ожидался числовой идентификатор");

}

else throw new Exception("Ожидалась (");

}

else throw new Exception("Ожидался буквенный идентификатор");

}

void AdditionB(CTree ctANode)

{

ctANode.AddSon(ctANode, "A");

AdditionA(ctANode.ctFSon);

if (Lex.enumPToken == TToken.lxmComma)

{

ctANode.ctFSon.AddBrother(ctANode.ctFSon, ",");

ctANode.ctFSon.ctFBrother.AddBrother(ctANode.ctFSon.ctFBrother, "C");

AdditionC(ctANode.ctFSon.ctFBrother.ctFBrother);

}

else throw new Exception("Ожидалась ,");

}

public void AdditionC(CTree ctANode)

{

if (Lex.enumPToken == TToken.lxmComma)

{

ctANode.AddSon(ctANode, ",");

Lex.NextToken();

ctANode.ctFSon.AddBrother(ctANode.ctFSon, "A");

AdditionA(ctANode.ctFSon.ctFBrother);

ctANode.ctFSon.ctFBrother.ctFBrother.AddBrother(ctANode.ctFSon.ctFBrother.ctFBrother, "C");

AdditionC(ctANode.ctFSon.ctFBrother.ctFBrother.ctFBrother);

}

else throw new Exception("Ожидалось ,");

}

}

}

**NodeTree.cs:**

using System;

using System.Collections.Generic;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

public class CTree

{

public CTree() { }

public string strFLabel;

public CTree ctFSon;

public CTree ctFBrother;

public void AddSon(CTree ctNode, String strALabel)

{

ctNode.ctFSon = new CTree();

ctNode.ctFSon.strFLabel = strALabel;

}

public void AddBrother(CTree ctNode, String strALabel)

{

ctNode.ctFBrother = new CTree();

ctNode.ctFBrother.strFLabel = strALabel;

}

struct CStackItem

{

public CTree ctNode;

public TreeNode tnNode;

}

public void ShowTree(CTree ctARoot, TreeView tvATree)

{

List<CStackItem> Stack = new List<CStackItem>(0);

CStackItem StackItem = new CStackItem();

CTree ctSelected = ctARoot;

tvATree.Nodes.Add(ctARoot.strFLabel);

TreeNode tnSelected = tvATree.Nodes[0];

while (ctSelected != null)

{

if (ctSelected.ctFSon != null)

{

tnSelected.Nodes.Add(ctSelected.ctFSon.strFLabel);

if (ctSelected.ctFBrother != null)

{

tnSelected.Parent.Nodes.Add(ctSelected.ctFBrother.strFLabel);

StackItem.ctNode = ctSelected.ctFBrother;

StackItem.tnNode = tnSelected.Parent.LastNode;

Stack.Add(StackItem);

}

ctSelected = ctSelected.ctFSon;

tnSelected = tnSelected.FirstNode;

}

else if (ctSelected.ctFBrother != null)

{

ctSelected = ctSelected.ctFBrother;

tnSelected.Parent.Nodes.Add(ctSelected.strFLabel);

tnSelected = tnSelected.NextNode;

}

else

{

int top = Stack.Count;

if (top > 0)

{

ctSelected = Stack[top - 1].ctNode;

tnSelected = Stack[top - 1].tnNode;

Stack.RemoveAt(top - 1);

}

else ctSelected = null;

}

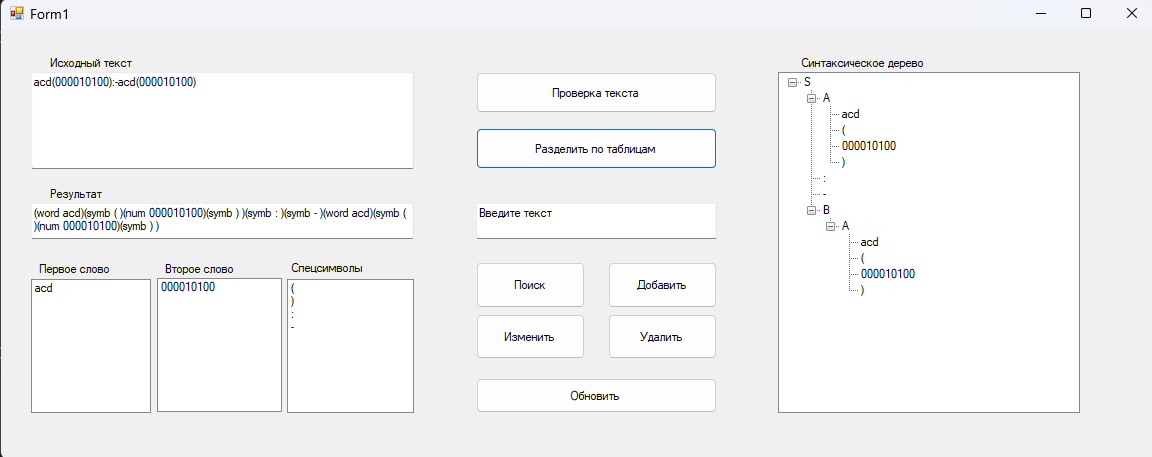
}

}

}

}

**Результат тестирования:**



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

**Задание:**

1. Перевести все числа в десятичное представление.

