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 №11.

Выполнил:  
Елфимов А.Г.

гр.6303

Проверил:  
Литвинов В. Г.

Самара 2017

**Задание:**

Написать программу синтаксического анализа автоматного языка операторов описания заголовков процедур и функций языка Turbo Pascal, имеющего вид:

**PROCEDURE** <идентификатор>[**(**<список формальных параметров>**)**]**;**

**FUNCTION** <идентификатор>[**(**<список формальных параметров>**)**]**:**<тип>**;**

<список формальных параметров> :: = <описание>[**;**<список формальных параметров>]

<описание> :: = [**VAR**]<список идентификаторов>**:**<тип>

<список идентификаторов> :: = <идентификатор>[**,**<список идентификаторов>]

<тип> :: = REAL|INTEGER|CHAR|BYTE|DOUBLE|STRING|BOOLEAN

<идентификатор> - идентификатор языка Turbo Pascal, начинается с буквы или знака подчеркивания, включает совокупность букв, цифр; не допускает использование пробелов и специальных символов, ввести ограничение на длину (не более 8 символов) и не может быть зарезервированным словом (PROCEDURE, FUNCTION, VAR, REAL, INTEGER, CHAR, BYTE, DOUBLE, STRING, BOOLEAN).

Семантика:

Построить таблицу идентификаторов с указанием типа и объема памяти под переменную. Учесть ограничения на идентификаторы.

Сообщать об ошибках в случае дублирования идентификаторов. Указывать курсором место ошибки при анализе и ее содержание.

Примеры правильных цепочек:

PROCEDURE \_CA\_B;

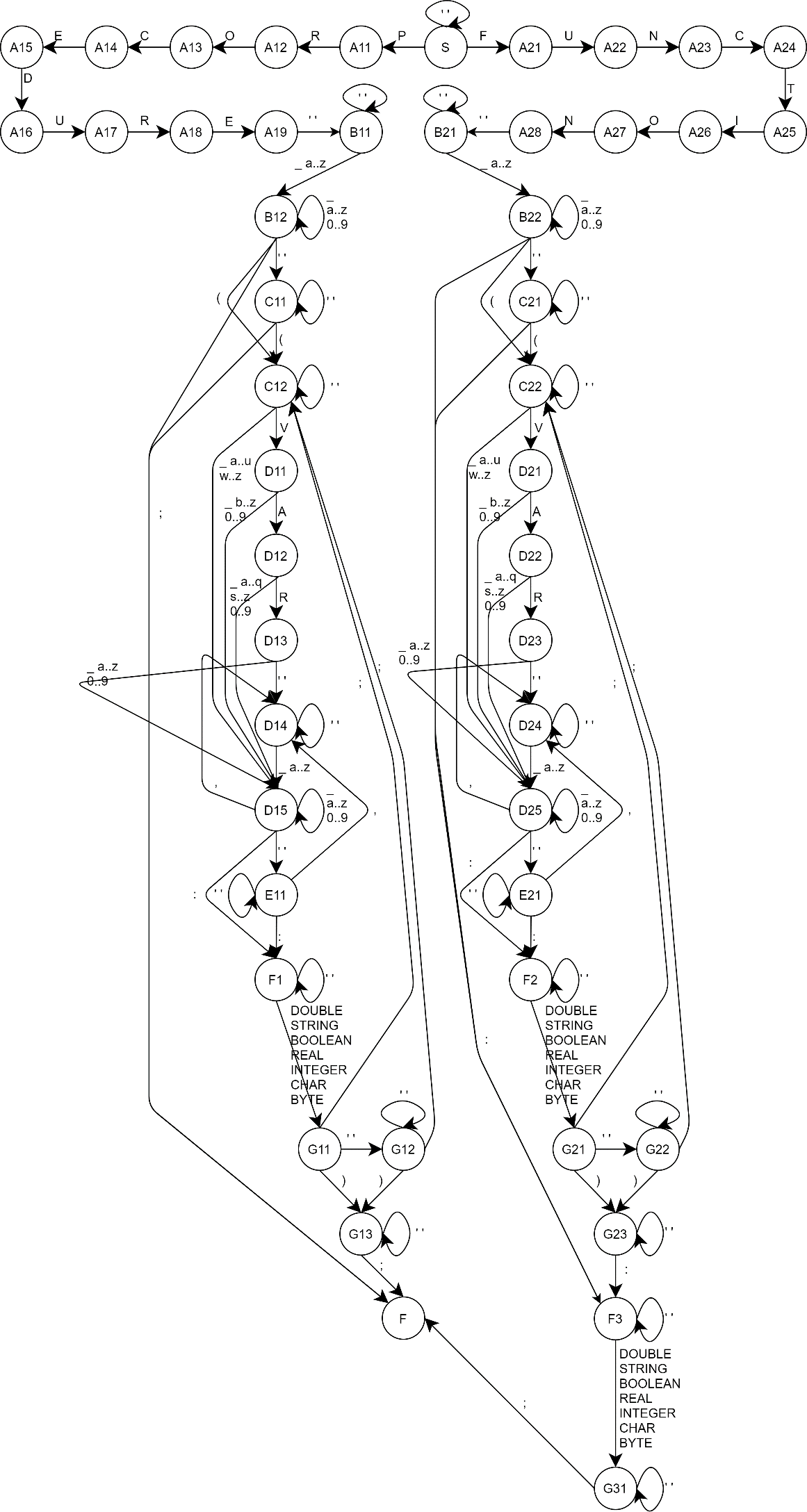
PROCEDURE SUBSTR ( S : CHAR; I : INTEGER );

