Министерство образования Российской Федерации

Федеральное агентство по образованию

КАЗАНСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. А.Н. Туполева - КАИ

Кафедра АСОИУ

Лабораторные работы №1-8

по дисциплине

«ТЕОРИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ» и «Методы трансляции»

Выполнил: студент группы 4309

Васильева В.В.

Проверила: Бикмуллина И. И.

Казань 2023

**Лабораторная работа № 1. Разработка транслитератора**

**Вариант 2.**

**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.
4. Для предложенного преподавателем варианта слов разработать лексический анализатор. На вход подаются два вида слов в любом порядке, разделенные любым количеством пробелов. Текст может быть многострочным. Текст может содержать комментарии. Шаблон комментария разработать самостоятельно. Комментарии и пробелы должны пропускаться.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Первое слово | Второе слово | Условие для второго слова |
| 2 | (000)\*001(100)\* | (a|b|c|d)+ | Первые два символа всегда ab |

**Краткое теоретическое обоснование:**

Несмотря на то, что лексический анализатор обрабатывает входную цепочку, удобнее на его вход подавать не просто отдельные символы, а символы, сгруппированные по категориям (классам). Поэтому перед лексическим анализом осуществляется дополнительная обработка, сопоставляющая каждому символу его класс, что позволяет лексическому анализатору манипулировать единым понятием для целой группы символов, иногда достаточно большой. Например, символы «a», «b», «c», и т.д. относятся к классу «буквы», а символы «0», «1», «2» и т.д. относятся к классу «цифры». Функция, осуществляющая чтение исходного текста и сопоставляющая каждому прочитанному символу значение его класса, называется **транслитератором**.

**Основные функции транслитератора**:

1. При каждом обращении к транслитератору прочитывается очередная литера из исходного текста и классифицируется по таблице кодов символов. Классификация литер осуществляется в зависимости от потребности лексического анализатора;
2. Читаемые строки текста нумеруются и выводятся на печать или экран (по необходимости);
3. Слежение за концом строки и переход на новую строку;
4. Слежение за концом текста.
5. Информирование пользователя об обнаруженных ошибках транслитерации (символ не входит в алфавит, попытка чтения символа за пределами исходного текста).

**Классы символов**. Наиболее типичными классами символов являются:

* **буква** - класс, с которым сопоставляется множество букв; буквы могут классифицироваться более детально, например, можно выделить класс **заглавная\_буква** или класс **малая\_буква\_кириллицы** и т.п
* **цифра** - множество символов, относящихся к цифрам (для десятичных цифр от символа 0 до символа 9);
* **разделитель** - пробел, перевод строки, возврат каретки, перевод формата; символ пробел появляется в исходном тексте довольно часто, поэтому для него можно создать свой класс **пробел**;
* **специальный** – в эту категорию обычно относят такие символы, как ‘\*’, ’:’, ’(’, ’!’, ’\’ и т.д.; часто эту категорию объединяют с классом **разделитель**;
* **игнорируемый** - может встречаться во входном потоке, но игнорируется и поэтому просто отфильтровывается из него (например, невидимый код звукового сигнала, символ табуляции и другие аналогичные коды);
* **запрещенный** - символы, которые не относятся к алфавиту языка, но могут по ошибке появиться в исходном тексте;
* **прочие** - символы, не вошедшие ни в одну из определенных категорий, обработка этих символов выполняется нестандартно.

Пара <**значение символа**, **класс символа**> будет использоваться лексическим анализатором, который проверяет пришедший символ на соответствие регулярной грамматике.

При лексическом анализе целого числа удобнее манипулировать понятием класса символов ("**цифра**"), тогда как при анализе действительного числа можно сразу смотреть значение символа: **"E"** или **"e"**, означающего начало порядка действительного числа.

**Программная реализация**. Реализация транслитератора, как и любого другого метода, во многом зависит от стиля программирования. Несмотря на простоту, эта задача может быть рассмотрена и в более широком контексте. Транслитератор можно использовать как совокупность нескольких функций, проверяющих принадлежность символа к одному из классов. Такая реализация выглядит вполне логичной при процедурном программировании. Вместе с тем следует отметить, что транслитератор не используется сам по себе. Он является промежуточным звеном между функциями, обеспечивающими чтение символов из входного потока и лексическим анализатором. Это промежуточное звено обычно скрывает от лексического анализатора механизм чтения и преобразования символов. Лексическому анализатору так же ничего не надо знать об открытии входного потока и манипуляции с ним. Поэтому при объектно-ориентированном подходе получение значения нового символа, его класса, а также манипуляция входным потоком реализуются, как единый объект (на основе класса). Этот класс инкапсулирует внутренние операции, а также объединяет разбросанные по программе структуры данных, используемые для ввода символов и их преобразования.

**Код программы:**

**Form1.cs:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using nsLex;

namespace Васильева\_4309

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

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

tbFSource.AppendText("01 \*|0110 \r\n");

tbFSource.AppendText("a\*|b|\* c\* |d");

}

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

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

Lex.enumPState = TState.stInitial;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.stExorbitant)

{

CLiteral Literal;

Literal = Lex.NextLiteral();

String s = "";

switch (Literal.enumFSelectionCharType)

{

case TCharType.Letter: { s = "Letter " + Literal.chrFSelection; break; }

case TCharType.Digit: { s = "Digit " + Literal.chrFSelection; break; }

case TCharType.ReservedSymbol: { s = "ReservedSymbol " + Literal.chrFSelection; break; }

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

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

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

}

tbFMessage.Text += "(" + s + ")";

}

tbFMessage.Text+= "Текст верный";

}

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;

}

}

}

}

**Program.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Васильева\_4309

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

**uLex.cs:**

using System;

namespace nsLex

{

public enum TState { Start, Continue, Finish };

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

public enum TToken { lxmUpperIdentifier, lxmNumber, lxmUnknown, lxmUl, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma, lxmText, lxmdt, lxmr, lxmrs, lxmls, lxmcommaDote };

public class CLex

//lxmUpperIdentifier - идентификатор с прописной буквы

//lxmNumber - целое число

{

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 CLex()

{

}

public void GetSymbol()

{

intFSourceColSelection++;

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];

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

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

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

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

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

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

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

enumFState = TState.Continue;

}

}

private void TakeSymbol()

{

char[] c = { chrFSelection };

String s = new string(c);

strFLexicalUnit += s;

GetSymbol();

}

public void NextToken()

{

strFLexicalUnit = "";

if (enumFState == TState.Start)

{

intFSourceRowSelection = 0;

intFSourceColSelection = -1;

GetSymbol();

}

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

{

GetSymbol();

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

}

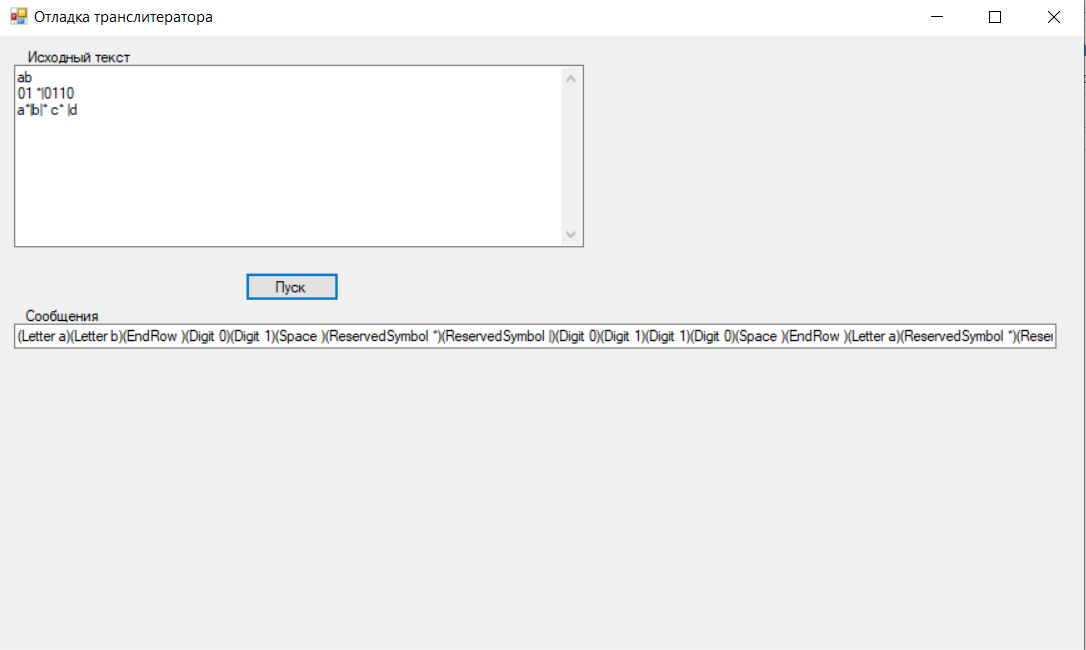
Введенное сообщение:

ab

01 \*|0110

a\*|b|\* c\* |d

**Результат выполнения программы:**



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

**Текст задания:**

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

**Краткое теоретическое обоснование:**

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

Основные функции лексического анализатора:

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

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

Конечные автоматы соответствуют регулярным грамматикам.

Способы описания конечного автомата:

1. Множество команд . Нетерминальным символам грамматики сопоставим состояния автомата, терминальным символам грамматики сопоставим символы входной ленты. Правилу вида A → aB сопоставим команду автомата (A,a) →B; правилу вида A → a (соответствует последнему символу в слове) сопоставим команду автомата (A,a) →Fin, где Fin - обозначение финального состояния. Для рассмотренного примера команды будут выглядеть следующим образом (Z,0) →U, (Z,1) →V, (U,1) →Z, (U,1) →Fin, (V,0) →Z, (V,0) →Fin.
2. Графический способ. Состояния автомата будут представлять собой узлы диаграммы состояний и переходов, а символы входной ленты будут помечать стрелку, соединяющую состояния. Каждому правилу вида A → aB сопоставим следующий фрагмент диаграммы состояний

**а**

Каждому правилу вида A → a сопоставим фрагмент диаграммы состояний вида:

**а**

1. Матричный способ. Строки матрицы помечаются состояниями (нетерминальным алфавитом), а столбцы – терминальным алфавитом (алфавитом обрабатываемого языка). В клетки матрицы заносятся состояния переходов.

**Недетерминированный/детерминированный графы.**

**Матрица и операции над матрицами.**

В лабораторной работе №2 используются понятия «Нетерминированный граф», «Детерминированный граф», «Матрица».

Недетерминированный граф - это граф, в котором существует несколько возможных переходов из одного состояния в другое. Детерминированный граф, наоборот, имеет единственный путь перехода между состояниями.

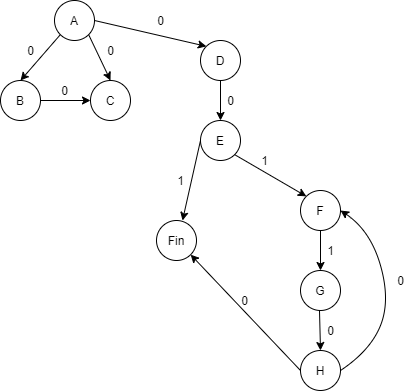
Матрица - это упорядоченный набор чисел, расположенных в виде прямоугольной таблицы.