2. Выполнить вывод исходного текста в структурированном виде.

**Теория:**

*Атрибутивная грамматика* — это формальный подход к определению атрибутов для создания формальной грамматики. Она связывает эти атрибуты со значениями, вычисляемыми в узлах абстрактного синтаксического дерева при обработке языка синтаксическим анализатором или компилятором.

Атрибуты делятся на две группы: синтезированные и унаследованные. Синтезированные атрибуты являются результатом правил оценки атрибутов и могут использовать значения унаследованных атрибутов. Унаследованные атрибуты передаются от родительских узлов.

В некоторых подходах, синтезированные атрибуты используются для передачи семантической информации вверх по дереву синтаксического анализа, в то время как унаследованные атрибуты помогают передавать семантическую информацию вниз и через него.

Атрибутивные грамматики могут использоваться для перевода синтаксического дерева непосредственно в код для некоторой конкретной машины или на некоторый промежуточный язык.

С помощью атрибутивных грамматик можно передавать информацию из любого места абстрактного синтаксического дерева в любое другое место контролируемым и формальным способом.

*Генератор* в контексте программирования представляет собой программный инструмент или устройство, способное автоматически создавать код, данные или другие выходные результаты на основе определенных исходных данных, спецификаций или правил. Генераторы используются для автоматизации и упрощения процессов разработки, а также для обеспечения консистентности и эффективности в создаваемых продуктах.

Проектирование генератора относится к разработке программного инструмента, предназначенного для автоматизации и упрощения процессов создания кода или других выходных результатов на основе предоставленных исходных данных или спецификаций. Этот подход включает в себя ряд этапов:

1. Определение основных исходных данных или спецификаций, на которых будет основан процесс генерации.

2. Разработка алгоритма или логики, лежащих в основе процесса генерации.

3. Создание генератора, который эффективно реализует предложенный алгоритм.

4. Проведение тестирования и отладки генератора для обеспечения его надежной и эффективной работы.

5. Интеграция генератора в рабочий процесс разработки или другие инструменты.

*Данный ниже код не является генератором*, так как он представляет собой программу, выполняющую конкретные задачи, в том числе взаимодействие с пользовательским интерфейсом и обработку входных данных. Генератор, в контексте программирования, обычно относится к инструменту, который автоматически создает код, данные или другие выходные результаты на основе определенных исходных данных или спецификаций.

Отсутствует следующие этапы, характерные для генераторов:

3. Создание генератора. Генератор как инструмент должен быть явно разработан для реализации алгоритма генерации.

4. Проведение тестирования и отладки. Обычно этот этап включает в себя специфические шаги для проверки корректности создаваемых выходных данных. Здесь это не реализовано.

5. Интеграция генератора. Генераторы обычно интегрируются в рабочий процесс разработки или другие инструменты. Данный код никуда не интегрируется.

**Код программы:**

**Form1.cs:**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

public partial class Form1 : Form

{

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

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

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

public MyHashFunction hashFunction = new MyHashFunction();

public CHashTableList htl = new CHashTableList(2);

public CTree ctTree;

public Form1()

{

InitializeComponent();

tbFSource.AppendText("acd(000010100):-acd(000010100),acd(000010100)");

int n = tbFSource.Lines.Length;

}

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

{

}

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

{

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

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

//htl.

htl.TableToStringList(0, listTable);

foreach (var entry in hashTableIdentifier)

{

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

}

listTable.Clear();

htl.TableToStringList(1, listTable);

foreach (var entry in hashTableDigital)

{

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

}

listTable.Clear();

foreach (var entry in hashTableRezerv)

{

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

}

listTable.Clear();

}

private void resetBtnsColor()

{

searchBtn.ResetBackColor();

searchBtn.UseVisualStyleBackColor = true;

addBtn.ResetBackColor();

addBtn.UseVisualStyleBackColor = true;

deleteBtn.ResetBackColor();

deleteBtn.UseVisualStyleBackColor = true;

changeBtn.ResetBackColor();

changeBtn.UseVisualStyleBackColor = true;