PROCEDURE IM1 ( A, B : BOOLEAN; VAR C, D : DOUBLE );

FUNCTION IM2 : REAL;

FUNCTION IM3 ( E, F : BYTE; G : STRING ) : INTEGER;

**Основной граф:**



**Листинг программы:**

*Класс Errors*

namespace Analyzer

{

public enum Error

{

NoError,

FunctionExpected,

ProcedureExpected,

ProcedureOrFunctionExpected,

VarExpected,

SpaceExpected,

TypeExpected,

DescriptionError,

IdentifierLengthError,

IdentifierFirstSymbolError,

IdentifierIsReservedWordError,

IdentifierError,

UnknownError,

OutOfRange,

OpenBracketExpected,

CloseBracketExpected,

OpenBracketOrColonExpected,

CloseBracketOrSemicolonExpected,

ListOfIdentifiersError,

ColonExpected,

RealExpected,

IntegerExpected,

CharExpected,

ByteExpected,

DoubleExpected,

StringExpected,

BooleanExpected,

IdentifierDuplicationError,

OpenBracketOrSemicolonExpected,

SemicolonExpected

}

public class Errors

{

private readonly int \_errorPosition;

private readonly Error \_error;

private string \_string;

public Errors(int errorPosition, Error error, string value)

{

this.\_errorPosition = errorPosition;

this.\_error = error;

this.\_string = value;

}

public int ErrorPosition

{

get => \_errorPosition;

}

public string ErrorMessage

{

get

{

switch (\_error)

{

case Error.NoError: return "Ошибок нет";

case Error.FunctionExpected: return "Ожидалось слово FUNCTION";

case Error.ProcedureExpected: return "Ожидалось слово PROCEDURE";

case Error.ProcedureOrFunctionExpected: return "Ожидалось слово PROCEDURE или FUNCTION";

case Error.VarExpected: return "Ожидалось слово VAR";

case Error.SpaceExpected: return "Ожидался пробел";

case Error.TypeExpected: return "Ожидался тип";

case Error.IdentifierFirstSymbolError: return "Идентификатор начинается с недопустимого символа";

case Error.IdentifierIsReservedWordError: return "Идентификатор совпадает с зарезервированным словом";

case Error.IdentifierLengthError: return "Превышена длина идентификатора";

case Error.IdentifierError: return "Ошибка в записи идентификатора";

case Error.UnknownError: return "Неизвестная ошибка";

case Error.OutOfRange: return "Выход за границы анализируемой строки";

case Error.OpenBracketExpected: return "Ожидалась открывающаяся скобка";

case Error.CloseBracketExpected: return "Ожидалась закрывающаяся скобка";

case Error.OpenBracketOrColonExpected: return "Ожидалась открывающаяся скобка или двоеточие";

case Error.OpenBracketOrSemicolonExpected: return "Ожидалась открывающаяся скобка или точка с запятой";

case Error.CloseBracketOrSemicolonExpected: return "Ожидалась закрывающаяся скобка или точка с запятой";

case Error.SemicolonExpected: return "Ожидалась точка с запятой";

case Error.ListOfIdentifiersError: return "Ошибка в списке идентификаторов";

case Error.IdentifierDuplicationError: return "Встречено дублирование идентификаторов";

case Error.ColonExpected: return "Ожидалось двоеточие";

case Error.RealExpected: return "Ожидался тип REAL";

case Error.IntegerExpected: return "Ожидался тип INTEGER";

case Error.CharExpected: return "Ожидался тип CHAR";

case Error.ByteExpected: return "Ожидался тип BYTE";

case Error.DoubleExpected: return "Ожидался тип DOUBLE";

case Error.StringExpected: return "Ожидался тип STRING";

case Error.BooleanExpected: return "Ожидался тип BOOLEAN";

default: return "Неизвестная ошибка";

