**Лабораторная работа №2**

Выполнил: студент группы 4309

Курбанов Д.И.

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

Основные функции лексического анализатора:

Чтение с помощью транслитератора очередного слова в исходном

тексте и его класси-фикация;

Пропуск пробелов и комментариев;

Выдача диагностических сообщений об обнаруженных

лексических ошибках.

Токен – некоторый класс слов (выделяется из потребностей синтаксического анализа в основном по принципу построения и назначения слов).

Лексический анализатор – программа, принимающая на вход текст (последовательность символов из алфавита) и разбивающая его на подстроки (лексемы) в соответствии с некоторым набором регулярных выражений. Последовательность лексем такова, что каждая лексема принадлежит хотя бы одному из языков, задаваемых регулярными выражениями.

В детерминированном алгоритме для данного конкретного входного сигнала компьютер всегда будет выдавать один и тот же выходной сигнал, проходящий через одни и те же состояния, но в случае недетерминированного алгоритма для одного и того же входного сигнала компилятор может выдавать разные выходные данные в разных запусках. На самом деле, недетерминированные алгоритмы не могут решить проблему за полиномиальное время и не могут определить, каков следующий шаг. Недетерминированные алгоритмы могут демонстрировать разное поведение для одних и тех же входных данных при разном выполнении, и в этом есть определенная степень случайности.

Элементарные операции над матрицами - это операции, выполняемые над строками и столбцами матрицы, которые не изменяют значение матрицы. Матрица - это способ представления чисел в виде массива, т.е. числа расположены в виде строк и столбцов. В матрице строки и столбцы содержат все значения в ячейке. Мы представляем матрицу в виде [A]m × n, где A - матрица, m - количество строк в матрице, n - количество столбцов матрицы.

**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**.

Для отладки лексического анализатора временно включить в обработчик нажатия кнопки цикл чтения слов исходного текста и вывода результатов лексического анализа.

**Первое слово:**

(010)\*000(001)\*

A-> 0B | 0С

B-> 1D

С-> 0E

D-> 0A

E-> 0F | 0

F-> 0G

G-> 0H

H -> 1 | 1F

Детерминированная :

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

Недетерминированная :

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

Второе слово:

(a | b | c | d)\*

A-> cB

B-> aC

C-> bC | aC | cC | dC

Недетерминированная:

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

Детерминированная:

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

текст программы:

using System;

using System.Collections.Generic;

using System.Text;

namespace \_2\_ая\_лаба

{

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

public enum TCharType { Letter, Digit, EndRow, EndText, Space, ReservedSymbol, Undefined, Opendoor, Closedoor, point, talepoint, talepointPoint }; // 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 == '(') enumFSelectionCharType = TCharType.Opendoor;

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

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

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

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

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();

}

// Вариант 14

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// A | | | B | |

// B |CFin| | | |

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

A:

{

if (chrFSelection == 'c')

{

TakeSymbol();

goto B;

}

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

}

B:

{

if (chrFSelection == 'a')

{

TakeSymbol();

goto CFin;

}

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

}

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 | BC | |

// BC | E | D |

// D | A | |

// E |FFin | |

// FFin | G | |

// G | | G |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto B;

}

else throw new Exception("Ожидался 0");

B:

if (chrFSelection == '0')

{

TakeSymbol();

goto E;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

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

D:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидался 0");

E:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидалось 0");

FFin:

if (chrFSelection == '0')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидалось 0");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 0");

H:

if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 1");

}

case TCharType.ReservedSymbol:

{

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

{

while (enumFSelectionCharType != TCharType.EndRow)

GetSymbol();

}

GetSymbol();

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

}

}

}

}

Результат:

