Министерство образования Российской Федерации

Федеральное агентство по образованию

КАЗАНСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. А.Н. Туполева - КАИ

Кафедра АСОИУ

Лабораторные работы №1-3

по дисциплине

«ТЕОРИЯ фОРМАЛЬНЫХ ГРАММАТИК И АВТОМАТОВ»

Выполнил:

Студент группы 4309

Перов А. М.

Проверила: Бикмуллина И. И.

Казань 2023

**Лабораторная работа № 1. Разработка транслитератора**

**Текст задания:**

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

**Код программы:**

**Form1.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp79

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

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

tbFSource.AppendText("1 a");

int n = tbFSource.Lines.Length;

}

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

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

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

Lex.NextToken();

String s = "";

String s1 = "";

switch (Lex.enumFSelectionCharType)

{

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

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

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

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

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

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

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

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

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

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

}

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

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

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

}

}

**uLex.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp79

{

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

public enum TCharType { Letter, Digit, EndRow, EndText, Space, ReservedSymbol, BracketOpen, BracketClosed, ExclamationMark, Comma, Semicolon }; // тип символа

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

public class CLex //класс лексический анализатор

{

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

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

public TCharType enumFSelectionCharType;

public char chrFSelection;

private TState enumFState;

private int intFSourceRowSelection;

private int intFSourceColSelection;

private String strFLexicalUnit;

private TToken enumFToken;

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

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

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

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

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

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

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

public CLex()

{

}

public void GetSymbol() //метод класса лексический анализатор

{

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

{

intFSourceRowSelection++;

if (intFSourceRowSelection <= strFSource.Length - 1)

{

intFSourceColSelection = -1;

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndRow;

enumFState = TState.Continue;

}

else

{

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndText;

enumFState = TState.Finish;

}

}

else

{

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

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

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

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

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

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

else if (chrFSelection == '(')

enumFSelectionCharType = TCharType.BracketOpen;

else if (chrFSelection == ')')

enumFSelectionCharType = TCharType.BracketClosed;

else if (chrFSelection == ',')

enumFSelectionCharType = TCharType.Comma;

else if (chrFSelection == '!')

enumFSelectionCharType = TCharType.ExclamationMark;

else if (chrFSelection == ';')

enumFSelectionCharType = TCharType.Semicolon;

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

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

enumFState = TState.Continue;

}

intFSourceColSelection++;

}

private void TakeSymbol()

{

char[] c = { chrFSelection };

String s = new string(c);

strFLexicalUnit += s;

GetSymbol();

}

public void NextToken()

{

strFLexicalUnit = "";

if (enumFState == TState.Start)

{

intFSourceRowSelection = 0;

intFSourceColSelection = -1;

GetSymbol();

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

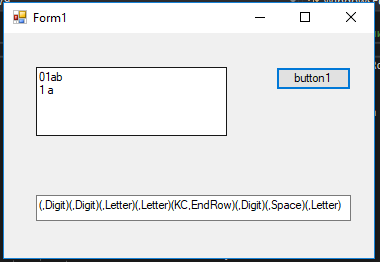
}

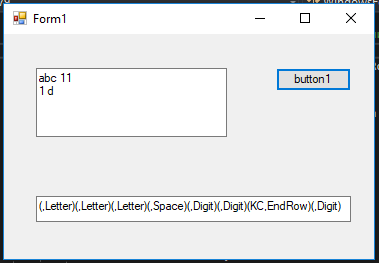
}

}

}

**Результаты тестирования:**





**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.

|  |  |  |
| --- | --- | --- |
| (011)\*000(001)\* | (a|b|c|d)+ | Вторые два символа всегда ab |

**Первое слово:**

(011)\*000(001)\*

A → 0B | 0C

B → 1D

C → 0Е

D → 1А

E → 0 | 0F

F → 0G

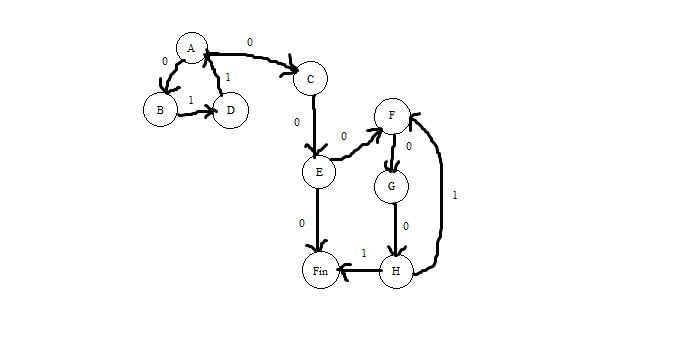
G → 0H

H → 1 | 1F

**Недетерминированная матрица:**

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

**Граф:**



**Детерминированная матрица:**

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| A | BC |  |
| BC | E | D |
| D |  | A |
| E | FFin |  |
| FFin | G |  |
| G | H |  |
| H | FFin |  |