}

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

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.Lines;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.Parser();

throw new Exception("Текст верный!!!");

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Synt.Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

treeView1.Nodes.Clear();

treeView1.Show();

Synt.ctTree.ShowTree(Synt.ctTree, treeView1);

treeView1.ExpandAll();

}

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

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmdt):

{

hashFunction.AddWord(hashTableRezerv, ":");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmComma):

{

hashFunction.AddWord(hashTableRezerv, ",");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmMinus):

{

hashFunction.AddWord(hashTableRezerv, "-");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmLeftParenth):

{

hashFunction.AddWord(hashTableRezerv, "(");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmRightParenth):

{

hashFunction.AddWord(hashTableRezerv, ")");

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

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

{

TablesToMemo(this, e);

}

break;

}

}

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

tbFMessage.Text += m;

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

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

{

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

{

searchBtn.BackColor = Color.Green;

}

else

{

searchBtn.BackColor = Color.Red;

}

}

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

{

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

addBtn.BackColor = Color.Green;

}

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

{

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

{

deleteBtn.BackColor = Color.Green;

}

else

{

deleteBtn.BackColor = Color.Red;

}

}

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

{

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

{

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

changeBtn.BackColor = Color.Green;

}

else

{

changeBtn.BackColor = Color.Red;

}

}

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

{

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

resetBtnsColor();

foreach (var entry in hashTableIdentifier)

{

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

}

foreach (var entry in hashTableDigital)

{

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

}

foreach (var entry in hashTableRezerv)

{

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

}

treeView1.Nodes.Clear();

}

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

{

CopyTreeView(treeView1, treeView2);

Converter converter = new Converter();

converter.ConvertBinaryToDecimal(treeView2);

treeView2.ExpandAll();

}

private void CopyTreeNodes(TreeNodeCollection sourceNodes, TreeNodeCollection targetNodes)

{

foreach (TreeNode sourceNode in sourceNodes)

{

TreeNode newNode = new TreeNode(sourceNode.Text);

targetNodes.Add(newNode);

CopyTreeNodes(sourceNode.Nodes, newNode.Nodes); // Рекурсивно копируем дочерние узлы

}

}

private void CopyTreeView(TreeView sourceTreeView, TreeView targetTreeView)

{

targetTreeView.Nodes.Clear(); // Очищаем существующие узлы в целевом TreeView

CopyTreeNodes(sourceTreeView.Nodes, targetTreeView.Nodes);

}

}

}

**uSyntAnalyzer.cs:**

using System;

namespace WindowsFormsApp4

{

class uSyntAnalyzer

{

private String[] strFSource;

private String[] strFMessage;

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

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

public CLex Lex = new CLex();

public CTree ctTree;

public void Parser()

{

ctTree = new CTree();

ctTree.strFLabel = "S";

AdditionS(ctTree);

}

void AdditionS(CTree ctANode)

{

ctANode.AddSon(ctANode, "A");

AdditionA(ctANode.ctFSon);

if (Lex.enumPToken == TToken.lxmdt)

{

ctANode.ctFSon.AddBrother(ctANode.ctFSon, ":");

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmMinus)

{

ctANode.ctFSon.ctFBrother.AddBrother(ctANode.ctFSon.ctFBrother, "-");

Lex.NextToken();

ctANode.ctFSon.ctFBrother.ctFBrother.AddBrother(ctANode.ctFSon.ctFBrother.ctFBrother, "B");

AdditionB(ctANode.ctFSon.ctFBrother.ctFBrother.ctFBrother);

if (Lex.enumPToken == TToken.lxmDot)

{

ctANode.ctFSon.ctFBrother.ctFBrother.ctFBrother.AddBrother(ctANode.ctFSon.ctFBrother.ctFBrother.ctFBrother, ".");

Lex.NextToken();

}

}

else throw new Exception("Ожидался минус");

}

else throw new Exception("Ожидалось двоеточие");

}

void AdditionA(CTree ctANode)

{

if (Lex.enumPToken == TToken.lxmIdentifier)

{

ctANode.AddSon(ctANode, Lex.strPLexicalUnit);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmLeftParenth)

{

ctANode.ctFSon.AddBrother(ctANode.ctFSon, "(");

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

ctANode.ctFSon.ctFBrother.AddBrother(ctANode.ctFSon.ctFBrother, Lex.strPLexicalUnit);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmRightParenth)

{

ctANode.ctFSon.ctFBrother.ctFBrother.AddBrother(ctANode.ctFSon.ctFBrother.ctFBrother, ")");

Lex.NextToken();

}

else throw new Exception("Ожидалась )");

}

else throw new Exception("Ожидался числовой идентификатор");

}

else throw new Exception("Ожидалась (");

}

else throw new Exception("Ожидался буквенный идентификатор");

}

void AdditionB(CTree ctANode)

{

ctANode.AddSon(ctANode, "A");

AdditionA(ctANode.ctFSon);

if (Lex.enumPToken == TToken.lxmComma)

{

ctANode.ctFSon.AddBrother(ctANode.ctFSon, ",");

ctANode.ctFSon.ctFBrother.AddBrother(ctANode.ctFSon.ctFBrother, "C");

AdditionC(ctANode.ctFSon.ctFBrother.ctFBrother);

}

else throw new Exception("Ожидалась ,");

}

public void AdditionC(CTree ctANode)

{

if (Lex.enumPToken == TToken.lxmComma)

{

ctANode.AddSon(ctANode, ",");

Lex.NextToken();

ctANode.ctFSon.AddBrother(ctANode.ctFSon, "A");

AdditionA(ctANode.ctFSon.ctFBrother);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmComma)

{

ctANode.ctFSon.ctFBrother.ctFBrother.AddBrother(ctANode.ctFSon.ctFBrother.ctFBrother, "C");

AdditionC(ctANode.ctFSon.ctFBrother.ctFBrother.ctFBrother);

}

}

else throw new Exception("Ожидалось ,");

}

}

}

**Converter.cs:**

using System;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

internal class Converter

{

// Метод для конвертации всех двоичных чисел в десятичные внутри TreeView

public void ConvertBinaryToDecimal(TreeView tree)

{

foreach (TreeNode treeNodes in tree.Nodes)

{

ConvertBinaryToDecimal(treeNodes);

}

}

// Вспомогательный метод для рекурсивной конвертации чисел в десятичные

private void ConvertBinaryToDecimal(TreeNode treeNodes)

{

// Обработка каждого узла внутри текущего узла дерева

foreach (TreeNode treeNode in treeNodes.Nodes)

{

string binaryValue = treeNode.Text;

// Проверка, является ли значение в узле двоичным числом

if (IsBin(binaryValue))

{

// Конвертация двоичного числа в десятичное и замена значения в узле

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

treeNode.Text = decimalValue.ToString();

}

// Перестановка узлов, если количество узлов равно 4

if (treeNode.Nodes.Count == 4)

{

TreeNode temp = treeNode.Nodes[2];

treeNode.Nodes[2].Remove();

treeNode.Nodes.Insert(0, temp);

temp = treeNode.Nodes[1];

treeNode.Nodes[1].Remove();

treeNode.Nodes.Insert(2, temp);

}

}

// Рекурсивный вызов для обработки дочерних узлов

foreach (TreeNode treeNode1 in treeNodes.Nodes)

{

ConvertBinaryToDecimal(treeNode1);

}

}

// Метод для проверки, является ли строка двоичным числом

private bool IsBin(string value)

{

foreach (var c in value)

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

return false;

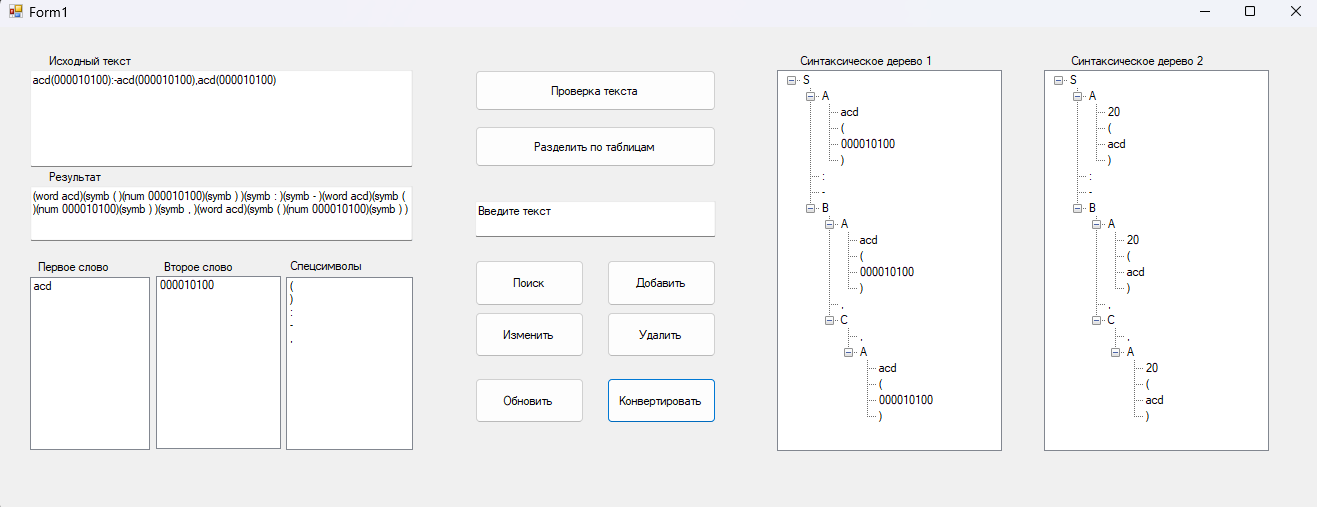
return true;

