Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

Тамбовский государственный технический университет

Кафедра САПР

**КУРСОВАЯ РАБОТА**

по дисциплине

«Лингвистическое программное обеспечение систем автоматизированного проектирования»

на тему: “Разработка однопроходного транслятора с исходного языка на язык Фортран”

Выполнил: студент группы

qwinmen

Проверил: преподаватель кафедры САПР

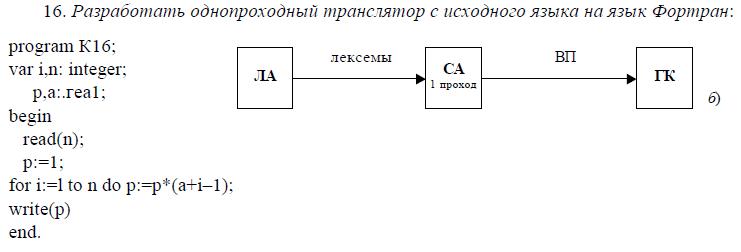
.

Тамбов 20

1. **Пояснительная записка**

## Задание на проектирование

Разработать однопроходный транслятор с исходного языка на язык Фортран. В качестве метода синтаксического анализа использовать метод операторного предшествования.

Задание вариант 16:

Дата принятия задания к исполнению . .201

Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Содержание

[*1* *Пояснительная записка* 2](#_Toc41668856)

[1.1 Задание на проектирование 2](#_Toc41668857)

[1.2 Содержание 3](#_Toc41668858)

[1.3 Введение 4](#_Toc41668859)

[1.4 Описание процесса решения задачи. 5](#_Toc41668860)

[1.5 Блок – схемы основных частей программы. 10](#_Toc41668861)

[*2* *Листинг программных модулей*  12](#_Toc41668862)

[*3* *Описание программы*  39](#_Toc41668863)

[3.1 Назначение и общее описание программы. 39](#_Toc41668864)

[3.2 Описание логической структуры программы. 39](#_Toc41668865)

[3.3 Способ обращения к программе 39](#_Toc41668866)

[3.4 Описание технических средств. 39](#_Toc41668867)

[3.5 Описание входных и выходных файлов. 39](#_Toc41668868)

[*4* *Текстовые примеры работы программы*  40](#_Toc41668869)

[4.1 Правильная программа 40](#_Toc41668870)

[4.2 Программа с ошибками. 40](#_Toc41668871)

[4.3 Программа с добавленными операторами. 42](#_Toc41668872)

## Введение

Транслятор – это программа, обеспечивающая автоматический перевод программ с алгоритмического языка в машинные коды. По функциональному назначению трансляторы делятся на:

• компиляторы (перевод программ на языке высокого уровня в машинные коды без выполнения);

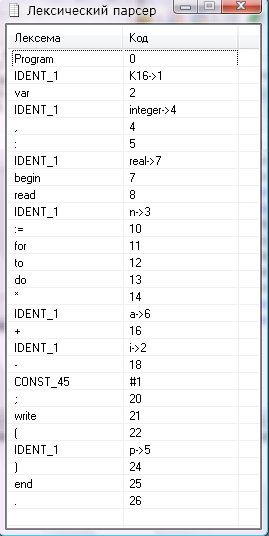
• интерпретаторы (перевод каждой конструкции алгоритмического языка в машинные коды с одновременным выполнением);

• ассемблеры (перевод программы с языка низкого уровня в машинные коды).

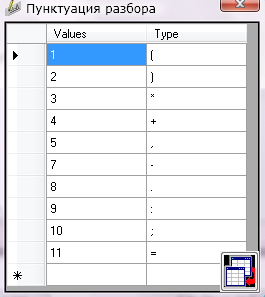
В задании требуется, что бы программа обеспечивала перевод текста программы написанного на структурно стилизованном языке на язык Fortran. А также проверяла исходный текст программы на наличие ошибок.

Курсовая работа содержит текст и описание программы, в которой реализовано задание, описание процесса решения задачи, текстовые примеры работы программы, блок-схемы основных процедур.

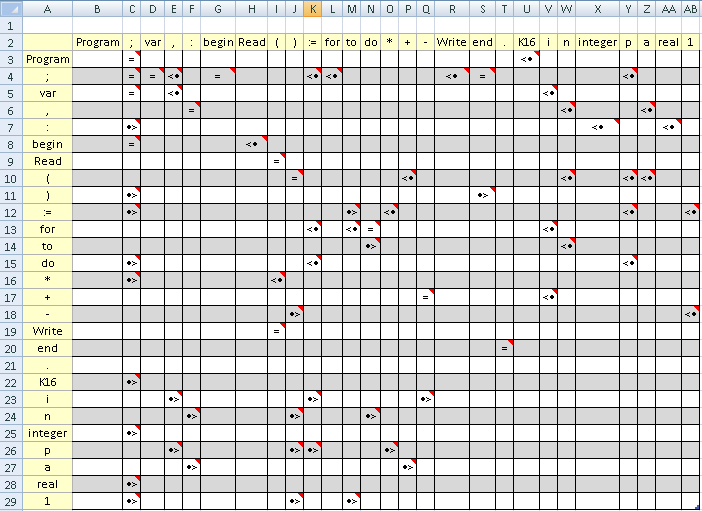
## Описание процесса решения задачи

Для лексического анализатора использовалась следующая таблица кодирования терминальных символов структурного языка:

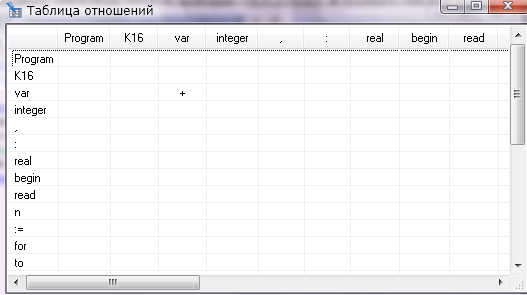
В ходе разработки этого модуля программы был воплощен динамический подход к поиску знаков препинания, которые являются некими «разграничителями» потока слов из исходного файла с кодом на Паскаль. Логика проста, мы можем выполнить анализ в жестких рамках, тоесть условия прописаны в коде программы и немогут быть изменены посторонним лицом (пользователем). Либо, и это даст существенное преимущество – динамический поиск и составление таблицы лексем на основе базы данных, к которой при желании можно подключиться, внести свои условия, удалить ненужное и выполнить сохранение новых критериев поиска.



Хранение записей пунктуаций вынесено в отдельный файл PunctuationDB.mdb

Для синтаксического анализатора была составлена следующая таблица предшествований:

Также как и с лексическим модулем, сдесь исходные-начальные отношения жестко закреплены, при внесении сторонних операторов паскать это гразит некоректной работой модуля разбора, а в последствии и всей программы. Была попытка реализаций динамического выставления приоритетов опираясь на таблицу лексем, однако воплотить задуманное не удалось, т.к. недостаточно знаний в этом направлении, хотя некий начальный каркас всёже есть в программе.



В качестве записи грамматики использовалась форма Бекуса-Наура. Грамматика БНФ состоит из множества правил вывода, каждое из которых определяет синтаксис некоторой конструкции языка программирования. Строки символов, заключённые в угловые скобки < и >, называются нетерминальными символами, то есть являются именами конструкций, определенных внутри грамматики. То, что не заключено в угловые скобки, называется терминальными символами (лексемами).

Грамматика Паскаль исходного фрагмента программы в форме Бекуса-Наура:

<**заг. прогр.**>----<**разд. опис.**>---- <**тело прогр.**>

<**заг. прогр.**>--> Program <имя прогр.>;

<имя прогр.>-->ид

<**разд. опис**.>--> Var <опис. перем.>{;<опис. перем.>};

<опис. перем.>--> <перем.>{,<перем.>}:<тип>

<перем.>-->ид

<тип>-->integer/real

<**тело прогр.**>--> Begin <оператор>{;<оператор>}end.

<оператор>--> <оп. присв.>/<оп. ввода>/<оп. вывода>/<оп. Цикла>

<оп. присв.>--> <перем.> := <значение>

<значение>--> <перем.>/konst/<выраж.>

<выраж.>--> <слаг.>{+<слаг.>}/{-<слаг.>}

<слаг.>--> <множ.>{\*<множ.>}

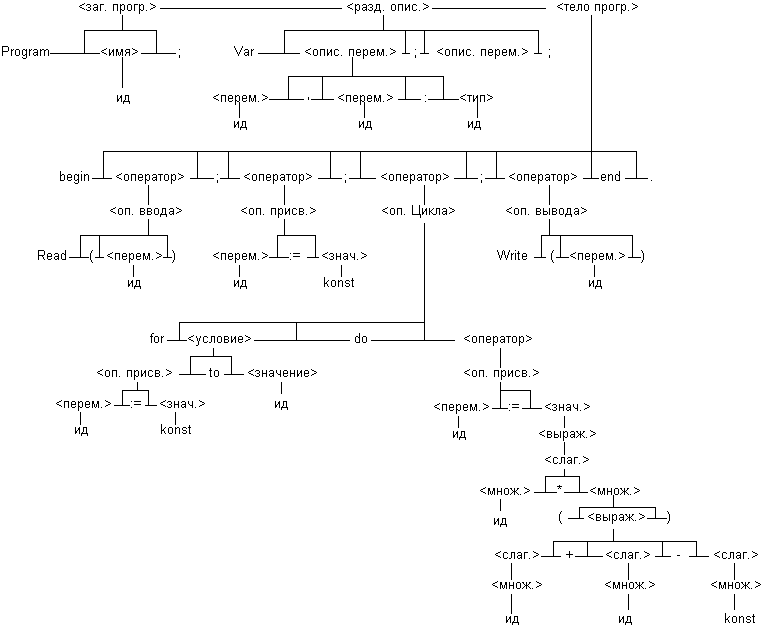
<множ.>--> ид/(<выраж.>)/konst

<оп. ввода>--> Read(<перем.>{,<перем.>})

<оп. вывода>--> Write(<перем.>{,<перем.>})

<оп. Цикла>--> For<условие>do<оператор>

<условие>--> <оп. присв.>to<значение>

Опираясь на грамматику построил синтаксическое дерево:

Работа синтаксического анализатора.

Получая строку, заканчивающуюся точкой запятой, он передает её на анализ лексическому анализатору. Тот в свою очередь разделяет строку на лексемы и заполняет массив по номерами лексем. Вовремя лексического анализа происходит проверка поступающей лексемы на вхождение ее в массив зарезервированных слов входного языка, а также на повторное вхождение. Если лексема ни в одном массиве не найдена и не является числовым значением, то транслятор допускает генерацию кода при условии не критичности допущенной конструкций.

После окончания работы лексического анализатора управление передается обратно синтаксическому анализатору. Тот, получив готовый к обработке массив с номерами лексем, расставляет в соответствии с таблицей предшествований приоритеты в тот же массив, но уже по четным индексам. За тем в цикле foreach происходит выталкивание из массива лексем с наивысшим приоритетом. Это продолжается до тех пор, пока массив не станет пустой.

Генератор кода построен на подобие лексического анализатора получая строку, он распознает ее на лексемы и в соответствии с грамматикой и зарезервированными словами генерирует выходную строку для отображения.

Грамматика Фортран конечной строки:

<**оп. заголовка**>----<**невып. опер.**>---- <**выполн. опер.**>----End

<**оп. заголовка**>--> Program <имя прогр.>

<имя прогр.>-->ид

<**невып. опер**.>--> <опис. типа>

<опис. типа>--> <тип> <перем.>{,<перем.>}

<перем.>-->ид

<тип>-->integer/real

<**выполн. опер.**>--> <оп. присв.>/<оп. ввода>/<оп. вывода>/<оп. Цикла>

<оп. ввода>--> Read(\*,\*)<перем.>{,<перем.>}

<оп. вывода>--> Write(\*,\*)<перем.>{,<перем.>}

<оп. присв.>--> <перем.> = <значение>

<значение>--> <перем.>/konst/<выраж.>

<выраж.>--> <слаг.>{+<слаг.>}/{-<слаг.>}

<слаг.>--> <множ.>{\*<множ.>}

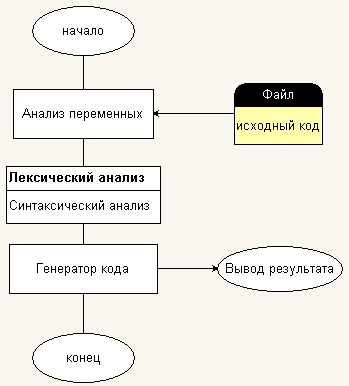
<множ.>--> ид/(<выраж.>)/konst

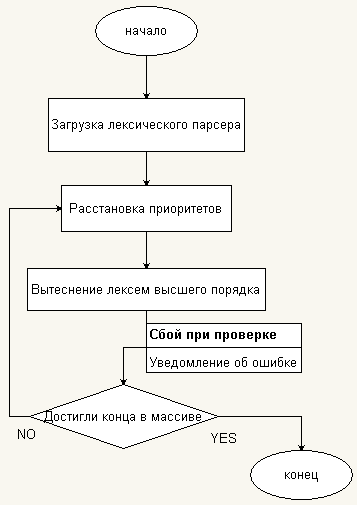
<оп. Цикла>--> do<метка><условие><выполн. опер.><метка>continue

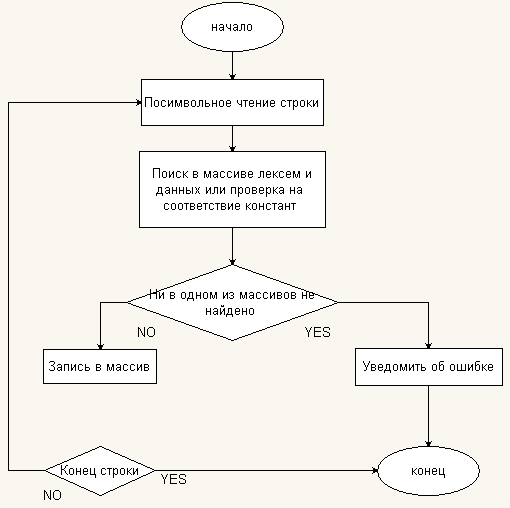
<метка>--> konst

<условие>--> <оп. присв.>,<значение>

## Блок – схемы основных частей программы.

Структура проекта

Структура синтаксического анализатора:

Структура Лексического анализатора:

Генератор кода.

Основан на структуре лексического анализатора, без блока проверки ошибок и вместо занесения в массив для синтаксического анализатора используется запись в выходную строку.

1. **Листинг програмных модулей**

**Модуль Form1.cs**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.IO;

using System.Text.RegularExpressions;

using System.Windows.Forms;

using FastColoredTextBoxNS;

namespace Mary

{

public partial class fMain : Form

{

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//

internal static class StaticData

{//Буфер данных

public static String DataBuffer = String.Empty;

}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//

public fMain(){InitializeComponent();}

/// <summary>

/// Событие по Выходу

/// </summary>

private void exitToolStripMenuItem\_Click(object sender, EventArgs e){ Close(); }

private OpenFileDialog \_openFileDialog;

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

{//Открытие файла входа

textBoxIN.ReadOnly = true;

textBoxIN.Visible = false;

\_openFileDialog = new OpenFileDialog {Filter = "Text files|\*.txt"};

if (\_openFileDialog.ShowDialog() != DialogResult.OK) return;

if(FormTable.TableCountStr!=0)//сначало Таблица, потом Парсер!!!

parserToolStripMenuItem.Visible = false;//т.к. инфа берется с Таблиц лексем

TextLoad();//Грузим тексты с файла внешнего

toolStripStatusLabel.Visible = toolStripStatusLPath.Visible = true;

toolStripStatusLPath.Text = \_openFileDialog.SafeFileName;

}

private StreamReader \_reader;//паток

/// <summary>

/// Коннектимся к файлу

/// </summary>

private void TextLoad()

{//Вывод текста с открытого файла в textBox.In

StaticData.DataBuffer = Path.GetFullPath(\_openFileDialog.FileName);

\_reader = new StreamReader(\_openFileDialog.OpenFile());

try

{

textBoxIN.Visible = true;

textBoxIN.Text = \_reader.ReadToEnd();

viewToolStripMenuItem.Enabled = tableToolStripMenuItem.Enabled = textBoxIN.ReadOnly = true;

/\*----------------------------------\*/

}

catch (OutOfMemoryException ex)

{

MessageBox.Show("Ошибка при чтении файла " + ex);

return;

}

}

/// <summary>

/// Когда грузимся с файла, поток открыт постоянно за счет TextLoad(); ниже в коде события

/// </summary>

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

{//View/Table/

TextLoad();

FormTable formTable = new FormTable();//создаем форму

textBoxOUT.Visible = parserToolStripMenuItem.Visible = true;

textBoxOUT.Text = formTable.SSS();

txtBoxFortranOUT.Text = formTable.SFP();

IndikatorProgram.Visible = IndikatorVar.Visible = IndikatorBegin.Visible = IndikatorEnd.Visible = true;

IndikLabel();

if (formTable.ShowDialog() != DialogResult.OK)//если нажата кн отмена то делаем ретурн

{

if (FormTable.ErrRepitId != null && FormTable.ErrRepitId.Length != 0)

\_errIDtoVar.fctb\_TextChanged(\_ergv, textBoxIN, FormTable.ErrRepitId);

return;

}

}

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

{//View/Parser/Восходящий

fVosxodParser fvosxod = new fVosxodParser();

fvosxod.Show();//Показать таблицу приоритетов

}

#region Размер Формы, Вызов Помощи

private void FormMain\_SizeChanged(object sender, EventArgs e)

{

if (WindowState == FormWindowState.Normal)

bHelp.Visible = false;

else

bHelp.Visible = true;

}

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

{

FHelp fHelp = new FHelp();

if (fHelp.ShowDialog() != DialogResult.OK)//если нажата кн отмена то делаем ретурн

return;

}

#endregion

private readonly Style \_styleGreen = new TextStyle(Brushes.Green, null, FontStyle.Italic);

private readonly Style \_styleBlue = new TextStyle(Brushes.Blue, null, FontStyle.Italic);

private int \_curPos = 0;//храним позицию курсора

private readonly clSyntaxErrIDtoVar \_errIDtoVar = new clSyntaxErrIDtoVar();

/// <summary>

/// когда FormTable закроется, необходимо передать в clSyntaxErrIDtoVar errIDtoVar состояние "е"

/// </summary>

private TextChangedEventArgs \_ergv;

private void textBoxIN\_TextChanged(object sender, TextChangedEventArgs e)

{

e.ChangedRange.ClearFoldingMarkers();

//маркеры на сворачивание блоков begin..end

e.ChangedRange.SetFoldingMarkers(@"begin\b", @"end\b");

KeyValuePair<Regex, Style>[] pairs = new []

{

new KeyValuePair<Regex, Style>(new Regex("(\r\nfor |for\r\n| for )| to | if | do ", RegexOptions.IgnoreCase), \_styleGreen),

new KeyValuePair<Regex, Style>(new Regex("program |(var |\r\nvar )|( begin |\r\nbegin |begin\r\n |\tbegin |\r\nbegin\r\n)|(end |\nend |end.)", RegexOptions.IgnoreCase), \_styleBlue)

};

\_curPos = e.ChangedRange.tb.SelectionStart;

foreach (KeyValuePair<Regex, Style> keyValuePair in pairs)

{

e.ChangedRange.ClearStyle(keyValuePair.Value);

e.ChangedRange.SetStyle(keyValuePair.Value, keyValuePair.Key);

}

\_ergv = e;

if (FormTable.ErrRepitId != null && FormTable.ErrRepitId.Length != 0)

\_errIDtoVar.fctb\_TextChanged(e, textBoxIN, FormTable.ErrRepitId);

e.ChangedRange.tb.SelectionStart = \_curPos;

TxtBoxInText = textBoxIN.Text;

}

/// <summary>

/// храним текст из панели txtBoxIN

/// </summary>

internal static string TxtBoxInText { get; private set; }

/// <summary>

/// это жать когда на textBoxIn чето было стерто или добавленно,

/// тк //View/Table/ работает непосредственно с потоком открытого файла!!

/// </summary>

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

{

FormTable formTable = new FormTable();//создаем форму

textBoxOUT.Visible = parserToolStripMenuItem.Visible = true;

textBoxOUT.Text = formTable.SSS();

txtBoxFortranOUT.Text = formTable.SFP();

IndikLabel();

if (formTable.ShowDialog() != DialogResult.OK)//если нажата кн отмена то делаем ретурн

{

if (FormTable.ErrRepitId != null && FormTable.ErrRepitId.Length != 0)

\_errIDtoVar.fctb\_TextChanged(\_ergv, textBoxIN, FormTable.ErrRepitId);

return;

}

}

/// <summary>

/// Установит связь с БазойДанных для внесения\удаления допЗнаков препинания на обработку

/// </summary>

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

{//окно с пунктуацией - связь с базой данных

fPunct dialog = new fPunct();

if(dialog.ShowDialog()!=DialogResult.OK) return;

}

/// <summary>

/// Эта кнопа даёт зелёный\красный свет на редактирование\нет текстовогоПоляIn

/// </summary>

private void roundButton1\_Click(object sender, EventArgs e)

{

if (textBoxIN.ReadOnly)

{

textBoxIN.ReadOnly = false;

hashToolStripMenuItem.Enabled = true;

roundButton1.BackColor = roundButton1.HoverColor = Color.LightGreen;

}

else

{

textBoxIN.ReadOnly = true;

hashToolStripMenuItem.Enabled = false;

roundButton1.BackColor = roundButton1.HoverColor = Color.Red;

}

textBoxIN.Focus();

}

/// <summary>

/// Подсветка наличия\отсутствия каркаса программы

/// </summary>

private void IndikLabel()

{

foreach (KeyValuePair<bool, int> keyValuePair in FormTable.Indikation)

{

if (keyValuePair.Value == 0 && keyValuePair.Key) IndikatorProgram.BackColor = Color.Green;

else if (keyValuePair.Value == 0 && !keyValuePair.Key) IndikatorProgram.BackColor = Color.Red;

if (keyValuePair.Value == 1 && keyValuePair.Key) IndikatorVar.BackColor = Color.Green;

else if (keyValuePair.Value == 1 && !keyValuePair.Key) IndikatorVar.BackColor = Color.Red;

if (keyValuePair.Value == 2 && keyValuePair.Key) IndikatorBegin.BackColor = Color.Green;

else if (keyValuePair.Value == 2 && !keyValuePair.Key) IndikatorBegin.BackColor = Color.Red;

if (keyValuePair.Value == 3 && keyValuePair.Key) IndikatorEnd.BackColor = Color.Green;

else if (keyValuePair.Value == 3 && !keyValuePair.Key) IndikatorEnd.BackColor = Color.Red;

}

if (FormTable.ZnakProgramToVar.Length != 0)

foreach (KeyValuePair<int, int> keyValuePair in FormTable.ZnakProgramToVar)

{

if (keyValuePair.Key == 0 && keyValuePair.Value == 1)

{//FormTable--SearchProgramToVar(idIndex[1], index, out pv);

IndikatorProgram.BorderSides = ToolStripStatusLabelBorderSides.Right;

IndikatorVar.BorderSides = ToolStripStatusLabelBorderSides.Left;

}

if (keyValuePair.Key == 1 && keyValuePair.Value == 2)

{//FormTable--SearchVarToBegin(idIndex[2], index, out pv);

IndikatorVar.BorderSides = ToolStripStatusLabelBorderSides.Right;

IndikatorBegin.BorderSides = ToolStripStatusLabelBorderSides.Left;

}

if (keyValuePair.Key == 2 && keyValuePair.Value == 3)

{

IndikatorBegin.BorderSides = ToolStripStatusLabelBorderSides.Right;

IndikatorEnd.BorderSides = ToolStripStatusLabelBorderSides.Left;

}

}

}

}

}

**Модуль FormTable.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Windows.Forms;

namespace Mary

{//FormTable.cs реализует лексический анализ

public partial class FormTable : Form

{

public FormTable()

{

InitializeComponent();

Leksema();

}

private string \_mmindex;

/// <summary>

/// Анализ в один проход

/// </summary>

private void Leksema()

{

for (int increment = 0; increment < 1; increment++)//Общий счетчик выражений

Kod(increment);

}

private static string[] \_words;

public void Kod(int index)

{

char[] razdelenie = { ' ', '\n', '\r', '\t' };

using (new StreamReader(fMain.StaticData.DataBuffer))

{//Безопасное открытие патока

int i = 0, j = 0;

string s = fMain.TxtBoxInText;

\_words = Filter(s).Split(razdelenie);//убираем пробелы\табы\&etc

/\*---parser----\*/ ParsFortran(\_words);

string[] Massiv = new string[\_words.Length], Massiv\_Index = new string[\_words.Length];

foreach (string ts in \_words)

{//нулим все повторения

Massiv[j] = ts;

j++;

for (int e = 0; e < i; e++)

if (ts.CompareTo(Massiv[e]) == 0)

{//Если есть результ, то есть совпадение с текущим значением ts!

Massiv\_Index[e] = ts;

Massiv[e] = null; //Занулим все совпадения

}

i++;

}

ЛексемыБезПовторов = Massiv;

int ks = 0;

foreach (string ss in Massiv)

{

if (ss != null) //пишeм в Таблицу и в txtOUT

{//Шарим в ЗАНУЛЕНОМ МАСИВЕ!

//----------------------------------------------------------------//

//----------------------------------------------------------------//

ks++;

IdConstVisual(ss, index);

index++;

//----------------------------------------------------------------//

//----------------------------------------------------------------//

}

else //Иначе надо null заменить значением

{//ss = null, пишем только в txtOUT

string vvv = Massiv\_Index[ks]; //";" || "Y\r"

int x = 0;

foreach (string vv in Massiv)

{

if (vv == vvv && vv != "") //vv != "" для отсева пустышек

{//Найден символ [2];<<=;[61]

IdConstVisual(vv, index, 1, x, Massiv);

}

x++; //контроль индекса у vv

}

ks++; //Поднятие индекса

}

} //End Foreach

}//END Using

}//END KOD()

#region Filter знаки препинания

private delegate string StrMod(string s);

/// <summary>

/// На вход строка с файла\\ на выход строка с отделёными зн. препинания

/// </summary>

private string Filter(string strIn)

{

StrMod aa = res =>

{

string temp = "";//для новой строки из старой

int index = 0;//для посмотреть символ за ':'

bool flag = false;//если true то 1 проход foreach надо пропустить

foreach (char ch in res)

{

if (fPunct.PunctArr!=null)

{//если в базе данных есть записи то:

if (!flag && index < res.Length)//чтоб не вылезти за [границы]

temp += Compare(ch, res, ref index, ref flag);

else flag = false;//сбросим flag после ':='

}

else

#region если база неподключена, то используем умолчания:

{

if (!flag && index < res.Length)//чтоб не вылезти за [границы]

switch (ch)

{//текущая пунктуация//

case '(': temp += " ( ";

break;

case ')': temp += " ) ";

break;

case '\*': temp += " \* ";

break;

case '+': temp += " + ";

break;

case ',': temp += " , ";

break;

case '-': temp += " - ";

break;

case '.': temp += " . ";

break;

case ':':

if (res[index + 1] == '=')//смотрим следущий за ':' символ

{//если за ':' идет '='

temp += " := ";

flag = true;//т.е. надо 1 проход foreach пропустить, т.к. там '='

}

else temp += " : ";//если за ':' идет лаб:уда

break;

case ';': temp += " ; ";

break;

case '=': temp += " = ";

break;

default: temp += ch;

break;

}

else flag = false;//сбросим flag после ':='

}

#endregion

index++;//храним текущую позицию ch в res

}

return temp;

};

StrMod strOp = aa;

return strOp(strIn);

}

/// <summary>

/// Используем знаки с базы данных fPunct.PunctArr вместо явной ветки с пунктуацией

/// </summary>

private string Compare(char ch, string res, ref int index, ref bool flag)

{

string temp = "";

foreach (string s in fPunct.PunctArr)//заглушка для break

{

if (frch(ch))

{

if (ch == ':')

{//если это двоеточие, то

if (res[index + 1] == '=')//смотрим следущий за ':' символ

{//если за ':' идет '='

temp += " := ";//значит это сцепка ":="

flag = true; //т.е. надо 1 проход foreach пропустить, т.к. там '='(void Filter)

}

else temp += " " + ch + " ";//иначе это просто одиночное двоеточие

break;

}

temp += " " + ch + " ";//если это другие знаки разделители, которые есть в базе:

break;

}

temp += ch;//иначе тут только лексемы без разделителей:

break;

}

return temp;

}

/// <summary>

/// если ch есть в БазеДанных, то вернет true

/// </summary>

private bool frch(char ch)

{

return fPunct.PunctArr.Any(s1 => s1 == ch.ToString());

}

#endregion

#region IdConstVisual визуально ID->1, Const#1 и т.п.

/// <summary>

/// Выводим в FormTable и txtOUT визуально ID->1, Const#1 и т.п.

/// </summary>

private void IdConstVisual(string ss, int index, params object[] flag)

{//если flag[]=true, то изменить только \_mmindex

//\*\*\*\*\*ЯВНОЕ\*ОБЬЯВЛЕНИЕ\*ЧТО\*СЧИТАТЬ\*ID\*CONST\*\*\*\*\*

const string K16\_atribute = "1",

i\_atribute = "2",

n\_atribute = "3",

integer\_atribute = "4",

p\_atribute = "5",

a\_atribute = "6",

real\_atribute = "7";

const int ind\_CONST = 45; //CONST\_0 #2.5

const int ind\_IDENT = 1; //ID X->1

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

switch (ss)

{

case "1"://ловим Konstant

if (flag.Length == 0)//flag=false

{

index = listViewKod\_Table.Items.Add("CONST\_" + ind\_CONST).Index; //\_0

listViewKod\_Table.Items[index].SubItems.Add("#" + ss); //#1

}

\_mmindex += (" " + ind\_CONST + "#" + ss + " "); // SSS()+="#"+ts;

break;

case "K16"://ловим идентификатор

if (flag.Length == 0)//flag=false

{

index = listViewKod\_Table.Items.Add("IDENT\_" + ind\_IDENT).Index; //\_1

listViewKod\_Table.Items[index].SubItems.Add(ss + "->" + K16\_atribute); //

}

\_mmindex += (" " + ind\_IDENT + "->" + K16\_atribute + " ");

break;

case "i":

if (flag.Length == 0)//flag=false

{

index = listViewKod\_Table.Items.Add("IDENT\_" + ind\_IDENT).Index; //\_1

listViewKod\_Table.Items[index].SubItems.Add(ss + "->" + i\_atribute); //

}

\_mmindex += (" " + ind\_IDENT + "->" + i\_atribute + " ");

break;

case "n":

if (flag.Length == 0)//flag=false

{

index = listViewKod\_Table.Items.Add("IDENT\_" + ind\_IDENT).Index; //\_1

listViewKod\_Table.Items[index].SubItems.Add(ss + "->" + n\_atribute); //

}

\_mmindex += (" " + ind\_IDENT + "->" + n\_atribute + " ");

break;

case "integer":

if (flag.Length == 0)//flag=false

{

index = listViewKod\_Table.Items.Add("IDENT\_" + ind\_IDENT).Index; //\_1

listViewKod\_Table.Items[index].SubItems.Add(ss + "->" + integer\_atribute); //

}

\_mmindex += (" " + ind\_IDENT + "->" + integer\_atribute + " ");

break;

case "p":

if (flag.Length == 0)//flag=false

{

index = listViewKod\_Table.Items.Add("IDENT\_" + ind\_IDENT).Index; //\_1

listViewKod\_Table.Items[index].SubItems.Add(ss + "->" + p\_atribute); //

}

\_mmindex += (" " + ind\_IDENT + "->" + p\_atribute + " ");

break;

case "a":

if (flag.Length == 0)//flag=false

{

index = listViewKod\_Table.Items.Add("IDENT\_" + ind\_IDENT).Index; //\_1

listViewKod\_Table.Items[index].SubItems.Add(ss + "->" + a\_atribute); //

}

\_mmindex += (" " + ind\_IDENT + "->" + a\_atribute + " ");

break;

case "real":

if (flag.Length == 0)//flag=false

{

index = listViewKod\_Table.Items.Add("IDENT\_" + ind\_IDENT).Index; //\_1

listViewKod\_Table.Items[index].SubItems.Add(ss + "->" + real\_atribute); //

}

\_mmindex += (" " + ind\_IDENT + "->" + real\_atribute + " ");

break;

default://лексемы-знаки и т.п.:

if (ss != "")

{

if (flag.Length == 0) //массив flag пуст

{

listViewKod\_Table.Items.Add(ss); //Колонка Лексемы

listViewKod\_Table.Items[listViewKod\_Table.Items.Count - 1].SubItems.Add(index.ToString());//Колонка Код

\_mmindex += (" $" + index + "$ "); //\_mmindex += (" " + index + " ");//

}

else

{//Какой x\_индекс у vvv в Massiv

int dex = 0;

//Сколько null от начала до vvv

for (int k = 0; k < (int) flag[1]; k++) //flag[1]=x

{

if (((string[]) flag[2])[k] == null) //Massiv[k]==null

dex++; //+1

}

//Вычтем подсчитаное количество нулов (x-dex)

\_mmindex += " " + ((int) flag[1] - dex) + " ";

}

}

break;

}//end Switch

}

#endregion

/// <summary>

/// Перехват значений и отправка с FormTable-->fMain

/// </summary>

internal string SSS()

{

fMain formMain = new fMain();

StaticDat.DataBuffer = \_mmindex;

TableCountStr = listViewKod\_Table.Items.Count;//сколько строк столько и лексем

return formMain.textBoxOUT.Text = \_mmindex;

}

private string \_rtf = "";

internal void ParsFortran(string[] index)

{

string[] fortranArr = {"Program", "Read", "=", "Write", "end", "stop", ",", "do", "continue"};

int i = 0, iBgn = 0;//s[i]//[iBgn]="begin"

int[] idIndex = new int[4];//для индексов каркаса

Indikation = new[]

{

new KeyValuePair<bool, int>(ПроверкаКаркасаПрограммы("Program", index, out idIndex[0]), 0),

new KeyValuePair<bool, int>(ПроверкаКаркасаПрограммы("Var", index, out idIndex[1]), 1),

new KeyValuePair<bool, int>(ПроверкаКаркасаПрограммы("Begin", index, out idIndex[2]), 2),

new KeyValuePair<bool, int>(ПроверкаКаркасаПрограммы("End", index, out idIndex[3]), 3)

};

ZnakProgramToVar = new KeyValuePair<int, int>[Indikation.Length];//4

if (Indikation[1].Key)//есть Var

{//от Program до Var

int pv;//idIndex[1]=Var

SearchProgramToVar(idIndex[1], index, out pv);//поиск ;

if (pv == 01) ZnakProgramToVar[0] = new KeyValuePair<int, int>(0, 1);

}if(Indikation[2].Key)//есть Begin

{//от Var до Begin

int pv;

if (idIndex[1] != index.Length) SearchVarToBegin(idIndex[2], index, out pv, idIndex[1]);//idIndex[2]=Begin

else SearchVarToBegin(idIndex[2], index, out pv);//иначе нету Var и поиск от начала до Begin

if (pv == 12) ZnakProgramToVar[1] = new KeyValuePair<int, int>(1, 2);

}if(Indikation[2].Key)//есть Begin

{//от Begin до End ИЛИ от Begin до конца

int pv;

if (idIndex[3] != index.Length) SearchBeginToEnd(idIndex[2], index, out pv, idIndex[3]);//idIndex[3]=end

else SearchBeginToEnd(idIndex[2], index, out pv);//иначе нету у нас end, ищем до конца

if (pv == 23) ZnakProgramToVar[2] = new KeyValuePair<int, int>(2, 3);