**Второе слово:**

(a|b|c|d)+

Вторые 2 символа всегда ab

A → aB | bB | cB | dB

B → aC

C → b | bD

D → a | b | c | d | aFin | bFin | cFin | dFin

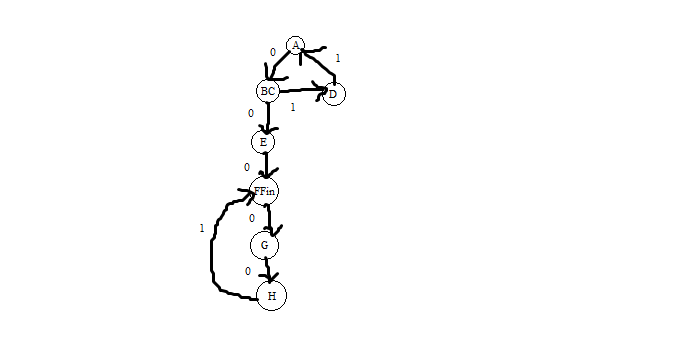
**Недетерминированная матрица:**

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

**Детерминированная матрица:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | a | b | c | d |
| A | B | B | B | B |
| B | C |  |  |  |
| C |  | DFin |  |  |
| DFin | DFin | DFin | DFin | DFin |

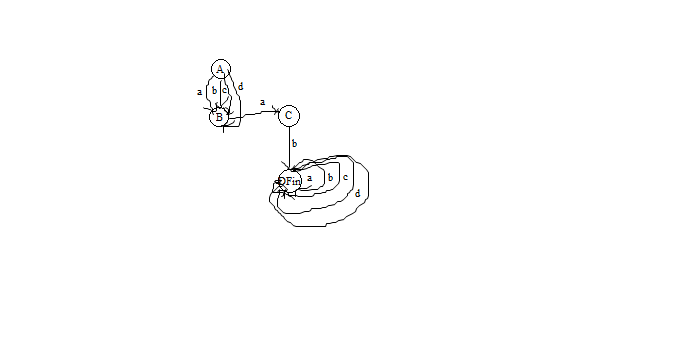
**Граф:**



**Граф:**



**Граф:**



**Код программы:**

**Form1.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp65

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

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

tbFSource.AppendText("");

int n = tbFSource.Lines.Length;

}

private void btnFStart\_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

{

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

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

String s = "";

String s1 = "";

switch (Lex.enumPToken)

{

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

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

}

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

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

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

}

}

**uLex.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp65

{

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

public enum TCharType { Letter, Digit, EndRow, EndText, Space, ReservedSymbol, LScob, RScob, Vosznak, Zapiatya, Tochkaszap }; // тип символа

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

public class CLex //класс лексический анализатор

{

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

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

public TCharType enumFSelectionCharType;

public char chrFSelection;

private TState enumFState;

private int intFSourceRowSelection;

private int intFSourceColSelection = -1;

private String strFLexicalUnit;

private TToken enumFToken;

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

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

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

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

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

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

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

public CLex()

{

}

public void GetSymbol() //метод класса лексический анализатор

{

intFSourceColSelection++; // продвигаем номер колонки

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

{

intFSourceRowSelection++;

if (intFSourceRowSelection <= strFSource.Length - 1)

{

intFSourceColSelection = -1;

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndRow;

enumFState = TState.Continue;

}

else

{

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndText;

enumFState = TState.Finish;

}

}

else

{

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

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

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

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

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

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

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

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

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

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

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

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

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

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

enumFState = TState.Continue;

}

}

private void TakeSymbol()

{

char[] c = { chrFSelection };

String s = new string(c);

strFLexicalUnit += s;

GetSymbol();

}

public void NextToken()

{

strFLexicalUnit = "";

if (enumFState == TState.Start)

{

intFSourceRowSelection = 0;

intFSourceColSelection = -1;

GetSymbol();

}

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

{

GetSymbol();

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// A | B | B | B | B |

// B | C | | | |

// C | |DFin| | |

// DFin|DFin|DFin|DFin|DFin|

A:

{

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

{

TakeSymbol();

goto B;

}

else throw new Exception("Ошибка");

}

B:

{

if (chrFSelection == 'a')

{

TakeSymbol();

goto C;

}

else throw new Exception("Вторым символом должен быть 'a'");

}

C:

{

if (chrFSelection == 'b')

{

TakeSymbol();

goto DFin;

}

else throw new Exception("Третьим символом должен быть 'b'");

}

DFin:

{

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

{

TakeSymbol();

goto DFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

case TCharType.Digit:

{

// 0 1

// A | BC | |

// BC | E | D |

// D | | A |

// E | FFin| |

// FFin| G | |

// G | H | |

// H | |FFin |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto BC;

}

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

BC:

if (chrFSelection == '0')

{

TakeSymbol();

goto E;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

else throw new Exception("Ожидался 0 или 1 #2");

D:

if (chrFSelection == '1')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидалась 1 #3");

E:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0 #4");

FFin:

if (chrFSelection == '0')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидался 0 #5");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 0 #6");

H:

if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидалась 1 #7");

}

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;

}

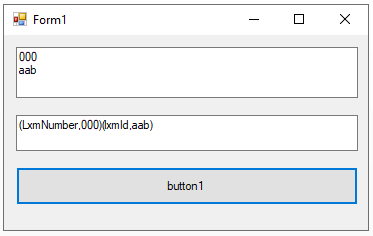
}

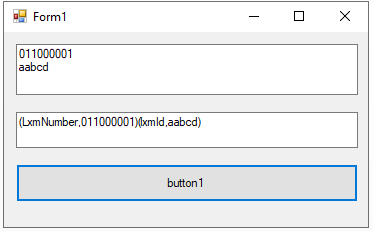
}

}

}

**Результаты тестирования:**

****

****

**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.

**Вариант 20:**

П 🡪 П;К | К

К 🡪 OR,A

O 🡪 <2>

R 🡪 <1>

A 🡪 <1> | <2>

Избавление от левой рекурсии:

П 🡪 K | ПC

C 🡪 ;K | ;C

K 🡪 OR,A

O 🡪 <2>

R 🡪 <1>

A 🡪 <1> | <2>

1: П 🡪 К 🡪 OR,A 🡪 <2><1>,<2>

2: П 🡪 K 🡪 OR,A 🡪 <2><1>,<1>

3: П 🡪 П;К 🡪 K;K 🡪 OR,A;OR,A 🡪 <2><1>,<2>;<2><1>,<2>

**Код программы:**

**Form1.cs**

namespace WinFormsApp1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

tbFSource.AppendText("");

int n = tbFSource.Lines.Length;

}

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

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.Lines;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.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;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WinFormsApp1

{

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

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

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

public class CLex //класс лексический анализатор

{

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

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

public TCharType enumFSelectionCharType;

public char chrFSelection;

private TState enumFState;

private int intFSourceRowSelection;

private int intFSourceColSelection;

private String strFLexicalUnit;

private TToken enumFToken;

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

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

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

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

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

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

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

public CLex()

{

}

public void GetSymbol() //метод класса лексический анализатор

{

intFSourceColSelection++; // продвигаем номер колонки

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

{

intFSourceRowSelection++;

if (intFSourceRowSelection <= strFSource.Length - 1)

{

intFSourceColSelection = -1;

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndRow;

enumFState = TState.Continue;

}

else

{

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndText;

enumFState = TState.Finish;

}

}

else

{

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

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

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

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

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

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

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

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

enumFState = TState.Continue;

}

}

private void TakeSymbol()

{

char[] c = { chrFSelection };

String s = new string(c);

strFLexicalUnit += s;

GetSymbol();

}

public void NextToken()

{

strFLexicalUnit = "";

if (enumFState == TState.Start)

{

intFSourceRowSelection = 0;

intFSourceColSelection = -1;

GetSymbol();

}

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

{

GetSymbol();

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// A | B | B | B | B |

// B | C | | | |

// C | |DFin| | |

// DFin|DFin|DFin|DFin|DFin|

A:

{

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

{

TakeSymbol();

goto B;

}

else throw new Exception("Ошибка");

}

B:

{

if (chrFSelection == 'a')

{

TakeSymbol();

goto C;

}

else throw new Exception("Вторым символом должен быть 'a'");

}

C:

{

if (chrFSelection == 'b')

{

TakeSymbol();

goto DFin;

}

else throw new Exception("Третьим символом должен быть 'b'");

}

DFin:

{

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

{

TakeSymbol();

goto DFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

case TCharType.Digit:

{

// 0 1

// A | BC | |

// BC | E | D |

// D | | A |

// E | FFin| |

// FFin| G | |

// G | H | |

// H | |FFin |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto BC;

}

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

BC:

if (chrFSelection == '0')

