Министерство образования Российской Федерации

Федеральное агентство по образованию

КАЗАНСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. А.Н. Туполева - КАИ

Кафедра АСОИУ

Лабораторная работа №1

по дисциплине

«ТЕОРИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ»

Выполнил:

Студент группы 4309

Ильичев Д.А.

Проверил: Бикмуллина И.АИ.

Казань 2024

**Лабораторная работа № 1. Разработка транслитератора**

**Литера** — буква, один из знаков азбуки; в типографии наборная буква.

**Транслитератор** — это инструмент, который помогает переводить текст с одного языка на другой, сохраняя при этом структуру и порядок слов.

Существует два основных подхода к созданию транслитераторов: машинное обучение (ML) и основанный на правилах (rule-based).

Подход с использованием машинного обучения строится на вероятностных моделях последовательностей, таких как марковские модели. В основе этого метода лежит алгоритм Витерби: сегментация латинских слов и поиск наиболее вероятной последовательности скрытых состояний — кириллических комбинаций.

Этот метод можно рассматривать как задачу машинного перевода: создаётся таблица соответствий букв, и на её основе модель обучается вероятностным преобразованиям. ML требуется качественный датасет, который необходимо собрать.

Теперь перейдём к подходу, основанному на правилах. Для этого используются словари, таблицы сочетаемости букв.

**Как работает транслитератор:**

1. Анализируем входную последовательность символов.
2. Ищет комбинацию в заданном классе, словаре или в таблице транслитерации.
3. Если комбинация найдена, то преобразуем её и корректируем несочетаемости символов с помощью регулярных выражений.

Некоторые последовательности могут иметь неоднозначные интерпретации на русском языке. Для решения таких ситуаций создаются правила:

1. Преобразование CH. В начале слова это может быть Ч, Х или К, в конце — Ч или Х. Простой способ: если перед CH стоит И или Ы, то это преобразуется в КР, иначе — в ХР.
2. Разделительный твёрдый знак. Проверяем приставки, и, если следующий символ — йотированная гласная (Е, Ё, Ю, Я), ставим твёрдый знак перед ней.
3. Йотированные гласные. Они идут после приставок с твёрдым знаком или в начале слова. Ошибки возможны (например, IONY → ЁНЫ вместо ИОНЫ), но это можно минимизировать через словари, где для слов с ИО часто следуют буквы ТАД.
4. ИЙ или ЫЙ? Если перед ними стоят звуки ГДЖКЦЧШЩ, то используется ИЙ, иначе — ЫЙ.

Как можно улучшить эту модель:

1. Распознавать мягкий знак.
2. Использовать актуальный орфографический словарь для корректировки правил и их веса, что может постепенно привести к ML-подходу.
3. Добавить спеллчекер (система проверки правописания) для улучшения результатов транслитерации.

**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.Text;

using System.Windows.Forms;

namespace Translator

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

tbFSource.Text = "";

tbFSource.Text = "";

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.EngLetter: { s1 = "EngLetter"; break; }

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

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

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

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

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

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

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

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

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

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

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

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

case TCharType.Minus: { s1 = "Mius"; break; }

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

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

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

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

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

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

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

}

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

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 Translator

{

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

public enum TCharType { EngLetter, RusLetter, Digit, EndRow, EndText, Space, Star, Slash, Exclamation, Equal, Semicolon, AnotherSymbol, OpenBracket, EndBracket, Colon, OpenSquadBracket, EndSquadBracket, Plus, Minus, Comma, Dot, NoInd}; // 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 <= 'z') enumFSelectionCharType = TCharType.EngLetter;

else if (chrFSelection >= 'а' && chrFSelection <= 'я') enumFSelectionCharType = TCharType.RusLetter;

else if (chrFSelection >= '0' && chrFSelection <= '9') enumFSelectionCharType = TCharType.Digit;

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

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

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

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

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

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

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

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

else if (chrFSelection == '[') enumFSelectionCharType = TCharType.OpenSquadBracket;

else if (chrFSelection == ']') enumFSelectionCharType = TCharType.EndSquadBracket;

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

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

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

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

else if (chrFSelection == '^' || chrFSelection == '%' || chrFSelection == '@' || chrFSelection == '<' || chrFSelection == '>' || chrFSelection == '?') enumFSelectionCharType = TCharType.AnotherSymbol;

else enumFSelectionCharType = TCharType.NoInd;

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();

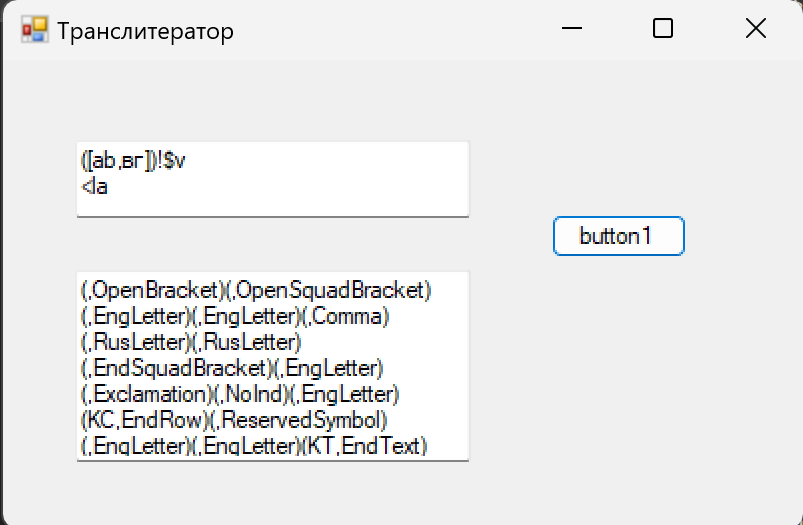
}

}

}

}

**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.

|  |  |  |
| --- | --- | --- |
| (001)\*101(110)\* | (a|b|c|d)+ | Длина не более 4 |

**Первое слово:**

(001)\*101(110)\*

A → 0B | 1C

B → 0D

C → 0Е

D → 1А

E → 1 | 1F

F → 1G

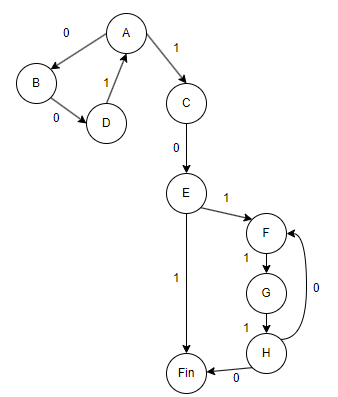
G → 1H

H → 0 | 0F

**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 | B | C |
| B | D |  |
| C | E |  |
| D |  | A |
| E |  | FFin |
| FFin |  | G |
| G |  | H |
| H | FFin |  |

**Второе слово:**

(a|b|c|d)+

Длина не более 4

A → a | b | c | d | aB | bB | cB | dB

B → a | b | c | d | aC | bC | cC | dC

C → a | b | c | d | aD | bD | cD | dD

D → a | b | c | d

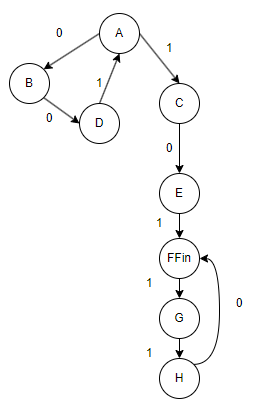
**Недетерминированная матрица:**

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

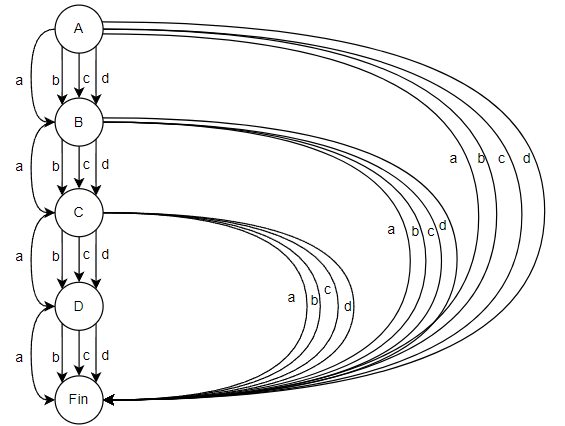
**Детерминированная матрица:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | a | b | c | d |
| A | BFin | BFin | BFin | BFin |
| BFin | CFin | CFin | CFin | CFin |
| CFin | D | D | D | D |
| D | Fin | Fin | Fin | Fin |
| Fin |  |  |  |  |

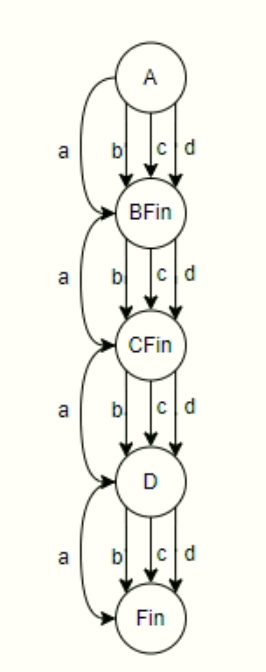
**Граф:**



**Граф:**



**Граф:**



**Код программы:**

**Form1.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

namespace Lab2

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

tbFSource.AppendText("000101110" + "\r\n");

tbFSource.AppendText("adbc");

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;

Lex.intPSourceColSelection = -1;

Lex.intPSourceRowSelection = 0;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

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

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

String s = "";

String s1 = "";

switch (Lex.enumPToken)

{

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

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

}

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

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

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

}

}

**uLex.cs**

using System;

using System.Collections.Generic;

using System.Text;

namespace Lab2

{

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 };

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 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();

}

// Вариант 6

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 'ad'");

}

B:

{

if (chrFSelection == 'd')

{

TakeSymbol();

goto CFin;

}

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

}

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 | B | C |

// B | D | |

// C | E | |

// D | A | |

// E | |FFin |

// FFin | | G |

// G | H | |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto B;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto C;

}

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

B:

if (chrFSelection == '0')

{

TakeSymbol();

goto D;

}

else throw new Exception("Ожидался 0");

C:

if (chrFSelection == '0')

{

TakeSymbol();

goto E;

}

else throw new Exception("Ожидался 0");

D:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидался 0");

E:

if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидалась 1");

FFin:

if (chrFSelection == '1')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидалась 1");

G:

if (chrFSelection == '1')

{

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;

}

}

}

}

}

**Результаты тестирования:**

