Министерство образования Российской Федерации

Федеральное агентство по образованию

КАЗАНСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. А.Н. Туполева - КАИ

Кафедра АСОИУ

Лабораторная работа №2

по дисциплине

«ТЕОРИЯ языков программирования и методов трансляций»

по теме «Разработка лексического анализатора»

Вариант 3

Выполнил:

Студент группы 4309

Габдуллин С.Ж.

Проверила: Бикмуллина И.И.

Казань 2023

**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.

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.

|  |  |  |  |
| --- | --- | --- | --- |
| 33 | (000)\*010(011)\* | (a|b|c|d)+ | Нет подстроки bb |

**Первое слово:**

(000)\*010(011)\*

A → 0B

B → 0C|1D

C → 0A

D → 0|0E

E → 0F

F → 1G

G → 1|1E

**Недетерминированная матрица:**

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

**Детерминированная матрица:**

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

**Второе слово:**

(a|b|c|d)+

Нет подстроки bb

A → a | b | c | d | aB | bC | cB | dB

B → a | b | c | d | aB | bC | cB | dB

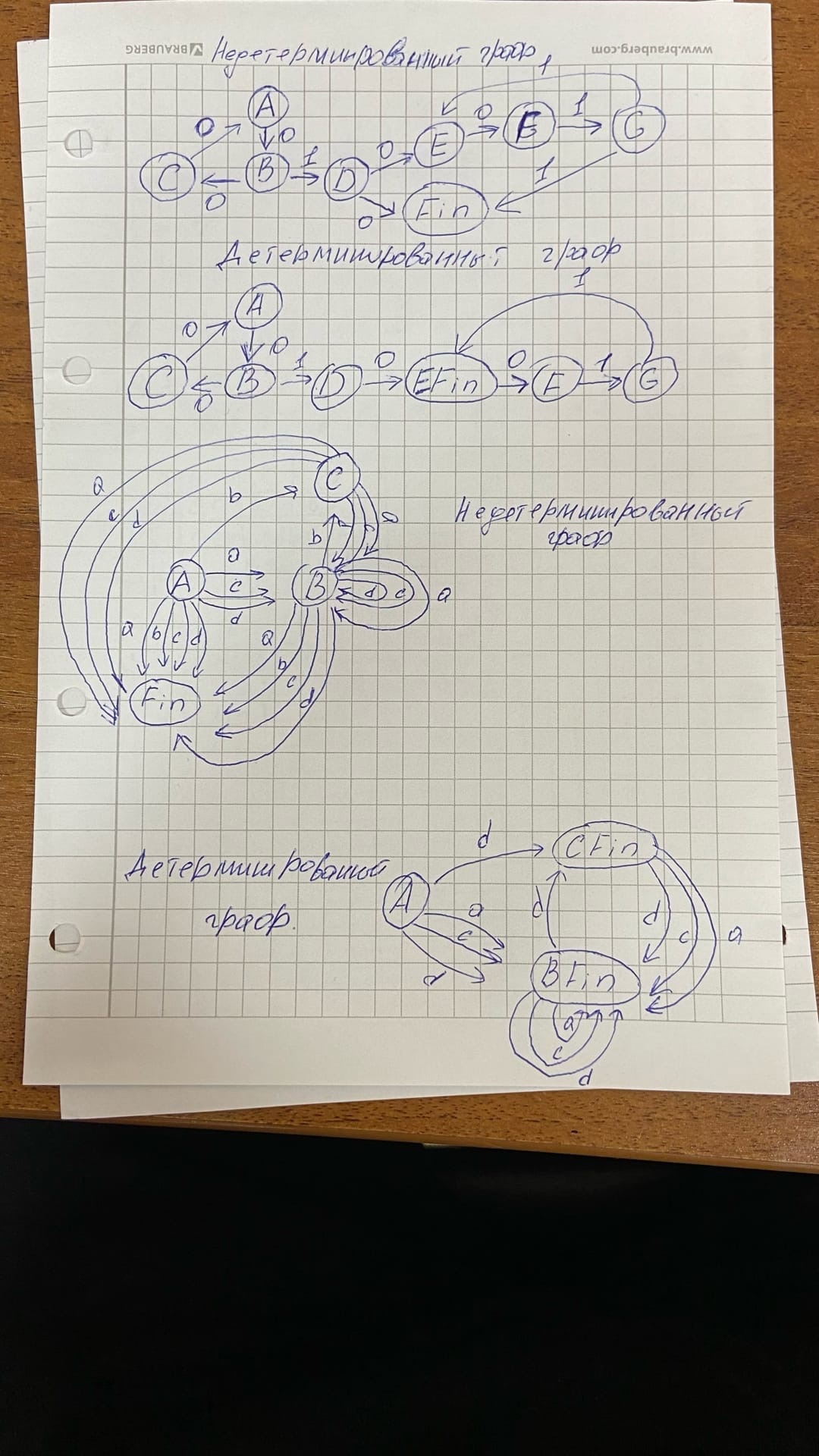
C → a | c | d | aB | cB | dB

**Недетерминированная матрица:**

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

**Детерминированная матрица:**

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



**Исходный текст программы:**

**uMainForm.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 nsLexMainForm

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

// textBox1.Lines.Count = 2;