}

foreach (string s in index)

{

if (";" == s)\_rtf += Environment.NewLine;

if (AravnoB("var",s)) \_rtf += ОписаниеПеременных(i, ref index, out iBgn);

foreach (string s1 in fortranArr)

{

if (AravnoB("for", s) & AravnoB("do", s1))//for i:=1 to n do <ТелоЦикла>;

{

\_rtf += s1 + " " + i + " ";//do\_metka\_

if (index[i + 2] == ":=")

{//i\_=\_1\_

\_rtf += index[i + 1] + " = " + index[i + 3] + " ";

index[i + 2] = index[i + 3] = "";//:= 1

if (AravnoB(index[i + 4],"to"))//if (index[i + 4] == "to")

{

\_rtf += ", " + index[i + 5] + " ";//,\_n\_

index[i + 5] = "";//n

\_rtf += ТелоЦикла(i + 7, ref index);

}

}

\_rtf += Environment.NewLine + i + " continue";//\_metka\_continue

}

else if (!AravnoB("do", s))

{

if (/\*iBgn != 0 &&\*/ s == ":=" && s1 == "=")//begin||;...:=...;

{

\_rtf += ОператорПрисваивания(i, iBgn, index);

}

if (AravnoB(s1, s))

{

bool flag = false;

for (int j = i; j < iBgn; j++)

{//var[i]...begin[iBgn]

if (s == ",")

{

\_rtf += "";//спрятать запятые в блоке обьявления переменых

flag = true;//без затирания в основном массиве index

}

}

if (AravnoB("end", s))

\_rtf += Environment.NewLine + s;

else if (AravnoB("write", s1) | AravnoB("read", s1))

{//ищем от write до ")"

if (FindAtoB(i + 1, ")", ref index) == "<Ошибка>")

\_rtf += s1 + "(\*,\* " + FindAtoB(i + 1, ")", ref index);

else

{

\_rtf += s1 + "(\*,\*) " + FindAtoB(i + 1, ")", ref index);

if (!SearchSkbToTzpt(i, index, ")")) ZnakProgramToVar[2] = new KeyValuePair<int, int>(2, 3);//MessageBox.Show("Нету ;");//искать от ) знак ';'

}//MessageBox.Show(FindAtoB(i, ".", ref index));

}

else if (!flag)

if (AravnoB("Program", s)) \_rtf += s + " " + FindAtoB(i, ";", ref index) + " ";//if (s == "K16") rtf += s + " ";

//rtf += s1 + " ";

}

}//end ElseIf

}//end foreach (string s1 in fortranArr)

i++;

}//end foreach (string s in index)

}

/// <summary>

/// поиск от iА до "B" в массиве index

/// </summary>

private string FindAtoB(int iA, string b, ref string[] index)

{//a="Program", b=";", regExp="<ID>"

int iB = 0,ut=0; string str = "";

for (int i = iA; i < index.Length; i++)//кто первый - того и тапки:

{//нашли ; стоп. Ненашли ; - нашли Var стоп

if (index[i] == ";" && b != ";") ut--;//ищем b до ; Если его нет, то понижаем индекс

if (index[i] == b || AravnoB("var", index[i]))

{

ut++;//ищем b до ; Если он есть, то повышаем индекс

break;

}

iB++;

}

if (iB == 0) MessageBox.Show("Знака " + b + " ненайдено!","\*Косяк в тексте");

if (ut <= 0) MessageBox.Show("Знака " + b + " ненайдено!", "\*\*Косяк в тексте");

for (int i = iA + 1; i < (iB+iA); i++)//[iA]=Program

{//от [iA]..[iB] чето найти

if (index[i] != "") str += index[i];

}

if (str == "" || ut <=0) str = "<Ошибка>";

return str;//K16

}

/// <summary>

/// Блок от Var до Begin

/// </summary>

private string ОписаниеПеременных(int iVar, ref string[] index, out int iBgn)

{//после var[5]

string stroka = "", strForRepitId = "";

int iBegin = 0, iTemp = iVar;

//от var до первого begin раздел описания переменных

iBegin = index.TakeWhile(s => !AravnoB("begin", s)).Count();

iBgn = iBegin;

for (int i = iVar; i < iBegin; i++)

{

if (index[i] == ";")

{

stroka += Environment.NewLine;

for (int j = iTemp; j < i; j++)

{

if (index[j] == ":")

{

//переставить местами А[iTemp..j]:B[j..i];

for (int k = j + 1; k < i; k++)//+1 = непишем :

{//<тип>

if (index[k] != "") stroka += index[k] + " ";

}

for (int l = iTemp + 1; l < j; l++)//+1 = непишем var||;

{//<ID,>

//MessageBox.Show("iTemp="+iTemp+" j="+j+" i="+i);

stroka += index[l] + " ";

//если возникло совпадение id=id, то надо подсветить:

//с индекса на длинну ID

if (index[l] != "" & index[l] != ",")//если там чето есть кроме зпт

strForRepitId += index[l] + ":" + l + "$" + index[l].Length + "%";// "id:index$length%"

}

}

}

iTemp = i;

}

}

Dictionary<int, int> dictionary = IndexLengthRepitId(strForRepitId);//MessageBox.Show(strForRepitId,"Надо подсветить:");

if (dictionary.Count != 0)

{

ErrRepitId = new KeyValuePair<int, int>[dictionary.Count/\*скоко ошибок\*/];

int indexator = 0;

foreach (KeyValuePair<int, int> keyValuePair in dictionary)

{

if (indexator < ErrRepitId.Length)

{

//Это история о том, сколько знаков от начала программы до [keyValuePair.Key]

string ads = "";

for (int i = 0; i < keyValuePair.Key; i++)

{

if (index[i] == "") ads += " ";

else ads += index[i];

}//Конец истории, знаки запакованны в строку.Length

ErrRepitId[indexator] = new KeyValuePair<int, int>(ads.Length + 2, keyValuePair.Value);

}

indexator++;

}

}else ErrRepitId = new[] {new KeyValuePair<int, int>(0, 0)};

return stroka;

}

/// <summary>

/// p:=p\*(a+i-1);

/// </summary>

private string ТелоЦикла(int i, ref string[] index)

{

string telo = "";

for (int j = i; j < index.Length; j++)

{

if (index[j]==";")

{

for (int k = i; k < j; k++)

{

if (index[k] == ":=")

{

index[k] = "";

telo += "=";

}

else

{

telo += index[k];

}

}

break;

}

}

return telo;

}

/// <summary>

/// Ищет выражения от begin или ; ДО ; или end

/// ПРИМЕР: begin a:=b; || Read(); a:=b end || Write(); a:=b;

/// </summary>

private string ОператорПрисваивания(int i, int iBegin, string[] index)

{//любое присваивание в begin...end

//знаем := //Находим ; или begin//Находим ; или (end-не реализовано!)

string str = ""; int temp = iBegin, endTemp = -1;

for (int j = iBegin; j < i; j++)//от begin до :=

if (index[j] == ";") temp = j;//ищем ;

//если не нашли, то в temp пишем iBegin

for (int j = i; j < index.Length; j++)//от := до ;||end

if(index[j]==";" | "end".Equals(index[j],StringComparison.OrdinalIgnoreCase))

{ endTemp = j; break; }

if (temp != -1 && endTemp != -1)//т.е. знак ; найден

{

for (int j = temp + 1; j < endTemp; j++)//begin a:=b; || ; a:=b;

{

if (index[j] == ":=") str += "=";

else if (index[j] != "") str += index[j];

}

}else if(temp == -1)MessageBox.Show("Нет begin ИЛИ ; до знака :=");

else if (endTemp == -1) MessageBox.Show("Нет ; ИЛИ end после знака :=");

return str;

}

#region Проверим наличие Program-;-Var-;-Begin-;-End

/// <summary>

/// Каркас проверки Program..;Var..;Begin..end

/// </summary>

private bool ПроверкаКаркасаПрограммы(string s, IEnumerable<string> index, out int idIndex)

{// s[i] in index

bool flag = false;

int sIndex = 0;

foreach (string s1 in index)

{

if (AravnoB(s1, s))

{

flag = true;

break;

}

sIndex++;

}

idIndex = sIndex;//if (flag) MessageBox.Show("Нет " + s + " "+sIndex,"Косяк каркаса");

return flag;

}

/// <summary>

/// От Program до Var долж быть только один ";" проверим

/// </summary>

private void SearchProgramToVar(int iVar, string[] res, out int pv)

{// res[index]="var"

int count = 0, iTo = 0, uInd=0;

for (int i = 0; i < iVar; i++)

{

if (res[i] == ";")

{

uInd = i;

count++;

}

} if (count != 1 && Indikation[0].Key) pv = 01;//MessageBox.Show("Нет ; между Program и Var", "Косяк пунктуации");

else pv = -1;

for (int i = iVar-1; i > uInd; i--)//реверс

{//от Var до ; если она есть

if (res[i] != "" && res[i] != ";") iTo++;//=0 то Ок, иначе Косяк

} if (iTo != 0) pv = 01;//MessageBox.Show(iTo.ToString());