{

TakeSymbol();

goto E;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

else throw new Exception("Ожидался 0 или 1 #2");

D:

if (chrFSelection == '1')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидалась 1 #3");

E:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0 #4");

FFin:

if (chrFSelection == '0')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидался 0 #5");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 0 #6");

H:

if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидалась 1 #7");

}

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.lxmtz;

GetSymbol();

return;

}

if (chrFSelection == '=')

{

enumFToken = TToken.lxmr;

GetSymbol();

return;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

}

}

}

}

uSyntAnalyzer.cs:

using System;

using System.Collections.Generic;

using System.Text;

namespace WinFormsApp1

{

class uSyntAnalyzer

{

private String[] strFSource;

private String[] strFMessage;

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

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

public CLex Lex = new CLex();

public void S()

{

K();

if (Lex.enumPToken == TToken.lxmtz)

C();

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

}

public void C()

{

if (Lex.enumPToken == TToken.lxmtz)

{

Lex.NextToken();

K();

if (Lex.enumPToken == TToken.lxmtz)

C();

}

}

public void K()

{

O();

if (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

R();

if (Lex.enumPToken == TToken.lxmComma)

{

Lex.NextToken();

A();

}

else throw new Exception("Ожидалась ,");

}

}

public void O()

{

if (Lex.enumFSelectionCharType == TCharType.Letter)

{

Lex.NextToken();

}

}

public void R()

{

if (Lex.enumPToken == TToken.lxmNumber)

{

Lex.NextToken();

}

else throw new Exception("Ожидался идентификатор (прим. 000)");

}

public void A()

{

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

{

Lex.NextToken();

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

{

throw new Exception("Ожидалась ;");

}

}

else throw new Exception("Ожидался идентификатор (aab или 000)");

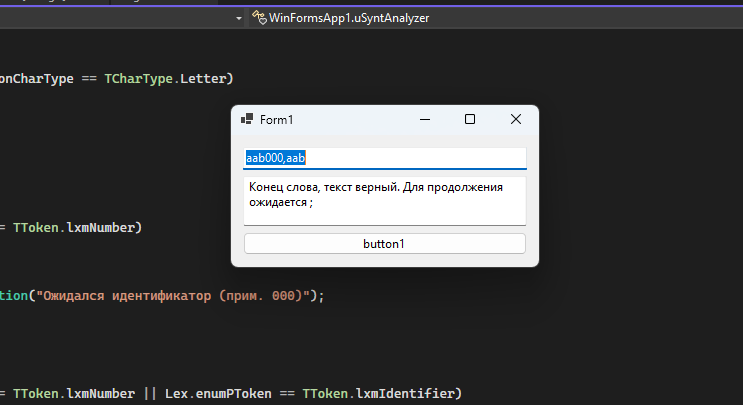
}

}

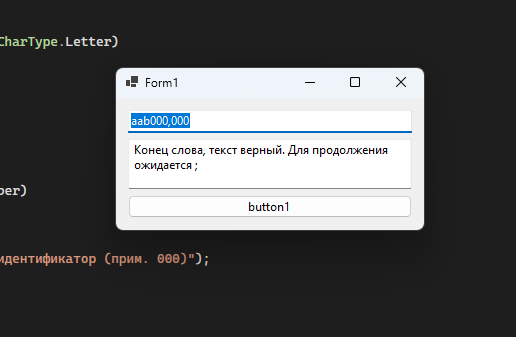
}

**Результаты тестирования:**

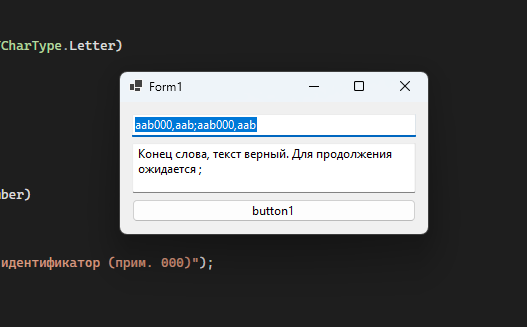
**1:**

****

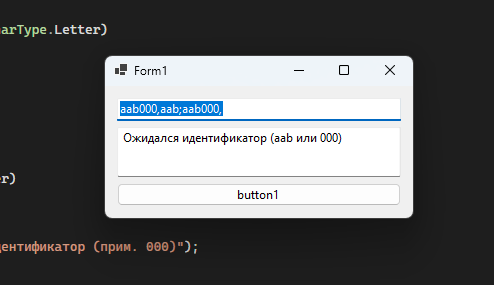
**2:**

****

**3:**

****

**4:**

****