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

textBox1.AppendText("110 111110 110000 bad\r\n");

textBox1.AppendText(" cb ab");

// int n = textBox1.Lines.Length;

// n = n;

}

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

{

CLex Lex = new CLex();

Lex.strPSource = textBox1.Lines;

Lex.strPMessage = textBox2.Lines;

Lex.enumPState = TState.Start;

textBox2.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

/\* отладка String s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier: { s = "Identifier"; break; }

case TToken.lxmNumber: { s = "Numder"; break; }

case TToken.lxmEmpty: { s = "EndText"; break; }

}

textBox2.Text += "(" + Lex.strPLexicalUnit + "," + s + ")";

\*/

}

textBox2.Text = "Текст верный";

}

catch (Exception exc)

{

textBox2.Text += exc.Message;

textBox1.Select();

textBox1.SelectionStart = 0;

int n = 0;

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

n += Lex.intPSourceColSelection;

textBox1.SelectionLength = n;

}

}

}

}

**uLex.cs:**

using System;

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(bool comm)

{

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 == ':') || (chrFSelection == ';') || (chrFSelection == ',') || (chrFSelection == '/')) enumFSelectionCharType = TCharType.ReservedSymbol;

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

enumFState = TState.Continue;

}

}

private void TakeSymbol()

{

char[] c = { chrFSelection };

String s = new string(c);

strFLexicalUnit += s;

GetSymbol(false);

}

public void NextToken()

{

strFLexicalUnit = "";

if (enumFState == TState.Start)

{

intFSourceRowSelection = 0;

intFSourceColSelection = -1;

GetSymbol(false);

}

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

{

GetSymbol(false);

}

if (chrFSelection == '/')

{

GetSymbol(false);

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol(true);

}

GetSymbol(false);

}

// Вариант 3

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

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

// BFin |BFin|CFin|BFin|BFin|

// CFin |BFin| |BFin|BFin|

A:

{

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

{

TakeSymbol();

goto BFin;

}

else if (chrFSelection == 'b')//|| chrFSelection == 'b' || chrFSelection == 'd')

{

TakeSymbol();

goto CFin;

}

else throw new Exception("Ошибка в идентификаторе");

}

BFin:

{

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

{

TakeSymbol();

goto BFin;

}

else if (chrFSelection == 'b')

{

TakeSymbol();

goto CFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

CFin:

{

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

{

TakeSymbol();

goto BFin;

}

else if (chrFSelection == 'b')

{

throw new Exception("Не соблюдено условие грамматики: Не должно быть подстроки bb");

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

}

case TCharType.Digit:

{

// 0 1

// A | B | |

// B | C | D |

// C | A | |

// D |EFin| |

// EFin | F | |

// F | | G |

// G | |EFin|

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto B;

}

else throw new Exception("Ожидалось 1");

B:

if (chrFSelection == '0')

{

TakeSymbol();

goto C;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

else throw new Exception("Ожидалось продолжение");

C:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

// else if (chrFSelection == '0')

// {

// TakeSymbol();

// goto DFin;

//}

else throw new Exception("Ожидалась цифра");

D:

if (chrFSelection == '0')

{

TakeSymbol();

goto EFin;

}

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

else throw new Exception("Ожидалось 0");

EFin:

if (chrFSelection == '0')

{

TakeSymbol();

goto F;

}

else

if (chrFSelection == '1') throw new Exception("Ожидалось 0 или конец слова");

else { enumFToken = TToken.lxmNumber; return; }

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

//else throw new Exception("Ожидалось 1");

F:

if (chrFSelection == '1')

{

TakeSymbol();

goto G;

}

else throw new Exception("Ожидалось 1");

G:

if (chrFSelection == '1')

{

TakeSymbol();

goto EFin;

}

else throw new Exception("Ожидалось 1");

}

case TCharType.ReservedSymbol:

{

if (chrFSelection == '/')

{

GetSymbol(false);

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol(true);

}

GetSymbol(false);

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

case TCharType.Space:

{

GetSymbol(false);

break;

}

}

}

}

}

**Результаты тестирования:**