}

}

}

**Результаты тестирования:**



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

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

**Вариант 4:** все числа должны быть разными.

**Теория:**

*Контекстный анализатор* — это компонент компилятора, который проверяет синтаксическую структуру кода, учитывая контекст, в котором используются определенные элементы. Он проверяет, что каждый идентификатор используется в соответствии с его объявлениями, и что объекты и операции совместимы.

*Этапы разработки контекстного анализатора* охватывают несколько ключевых шагов, обеспечивающих корректное понимание и обработку программного кода:

1. Формулирование семантических правил: Инициируется определением набора семантических правил, характеризующих функциональность вашего языка программирования. Эти правила точно определяют разрешенные операции и действия в программе, а также накладываемые ограничения на типы данных и выражения.

2. Аннотация синтаксического дерева: В рамках контекстного анализа формируется синтаксическое дерево, выстраиваемое на основе результатов синтаксического анализа. Эта иерархическая структура программы дополняется аннотациями семантической информацией. К этим аннотациям относится привязка имен к их объявлениям (разрешение идентификаторов), определение типов данных выражений и другие проверки на семантическую корректность.

3. Проведение проверок на семантические ошибки: Контекстный анализатор осуществляет проверки с целью выявления семантических ошибок в программе. Эти ошибки включают в себя, например, попытки использования необъявленных переменных или присвоение значений переменным с неподходящими типами данных.

4. Генерация семантического дерева или промежуточного представления: При успешном прохождении проверок контекстный анализатор может сгенерировать семантическое дерево или промежуточное представление программы. Эти структуры будут использованы для последующего анализа и генерации кода.

5. Проверка соответствия типов данных: Один из важных аспектов контекстного анализа заключается в проверке соответствия типов данных. Контекстный анализатор выявляет типы данных выражений и операций, а также убеждается в их совместимости. Например, он проверяет, что операторы, применяемые к переменным или значениям, имеют совместимые типы.

6. Применение семантических правил: Контекстный анализатор активно применяет семантические правила языка программирования, включая ограничения на использование ключевых слов, операторов и структур данных.

7. Выдача сообщений об ошибках: В случае обнаружения нарушений семантических правил контекстный анализатор формирует сообщения об ошибках. Эти сообщения направлены на то, чтобы помочь разработчикам устранить проблемы в коде, повышая уровень его корректности и надежности.

**Код программы:**

**uSyntAnalyzer.cs:**

using System;

using System.Collections.Generic;

namespace WindowsFormsApp4

{

class uSyntAnalyzer

{

private String[] strFSource;

private String[] strFMessage;

private HashSet<string> encounteredNumbers = new HashSet<string>(); // Коллекция для отслеживания чисел

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

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

public CLex Lex = new CLex();

public CTree ctTree;

public void Parser()

{

ctTree = new CTree();

ctTree.strFLabel = "S";

AdditionS(ctTree);

}

void AdditionS(CTree ctANode)

{

ctANode.AddSon(ctANode, "A");

AdditionA(ctANode.ctFSon);

if (Lex.enumPToken == TToken.lxmdt)

{

ctANode.ctFSon.AddBrother(ctANode.ctFSon, ":");

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmMinus)

{

ctANode.ctFSon.ctFBrother.AddBrother(ctANode.ctFSon.ctFBrother, "-");

Lex.NextToken();

ctANode.ctFSon.ctFBrother.ctFBrother.AddBrother(ctANode.ctFSon.ctFBrother.ctFBrother, "B");

AdditionB(ctANode.ctFSon.ctFBrother.ctFBrother.ctFBrother);

if (Lex.enumPToken == TToken.lxmDot)

{

ctANode.ctFSon.ctFBrother.ctFBrother.ctFBrother.AddBrother(ctANode.ctFSon.ctFBrother.ctFBrother.ctFBrother, ".");

