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 фОРМАЛЬНЫХ ГРАММАТИК И АВТОМАТОВ»

Выполнила:

Студентка группы 4309

Газимзянова А.Д.

Проверила: Бикмуллина И.И.

Казань 2023

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

**Теория**

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

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

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

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

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

*Существует несколько основных операций над матрицами:*

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

**Текст задания:**

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

**Вариант задания:**



**Первое слово:**

(000)\*010(100)\*

A → 0B

B → 0C|1D

C → 0A

D → 0|0F

F → 1G

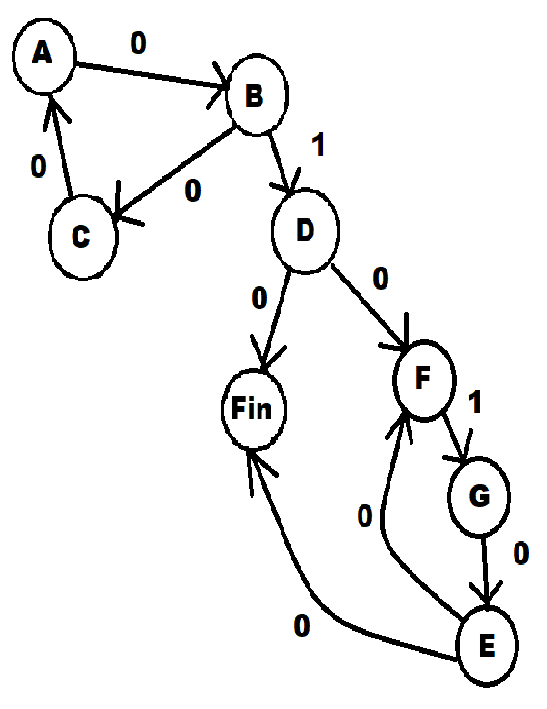
G → 0E

E → 0|0F

**1. Недетерминированная матрица:**

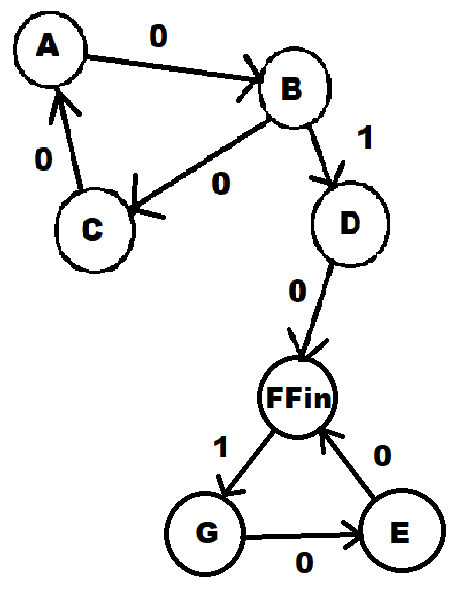
|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| A | B |  |
| B | C | D |
| C | A |  |
| D | F,Fin |  |
| E | F,Fin |  |
| F |  | G |
| G | E |  |

**Граф:**



**2. Детерминированная матрица: Граф:**

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

****

**Второе слово:**

(a|b|c|d)+

Не должно начинаться с ab

A → a|b|c|d| aB|bC|cC|dC

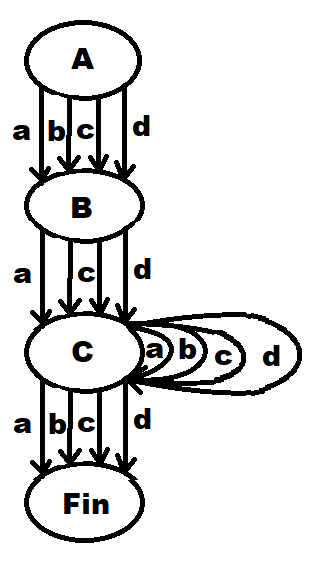
B → a|c|d| aC|cC|dC

C → a|b|c|d| aC|bC|cC|dC

**1. Недетерминированная матрица:**

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

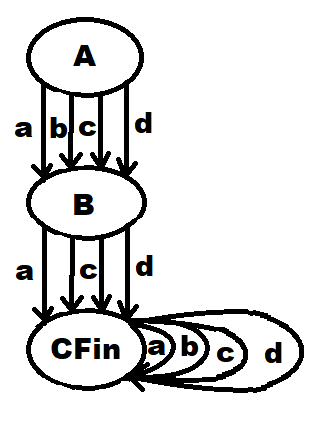
**Граф:**



**2. Детерминированная матрица:**

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

**Граф:**



**Код программы:**

**Form1.cs**

using System;

using System.Windows.Forms;

namespace WindowsFormsApp46

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

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

textBox1.AppendText("000010100");

}

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

{

}

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

{

uLex Lex = new uLex();

Lex.strPSource = textBox1.Lines;

Lex.strPMessage = textBox2.Lines;

Lex.enumPState = TState.Start;

textBox2.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

}

textBox2.Text = "Текст верный";

}

catch (Exception exc)

{

textBox2.Text += exc.Message;

textBox1.Select();

textBox1.SelectionStart = 0;

int n = 0;

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

n += Lex.intPSourceColSelection;

textBox1.SelectionLength = n;

}

}

}

}

**uLex.cs**

using System;

namespace WindowsFormsApp46

{

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

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

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

public class uLex

{

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

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

public TCharType enumFSelectionCharType;

public char chrFSelection;

private TState enumFState;

private int intFSourceRowSelection;

private int intFSourceColSelection;

private String strFLexicalUnit;

private TToken enumFToken;

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

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

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

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

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

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

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

public uLex()

{

}

public void GetSymbol() //метод класса лексический анализатор

{

// продвигаем номер колонки

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

{

intFSourceRowSelection++;

if (intFSourceRowSelection <= strFSource.Length - 1)

{

intFSourceColSelection = -1;

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndRow;

enumFState = TState.Continue;

}

else

{

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndText;

enumFState = TState.Finish;

}

}

else

{

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

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

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

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

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

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

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

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

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

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

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

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

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

enumFState = TState.Continue;

}

}

private void TakeSymbol()

{

char[] c = { chrFSelection };

String s = new string(c);

strFLexicalUnit += s;

GetSymbol();

}

public void NextToken()

{

strFLexicalUnit = "";

if (enumFState == TState.Start)

{

intFSourceRowSelection = 0;

intFSourceColSelection = 1;

GetSymbol();

}

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

{

GetSymbol();

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

//вариант 4

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

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

// B | CFin | |CFin| CFin |

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

A:

{

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

{

TakeSymbol();

goto B;

}

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

}

B:

{

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

{

TakeSymbol();

goto CFin;

}

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

}

CFin:

{

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

{

TakeSymbol();

goto CFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

}

case TCharType.Digit:

{

// 0 1

// A | B | |

// B | C | D |

// C | A | |

// D |FFin | |

// E |FFin | |

// FFin | | G |

// G | E | |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto B;

}

else throw new Exception("Ожидался 0");

B:

if (chrFSelection == '0')

{

TakeSymbol();

goto C;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

else throw new Exception("Ожидался 0");

C:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидался 0");

D:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0");

FFin:

if (chrFSelection == '1')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидался 1");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto E;

}

else throw new Exception("Ожидался 0");

E:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

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

else throw new Exception("Ожидался 0");

}

case TCharType.ReservedSymbol:

{

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

{

while (enumFSelectionCharType != TCharType.EndRow)

GetSymbol();

}

GetSymbol();

}

if (chrFSelection == '(')

{

enumFToken = TToken.lxmLeftParenth;

GetSymbol();

return;

}

if (chrFSelection == ')')

{

enumFToken = TToken.lxmRightParenth;

GetSymbol();

return;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

}

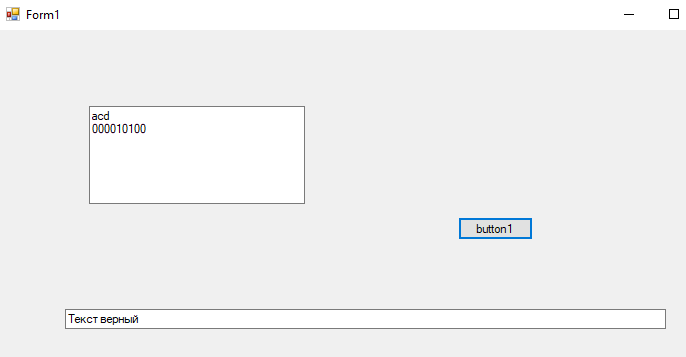
}

}

}

**Результат работы программы:**

Верно:



С ошибкой:

