МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С. П. КОРОЛЕВА»

(САМАРСКИЙ УНИВЕРСИТЕТ)

Отчёт по лабораторной работе

по курсу «Теория формальных языков и грамматик»

Вариант №10.

Выполнил:  
Филатов В. В.

гр.6303

Проверил:  
Литвинов В. Г.

Самара 2017

**Задание:**

Написать программу синтаксического анализа автоматного языка операторов описания файлов языка Turbo Pascal, имеющего вид:

**VAR**<идентификатор переменной>**:**<описание файла>**;**

<описание файла> :: = **FILE OF**<тип>|**TEXT**|**FILE**|<иденти-фикатор типа>

<тип> :: = CHAR|STRING[**[**<целая константа>**]**]|INTEGER|REAL|DOUBLE|SINGLE|BYTE

<идентификатор> - идентификатор языка Turbo Pascal, начинается с буквы или знака подчеркивания, включает совокупность букв, цифр, подчеркиваний; не допускает пробелы и специальные символы, ввести ограничение на длину (не более 8 символов) и не может быть зарезервированным словом (VAR, FILE OF, TEXT, FILE, CHAR, STRING, DOUBLE, SINGLE, BYTE, REAL, INTEGER);

<целая константа> - число в диапазоне 0 ÷ 255;

Семантика:

Построить и вывести на печать таблицу идентификаторов и констант. Учесть перечисленные выше ограничения на идентификаторы и константы. Не допускать совпадение идентификаторов типа и переменной.

Сообщать об ошибках при анализе, указывая курсором место возникновения ошибки и ее содержание.

Примеры правильных цепочек:

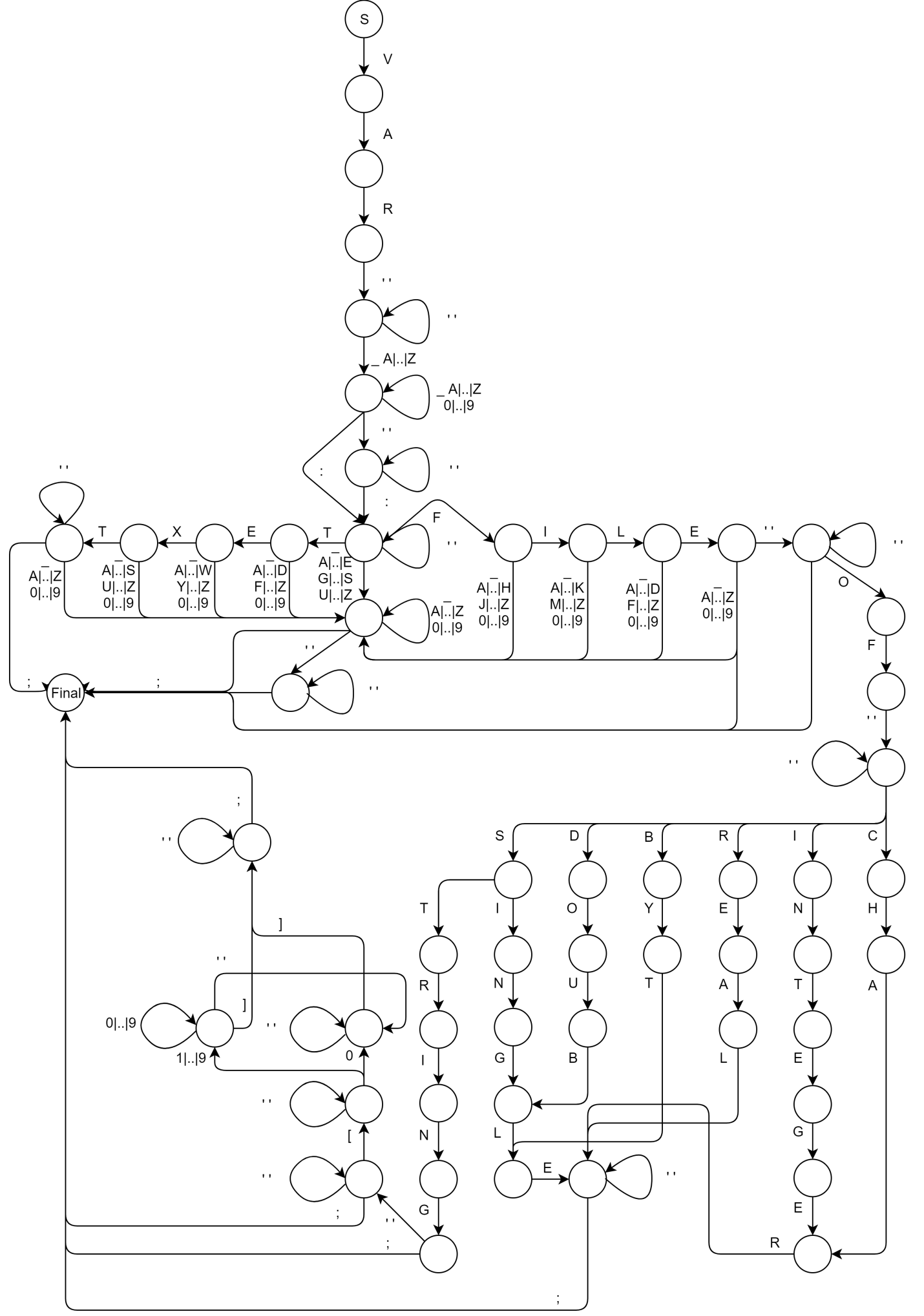
VAR name : file of char;

VAR \_A\_C : TEXT;

VAR C12 : file of STRING [10];

VAR D : myfile;

**Основной граф:**

****

**Листинг программы:**

*Класс Result*

namespace Analizator

{

public enum Err

{

NoError,

UnknownError,

IdReservedError,

OverflowCharacters,

OutOfRange,

VarExpected,

OfExpected,

IdOrKeywordExpected,

IdExpected,

СolonExpected,

SemicolonExpected,

СonstantExpected,

NotFoundTypeOfConstantExpected,

OpenBracketsExpected,

CloseBracketsExpected,

IdTypeVarExpected

}

class Result

{

int ErrPos;

Err Err;

string \_Str;

public Result(int ErrPos, Err Err, string Value)

{

this.ErrPos = ErrPos;

this.Err = Err;

\_Str = Value;

}

public int ErrPosition

{

get

{

return ErrPos;

}

}

public string ErrMessage

{

get

{

switch (Err)

{

case Err.NoError:

{

return "Нет ошибок";

}

case Err.UnknownError:

{

return "Неизвестная ошибка";

}

case Err.IdReservedError:

{

return "Ид-р - зарезервированное слово";

}

case Err.OverflowCharacters:

{

return "кол-во символов в идентиф-ре превышает 8";

}

case Err.OutOfRange:

{

return "целая константа вне диапазона 0 - 255";

}

case Err.VarExpected:

{

return "ожидалось кл.слово VAR";

}

case Err.OfExpected:

{

return "ожидалось кл.слово OF";

}

case Err.IdOrKeywordExpected:

{

return "ожидался идентиф-р или кл. слово";

}

case Err.IdExpected:

{

return "ожидался идентиф-р";

}

case Err.СolonExpected:

{

return "ожидалось двоеточие";

}

case Err.SemicolonExpected:

{

return "ожидалась точка с зпт";

}

case Err.СonstantExpected:

{

return "ожидалась целая константа";

}

case Err.NotFoundTypeOfConstantExpected:

{

return "такого типа константы нет";

}

case Err.OpenBracketsExpected:

{

return "ожидалась открывающая скобка";

}

case Err.CloseBracketsExpected:

{

return "ожидалась закрывающая скобка";

}

case Err.IdTypeVarExpected:

{

return "совпадение ид ра типа и пер ой";

}

default:

{

return "Неизвестная ошибка";

}

}

}

}

public string Str

{

get

{

return \_Str;

}

}

}

}

*Класс Analizator*

using System;

using System.Collections.Generic;

using System.Linq;

namespace Analizator

{

static class Analizator

{

enum State { Start, Error, Final, PredFinal, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, AA, BB, CC, DD, EE, FF, GG, HH, II, JJ, KK, LL, MM, NN, OO, PP, QQ, RR, SS};

private static int i = 0;

private static int max = 8;

private static int cnt;

private static List<string> l = new List<string>();

private static List<string> ll = new List<string>();

public static List<string> L

{

get { return l; }

}

public static List<string> LL

{

get { return ll; }

}

private static string st;

private static int len;

private static Err Err;

private static int ErrPos;

private static string str;

public static Result Result(string value)

{

str = "";

st = value;

i = 0;

len = st.Length;

SetError(Err.NoError, -1);

Analiz();

return new Result(ErrPos, Err, str);

}

private static void SetError(Err ErrorType, int ErrorPosition)

{

Err = ErrorType;

ErrPos = ErrorPosition;

}

public static bool ReservedWord(string word)

{

return (word == "VAR" || word == "FILE" || word == "TEXT" || word == "CHAR" || word == "STRING" || word == "DOUBLE" || word == "SINGLE" || word == "BYTE" || word == "REAL" || word == "INTEGER");

}

private static bool Analiz()

{

l.Clear();

ll.Clear();

State Sta = State.Start;

int TmpPos = i;

string s = "";

while ((Sta != State.Error) && (Sta != State.Final))

{

if (i >= len)

{

SetError(Err.SemicolonExpected, i - 1);

Sta = State.Error;

}

else

{

switch (Sta)

{

case State.Start:

{

if (st[i] == ' ' || st[i] == '\r' || st[i] == '\n')

{

i++;

Sta = State.Start;

}

else if (st[i] == 'V')

{

i++;

Sta = State.A;

}

else

{

SetError(Err.VarExpected, i);

Sta = State.Error;

}

}

break;

case State.A:

{

if (st[i] == 'A')

{

i++;

Sta = State.B;

}

else

{

SetError(Err.VarExpected, i);

Sta = State.Error;

}

}

break;

case State.B:

{

if (st[i] == 'R')

{

i++;

Sta = State.C;

}

else

{

SetError(Err.VarExpected, i);

Sta = State.Error;

}

}

break;

case State.C:

{

if (st[i] == ' ')

{

i++;

Sta = State.C;

}

else if (char.IsLetter(st[i]) || st[i] == '\_')

{

s += st[i];

i++;

cnt++;

Sta = State.D;

}

else

{

SetError(Err.IdExpected, i - 1);

Sta = State.Error;

}

}

break;

case State.D:

{

if (char.IsLetter(st[i]) || st[i] == '\_' || char.IsDigit(st[i]))

{

s += st[i];

i++;

cnt++;

Sta = State.D;

}

else if (st[i] == ' ')

{

if (ReservedWord(s))

{

s = "";

SetError(Err.IdReservedError, i - 1);

Sta = State.Error;

}

else

{

l.Add(s);

s = "";

if (cnt > max)

{

cnt = 0;

SetError(Err.OverflowCharacters, i - 1);

Sta = State.Error;

}

cnt = 0;

i++;

Sta = State.E;

}

}

else if (st[i] == ':')

{

if (ReservedWord(s))

{

s = "";

SetError(Err.IdReservedError, i - 1);

Sta = State.Error;

}

else

{

l.Add(s);

s = "";

if (cnt > max)

{

cnt = 0;

SetError(Err.OverflowCharacters, i - 1);

Sta = State.Error;

}

cnt = 0;

i++;

Sta = State.F;

}

}

else

{

SetError(Err.IdExpected, i - 1);

Sta = State.Error;

}

}

break;

case State.E:

{

if (st[i] == ' ')

{

i++;

Sta = State.E;

}

else if (st[i] == ':')

{

i++;

Sta = State.F;

}

else

{

SetError(Err.СolonExpected, i - 1);

Sta = State.Error;

}

}

break;

case State.F:

{

if (st[i] == ' ')

{

i++;

Sta = State.F;

}

else if (st[i] == 'F')

{

i++;

Sta = State.G;

}

else if (st[i] == 'T')

{

i++;

Sta = State.PP;

}

else if (char.IsLetter(st[i]))

{

s += st[i];

i++;

cnt++;

Sta = State.SS;

}

else if (st[i] == '\_')

{

s += st[i];

i++;

cnt++;

Sta = State.SS;

}

else

{

SetError(Err.IdOrKeywordExpected, i - 1);

Sta = State.Error;

}

}

break;

case State.G:

{

if (st[i] == 'I')

{

i++;

Sta = State.H;

}

else if (st[i] != 'I')

{

Sta = State.SS;

}

else

{

Sta = State.Error;

}

}

break;

case State.H:

{

if (st[i] == 'L')

{

i++;

Sta = State.I;

}

else if (st[i] != 'L')

{

Sta = State.SS;

}

else

{

Sta = State.Error;

}

}

break;

case State.I:

{

if (st[i] == 'E')

{

i++;

Sta = State.J;

}

else if (st[i] != 'E')

{

Sta = State.SS;

}

else

{

Sta = State.Error;

}

}

break;

case State.J:

{

if (st[i] == ' ')

{

i++;

Sta = State.J;

}

else if (st[i] == ';')

{

i++;

Sta = State.Final;

}

else if (st[i] == 'O')

{

i++;

Sta = State.K;

}

else

{

SetError(Err.OfExpected, i - 1);

Sta = State.Error;

}

}

break;

case State.K:

{

if (st[i] == 'F')

{

i++;

Sta = State.L;

}

else

{

SetError(Err.OfExpected, i - 1);

Sta = State.Error;

}

}

break;

case State.L:

{

if (st[i] == ' ')

{

i++;

Sta = State.L;

}

else if (st[i] == 'R')//1)+

{

i++;

Sta = State.M;

}

else if (st[i] == 'S')//4)+

{

i++;

Sta = State.X;//

}

else if (st[i] == 'B')//5)+

{

i++;

Sta = State.KK;//

}

else if (st[i] == 'D')//6)+

{

i++;

Sta = State.MM;//

}

else if (st[i] == 'I')//2)+

{

i++;

Sta = State.P;//

}

else if (st[i] == 'C')//3)+

{

i++;

Sta = State.V;//

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.M:

{

if (st[i] == 'E')

{

i++;

Sta = State.N;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.N:

{

if (st[i] == 'A')

{

i++;

Sta = State.O;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.O:

{

if (st[i] == 'L')

{

i++;

Sta = State.PredFinal;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.P:

{

if (st[i] == 'N')

{

i++;

Sta = State.Q;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.Q:

{

if (st[i] == 'T')

{

i++;

Sta = State.R;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.R:

{

if (st[i] == 'E')

{

i++;

Sta = State.S;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.S:

{

if (st[i] == 'G')

{

i++;

Sta = State.T;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.T:

{

if (st[i] == 'E')

{

i++;

Sta = State.U;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.U:

{

if (st[i] == 'R')

{

i++;

Sta = State.PredFinal;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.V:

{

if (st[i] == 'H')

{

i++;

Sta = State.W;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.W:

{

if (st[i] == 'A')

{

i++;

Sta = State.U;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.X:

{

if (st[i] == 'I')

{

i++;

Sta = State.Y;

}

else if (st[i] == 'T')

{

i++;

Sta = State.CC;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.Y:

{

if (st[i] == 'N')

{

i++;

Sta = State.Z;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.Z:

{

if (st[i] == 'G')

{

i++;

Sta = State.AA;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.AA:

{

if (st[i] == 'L')

{

i++;

Sta = State.BB;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.BB:

{

if (st[i] == 'E')

{

i++;

Sta = State.PredFinal;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.CC:

{

if (st[i] == 'R')

{

i++;

Sta = State.DD;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.DD:

{

if (st[i] == 'I')

{

i++;

Sta = State.EE;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.EE:

{

if (st[i] == 'N')

{

i++;

Sta = State.FF;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.FF:

{

if (st[i] == 'G')

{

i++;

Sta = State.GG;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.GG:

{

if (st[i] == ';')

{

i++;

Sta = State.Final;

}

else if (st[i] == ' ')

{

i++;

Sta = State.GG;

}

else if (st[i] == '[')

{

i++;

Sta = State.HH;

}

else

{

SetError(Err.OpenBracketsExpected, i);

Sta = State.Error;

}

}

break;

case State.HH:

{

if (st[i] == ' ')

{

i++;

Sta = State.HH;

}

else if (st[i] == '0')

{

ll.Add(st[i].ToString());

i++;

Sta = State.JJ;

}

else if (char.IsDigit(st[i]))//1-9

{

s += st[i];

i++;

Sta = State.II;

}

else

{

SetError(Err.СonstantExpected, i);

Sta = State.Error;

}

}

break;

case State.II:

{

if (char.IsDigit(st[i]))

{

s += st[i];

i++;

Sta = State.II;

}

else if (st[i] == ']')

{

ll.Add(s);

byte c;

Byte.TryParse(s, out c);

s = "";

if (c == 0)

{

SetError(Err.OutOfRange, i);

Sta = State.Error;

}

else

{

i++;

Sta = State.PredFinal;

}

}

else if (st[i] == ' ')

{

i++;

Sta = State.JJ;

}

else

{

SetError(Err.CloseBracketsExpected, i);

Sta = State.Error;

}

}

break;

case State.JJ:

{

if (st[i] == ' ')

{

i++;

Sta = State.JJ;

}

else if (st[i] == ']')

{

if (ll.Count == 0)

{

ll.Add(s);

byte c;

Byte.TryParse(s, out c);

s = "";

if (c == 0)

{

SetError(Err.OutOfRange, i);

Sta = State.Error;

}

else

{

i++;

Sta = State.PredFinal;

}

}

else

{

i++;

Sta = State.PredFinal;

}

}

else

{

SetError(Err.CloseBracketsExpected, i);

Sta = State.Error;

}

}

break;

case State.KK:

{

if (st[i] == 'Y')

{

i++;

Sta = State.LL;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.LL:

{

if (st[i] == 'T')

{

i++;

Sta = State.BB;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.MM:

{

if (st[i] == 'O')

{

i++;

Sta = State.NN;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.NN:

{

if (st[i] == 'U')

{

i++;

Sta = State.OO;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.OO:

{

if (st[i] == 'B')

{

i++;

Sta = State.AA;

}

else

{

SetError(Err.NotFoundTypeOfConstantExpected, i);

Sta = State.Error;

}

}

break;

case State.PP:

{

if (st[i] == 'E')

{

i++;

Sta = State.QQ;

}

else if (st[i] != 'E')

{

Sta = State.SS;

}

else

{

Sta = State.Error;

}

}

break;

case State.QQ:

{

if (st[i] == 'X')

{

i++;

Sta = State.RR;

}

else if (st[i] != 'X')

{

Sta = State.SS;

}

else

{

Sta = State.Error;

}

}

break;

case State.RR:

{

if (st[i] == 'T')

{

i++;

Sta = State.PredFinal;

}

else if (st[i] != 'T')

{

Sta = State.SS;

}

else

{

Sta = State.Error;

}

}

break;

case State.SS:

{

if (char.IsLetter(st[i]) || st[i] == '\_' || char.IsDigit(st[i]))

{

s += st[i];

i++;

cnt++;

Sta = State.SS;

}

else if (st[i] == ' ')

{

if (l.First() == s)

{

SetError(Err.IdTypeVarExpected, i - 1);

Sta = State.Error;

}

else {

l.Add(s);

}

s = "";

if (cnt > max)

{

cnt = 0;

SetError(Err.OverflowCharacters, i - 1);

Sta = State.Error;

}

cnt = 0;

i++;

Sta = State.PredFinal;

}

else if (st[i] == ';')

{

if (ReservedWord(s))

{

s = "";

SetError(Err.IdReservedError, i - 1);

Sta = State.Error;

}

else

{

if (l.First() == s)

{

SetError(Err.IdTypeVarExpected, i - 1);

Sta = State.Error;

}

else

{

l.Add(s);

}

s = "";

cnt = 0;

Sta = State.Final;

}

}

else

{

SetError(Err.IdExpected, i - 1);

Sta = State.Error;

}

}

break;

case State.PredFinal:

{

if (st[i] == ' ')

{

i++;

Sta = State.PredFinal;

}

else if (st[i] == ';')

{

Sta = State.Final;

}

else

{

SetError(Err.SemicolonExpected, i - 1);

Sta = State.Error;

}

}

break;

default:

{

SetError(Err.UnknownError, i);

Sta = State.Error;

}

break;

}

}

}

if (Sta == State.Error)

{

TmpPos = i;

return false;

}

else

{

return true;

}

}

}

}

*Класс пользовательского интерфейса*

using System;

using System.Windows.Forms;

namespace Analizator

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

{

label2.Text = "";

label3.Text = "";

string str = textBox1.Text.ToUpper();

Result r = Analizator.Result(str);

if (r.ErrPosition == -1)

{

richTextBox1.Text = "Введенная строка соответствует синтаксису языка Turbo Pascal\r\n";

foreach (string item in Analizator.L)

{

richTextBox1.Text += item + "\r\n";

}

foreach (string item in Analizator.LL)

{

richTextBox1.Text += item + "\r\n";

}

}

else

{

string strs = r.ErrPosition.ToString() + "\n" + r.ErrMessage;

richTextBox1.Text = strs;

}

}

}

}

**Результат выполнения программы:**