Lex.NextToken();

}

}

else throw new Exception("Ожидался минус");

}

else throw new Exception("Ожидалось двоеточие");

}

void AdditionA(CTree ctANode)

{

if (Lex.enumPToken == TToken.lxmIdentifier)

{

ctANode.AddSon(ctANode, Lex.strPLexicalUnit);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmLeftParenth)

{

ctANode.ctFSon.AddBrother(ctANode.ctFSon, "(");

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

string number = Lex.strPLexicalUnit;

// Проверка на повторение числа

if (encounteredNumbers.Contains(number))

{

throw new Exception($"Повторяющееся число: {number}");

}

encounteredNumbers.Add(number);

ctANode.ctFSon.ctFBrother.AddBrother(ctANode.ctFSon.ctFBrother, Lex.strPLexicalUnit);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmRightParenth)

{

ctANode.ctFSon.ctFBrother.ctFBrother.AddBrother(ctANode.ctFSon.ctFBrother.ctFBrother, ")");

Lex.NextToken();

}

else throw new Exception("Ожидалась )");

}

else throw new Exception("Ожидался числовой идентификатор");

}

else throw new Exception("Ожидалась (");

}

else throw new Exception("Ожидался буквенный идентификатор");

}

void AdditionB(CTree ctANode)

{

ctANode.AddSon(ctANode, "A");

AdditionA(ctANode.ctFSon);

if (Lex.enumPToken == TToken.lxmComma)

{

ctANode.ctFSon.AddBrother(ctANode.ctFSon, ",");

ctANode.ctFSon.ctFBrother.AddBrother(ctANode.ctFSon.ctFBrother, "C");

AdditionC(ctANode.ctFSon.ctFBrother.ctFBrother);

}

else throw new Exception("Ожидалась ,");

}

public void AdditionC(CTree ctANode)

{

if (Lex.enumPToken == TToken.lxmComma)

{

ctANode.AddSon(ctANode, ",");

Lex.NextToken();

ctANode.ctFSon.AddBrother(ctANode.ctFSon, "A");

AdditionA(ctANode.ctFSon.ctFBrother);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmComma)

{

ctANode.ctFSon.ctFBrother.ctFBrother.AddBrother(ctANode.ctFSon.ctFBrother.ctFBrother, "C");

AdditionC(ctANode.ctFSon.ctFBrother.ctFBrother.ctFBrother);

}

}

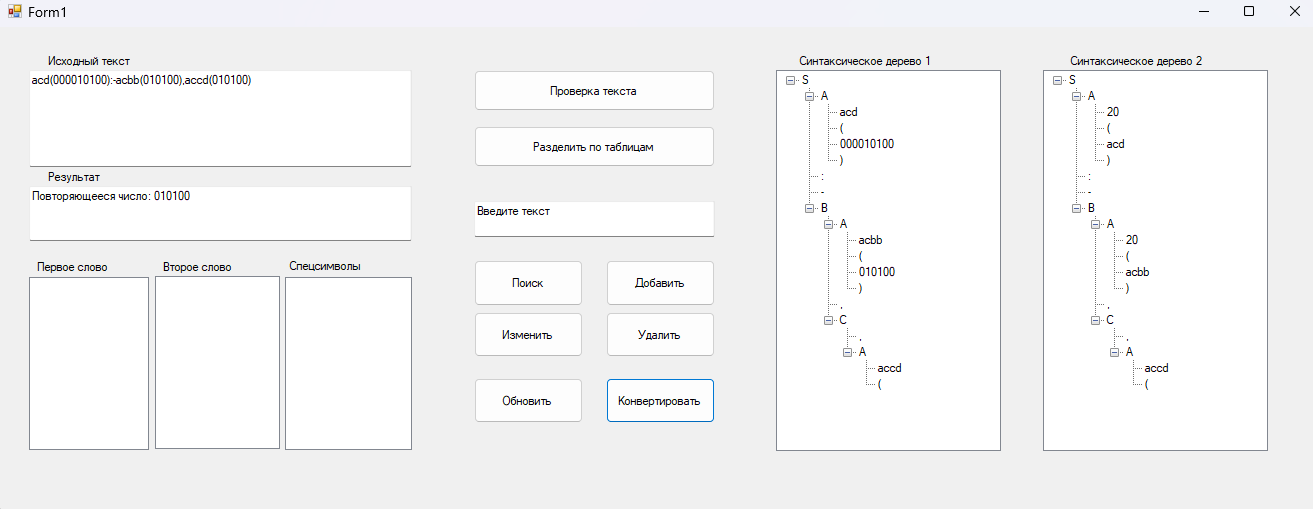
else throw new Exception("Ожидалось ,");