}

/// <summary>

/// От Var до Begin сколько ':' столько ';'

/// Если нет Var, то от Begin до 0-начала программы

/// </summary>

private void SearchVarToBegin(int iBegin, string[] res, out int pv, params int[] iVar)

{//idIndex[2], index, out pv

int iV = 0;//если iVar пуст(те нету ID), то ';' на один больше чем ':'

List<int> ttIndex=new List<int>();//лист тк непойми сколько этих ':'

List<int> zptIndex = new List<int>();//лист тк непойми сколько этих ';'

if (iVar.Length != 0) iV = iVar[0];//MessageBox.Show(iV.ToString());

for (int i = iBegin; i > iV; i--)//реверс от begin до var

{

if (res[i] == ":") ttIndex.Add(i);

if(res[i]==";")zptIndex.Add(i);

} if (ttIndex.Count > zptIndex.Count && iV!=0) pv = 12;//MessageBox.Show("Нехватает ;", "Var Begin", MessageBoxButtons.OK, MessageBoxIcon.Error);

else if (ttIndex.Count < zptIndex.Count && iV!=0) pv = 12;//MessageBox.Show("Нехватает :", "Var Begin", MessageBoxButtons.OK, MessageBoxIcon.Error);

else pv = 00;

if (ttIndex.Count == zptIndex.Count && ttIndex.Count != 0 && zptIndex.Count != 0)

for (int q = 0; q < ttIndex.Count; q++)

{//от ':' до ';' ровно один ID-типа, если это не так - то ошибка

int countId = 0;

for (int i = ttIndex[q] + 1; i < zptIndex[q]; i++)

if (res[i] != "") countId++;

if (countId != 1) pv = 12;//MessageBox.Show("Между : и ; перебор/нехватка ID-типов", "Var Begin", MessageBoxButtons.OK,MessageBoxIcon.Error);

}

pv = pv != 12 ? 00 : 12;//при косяке = 12

}

/// <summary>

/// От Begin до End

/// Если нет End, то от Begin до конца

/// </summary>

private void SearchBeginToEnd(int iBegin, string[] res, out int pv, params int[] iEnd)

{

int iE = res.Length;//самый конец

if (iEnd.Length != 0) iE = iEnd[0];//MessageBox.Show(iE.ToString());

for (int i = iBegin; i < iE; i++)

{//от Begin до End|res.Length

//надо обработать а=и выражения //и фот ту ду

}

pv = 00;

}

/// <summary>

/// Write\Read При поиске от ) до ; недолжно быть ID иначе косяк ИЛИ кроме ID=End

/// </summary>

private bool SearchSkbToTzpt(int i, string[] index, string str)

{//index[i]='Id' надо найти ')', и потом ';'

bool state = false;

int iSkb = 0;//индекс )

for (int j = i; j < index.Length; j++)

if (AravnoB(index[j],str))

{

iSkb = j;

break;

}

int iId = 0;

for (int j = iSkb+1; j < index.Length; j++)//от ) до первой ; ИЛИ до конца

{

if (index[j] == ";")

{

state = true;

break;

}

if (index[j] != "" && !AravnoB(index[j],"end")) iId++;//т.е. есть между ) ; какието знаки

}

if (iId != 0 && state) state=false;//MessageBox.Show("Между ) и ; есть лишние ID");

else if (iId == 1 && !state) state=true;//MessageBox.Show("Между ) и Концом есть ID=end");

else if(iId==0 && state) state = true;

return state;//false при 'ненайден знак'

}

#endregion

/// <summary>

/// if(a==b)return true. Без учета регистра!

/// </summary>

private bool AravnoB(string a, string b){return a.Equals(b, StringComparison.InvariantCultureIgnoreCase);}

/// <summary>

/// На вход все подряд ID, на выходе те ID, которые нужно подкрасить красным

/// index[l] + ":" + l + "$" + index[l].Length + "%";

/// </summary>

private Dictionary<int,int> IndexLengthRepitId(string strFull)

{

string str = ""; Dictionary<int,int> keyValue=new Dictionary<int, int>();

string[] arr = strFull.Split('%'), massiv = new string[arr.Length];

int i = 0, j = 0;

foreach (string ts in arr)

{

if(ts!="")

{

massiv[j] = ts;

j++;

string tmp = ts.Substring(0, ts.IndexOf(':'));//id без хвоста

for (int e = 0; e < i; e++)

if (massiv[e] != null && AravnoB(tmp, massiv[e].Substring(0, massiv[e].IndexOf(':'))) )

{//Если есть результ, то есть совпадение с текущим значением ts!

string key = ts.Substring(ts.IndexOf(':') + 1, ts.IndexOf('$') - ts.IndexOf(':') - 1 );

string value = ts.Substring(ts.IndexOf('$') + 1);

keyValue.Add(Convert.ToInt32(key), Convert.ToInt32(value));

str += ts+"%";//необязательно

massiv[e] = null;//Занулим все совпадения

}

i++;

}

}//MessageBox.Show(str, "Это:надо$подсветить%");

return keyValue;//это надо выделить

}

/// <summary>

/// вывод в текстовую область Фортран

/// </summary>

internal string SFP()

{

fMain formMain = new fMain();

return formMain.txtBoxFortranOUT.Text = \_rtf.ToUpper();//fortranArr[i].ToUpper();

}

/// <summary>

/// храним количество строк в таблице

/// </summary>

internal static int TableCountStr { get; private set; }

/// <summary>

/// храним лексемы из входного файла

/// </summary>

internal static string[] ЛексемыБезПовторов { get; private set; }

/// <summary>

/// ошибочный Id. Key=от, Value=id.Length выделить

/// </summary>

internal static KeyValuePair<int,int>[] ErrRepitId { get; private set; }

/// <summary>

/// Неполадки в каркасе bool=Red\Green int=0\_program..3\_End

/// </summary>

internal static KeyValuePair<bool, int>[] Indikation { get; private set; }

/// <summary>

/// 0\_1 отобразить косяк отсутствия ;

/// </summary>

internal static KeyValuePair<int, int>[] ZnakProgramToVar { get; private set; }

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//

internal static class StaticDat

{//ПЕРЕДАЧА КОДОВ ЛЕКСЕМ ТЕКСТА В PARSER.cs

//Буфер данных

internal static String DataBuffer = String.Empty;

}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//

internal static class BuffWords

{//ПЕРЕДАЧА ТЕКСТА В fParserLexical\_OperatPredshest.cs

//Буфер данных

internal static String DataBuffer = String.Empty;

}

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

{

try

{

columnHeader1.Width = columnHeader2.Width = 115;

ClientSize = new System.Drawing.Size(columnHeader1.Width + 140, \_mmindex.Length + 200);

}

catch (NullReferenceException)

{

columnHeader1.Width = columnHeader2.Width = 115;

}

}

}

}

**Модуль clSyntaxErrIDtoVar.cs**

using System.Collections.Generic;

using System.Drawing;

using FastColoredTextBoxNS;

namespace Mary

{

/// <summary>

/// надо обработать повторяющиеся ID в блоке Var, подкрасить ошибки блока

/// </summary>

class clSyntaxErrIDtoVar

{

//create my custom style

readonly EllipseStyle \_ellipseStyle = new EllipseStyle();

/// <summary>

/// В блоке Var выделить все повторяющиеся идентификаторы

/// </summary>

internal void fctb\_TextChanged(TextChangedEventArgs e, FastColoredTextBox textBoxIn, KeyValuePair<int,int>[] err)

{

//clear old styles of chars

e.ChangedRange.ClearStyle(\_ellipseStyle);

if(err!=null)

foreach (KeyValuePair<int, int> keyValuePair in err)

{

e.ChangedRange.tb.SelectionStart = keyValuePair.Key;//откуда

e.ChangedRange.tb.SelectionLength = keyValuePair.Value;//и на сколько

textBoxIn.Selection.SetStyle(\_ellipseStyle);

}

}

}

/// <summary>

/// Этот стиль отрисует рамку-элипсис вокруг косячного обьявления

/// </summary>

class EllipseStyle : Style

{

/// <summary>

/// типо конструктор position(x,y) range(width,height)

/// </summary>

public override void Draw(Graphics gr, Point position, Range range)

{

//get size of rectangle

Size size = GetSizeOfRange(range);

//create rectangle

Rectangle rect = new Rectangle(position, size);

//inflate it

rect.Inflate(2, 2);

//get rounded rectangle

var path = GetRoundedRectangle(rect, 7);

//draw rounded rectangle

gr.DrawPath(Pens.Red, path);

}

}

}

**Модуль fPunct.cs**

using System.Data;

using System.Data.OleDb;

using System.Windows.Forms;

//работа с бд пунктуации

namespace Mary

{

/// <summary>

/// Здесь описаны все знаки препинания, которые являются разделителями между слов

/// Идея проста - на форме есть сетка, в неё грузим пунктуацию из файла-базы.

/// Также можно чтото удалить, чтото добавить. Все хранение в файл базы данных.

/// </summary>

public partial class fPunct : Form

{

public fPunct()

{

InitializeComponent();

ReadData();

}

private BindingSource \_bindingSource = new BindingSource();

private OleDbConnection connection;

private DataSet dataSet = new DataSet();

private void ReadData()

{

//соединяемся с сервером

if(CreateConnection()!=null)

{

connection = CreateConnection();

//подготовим команду

OleDbCommand command = new OleDbCommand("SELECT \* FROM Punctuation");

command.Connection = connection;

//создаем адаптер и набор данных

OleDbDataAdapter adapter = new OleDbDataAdapter(command);

//заполняем набор данных

adapter.Fill(dataSet);

//закрываем больше ненужное соединение

connection.Close();

//связываем набор данных с сеткой через посредника bindingSource

dataGridView1.AutoGenerateColumns = true;

\_bindingSource.DataSource = dataSet.Tables[0];

dataGridView1.DataSource = \_bindingSource;

}

}

OleDbConnection CreateConnection()

{//C:\Users\user\Desktop\Учеба\2 курс Февр-Июль\ЛПО-КУРСАЧ\Mary\bin\Debug\

OleDbConnection connection = new OleDbConnection();

connection.ConnectionString =

@"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=PunctuationDB.mdb;Persist Security Info=False";

try

{

connection.Open();

}

catch

{

MessageBox.Show("Ошибка соединения с базой данных");

connection.Close();

}

return connection;

}

private void btnSave\_Click(object sender, System.EventArgs e)

{

OleDbDataAdapter adapter = new OleDbDataAdapter("SELECT \* FROM Punctuation", connection);

//создаем обьект команды

adapter.UpdateCommand =

new OleDbCommand("UPDATE Punctuation SET Type = ?"+"WHERE Values = ?");

//создать параметры связи данных

adapter.UpdateCommand.Parameters.Add("Type", OleDbType.VarChar, 50, "Type");

adapter.UpdateCommand.Parameters.Add("Values", OleDbType.Integer, 50, "Values");

//++++++++

//Реализация добавления записи из таблицы

adapter.InsertCommand =

new OleDbCommand("INSERT INTO Punctuation (Type)" + "VALUES (?)");

adapter.InsertCommand.Parameters.Add("Type", OleDbType.VarChar, 50, "Type");

adapter.InsertCommand.Connection = connection;

//Реализация команды удаления поля из таблицы

adapter.DeleteCommand = new OleDbCommand("DELETE FROM Punctuation WHERE Type=?");

//adapter.DeleteCommand.Parameters.Add("Values", OleDbType.Integer, 10, "Values");

adapter.DeleteCommand.Parameters.Add("Type", OleDbType.VarChar, 50, "Type");

adapter.DeleteCommand.Connection = connection;

//указать обьект соединения

adapter.UpdateCommand.Connection = connection;

//вызов обновления данных в базе

adapter.Update(dataSet.Tables[0]);

DataGridToArray(dataGridView1);

//зактырь таблицу-форму

if (fPunct.ActiveForm != null)

ActiveForm.Close();

}

void DataGridToArray(DataGridView dataGridView)

{

string[] arr = new string[dataGridView.RowCount];

for (int i = 0; i < arr.Length-1; i++)

{

if (dataGridView.Rows[i].Cells[1].Value.ToString() != "" || dataGridView.Rows[i].Cells[1].Value != null)

arr[i] = dataGridView.Rows[i].Cells[1].Value.ToString();

}

if (arr.Length != 0) PunctArr = arr;

}

/// <summary>

/// хранит строки с базы данных - знаки пунктуации

/// </summary>

internal static string[] PunctArr { get; private set; }//знаки препинания для FormTable.Filter()

}

}

**Модуль fVosxodParser.cs**

using System;

using System.Windows.Forms;

namespace Mary

{//Метод операторного предшествования

public partial class fVosxodParser : Form

{

public fVosxodParser()

{

InitializeComponent();

CreateTablePriorityVisual();

}

/// <summary>

/// Создать, заполнить колонки у listViewTablVosxod

/// </summary>

private void CreateTablePriorityVisual()

{

int i = 1;

listViewTablVosxod.Columns.Add((i-1).ToString(), "", 50);

foreach (string str in FormTable.ЛексемыБезПовторов)

{//создать колонки

if (str != null & str != "")

{

listViewTablVosxod.Columns.Add(i.ToString(), str, str.Length + 45, HorizontalAlignment.Center, 0);

listViewTablVosxod.Items.Add(str);//строки

i++;

}

}

AddStrListView("+","2","2");//значение-строка-столбец

}

/// <summary>

/// str - значение, znak[0]=строка, znak[1]=столбец

/// </summary>

void AddStrListView(string str, params string[] znak)

{

if (znak.Length != 0 & znak[0] != null & znak[1] != null

& Convert.ToInt32(znak[1]) < FormTable.TableCountStr

& Convert.ToInt32(znak[0]) < FormTable.TableCountStr)

for (int i = 0; i < FormTable.TableCountStr; i++)

{//по строке

for (int j = 0; j < Convert.ToInt32(znak[1]); j++)//по столбцу

listViewTablVosxod.Items[i].SubItems.Add("");

if (i == Convert.ToInt32(znak[0]))//по строке

listViewTablVosxod.Items[i].SubItems.Add(str);

}

}

}

internal class Priority

{

public Priority() { BufferWords(); }

private string outMessage;

private string[] arrPriority = {"="/\*0\*/, "<•"/\*1\*/, "•>"/\*2\*/, "<•"/\*3\*/, "•>"/\*4\*/, "="/\*5\*/, "•>"/\*6\*/,

"<•"/\*7\*/, "<•"/\*8\*/, "•>"/\*9\*/, "<•"/\*10\*/,"•>"/\*11\*/,"<•"/\*12\*/,"•>"/\*13\*/,

"<•"/\*14\*/,"•>"/\*15\*/,"<•"/\*16\*/,"="/\*17\*/, "<•"/\*18\*/,"•>"/\*19\*/

};

private string[] arrZapas = { "<•"/\*( +\*/, "•>"/\*+ <=\*/};//ДопЗначения при колизии

public static string[] words;//для слов

string text;//для строки

void BufferWords()

{//Вытаскиваем из буфера Текст "как есть" и переводим в массив

text = FormTable.BuffWords.DataBuffer;

char[] razdelenie = { ' ', '\n' };

words = text.Split(razdelenie); //Делим строку на ПОДСТРОКИ

for (int i = 0; i < words.Length; i++)

{//Поиск :

if (words[i] == ":")

{

if (words[i + 1] == ":\r" || words[i + 1] == ":")

{//Обьединить

words[i] += " :=";

for (int j = i + 1; j < words.Length - 1; j++)

{//Делаем сдвиг в лево оставшейся части

words[j] = words[j + 1];

}

words[words.Length - 1] = null;//Последнюю занулим

}

}

if (words[i] == "'")//i=[19]

{

if (words[i + 1] == "Верно" && words[i + 2] == "'")

{//Обьединить ' и Верно в одну ячейку

words[i] += "Верно'";//'Верно'

for (int j = i + 1; j < words.Length - 1; j++)

{//Делаем сдвиг в лево оставшейся части

words[j] = words[j + 1];

}

words[i + 1] = words[i + 2];//[)]

for (int j = i + 2; j < words.Length; j++)

{

words[j] = null;//Последнюю занулим

}

}

}

}

}

void OutRes()

{//Или 8 раз или сделать поиск \*> наперёд для подстраховки

for (int i = 0; i < 5; i++)

{//Вывод построчно

outMessage += i + "] "; perebor(); outMessage += Environment.NewLine;

}

for (int i = 5; i < 12; i++)

{//Вывод построчно

outMessage += i + "] "; pereborTwo(); outMessage += Environment.NewLine;

}

for (int i = 12; i < 13; i++)

{//Вывод построчно

outMessage += i + "] "; PereborFree(); outMessage += Environment.NewLine;

}

}

void perebor()

{//p\*(a+i-1);

int flagStop = 0, flagStop1 = 0;

if (words[3] == null)//если + занулен

{//<\* <==2.8 \*>//Слив на запасной массив при колизии

arrPriority[4] = arrZapas[0];//ставим <\*

for (int e = 0; e < words.Length; e++)

{

if (words[e] != null)//Нулы пропускаем

outMessage += " " + words[e] + " ";

}

arrPriority[4] = null; words[5] = null;

arrPriority[5] = null; words[6] = null; arrPriority[6] = null;

}

else if (words[3 - 1] == null && words[3] != null && words[3 + 1] == null)

{//\_+\_//Слив на запасной массив при колизии

arrPriority[1] = arrZapas[0];

arrPriority[2] = arrZapas[1];

for (int e = 0; e < words.Length; e++)

{

if (words[e] != null)//Нулы пропускаем

outMessage += " " + words[e] + " ";

}

arrPriority[1] = null; words[3] = null; arrPriority[2] = null;

}else

for (int p = 0; flagStop != 1; p++)

{

if (arrPriority[p] == "•>")

{//outMessage += "найден •>";

flagStop = 1;

for (int k = p; flagStop1 != 1; k--)

{

if (arrPriority[k] == "<•" || arrPriority[k] == "=")

{

for (int e = 0; e < words.Length; e++)

{

if (words[e] != null)//Нулы пропускаем

outMessage += " " + words[e] + " ";

}

//arrPriority[p] = null; arrPriority[k] = null;

arrPriority[k] = null; words[p] = null; arrPriority[p] = null;

flagStop1 = 1;

}

}

}

}

}

void pereborTwo()

{//begin write() read()

int flagStop = 0, flagStop1 = 0;

if (words[14 - 1] == null && words[14] != null && words[14 + 1] == null)

{//\_Write(\_//Слив на запасной массив при колизии

arrPriority[13] = arrZapas[0];

arrPriority[14] = arrZapas[1];

arrPriority[13] = null; words[14] = null; arrPriority[14] = null;

for (int e = 0; e < words.Length; e++)

{

if (words[e] != null)//Нулы пропускаем

outMessage += " " + words[e] + " ";

}

}else

if (words[10 - 1] == null && words[10] != null && words[10 + 1] == null)

{//\_Read(\_//Слив на запасной массив при колизии

arrPriority[9] = arrZapas[0];

arrPriority[10] = arrZapas[1];

arrPriority[9] = null; words[10] = null; arrPriority[10] = null;

for (int e = 0; e < words.Length; e++)

{

if (words[e] != null)//Нулы пропускаем

outMessage += " " + words[e] + " "; //to do

}

}else

for (int p = 0; flagStop != 1; p++)

{//5]

if (arrPriority[p] == "•>")

{//outMessage += "найден •>";

flagStop = 1;

for (int k = p; flagStop1 != 1; k--)

{

if (arrPriority[k] == "<•" || arrPriority[k] == "=")

{

arrPriority[k] = null; words[p] = null; arrPriority[p] = null;

for (int e = 0; e < words.Length; e++)

{

if (words[e] != null)//Нулы пропускаем

outMessage += " " + words[e] + " "; //for to do

}

//arrPriority[p] = null; arrPriority[k] = null;

flagStop1 = 1;

}

}

}

}

}

void PereborFree()

{

if (words[10] == null && words[12] != null && words[11] == null && words[8] != null)

{//\_;=;\_//Слив на запасной массив при колизии

arrPriority[7] = arrZapas[0];

arrPriority[12] = arrZapas[1];

arrPriority[7] = null; words[8] = null;

words[12] = null; arrPriority[12] = null;

for (int e = 0; e < words.Length; e++)

{

if (words[e] != null)//Нулы пропускаем

outMessage += " " + words[e] + " "; //for to do

}

}

}

private string inOutTextBox()

{//Выводим результаты в textBoxOut

var formMain = new fMain();

OutRes();

return formMain.txtBoxFortranOUT.Text = outMessage;

}

}

}

# Описание программы.

## Назначение и общее описание программы.

Приведенная программа предназначена для трансляции фрагмента программы с языка паскаль на язык Fortran 90. Программа реализована на основе однопроходного транслятора.

На вход синтаксическому анализатору, поступает цепочка отдельных терминальных символов, которые группируются им в единые синтаксические объекты – лексемы. Цепочка лексем представляет собой массив чисел. После распознания лексемы, исследуется цепочка лексем и устанавливается, принадлежит ли она структурным объектам заданного языка.

Синтаксический и лексический анализаторы распознают следующие ошибки:

• Двойное описание идентификатора

• Отсутствие блоков верхнего уровня языка паскаль

• Синтаксические ошибки

• Пунктуационные ошибки

При обнаружении ошибки транслятор выдает сообщение и подсвечивает возможный блок с неисправностью\ошибкой.

## Описание логической структуры программы.

Программа состоит из одного логического модуля.

## Способ обращения к программе

Обращение к программе производится напрямую запуском exe файла.

## Описание технических средств.

Программа была разработана на IBM совместимой машине с процес-сором AMD Athlon 3D-Now 850 MHz на языке C# под управле-нием ОС Windows 98Se. Особых системных требований не имеет. Предполагается ее работа на любом IBM-совместимом компьютере под управлением ОС Windows и установленном DotNetFramework 3. Для запуска без DotNetFramework можно воспользоваться сторонней перекомпановкой для Visual Studio, XenoCode\_Postbuild с платной лицензией.

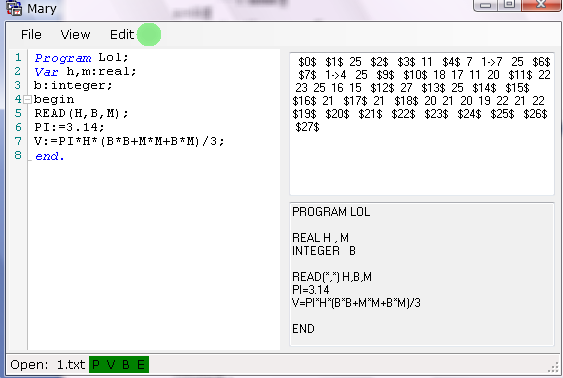
## Описание входных и выходных файлов.

Входным файлом является текстовый файл, содержащий программу на языке паскаль. Отображение результатов происходит в программе, панель Fortran.

# Текстовые примеры работы программы.

## Правильная программа

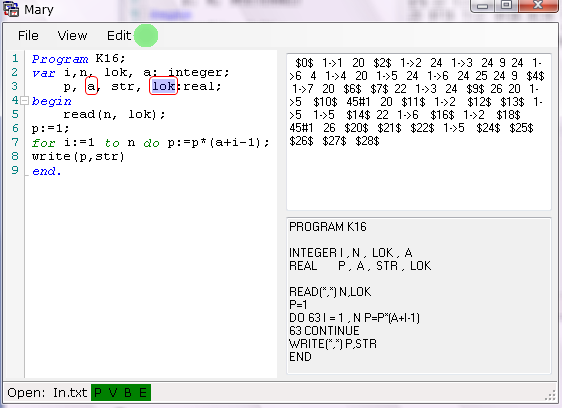
## Программа с ошибками.

За счет динамических модулей, которые производят разбор не завися от одной базовой конструкций, я добился отработки программы при разных «боевых» условиях.

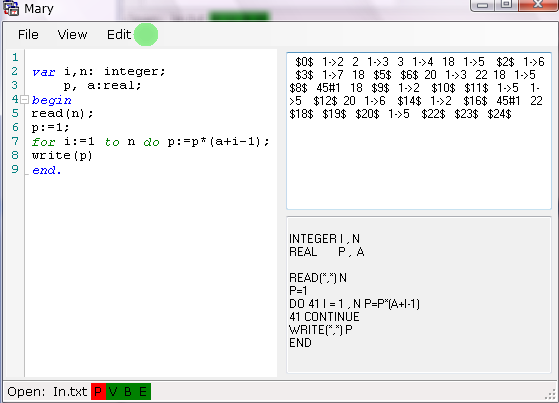
На рисунке:

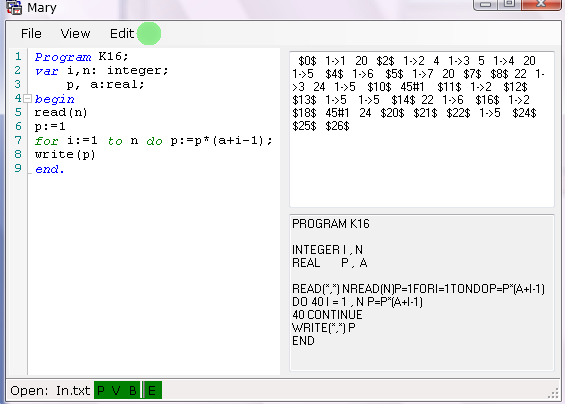
* Левая панель отображает загруженный файл, для наглядности используется подсветка синтаксических конструкций языка.
* В верхней правой части отображается транслит «слов» из правой панели в готовые коды лексем.
* В нижней правой части отображается сделанный перевод паскаль-фортран.
* Об открытии файла сообщает надпись «Open» в нижней панели состояния.
* Индикаторы PVBE отвечают за модули Program—Var—Begin—End.
* На верхней панели меню размещены пункты выбора File, View, Edit.
* Зеленая кнопка\индикатор рядом с пунктом «Edit» позволяет вносить правки непосредственно в код правой панели без необходимости открытия\закрытия файла. Поле «Edit» работает по алгоритму лексического парсера с той лишь разницой, что разбор идет не из файла, а с панели (это подразумевает динамически вносимые изменения и последующий анализ запуском Edit).

## Программа с добавленными операторами.

Внесём повторные обьявления в блок Var:

Красным программа выделила возможные ошибки – повторное обьявление.

Теперь удалим что-нибудь, например блок Program:

Загарелось оповещение об отсутствии блока Program.

О внутренних ошибках известит индикация межблоков:

На рисунке удалены знаки «;» операторов read() и p:=1. Оповещение сообщает о возникших ошибках.