Операции над матрицами включают в себя сложение, вычитание и умножение. Сложение матриц происходит путем сложения соответствующих элементов матриц. Вычитание матриц проводится аналогично. Умножение матриц требует перемножения элементов строк первой матрицы на элементы столбцов второй матрицы с последующим сложением результатов.

Таким образом, недетерминированный граф представляет собой множество возможных путей, детерминированный граф имеет единственный путь, а матрица используется для представления и выполнения операций над числовыми данными.

**Задание:**

|  |  |  |  |
| --- | --- | --- | --- |
| 22 | (000)\*001(100)\* | (a|b|c|d)+ | Первые два символа всегда ab |

**Первое слово:**

(000)\*001(100)\*

A->0B|0D

B->0C

C->0A

D->0E

E->1|1F

F->1G

G->0H

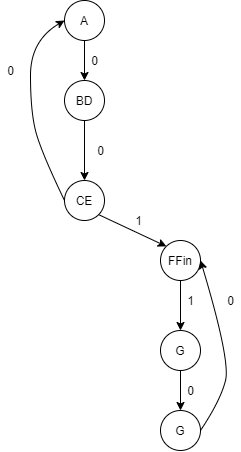
H->0|0H

**Недетерминированная матрица (1):**

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

**Недетерминированная матрица (2):**

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| A | BD |  |
| BD | CE |  |
| C | A |  |
| E |  | FFin |
| FFin |  | G |
| G | H |  |
| H | FFin |  |

**Детерминированная матрица и граф:**

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| A | BD |  |
| BD | CE |  |
| CE | A | FFin |
| FFin |  | G |
| G | H |  |
| H | FFin |  |

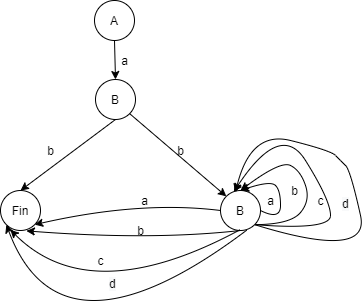
**Второе слово:**

(a|b|c|d)+

Первые два символа всегда ab

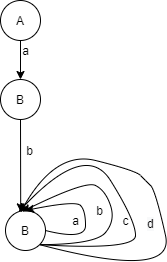
A->aB

B->b|bC

C->a|b|c|d|aC|bC|cC|dC

**Недетерминированная матрица:**

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



**Детерминированная матрица:**

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

**Код программы:**

**Form1.cs:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using nsLex;

namespace Васильева\_4309

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

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

tbFSource.AppendText("00001100ab \r\n");

}

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

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

Lex.enumPState = TState.stInitial;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.stExorbitant)

{

CLiteral Literal;

Literal = Lex.NextLiteral();

String s = "";

switch (Literal.enumFSelectionCharType)

{

case TCharType.Letter: { s = "Letter " + Literal.chrFSelection; break; }

case TCharType.Digit: { s = "Digit " + Literal.chrFSelection; break; }

case TCharType.ReservedSymbol: { s = "ReservedSymbol " + Literal.chrFSelection; break; }

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

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

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

}

tbFMessage.Text += "(" + s + ")";

}

tbFMessage.Text+= "Текст верный";

}

catch (Exception exc)

{

tbFMessage.Text += "(" + exc.Message + ")";

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

}

}

**uLex.cs**

using System;

namespace nsLex

{

public enum TState { Start, Continue, Finish };

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

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

public class CLex

{

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 CLex()

{

}

public void GetSymbol()

{

intFSourceColSelection++;

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];

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

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

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

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

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

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

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

enumFState = TState.Continue;

}

}

private void TakeSymbol()

{

char[] c = { chrFSelection };

String s = new string(c);

strFLexicalUnit += s;

GetSymbol();

}

public void NextToken()

{

strFLexicalUnit = "";

if (enumFState == TState.Start)

{

intFSourceRowSelection = 0;

intFSourceColSelection = -1;

GetSymbol();

}

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

{

GetSymbol();

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

// Вариант 2

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// A | B | | | |

// B | |CFin| | |

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

A:

{

if (chrFSelection == 'a')

{

TakeSymbol();

goto B;

}

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

}

B:

{

if (chrFSelection == 'b')

{

TakeSymbol();

goto CFin;

}

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

}

CFin:

{

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

{

TakeSymbol();

goto CFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

case TCharType.Digit:

{

// 0 1

// A | BD | |

// BD | CE | |

// CE | A | FFin|

// FFin | | G |

// G | H | |

// H | FFin| |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto BD;

}

else throw new Exception("Ожидался 0");

BD:

if (chrFSelection == '0')

{

TakeSymbol();

goto CE;

}

else throw new Exception("Ожидался 0");

CE:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 1");

FFin:

if (chrFSelection == '1')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидалась 1");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 0");

H:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

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;

}

}

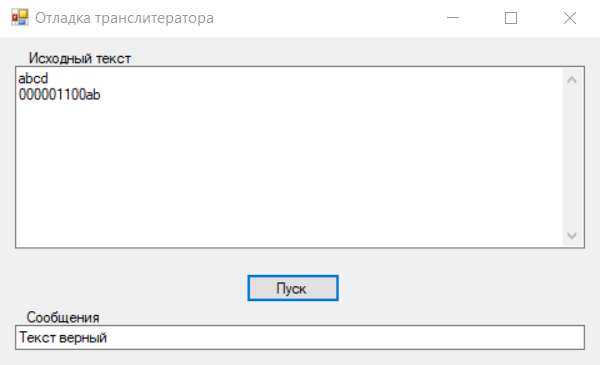
}

}

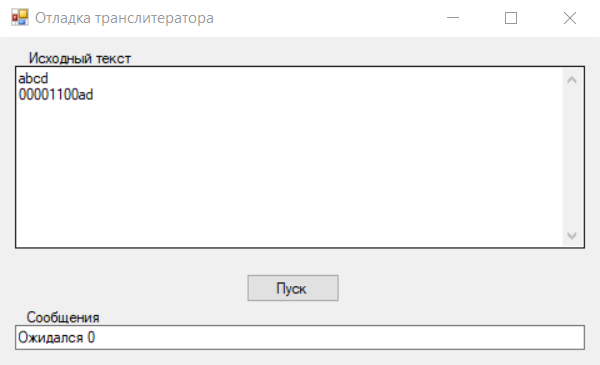
}

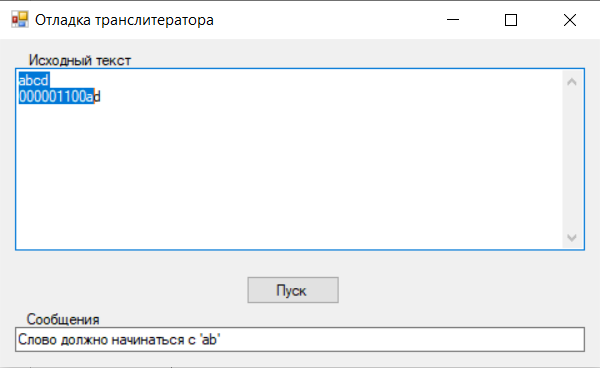
**Результат выполнения программы:**

**Без ошибки:**



**С ошибкой:**





**Лабораторная работа № 3. Разработка транслитератора**

**Текст задания:**

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

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

Указания:

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

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

3. Если грамматика леворекурсивная, то устранить левую рекурсию.

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

**Текст задания:**

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

**Грамматика: Избавление от левой рекурсии:**

|  |  |  |
| --- | --- | --- |
| S → S A |  | S →B B | S A |
| S → B B |  | A →<1> | <2> AА |
| A → <2> A A |  | B → <2> |
| A→ <1> |  |  |
| B → <2> |  |  |

В строке «S→S A» присутствует левая рекурсия

**Устранение левой рекурсии:**

1. S → B B

S → <2><2>

1. S → BBA

S → <2><2> <1>

1. S → BBAAA-><2><2><1>AA-><2><2><1><2> AA->

<2><2><1><2><1> <1>

**Краткое теоретическое обоснование:**

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

В нашей модели компилятора синтаксический анализатор получает строку токенов из лексического анализатора и проверяет, может ли эта строка токенов порождаться контекстно-свободной грамматикой. Мы также ожидаем от синтаксического анализатора сообщений обо всех выявленных ошибках. Часто синтаксический анализатор строит синтаксическое дерево (дерево разбора) и передает его для дальнейшей обработки.

Одним из ключевых элементов для разработки КС-синтаксического анализатора является определение грамматики языка, для которого разрабатывается анализатор. Это позволяет определить структуру предложений и выражений в данном языке, что необходимо для правильной обработки текста.

Другим важным компонентом является выбор алгоритма анализа, который определяет способ обхода и проверки правильности структуры текста. Этот выбор влияет на производительность и точность анализатора, поэтому необходимо тщательно подходить к его выбору.

Также важным компонентом разработки КС-синтаксического анализатора является работа с лексическими анализаторами, которые отвечают за разбор отдельных лексем в тексте. Их правильная работа влияет на качество анализа и скорость работы анализатора в целом.

Наконец, важным компонентом является тщательное тестирование анализатора на различных входных данных, чтобы убедиться в его правильной работе во всех возможных сценариях использования.

Существует целый ряд алгоритмов синтаксического анализа. Мы рассмотрим наиболее простой и часто используемый: избавление от левой рекурсии. Избавление от левой рекурсии в программировании очень важно для оптимизации кода. Левая рекурсия может привести к переполнению стека, что может вызвать непредсказуемое поведение программы. Поэтому разработчикам необходимо активно избавляться от левой рекурсии в своем коде. Для этого можно использовать различные методы, такие как преобразование рекурсивной функции в итеративную, использование хвостовой рекурсии или применение специальных оптимизаций компилятора. Важно помнить, что избавление от левой рекурсии может значительно улучшить производительность программы и сделать ее более надежной.

В целом, разработка КС-синтаксического анализатора требует учета множества компонентов, от выбора алгоритма анализа до тестирования и оптимизации его работы. Это сложный процесс, который требует внимания к деталям и тщательного подхода.

**Код программы:**

**Form1.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using nsSynt;

using nsLex;

namespace Васильева\_4309

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

int n = tbFSource.Lines.Length;

}

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

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.Lines;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.S();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Synt.Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

}

}

**uLex.cs**

using System;

namespace nsLex

{

public enum TState { Start, Continue, Finish };

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

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

public class CLex

//lxmUpperIdentifier - идентификатор с прописной буквы

//lxmNumber - целое число

{

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 CLex()

{

}

public void GetSymbol()

{

intFSourceColSelection++;

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];

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

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

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

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

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

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

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

enumFState = TState.Continue;

}

}

private void TakeSymbol()

{

char[] c = { chrFSelection };

String s = new string(c);

strFLexicalUnit += s;

GetSymbol();

}

public void NextToken()

{

strFLexicalUnit = "";

if (enumFState == TState.Start)

{

intFSourceRowSelection = 0;

intFSourceColSelection = -1;

GetSymbol();

}

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

{

GetSymbol();

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

// Вариант 2

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// A | B | | | |

// B | |CFin| | |

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

A:

{

if (chrFSelection == 'a')

{

TakeSymbol();

goto B;

}

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

}

B:

{

if (chrFSelection == 'b')

{

TakeSymbol();

goto CFin;

}

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

}

CFin:

{

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

{

TakeSymbol();

goto CFin;

}

else

{

enumFToken = TToken.lxmUpperIdentifier;

return;

}

}

}

case TCharType.Digit:

{

// 0 1

// A | BD | |

// BD | CE | |

// CE | A | FFin|

// FFin | | G |

// G | H | |

// H | FFin| |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto BD;

}

else throw new Exception("Ожидался 0");

BD:

if (chrFSelection == '0')

{

TakeSymbol();

goto CE;

}

else throw new Exception("Ожидался 0");

CE:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 1");

FFin:

if (chrFSelection == '1')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидалась 1");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 0");

H:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

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.lxmls;

GetSymbol();

return;

}

if (chrFSelection == ']')

{

enumFToken = TToken.lxmrs;

GetSymbol();

return;

}

if (chrFSelection == ',')

{

enumFToken = TToken.lxmComma;

GetSymbol();

return;

}

if (chrFSelection == ':')

{

enumFToken = TToken.lxmdt;

GetSymbol();

return;

}

if (chrFSelection == '=')

{

enumFToken = TToken.lxmr;

GetSymbol();

return;

}

if (chrFSelection == '\_')

{

enumFToken = TToken.lxmUl;

GetSymbol();

return;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

}

}

}

}

**uSyntAnalyzer.cs**

using System;

using System.Collections.Generic;

using System.Text;

using nsLex;

namespace nsSynt

{

class uSyntAnalyzer

{

private String[] strFSource;

private String[] strFMessage;

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

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

public CLex Lex = new CLex(); // Создание экземпляра класса CLex

public void S() //метод S

{

B();

if (Lex.enumPToken == TToken.lxmUl) // Проверка на наличие токена нижнего подчеркивания

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

throw new Exception("На второй позиции число!");

}

else

{

B();

Lex.NextToken();

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

{

A();

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

throw new Exception("Ожидалось число");

}

}

else throw new Exception("Конец строки, текст верный");

}

}

}

public void A() //метод А, отвечающий за численный идентификатор

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmUl) // Проверка на наличие токена нижнего подчеркивания

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber || Lex.enumPToken == TToken.lxmUpperIdentifier) // Проверка на наличие токена числа или буквенного идентификатора

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmUl)

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber || Lex.enumPToken == TToken.lxmUpperIdentifier || Lex.enumPToken == TToken.lxmUl)

{

A();

}

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

}

else throw new Exception("Ожидалось нижнее подчеркивание");

}

else throw new Exception("Конец строки, текст верный");

}

}

public void B() //метод В, отвечающий за буквенный идентификатор

{

if (Lex.enumPToken == TToken.lxmUpperIdentifier)

{

Lex.NextToken(); //переход к следующему токену

}

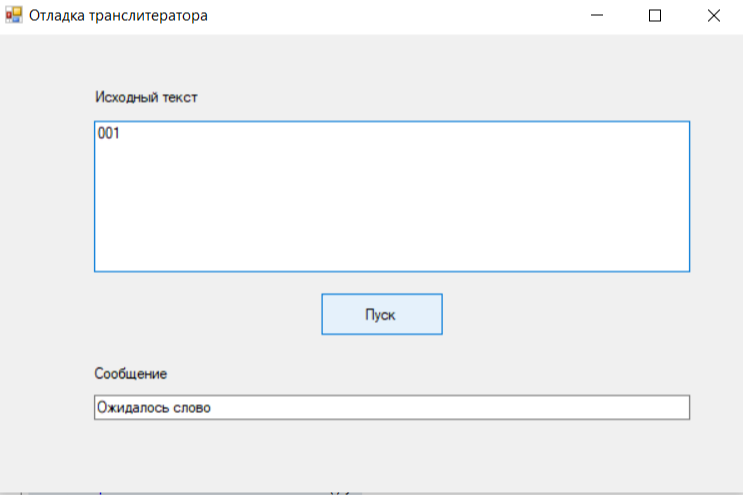
else throw new Exception("Ожидалось слово");

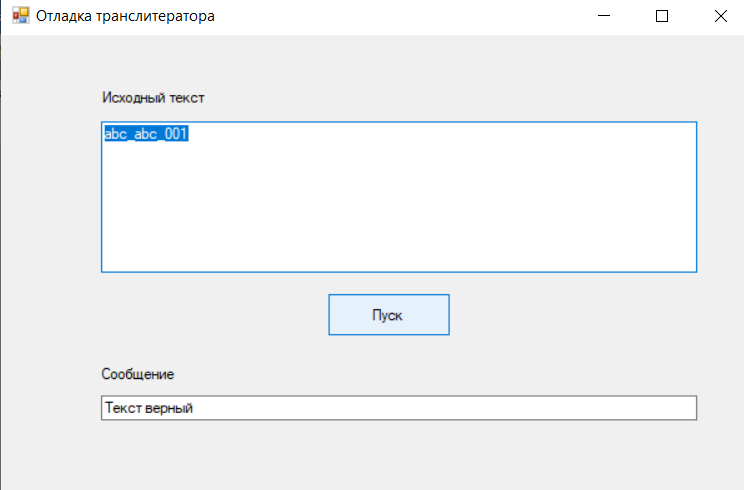
}

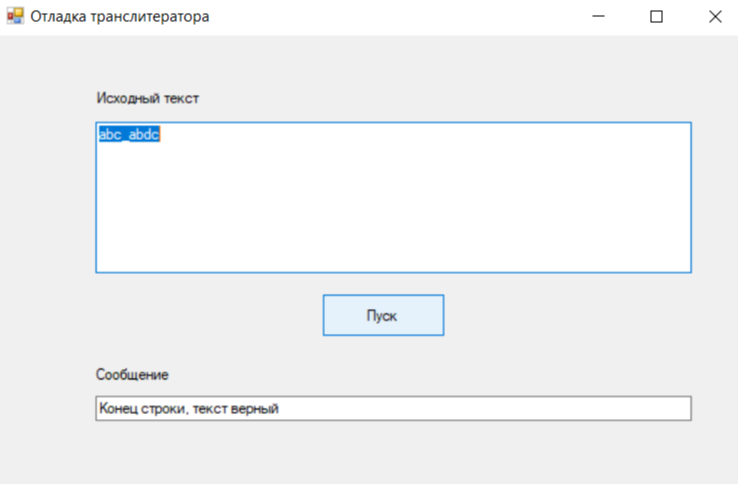
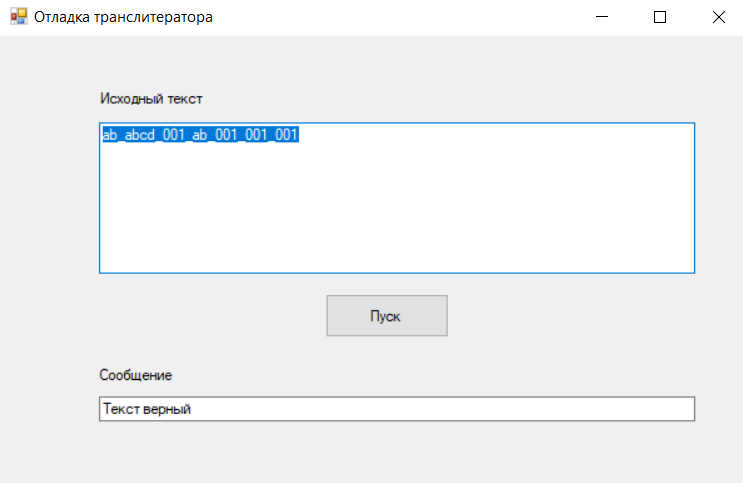
}

}

**Результат выполнения программы:**







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

**Задание:**

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

**Краткое теоретическое обоснование:**

Хеш-таблица - это структура данных, которая использует хеш-функцию для преобразования ключей в индексы массива. Хеш-таблицы используют хеш-функции для быстрого и эффективного поиска, вставки и удаления элементов. Хеш-функция преобразует ключи (например, строки или числа) в индексы (хеши), по которым элементы хранятся в массиве (бакете) с целью обеспечения быстрого доступа к ним. Ключи - это уникальные идентификаторы, которые используются для доступа к значениям в хеш-таблице. Хеш-функция - это функция, которая принимает в качестве входных данных ключ и возвращает соответствующий ему индекс в массиве.

Коллизия - это ситуация, когда два разных ключа после хеширования с помощью хеш-функции дают одинаковый индекс в массиве. Это может привести к конфликту при поиске значения по ключу.

В лабораторной работе №4 используется метод цепочек. При использовании этого метода каждая ячейка массива представляет собой связанный список или другую структуру данных, такую как дерево или хеш-таблица, где каждый элемент хранит пару ключ-значение. В случае коллизии элементы с одинаковыми ключами добавляются в эту структуру данных, что позволяет избежать потери информации.

Преимущества метода цепочек для избавления от коллизий в хеш-таблицах включают:

1. Простота реализации: Метод цепочек относительно прост в реализации, поскольку требует только использования связанных списков или других структур данных для хранения элементов с одинаковыми индексами.

2. Эффективность в случае высокой загрузки таблицы: При большом количестве элементов и маленьком размере массива метод цепочек может обеспечивать хорошую производительность и уменьшать вероятность коллизий.

Однако у метода цепочек есть и недостатки:

1. Потребление памяти: Использование связанных списков или других структур данных для хранения коллизионных элементов требует дополнительной памяти, что может быть неэффективно для небольших коллизий.

2. Неоптимальное использование памяти: В случае, если загрузка таблицы невелика, метод цепочек может привести к избыточному использованию памяти из-за хранения пустых ячеек в связанных списках.

3. Сложность обработки: Удаление или поиск элементов в связанных списках может привести к увеличению времени выполнения операций, особенно если списки становятся слишком длинными.

При выборе метода для решения коллизий следует учитывать особенности конкретной задачи, а также размер и ожидаемую загрузку хеш-таблицы.

**Код программы:**

**Form1.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using nsSynt;

using nsLex;

namespace Васильева\_4309

{

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("abc\_abcd\_001" + "\r\n");

int n = tbFSource.Lines.Length;

}

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

{

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

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

htl.TableToStringList(0, listTable);

foreach (var entry in hashTableIdentifier)

{

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

}

listTable.Clear();

htl.TableToStringList(1, listTable);

foreach (var entry in hashTableDigital)

{

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

}

listTable.Clear();

foreach (var entry in hashTableRezerv)

{

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

}

listTable.Clear();

}

private void start\_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 record\_Click(object sender, EventArgs e)

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmUpperIdentifier: //добавление токена буквенного идентификатора в ListBox1

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber: //добавление токена числового идентификатора в ListBox2

{

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): //добавление токена : в резерв

{

hashFunction.AddWord(hashTableRezerv, ":");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmComma): //добавление токена ;

{

hashFunction.AddWord(hashTableRezerv, ",");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmUl): //добавление токена \_

{

hashFunction.AddWord(hashTableRezerv, "\_");

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

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

{

TablesToMemo(this, e);

}

break;

}

}

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

tbFMessage.Text += m;

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

private void search\_Click(object sender, EventArgs e) //кнопка поиска

{

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

{

search.BackColor = Color.Green;

}

else

{

search.BackColor = Color.Red;

}

}

private void add\_Click(object sender, EventArgs e) //кнопка добавления слова

{

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

}

private void delete\_Click(object sender, EventArgs e) //кнопка удаления слова

{

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

{

delete.BackColor = Color.Green;

}

else

{

delete.BackColor = Color.Red;

}

}

private void change\_Click(object sender, EventArgs e) //кнопка изменения

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox1.SelectedItem.ToString())) //удаляется слово, выбранное в листе 1

{

hashFunction.AddWord(hashTableIdentifier, textBox1.Text.ToString()); //добавляется слово

change.BackColor = Color.Green; //кнопка меняется на зеленый цввет

}

else

{

change.BackColor = Color.Red; //иначе кнопка меняется на красный цвет

}

}

private void reload\_Click(object sender, EventArgs e) //кнопка обновления

{

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

foreach (var entry in hashTableIdentifier)

{

listBox1.Items.Add(string.Join(", ", entry.Value)); //для буквенного идентификатора

}

foreach (var entry in hashTableDigital)

{

listBox2.Items.Add(string.Join(", ", entry.Value)); //для числового идентификатора

}

foreach (var entry in hashTableRezerv)

{

listBox3.Items.Add(string.Join(", ", entry.Value)); //для других идентификаторов

}

}

}

}

**MyHashFunctions.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Васильева\_4309

{

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) //добавление слова

{

int hashValue = HashFunction(word); //хэш-значение слова

if (!hashTable.ContainsKey(hashValue)) //если в хэш-таблице не существует хэш-значение слова

{

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

}

else

{

if (hashTable[hashValue].Contains(word)) //иначе выходит надпись «Слово уже существует»

{

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

return;

}

}

hashTable[hashValue].Add(word);

}

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

{

int hashValue = HashFunction(word);

if (hashTable.ContainsKey(hashValue))

{

return 1;

}

else

{

return 0;

}

}

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

{

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;

}

}

}

**HashTables.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Васильева\_4309

{

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.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Васильева\_4309

{

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;

}

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

}

}

**Heap.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Васильева\_4309

{

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++)

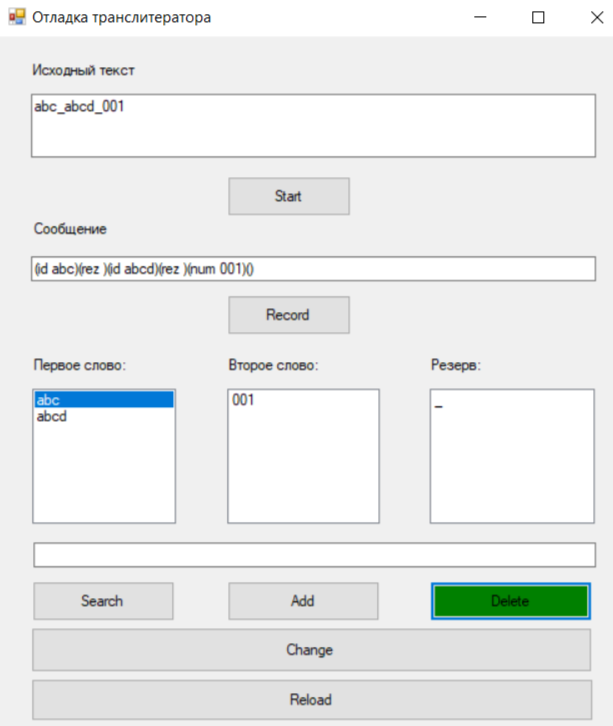
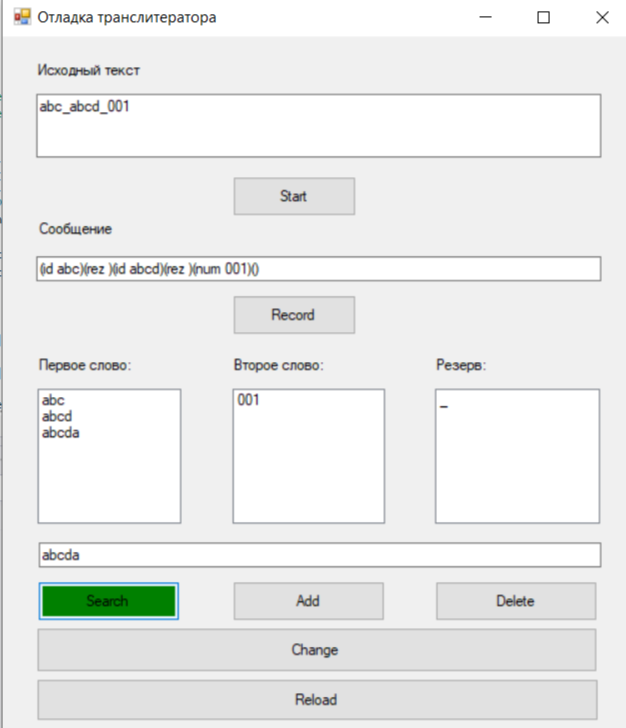
sList.Add(arrFHeapTable[i].strFLexicalUnit);

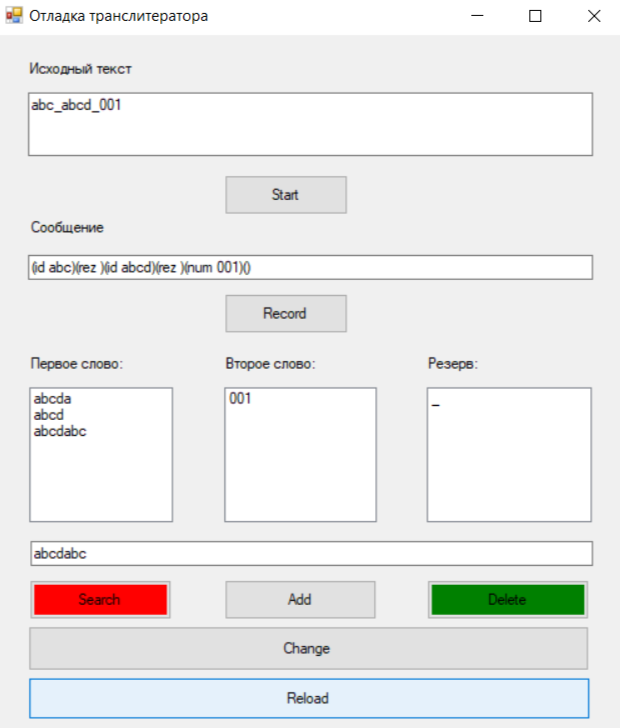
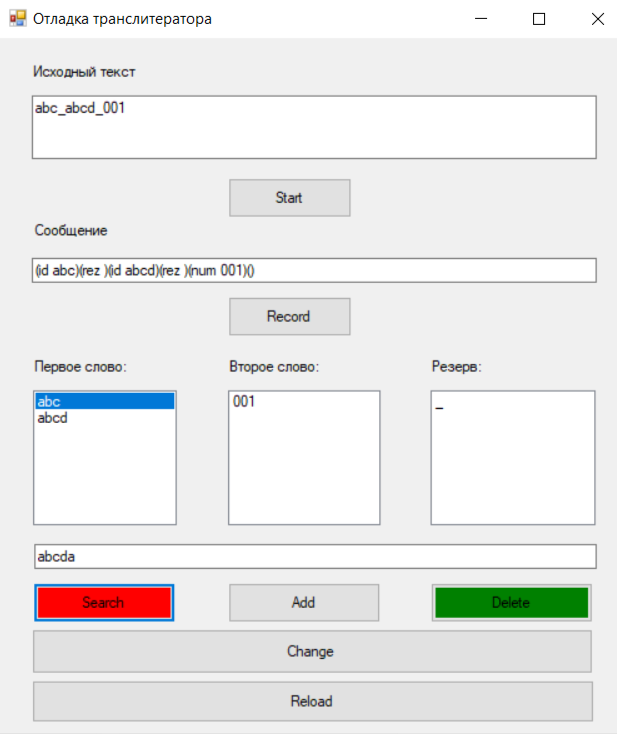
}

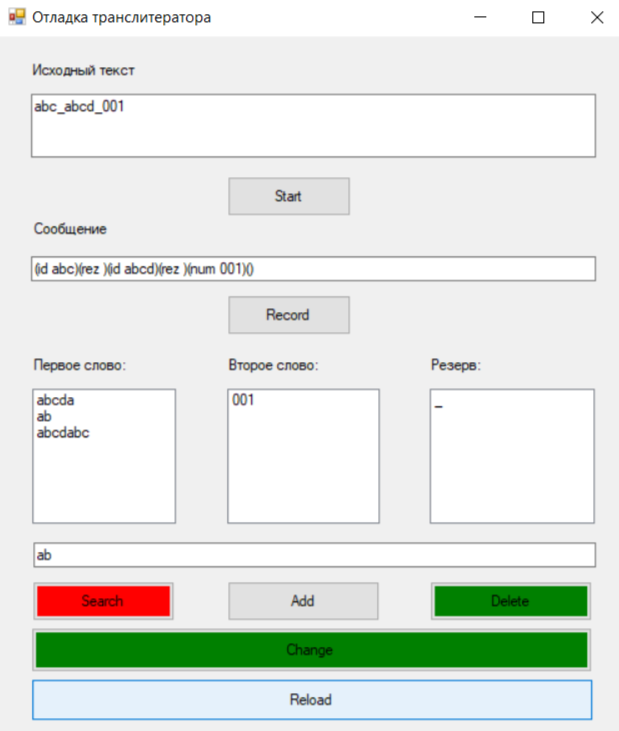
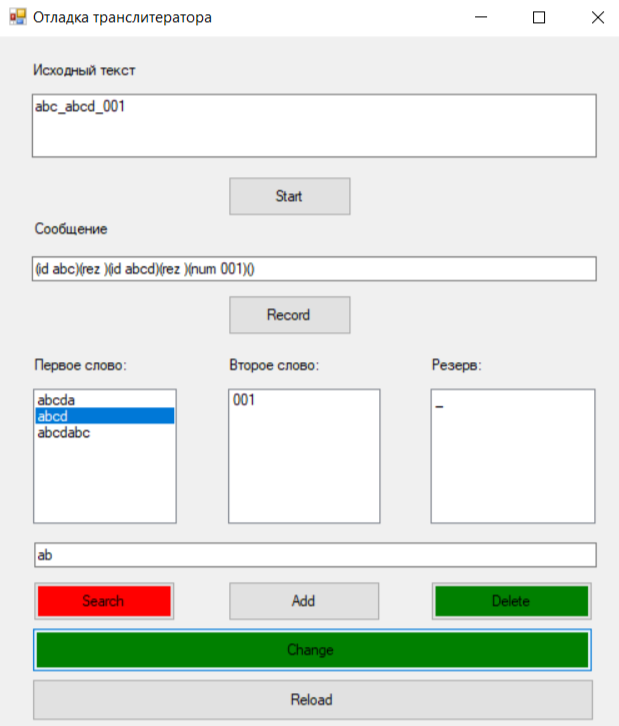
}

}

**Результат выполнения программы:**







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

**Задание:**

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

**Краткое теоретическое обоснование:**

Синтаксическое дерево, также известное как дерево разбора, представляет собой иерархическую структуру, которая отображает синтаксическую структуру предложения в естественном или формальном языке. Оно позволяет анализировать структуру предложения, а также выделять семантические и синтаксические компоненты. Построение синтаксического дерева включает несколько основных шагов.

1. Токенизация: В первую очередь предложение должно быть разбито на отдельные токены, такие как слова, знаки препинания и другие части речи. Каждый токен становится узлом в синтаксическом дереве.

2. Морфологический анализ: Для каждого токена определяются его морфологические характеристики, такие как часть речи, падеж, число и т.д. Эта информация связывается с соответствующими узлами в дереве.

3. Синтаксический анализ: На этом этапе проводится анализ синтаксической структуры предложения. Алгоритмы синтаксического анализа используются для построения иерархической структуры дерева, отображающей зависимости между словами в предложении.

4. Построение дерева: На основе результатов морфологического и синтаксического анализа строится само синтаксическое дерево. Узлы дерева представляют собой токены, а рёбра - связи и зависимости между ними.

Treeview, с другой стороны, является графическим элементом пользовательского интерфейса, который представляет дерево-подобную структуру для навигации и отображения иерархически организованных данных, таких как файловая система, структура документа или синтаксическое дерево. В компьютерных приложениях Treeview позволяет пользователям удобно просматривать и выбирать элементы из иерархических структур.

Синтаксическое дерево и Treeview относятся к разным областям компьютерных наук и имеют различные функции, но оба служат для представления иерархических структур данных.

**Код программы:**

**uSyntAnalyzer.cs**

using System;

using System.Collections.Generic;

using System.Windows.Forms;

using nsLex;

using Васильева\_4309;

namespace nsSynt

{

class uSyntAnalyzer

{

const int word = 0;

const int number = 0;

const int reserved = 0;

public TreeNode[] treeNodes = new TreeNode[1024];

int i = 1;

public String[] str = new String[20];

public int strNumber = 0;

public int indId = 0;

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 CHashTableList htl = new CHashTableList(3);

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

int intVLexicalCode = 0;

public uSyntAnalyzer()

{

treeNodes[0] = new TreeNode("S"); //создается дерево

}

public void S()

{

if (i > 0)

{

treeNodes[i] = new TreeNode("S");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

}

i++;

B();

if (Lex.enumPToken == TToken.lxmcommaDote)

{

Lex.NextToken();

B();

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, number, ref intVLexicalCode))

throw new Exception("В тексте не должно быть повторных чисел");

A();

i--;

if (Lex.enumPToken == TToken.lxmcommaDote)

{

treeNodes[i] = new TreeNode(";");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

Lex.NextToken();

A();

}

else throw new Exception("Ожидалась точка с запятой");

}

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

{

A();

i--;

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmcommaDote)

{

treeNodes[i] = new TreeNode(";");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

Lex.NextToken();

A();

i--;

if (Lex.enumPToken == TToken.lxmcommaDote)

{

treeNodes[i] = new TreeNode(";");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

Lex.NextToken();

A();

}

}

}

}

else throw new Exception("Ожидалась точка с запятой");

}

public void A()

{

if (i > 0)

{

treeNodes[i] = new TreeNode("A"); //создается ветвь дерева

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

}

i++;

if (Lex.enumPToken == TToken.lxmNumber)

{

if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, number, ref intVLexicalCode))

throw new Exception("В тексте не должно быть повторных чисел");

treeNodes[i] = new TreeNode("001");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(Lex.strPLexicalUnit, number, ref intVLexicalCode);

Lex.NextToken();

i--;

if (Lex.enumPToken == TToken.lxmcommaDote)

{

treeNodes[i] = new TreeNode(";");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(";", reserved, ref intVLexicalCode);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmUpperIdentifier)

{

treeNodes[i] = new TreeNode("A");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

i++;

treeNodes[i] = new TreeNode("ab");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(Lex.strPLexicalUnit, word, ref intVLexicalCode);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmcommaDote)

{

treeNodes[i] = new TreeNode(";");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(";", reserved, ref intVLexicalCode);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

// if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, number, ref intVLexicalCode))

// throw new Exception("В тексте не должно быть повторных чисел");

treeNodes[i] = new TreeNode("A");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

i++;

treeNodes[i] = new TreeNode(Lex.strPLexicalUnit);

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(Lex.strPLexicalUnit, number, ref intVLexicalCode);

i--;

}

}

}

}

}

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

{

treeNodes[i] = new TreeNode(Lex.strPLexicalUnit);

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(Lex.strPLexicalUnit, word, ref intVLexicalCode);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmcommaDote)

{

treeNodes[i] = new TreeNode(";");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(";", reserved, ref intVLexicalCode);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, number, ref intVLexicalCode))

throw new Exception("В тексте не должно быть повторных чисел");

treeNodes[i] = new TreeNode("A");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

i++;

treeNodes[i] = new TreeNode(Lex.strPLexicalUnit);

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(Lex.strPLexicalUnit, number, ref intVLexicalCode);

i--;

}

}

}

}

public void B()

{

if (i > 0)

{

treeNodes[i] = new TreeNode("B"); //создается ветвь дерева

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

}

i++;

if (Lex.enumPToken == TToken.lxmUpperIdentifier)

{

Lex.NextToken();

treeNodes[i] = new TreeNode("abcd");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(Lex.strPLexicalUnit, word, ref intVLexicalCode);

i--;

if (Lex.enumPToken == TToken.lxmcommaDote)

{

treeNodes[i] = new TreeNode(";");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

}

}

else throw new Exception("Ожидалось слово");

}

}

}

**Form1.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using nsSynt;

using nsLex;

namespace Васильева\_4309

{

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("abcd;abcd;abcddc;001100;001;ab;001100100" + "\r\n");

int n = tbFSource.Lines.Length;

}

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

{

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

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

htl.TableToStringList(0, listTable);

foreach (var entry in hashTableIdentifier)

{

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

}

listTable.Clear();

htl.TableToStringList(1, listTable);

foreach (var entry in hashTableDigital)

{

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

}

listTable.Clear();

foreach (var entry in hashTableRezerv)

{

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

}

listTable.Clear();

}

private void start\_Click(object sender, EventArgs e) //кнопка старта

{

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;

}

SyntTree.Nodes.Clear();

SyntTree.Nodes.Add(Synt.treeNodes[1]);

SyntTree.ExpandAll();

}

private void record\_Click(object sender, EventArgs e) //кнопка записи

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmUpperIdentifier:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmdt):

{

hashFunction.AddWord(hashTableRezerv, ":");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmComma):

{

hashFunction.AddWord(hashTableRezerv, ",");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmcommaDote):

{

hashFunction.AddWord(hashTableRezerv, ";");

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

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

{

TablesToMemo(this, e);

}

break;

}

}

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

tbFMessage.Text += m;

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

private void search\_Click(object sender, EventArgs e) //кнопка поиска

{

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

{

search.BackColor = Color.Green;

}

else

{

search.BackColor = Color.Red;

}

}

private void add\_Click(object sender, EventArgs e) //кнопка добавления слова

{

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

}

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

{

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

{

delete.BackColor = Color.Green;

}

else

{

delete.BackColor = Color.Red;

}

}

private void change\_Click(object sender, EventArgs e) //кнопка изменениия слова

{

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

{

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

change.BackColor = Color.Green;

}

else

{

change.BackColor = Color.Red;

}

}

private void reload\_Click(object sender, EventArgs e) //кнопка обновления

{

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

foreach (var entry in hashTableIdentifier)

{

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

}

foreach (var entry in hashTableDigital)

{

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

}

foreach (var entry in hashTableRezerv)

{

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

}

}

private void Convert\_Click(object sender, EventArgs e) //кнопка конвертации дляпостроения структурного дерева

{

CopyTreeView(SyntTree, SyntTree2);

TreeConverter treeConverter = new TreeConverter();

treeConverter.ConvertBinaryToDecimal(SyntTree2);

SyntTree2.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);

}

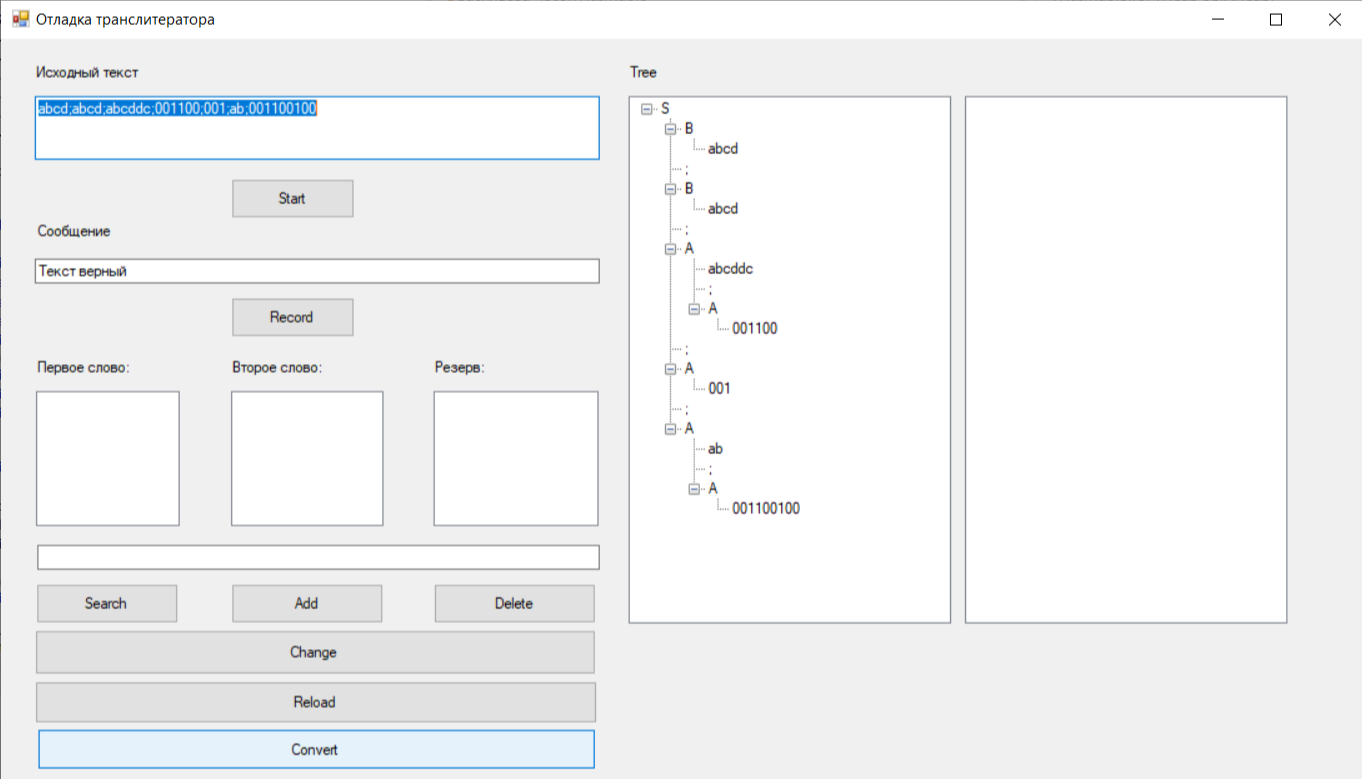
private void Form1\_Load(object sender, EventArgs e)

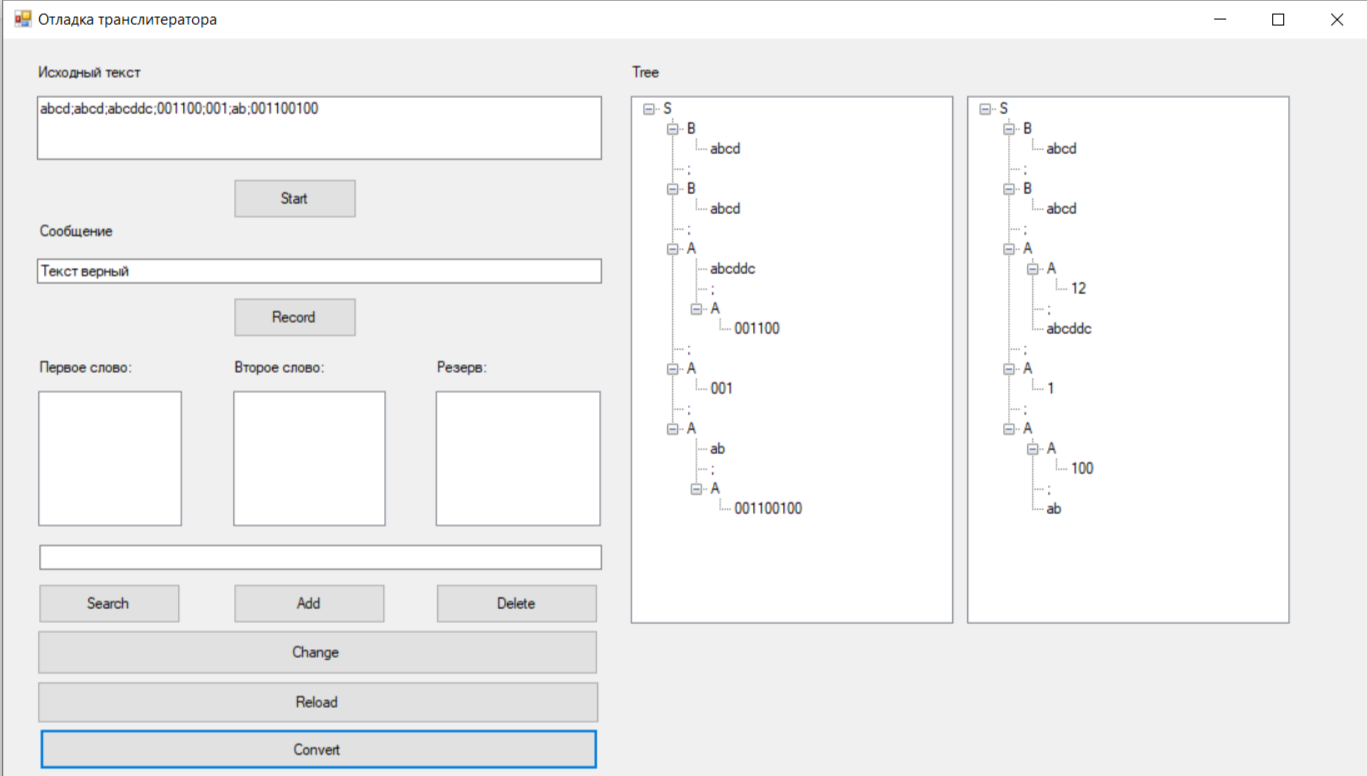
{

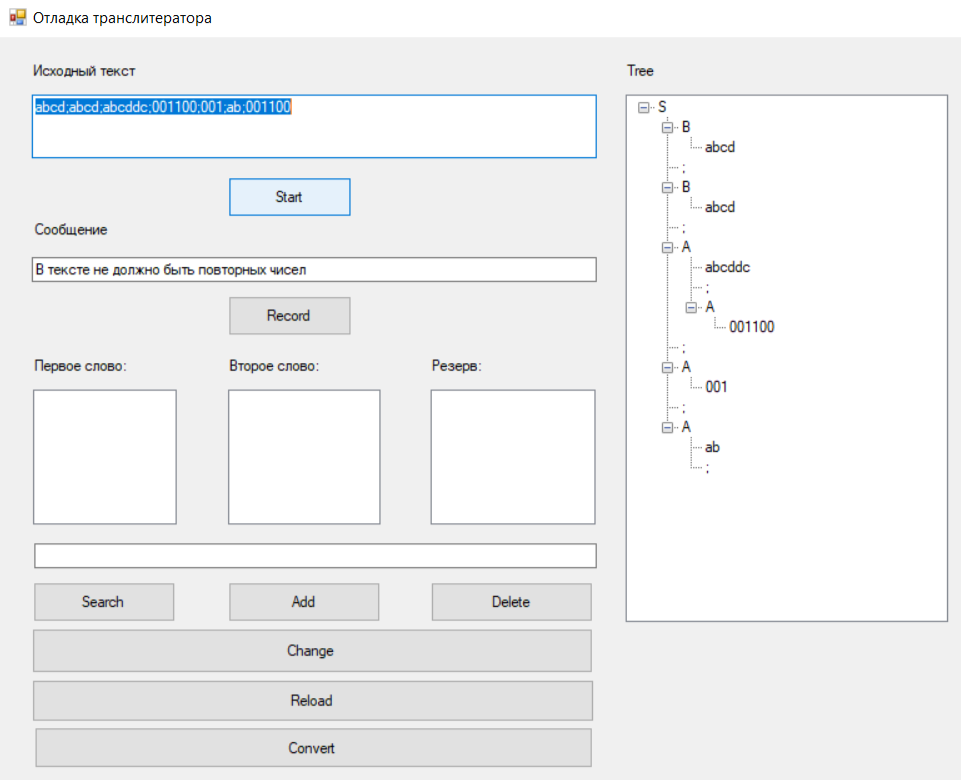
}

}

}







**Лабораторная работа № 6-7. Разработка генератора (2 часа)**

**Задание:**

1. Перевести все числа в десятичное представление.

2. Выполнить вывод исходного текста в структурированном виде.

**Краткое теоретическое обоснование:**

TreeView (дерево просмотра) - это элемент пользовательского интерфейса, который отображает данные в виде иерархического дерева. Он состоит из узлов, которые могут содержать данные, и связей между узлами, представляющих их иерархическую структуру.

Атрибутивная грамматика - это формальная система для описания языка или множества строк. В атрибутной грамматике каждому символу или правилу грамматики сопоставляются некоторые атрибуты, которые могут использоваться для вычисления значений этих символов или правил. Атрибуты могут использоваться для определения семантики языка или для выполнения другой полезной работы, связанной с грамматикой.

Генератор в разработке транслятора — это ключевой элемент, который преобразует высокоуровневое представление программы в код, который может быть выполнен на целевой платформе. Его эффективность и качество влияют на производительность и правильность выполнения программы на целевой машине.

**Код программы:**

**Form1.cs:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using nsSynt;

using nsLex;

namespace Васильева\_4309

{

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("abcd;abcd;abcddc;001100;001;ab;001100100" + "\r\n");

int n = tbFSource.Lines.Length;

}

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

{

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

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

htl.TableToStringList(0, listTable);

foreach (var entry in hashTableIdentifier)

{

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

}

listTable.Clear();

htl.TableToStringList(1, listTable);

foreach (var entry in hashTableDigital)

{

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

}

listTable.Clear();

foreach (var entry in hashTableRezerv)

{

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

}

listTable.Clear();

}

private void start\_Click(object sender, EventArgs e) //кнопка старта

{

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;

}

SyntTree.Nodes.Clear();

SyntTree.Nodes.Add(Synt.treeNodes[1]);

SyntTree.ExpandAll(); //открывает дерево полностью

}

private void record\_Click(object sender, EventArgs e) //кнопка записи

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmUpperIdentifier:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmdt):

{

hashFunction.AddWord(hashTableRezerv, ":");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmComma):

{

hashFunction.AddWord(hashTableRezerv, ",");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmcommaDote):

{

hashFunction.AddWord(hashTableRezerv, ";");

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

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

{

TablesToMemo(this, e);

}

break;

}

}

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

tbFMessage.Text += m;

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

private void search\_Click(object sender, EventArgs e) //кнопка поиска

{

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

{

search.BackColor = Color.Green;

}

else

{

search.BackColor = Color.Red;

}

}

private void add\_Click(object sender, EventArgs e) //кнопка добавления слова

{

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

}

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

{

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

{

delete.BackColor = Color.Green;

}

else

{

delete.BackColor = Color.Red;

}

}

private void change\_Click(object sender, EventArgs e) //кнопка изменениия слова

{

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

{

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

change.BackColor = Color.Green;

}

else

{

change.BackColor = Color.Red;

}

}

private void reload\_Click(object sender, EventArgs e) //кнопка обновления

{

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

foreach (var entry in hashTableIdentifier)

{

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

}

foreach (var entry in hashTableDigital)

{

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

}

foreach (var entry in hashTableRezerv)

{

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

}

}

private void Convert\_Click(object sender, EventArgs e) //кнопка конвертации дляпостроения структурного дерева

{

CopyTreeView(SyntTree, SyntTree2);

TreeConverter treeConverter = new TreeConverter();

treeConverter.ConvertBinaryToDecimal(SyntTree2);

SyntTree2.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);

}

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

{

}

}

}

**TreeConverter.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Васильева\_4309

{

internal class TreeConverter

{

public void ConvertBinaryToDecimal(TreeView tree)

{

foreach (TreeNode treeNodes in tree.Nodes)

{

ConvertBinaryToDecimal(treeNodes);

}

}

private void ConvertBinaryToDecimal(TreeNode treeNodes)

{

foreach (TreeNode treeNode in treeNodes.Nodes)

{

string binaryValue = treeNode.Text;

if (IsBin(binaryValue))

{

int decimalValue = Convert.ToInt32(binaryValue, 2); //конвертация в 10-сс

treeNode.Text = decimalValue.ToString(); //преобразование текста в строку

}

if (treeNode.Nodes.Count == 3) // если ветвь имеет 3 узла

{

TreeNode temp = treeNode.Nodes[2]; //находим 2 узел

treeNode.Nodes[2].Remove(); //удаляем данные из него

treeNode.Nodes.Insert(0, temp); //вставляем данные из 0 узла

temp = treeNode.Nodes[1]; //находим 1 узел

treeNode.Nodes[1].Remove(); //удаляем данные из него

treeNode.Nodes.Insert(2, temp); //вставляем данные из 0 узла

}

}

foreach (TreeNode treeNode1 in treeNodes.Nodes)

{

ConvertBinaryToDecimal(treeNode1);

}

}

private bool IsBin(string value)

{

foreach (var c in value) //проходим по всем символам

if (c != '0' && c != '1') //проверка на наличие символов 0 и 1

return false;

return true;

}

}

}

**uSyntAnalyzer.cs:**

using System;

using System.Collections.Generic;

using System.Windows.Forms;

using nsLex;

using Васильева\_4309;

namespace nsSynt

{

class uSyntAnalyzer

{

const int word = 0;

const int number = 0;

const int reserved = 0;

public TreeNode[] treeNodes = new TreeNode[1024];

int i = 1;

public String[] str = new String[20];

public int strNumber = 0;

public int indId = 0;

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 CHashTableList htl = new CHashTableList(3);

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

int intVLexicalCode = 0;

public uSyntAnalyzer()

{

treeNodes[0] = new TreeNode("S"); //создается дерево

}

public void S()

{

if (i > 0)

{

treeNodes[i] = new TreeNode("S");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

}

i++;

B();

if (Lex.enumPToken == TToken.lxmcommaDote)

{

Lex.NextToken();

B();

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, number, ref intVLexicalCode))

throw new Exception("В тексте не должно быть повторных чисел");

A();

i--;

if (Lex.enumPToken == TToken.lxmcommaDote)

{

treeNodes[i] = new TreeNode(";");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

Lex.NextToken();

A();

}

else throw new Exception("Ожидалась точка с запятой");

}

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

{

A();

i--;

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmcommaDote)

{

treeNodes[i] = new TreeNode(";");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

Lex.NextToken();

A();

i--;

if (Lex.enumPToken == TToken.lxmcommaDote)

{

treeNodes[i] = new TreeNode(";");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

Lex.NextToken();

A();

}

}

}

}

else throw new Exception("Ожидалась точка с запятой");

}

public void A()

{

if (i > 0)

{

treeNodes[i] = new TreeNode("A"); //создается ветвь дерева

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

}

i++;

if (Lex.enumPToken == TToken.lxmNumber)

{

if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, number, ref intVLexicalCode))

throw new Exception("В тексте не должно быть повторных чисел");

treeNodes[i] = new TreeNode("001");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(Lex.strPLexicalUnit, number, ref intVLexicalCode);

Lex.NextToken();

i--;

if (Lex.enumPToken == TToken.lxmcommaDote)

{

treeNodes[i] = new TreeNode(";");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(";", reserved, ref intVLexicalCode);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmUpperIdentifier)

{

treeNodes[i] = new TreeNode("A");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

i++;

treeNodes[i] = new TreeNode("ab");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(Lex.strPLexicalUnit, word, ref intVLexicalCode);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmcommaDote)

{

treeNodes[i] = new TreeNode(";");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(";", reserved, ref intVLexicalCode);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

// if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, number, ref intVLexicalCode))

// throw new Exception("В тексте не должно быть повторных чисел");

treeNodes[i] = new TreeNode("A");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

i++;

treeNodes[i] = new TreeNode(Lex.strPLexicalUnit);

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(Lex.strPLexicalUnit, number, ref intVLexicalCode);

i--;

}

}

}

}

}

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

{

treeNodes[i] = new TreeNode(Lex.strPLexicalUnit);

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(Lex.strPLexicalUnit, word, ref intVLexicalCode);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmcommaDote)

{

treeNodes[i] = new TreeNode(";");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(";", reserved, ref intVLexicalCode);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, number, ref intVLexicalCode))

throw new Exception("В тексте не должно быть повторных чисел");

treeNodes[i] = new TreeNode("A");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

i++;

treeNodes[i] = new TreeNode(Lex.strPLexicalUnit);

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(Lex.strPLexicalUnit, number, ref intVLexicalCode);

i--;

}

}

}

}

public void B()

{

if (i > 0)

{

treeNodes[i] = new TreeNode("B"); //создается ветвь дерева

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

}

i++;

if (Lex.enumPToken == TToken.lxmUpperIdentifier)

{

Lex.NextToken();

treeNodes[i] = new TreeNode("abcd");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(Lex.strPLexicalUnit, word, ref intVLexicalCode);

i--;

if (Lex.enumPToken == TToken.lxmcommaDote)

{

treeNodes[i] = new TreeNode(";");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

}

}

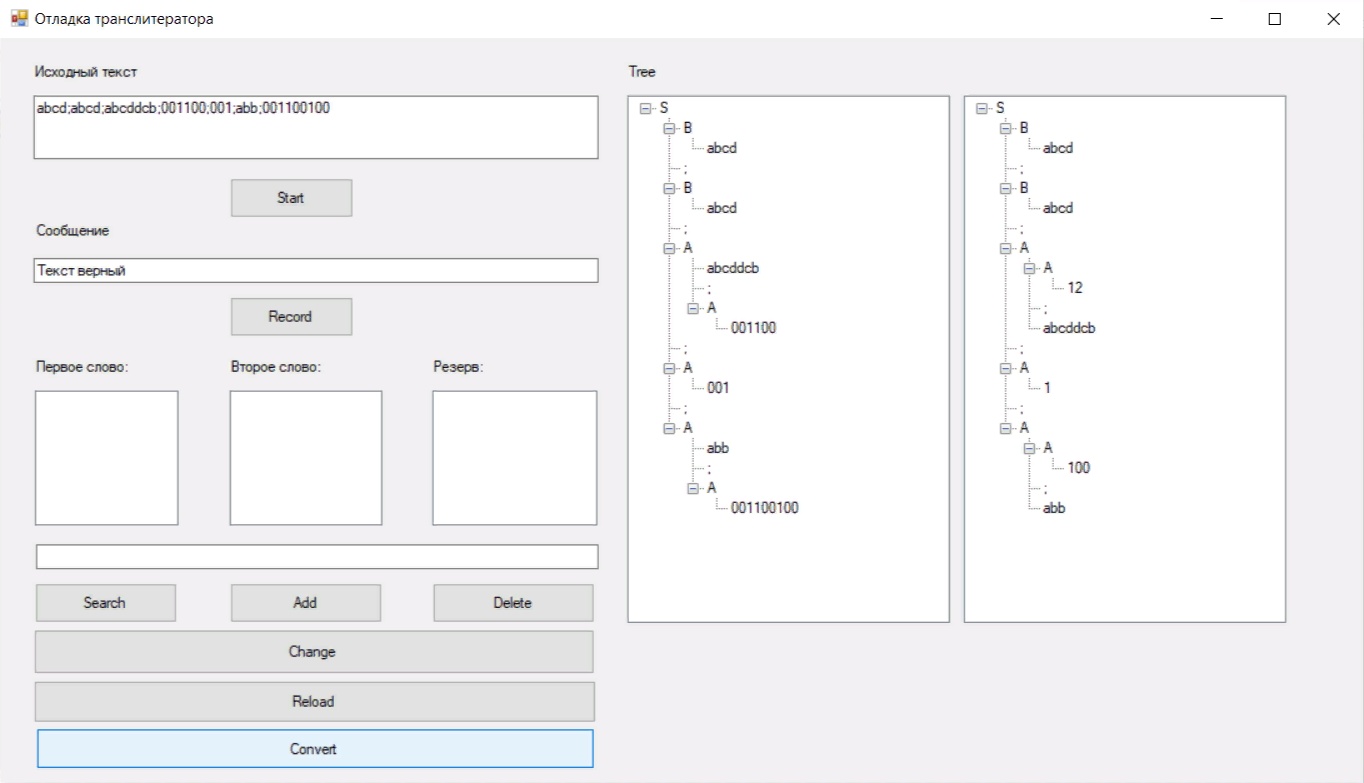
else throw new Exception("Ожидалось слово");

}

}

}

**Результат работы программы:**



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

**Задание:**

Lля 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 2:** все числа должны быть разными.

**Краткое теоретическое обоснование:**

Транслятор - это программное обеспечение или устройство, которое выполняет преобразование кода программы из одного языка программирования в другой. Трансляторы используются для обеспечения переносимости программного кода, так как разные языки программирования имеют разные синтаксисы и семантику. Они могут быть компилирующими (полностью переводящими исходный код в машинный код) или интерпретирующими (построчно выполняющими код на другом языке). Трансляторы также могут выполнять другие задачи, такие как оптимизация кода или генерация документации.

Чтобы генератор стал реальным и превратился в транслятор, нужно выполнить следующие основные этапы:

1. Анализ исходного языка: изучение синтаксиса и семантики языка, анализ его структуры и функциональности.

2. Разработка грамматики: создание формальной грамматики для представления синтаксиса языка. Это может включать в себя определение терминалов (лексем) и нетерминалов (синтаксических конструкций), а также правил вывода.

3. Создание лексического анализатора: разработка компонента, который будет анализировать исходный код на предмет лексем (например, идентификаторы, константы, операторы) и создавать токены для передачи их синтаксическому анализатору.

4. Синтаксический анализ: разработка компонента, который анализирует лексический поток токенов и проверяет его на соответствие синтаксису грамматики. Обычно это выполняется с использованием алгоритма разбора сверху вниз (например, метод рекурсивного спуска) или снизу вверх (например, метод анализа LL(k)).

5. Построение дерева разбора: создание структуры данных (обычно дерева) для представления синтаксического анализа и связи между различными синтаксическими конструкциями. Дерево разбора может быть использовано для дальнейшего анализа и трансляции.

6. Семантический анализ: выполнение проверок и вычислений, связанных с семантикой исходного языка. Это может включать в себя проверки типов, разрешение ссылок, оптимизацию кода и другие операции.

7. Генерация целевого кода: создание выходного кода или исполняемого файла на основе анализа исходного кода и его семантики. Это может включать в себя создание промежуточного представления кода, оптимизацию и генерацию конечного кода.

8. Тестирование и отладка: проведение тестов для проверки работоспособности и корректности транслятора. Отладка ошибок и улучшение производительности и функциональности программы.

**Код программы:**

**uSyntAnalyzer.cs:**

using System;

using System.Collections.Generic;

using System.Windows.Forms;

using nsLex;

using Васильева\_4309;

namespace nsSynt

{

class uSyntAnalyzer

{

const int word = 0;

const int number = 0;

const int reserved = 0;

public TreeNode[] treeNodes = new TreeNode[1024];

int i = 1;

public String[] str = new String[20];

public int strNumber = 0;

public int indId = 0;

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 CHashTableList htl = new CHashTableList(3);

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

int intVLexicalCode = 0;

public uSyntAnalyzer()

{

treeNodes[0] = new TreeNode("S"); //создается дерево

}

public void S()

{

if (i > 0)

{

treeNodes[i] = new TreeNode("S");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

}

i++;

B();

if (Lex.enumPToken == TToken.lxmcommaDote)

{

Lex.NextToken();

B();

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, number, ref intVLexicalCode))

throw new Exception("В тексте не должно быть повторных чисел");

A();

i--;

if (Lex.enumPToken == TToken.lxmcommaDote)

{

treeNodes[i] = new TreeNode(";");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

Lex.NextToken();

A();

}

else throw new Exception("Ожидалась точка с запятой");

}

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

{

A();

i--;

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmcommaDote)

{

treeNodes[i] = new TreeNode(";");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

Lex.NextToken();

A();

i--;

if (Lex.enumPToken == TToken.lxmcommaDote)

{

treeNodes[i] = new TreeNode(";");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

Lex.NextToken();

A();

}

}

}

}

else throw new Exception("Ожидалась точка с запятой");

}

public void A()

{

if (i > 0)

{

treeNodes[i] = new TreeNode("A"); //создается ветвь дерева

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

}

i++;

if (Lex.enumPToken == TToken.lxmNumber)

{

if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, number, ref intVLexicalCode))

throw new Exception("В тексте не должно быть повторных чисел");

treeNodes[i] = new TreeNode("001");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(Lex.strPLexicalUnit, number, ref intVLexicalCode);

Lex.NextToken();

i--;

if (Lex.enumPToken == TToken.lxmcommaDote)

{

treeNodes[i] = new TreeNode(";");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(";", reserved, ref intVLexicalCode);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmUpperIdentifier)

{

treeNodes[i] = new TreeNode("A");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

i++;

treeNodes[i] = new TreeNode("ab");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(Lex.strPLexicalUnit, word, ref intVLexicalCode);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmcommaDote)

{

treeNodes[i] = new TreeNode(";");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(";", reserved, ref intVLexicalCode);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, number, ref intVLexicalCode)) // Проверяем, существует ли уже такое числовое значение в хэш-таблице

throw new Exception("В тексте не должно быть повторных чисел"); //в случае,если такое значение уже есть,вызывается исключение

treeNodes[i] = new TreeNode("A");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

i++;

treeNodes[i] = new TreeNode(Lex.strPLexicalUnit);

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(Lex.strPLexicalUnit, number, ref intVLexicalCode);

i--;

}

}

}

}

}

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

{

treeNodes[i] = new TreeNode(Lex.strPLexicalUnit);

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(Lex.strPLexicalUnit, word, ref intVLexicalCode);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmcommaDote)

{

treeNodes[i] = new TreeNode(";");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(";", reserved, ref intVLexicalCode);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

if (htl.SearchLexicalUnit(Lex.strPLexicalUnit, number, ref intVLexicalCode)) // Проверяем, существует ли уже такое числовое значение в хэш-таблице

throw new Exception("В тексте не должно быть повторных чисел"); //в случае,если такое значение уже есть,вызывается исключение treeNodes[i] = new TreeNode("A");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

i++;

treeNodes[i] = new TreeNode(Lex.strPLexicalUnit);

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(Lex.strPLexicalUnit, number, ref intVLexicalCode);

i--;

}

}

}

}

public void B()

{

if (i > 0)

{

treeNodes[i] = new TreeNode("B"); //создается ветвь дерева

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

}

i++;

if (Lex.enumPToken == TToken.lxmUpperIdentifier)

{

Lex.NextToken();

treeNodes[i] = new TreeNode("abcd");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

htl.AddLexicalUnit(Lex.strPLexicalUnit, word, ref intVLexicalCode);

i--;

if (Lex.enumPToken == TToken.lxmcommaDote)

{

treeNodes[i] = new TreeNode(";");

treeNodes[i - 1].Nodes.Add(treeNodes[i]);

}

}

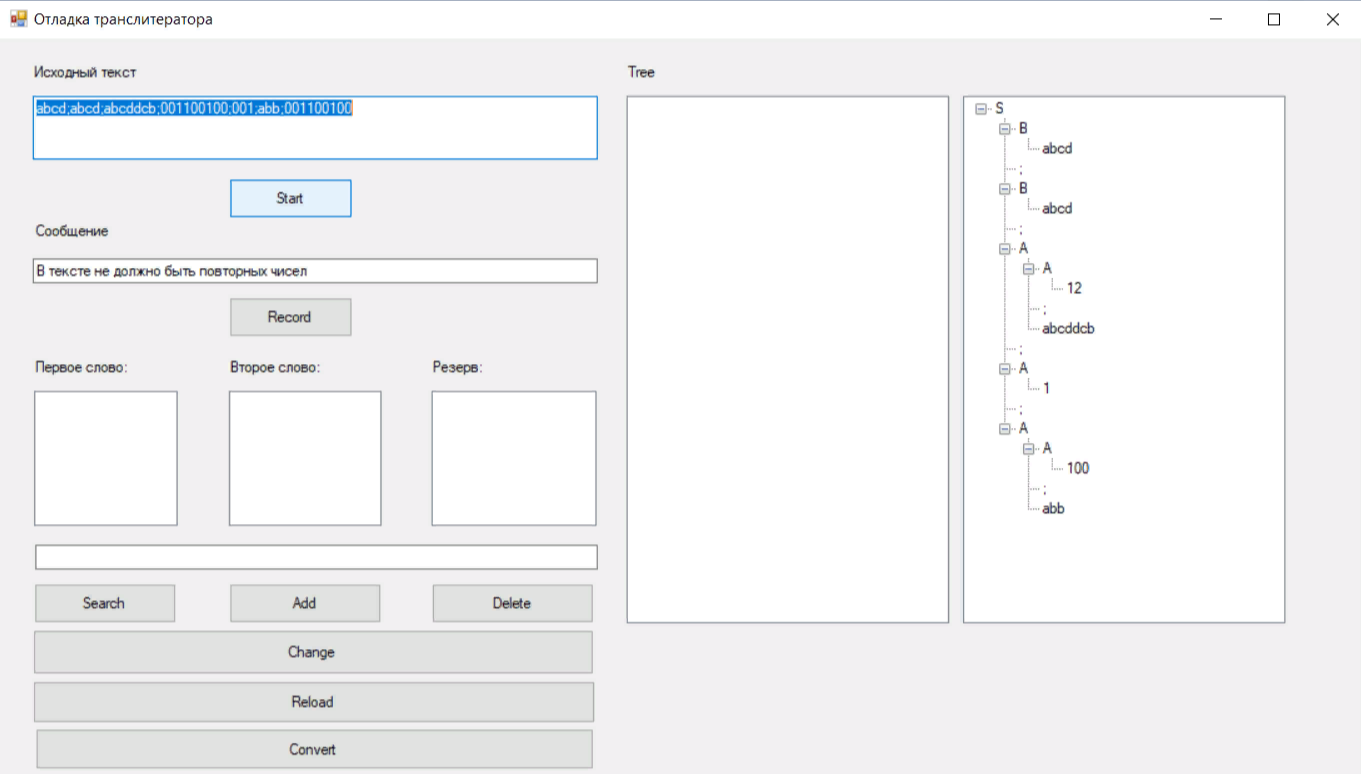
else throw new Exception("Ожидалось слово");

}

}

}

**Результат работы программы:**



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

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

**Вариант:** Проверка идентификатора слова <1> на наличие не более 2 букв ‘a’, проверка того, что длинна идентификатора слова <1> будет больше десятичного представления идентификатора числа <2>.

**Краткое теоретическое обоснование:**

Семантика – раздел лингвистики, изучающий смысловое значение единиц языка. Семантический анализатор - это компонент или алгоритм, который выполняет анализ семантики. Он позволяет понимать и интерпретировать значение слов, предложений и текстов.

Основная цель семантического анализатора - выявление смысловых связей между словами, обнаружение ключевых тем или концепций, анализ оттенков значения слов и выражений, а также понимание общего контекста коммуникации.

Основные методы, используемые в семантическом анализаторе, включают:

1. Лексико-семантический анализ: этот метод основан на анализе значений отдельных слов и их сочетаемости. Анализатор обращается к словарным базам данных или онтологиям, чтобы определить значение и различные семантические свойства слова.

2. Синтактический анализ: в этом методе анализируется структура предложений и текстов. Анализатор проверяет правильность грамматической конструкции и ищет зависимости между словами и их ролями (субъект, объект и т. д.).

3. Семантический ролевой анализ: здесь проводится анализ роли, которую играет каждое слово или выражение в предложении. Это может включать определение агента, пациента, инструмента и других элементов предложения.

4. Анализ контекста: анализатор учитывает текстовый контекст, т. е. учитывает смысл и значение, которые могут быть определены только на основе информации из предыдущих или последующих предложений в тексте.

5. Семантическая классификация: анализатор может классифицировать тексты или фразы по определенным категориям или темам, используя методы машинного обучения или статистические алгоритмы.

6. Анализ семантических связей: этот метод позволяет определить семантические связи между различными элементами, такими как синонимы, антонимы, гиперонимы и гипонимы, меры сходства и различия и т. д.

7. Экстракция информации: анализатор может извлекать информацию из текстов, например, имена собственные, даты, события, факты и т. д.

Все эти методы обычно комбинируются в различных сочетаниях, чтобы достичь лучшего понимания семантики и содержания текстов. Семантический анализатор может использоваться в различных приложениях, таких как поиск информации, обработка естественного языка, анализ текстов, виртуальные ассистенты и многих других.

**Код программы**:

**SemantAnalyzer.cs:**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Security.AccessControl;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Васильева\_4309

{

public class uSemantAnalyzer

{

public int i = 0;

public string strIndentifier;

public string strDigital;

private TreeView tree;

private TreeNode[] treeNodes;

public uSemantAnalyzer(TreeView treeView)

{

tree = treeView;

TreeController(tree);

}

public uSemantAnalyzer(TreeNode[] treeNodes)

{

this.treeNodes = treeNodes;

}

public void TreeController(TreeView tree)

{

foreach (TreeNode treeNodes in tree.Nodes)

{

TreeController(treeNodes);

}

}

public void TreeController(TreeNode treeNodes)

{

if (treeNodes.Text == "A")

{

if (treeNodes.Nodes.Count > 1)

{

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

strDigital = treeNodes.Nodes[2].Nodes[0].Text.ToString();

Check(strIndentifier, strDigital, treeNodes);

}

}

foreach (TreeNode treeNode1 in treeNodes.Nodes)

{

TreeController(treeNode1);

}

}

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

{

int countI = 0;

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

{

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

{

countI++; //Подсчет букв b

}

else

{

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

}

}

int countB = digit.Length / 3 - 1; //Подсчет допольнительных частей

Console.WriteLine(ident + " " + countI);

Console.WriteLine(digit + " " + countB);

if (countB != countI) //если количество дополнительных частей не равно количеству b

{

tree.SelectedNode = treeNodes;

tree.SelectedNode.BackColor = Color.Blue;

throw new Exception("Error: count b > count dop digit"); //вызывается исключение

}

}

}

}

**Form1.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using nsSynt;

using nsLex;

namespace Васильева\_4309

{

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("abcd;abcd;abcddc;001100;001;ab;001100100" + "\r\n");

int n = tbFSource.Lines.Length;

}

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

{

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

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

htl.TableToStringList(0, listTable);

foreach (var entry in hashTableIdentifier)

{

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

}

listTable.Clear();

htl.TableToStringList(1, listTable);

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

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

foreach (var entry in hashTableDigital)

{

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

}

listTable.Clear();

foreach (var entry in hashTableRezerv)

{

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

}

listTable.Clear();

}

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

{

tbFMessage.Clear();

SyntTree.Nodes.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();

SyntTree.Nodes.Clear();

SyntTree.Nodes.Add(Synt.treeNodes[1]);

SyntTree.ExpandAll();

SemantFunction();

}

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;

}

if (tbFMessage.Text == "") tbFMessage.Text = "Текст верный";

}

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

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmUpperIdentifier:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmdt):

{

hashFunction.AddWord(hashTableRezerv, ":");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmComma):

{

hashFunction.AddWord(hashTableRezerv, ",");

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

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

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmcommaDote):

{

hashFunction.AddWord(hashTableRezerv, ";");

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

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

{

TablesToMemo(this, e);

}

break;

}

}

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

tbFMessage.Text += m;

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

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

{

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

{

search.BackColor = Color.Green;

}

else

{

search.BackColor = Color.Red;

}

}

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

{

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

}

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

{

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

{

delete.BackColor = Color.Green;

}

else

{

delete.BackColor = Color.Red;

}

}

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

{

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

{

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

change.BackColor = Color.Green;

}

else

{

change.BackColor = Color.Red;

}

}

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

{

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

foreach (var entry in hashTableIdentifier)

{

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

}

foreach (var entry in hashTableDigital)

{

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

}

foreach (var entry in hashTableRezerv)

{

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

}

}

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

{

CopyTreeView(SyntTree, SyntTree2);

TreeConverter treeConverter = new TreeConverter();

treeConverter.ConvertBinaryToDecimal(SyntTree2);

SyntTree2.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(SyntTree); // Внутри блока try инициализируется объект Semant класса uSemantAnalyzer с передачей значения переменной SyntTree в качестве аргумента конструктора.

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message; //Если при инициализации объекта Semant возникнет исключение, оно будет перехвачено блоком catch, и сообщение об ошибке будет добавлено к тексту элемента управления tbFMessage.

}

}

}

**Результат работы кода:**