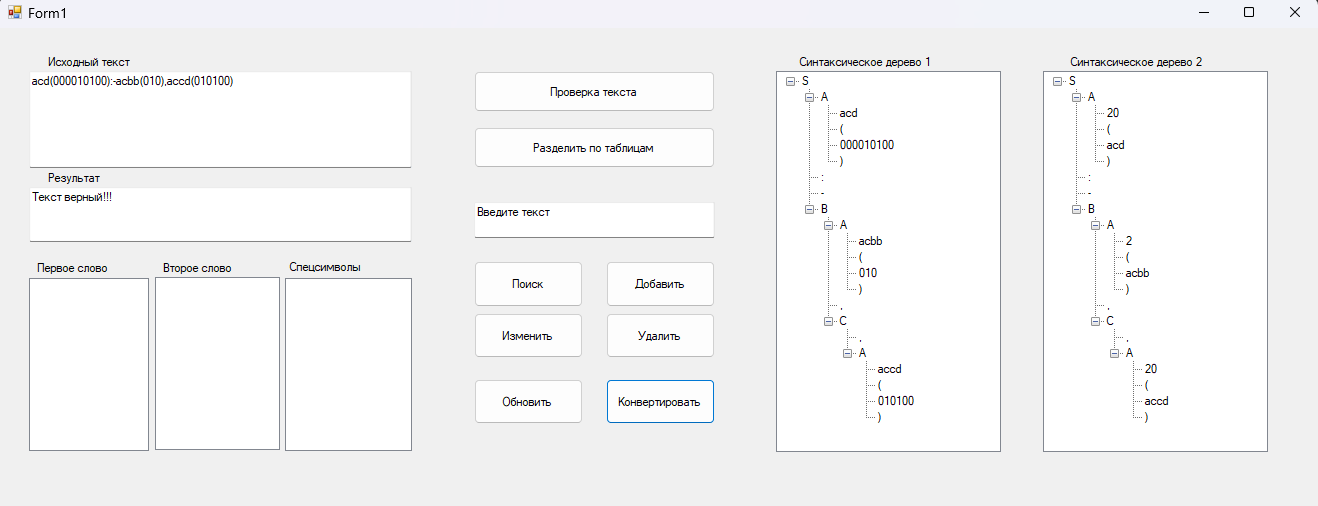
}

}

}

**Результат тестирования:**





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

**Теория:**

*Семантический анализ* - это этап в последовательности действий алгоритма автоматического понимания текстов, заключающийся в выделении семантических отношений и формировании семантического представления текстов. В общем случае семантическое представление является графом, семантической сетью, отражающим бинарные отношения между двумя узлами — смысловыми единицами текста.

Семантика — раздел лингвистики, изучающий смысловое значение единиц языка. В качестве инструмента изучения применяют семантический анализ.

*Основные методы семантического анализа*:

* Использование формальных грамматик: При этом подходе используются формальные правила для анализа структуры предложений и определения их значения.
* Использование онтологий: Онтологии представляют собой формальные описания терминов предметной области и отношений между ними. Они могут использоваться как основа для семантического анализа.
* Использование статистических методов: Например, при анализе текста могут применяться методы машинного обучения, которые позволяют "научить" систему распознавать и интерпретировать паттерны в данных.

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

**Вариант:** Проверка того, что после идентификатора accd стоит число 010100.

**Код программы:**

**uSemantAnalyzer.cs:**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

class uSemantAnalyzer

{

public int i = 0; // Публичное поле для хранения целочисленного значения i.

public string strIndentifier; // Публичное поле для хранения строки идентификатора.

public string strDigital; // Публичное поле для хранения строки числа.

TreeView tree; // Поле для хранения ссылки на объект TreeView.

public uSemantAnalyzer(){} // Конструктор без параметров.

// Конструктор с параметром, принимающий объект TreeView.

public uSemantAnalyzer(TreeView treeView)

{

tree = treeView; // Инициализация поля tree переданным объектом TreeView.

TreeController(tree); // Вызов метода для обхода дерева.

}

// Метод для обхода дерева.

public void TreeController(TreeView tree)

{

// Обход всех узлов верхнего уровня.

foreach (TreeNode node in tree.Nodes)

{

TreeController(node); // Вызов перегруженного метода для обработки каждого узла.

}

}

// Перегруженный метод для обхода дерева, принимающий узел TreeNode.

public void TreeController(TreeNode node)

{

if (node.Text == "A") // Если текст текущего узла равен "A".

{

// Если есть больше одного дочернего узла.

if (node.Nodes.Count > 1)

{

// Получение текста первого и третьего дочернего узла.

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

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

Check(strIndentifier, strDigital, node); // Вызов метода проверки.

}

}

// Рекурсивный обход всех дочерних узлов текущего узла.

foreach (TreeNode childNode in node.Nodes)

{

TreeController(childNode); // Вызов перегруженного метода для каждого дочернего узла.

}

}

// Метод для проверки семантических правил.

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

{

if (ident == "accd" && digit != "010100")

{

throw new Exception("После слова accd должно быть число 010100!");

}

if (ident == "accd" && digit == "010100")

{

throw new Exception("Текст верный!!!");

}

}

}

}

**Form1.cs:**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

public partial class Form1 : Form

{

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

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

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

public MyHashFunction hashFunction = new MyHashFunction();

public CHashTableList htl = new CHashTableList(2);

public CTree ctTree;

public Form1()

{

InitializeComponent();

//tbFSource.AppendText("acd(000010100):-acd(000010100),acd(000010100)");

tbFSource.AppendText("acccd(000010100):-acbb(010),accd(000010)");

int n = tbFSource.Lines.Length;

}

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

{

}

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

{

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

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

//htl.

htl.TableToStringList(0, listTable);

foreach (var entry in hashTableIdentifier)

{

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

}

listTable.Clear();

htl.TableToStringList(1, listTable);

foreach (var entry in hashTableDigital)

{

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

}

listTable.Clear();

foreach (var entry in hashTableRezerv)

{

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

}

listTable.Clear();

}

private void resetBtnsColor()

{

searchBtn.ResetBackColor();

searchBtn.UseVisualStyleBackColor = true;

addBtn.ResetBackColor();

addBtn.UseVisualStyleBackColor = true;

deleteBtn.ResetBackColor();

deleteBtn.UseVisualStyleBackColor = true;

changeBtn.ResetBackColor();

changeBtn.UseVisualStyleBackColor = true;

}

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

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.Lines;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.Parser();

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Synt.Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

treeView1.Nodes.Clear();

treeView1.Show();

Synt.ctTree.ShowTree(Synt.ctTree, treeView1);

treeView1.ExpandAll();

SemantFunction();

}

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

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmdt):

{

hashFunction.AddWord(hashTableRezerv, ":");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmComma):

{

hashFunction.AddWord(hashTableRezerv, ",");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmMinus):

{

hashFunction.AddWord(hashTableRezerv, "-");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmLeftParenth):

{

hashFunction.AddWord(hashTableRezerv, "(");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmRightParenth):

{

hashFunction.AddWord(hashTableRezerv, ")");

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

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

{

TablesToMemo(this, e);

}

break;

}

}

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

tbFMessage.Text += m;

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

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

{

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

{

searchBtn.BackColor = Color.Green;

}

else

{

searchBtn.BackColor = Color.Red;

}

}

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

{

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

addBtn.BackColor = Color.Green;

}

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

{

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

{

deleteBtn.BackColor = Color.Green;

}

else

{

deleteBtn.BackColor = Color.Red;

}

}

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

{

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

{

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

changeBtn.BackColor = Color.Green;

}

else

{

changeBtn.BackColor = Color.Red;

}

}

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

{

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

resetBtnsColor();

foreach (var entry in hashTableIdentifier)

{

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

}

foreach (var entry in hashTableDigital)

{

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

}

foreach (var entry in hashTableRezerv)

{

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

}

treeView1.Nodes.Clear();

}

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

{

CopyTreeView(treeView1, treeView2);

Converter converter = new Converter();

converter.ConvertBinaryToDecimal(treeView2);

treeView2.ExpandAll();

}

private void CopyTreeNodes(TreeNodeCollection sourceNodes, TreeNodeCollection targetNodes)

{

foreach (TreeNode sourceNode in sourceNodes)

{

TreeNode newNode = new TreeNode(sourceNode.Text);

targetNodes.Add(newNode);

CopyTreeNodes(sourceNode.Nodes, newNode.Nodes); // Рекурсивно копируем дочерние узлы

}

}

private void CopyTreeView(TreeView sourceTreeView, TreeView targetTreeView)

{

targetTreeView.Nodes.Clear(); // Очищаем существующие узлы в целевом TreeView

CopyTreeNodes(sourceTreeView.Nodes, targetTreeView.Nodes);

}

public void SemantFunction()

{

try

{

uSemantAnalyzer Semant = new uSemantAnalyzer(treeView1);

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

}

}

}

}

**Результат тестирования:**