}

}

}

}

}

*Класс Analyzer*

using System.Collections.Generic;

namespace Analyzer

{

public static class Analyzer

{

private enum State {Start, Error, Final, A11, A12, A13, A14, A15, A16, A17, A18, A19, B11, B12, C11, C12, D11, D12, D13, D14, D15,

E11, F1, F1A, F1B, F1C, F1D, F1E, F1F, F1G, F1H, F1I, F1J, F1K, F1L, F1O, F1P, F1Q, F1R, F1S, F1T, F1U, F1V, F1W, F1X, F1Y,

G11, G12, G13, A21, A22, A23, A24, A25, A26, A27, A28, B21, B22, C21, C22, D21, D22, D23, D24, D25, E21, F2, F2A, F2B, F2C, F2D,

F2E, F2F, F2G, F2H, F2I, F2J, F2K, F2L, F2O, F2P, F2Q, F2R, F2S, F2T, F2U, F2V, F2W, F2X, F2Y, G21, G22, G23, F3, F3A, F3B, F3C,

F3D, F3E, F3F, F3G, F3H, F3I, F3J, F3K, F3L, F3O, F3P, F3Q, F3R, F3S, F3T, F3U, F3V, F3W, F3X, F3Y, G31};

private static Error \_error;

private static string \_str, \_stringOfError;

private static int \_ind, \_errPos, \_length, \_identifierLength, \_identifierCount;

public static List<string> ListOfIdentifiers = new List<string>() , ListOfTypes = new List<string>(), ListOfSizes = new List<string>();

public static Errors Test(string value)

{

\_stringOfError = "";

\_str = value;

\_ind = 0;

\_length = \_str.Length;

SetError(Error.NoError, -1);

Analyze();

return new Errors(\_errPos, \_error, \_stringOfError);

}

private static void SetError(Error errorType, int errorPosition)

{

\_error = errorType;

\_errPos = errorPosition;

}

public static bool ReservedWord(string word)

{

if (word == "PROCEDURE" || word == "FUNCTION" || word == "VAR" ||

word == "REAL" || word == "INTEGER" || word == "CHAR" ||

word == "BYTE" || word == "DOUBLE" || word == "STRING" || word == "BOOLEAN")

return true;

return false;

}

private static bool Analyze()

{

var state = State.Start;

var tmpPos = \_ind;

\_identifierLength = 0;

\_identifierCount = 0;

var c = "";

SetError(Error.NoError, 0);

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

{

if (\_ind >= \_length)

{

SetError(Error.OutOfRange, \_ind - 1);

state = State.Error;

}

else

{

switch (state)

{

case State.Start:

{

if (\_str[\_ind] == ' ') state = State.Start;

else if (\_str[\_ind] == 'P')

{

ListOfTypes.Add("");

ListOfSizes.Add("");

state = State.A11;

}

else if (\_str[\_ind] == 'F') state = State.A21;

else

{

SetError(Error.ProcedureOrFunctionExpected, \_ind);

state = State.Error;

}

}

break;

//PROCEDURE

case State.A11:

{

if (\_str[\_ind] == 'R') state = State.A12;

else

{

SetError(Error.ProcedureExpected, \_ind);

state = State.Error;

}

}

break;

case State.A12:

{

if (\_str[\_ind] == 'O') state = State.A13;

else

{

SetError(Error.ProcedureExpected, \_ind);

state = State.Error;

}

}

break;

case State.A13:

{

if (\_str[\_ind] == 'C') state = State.A14;

else

{

SetError(Error.ProcedureExpected, \_ind);

state = State.Error;

}

}

break;

case State.A14:

{

if (\_str[\_ind] == 'E') state = State.A15;

else

{

SetError(Error.ProcedureExpected, \_ind);

state = State.Error;

}

}

break;

case State.A15:

{

if (\_str[\_ind] == 'D') state = State.A16;

else

{

SetError(Error.ProcedureExpected, \_ind);

state = State.Error;

}

}

break;

case State.A16:

{

if (\_str[\_ind] == 'U') state = State.A17;

else

{

SetError(Error.ProcedureExpected, \_ind);

state = State.Error;

}

}

break;

case State.A17:

{

if (\_str[\_ind] == 'R') state = State.A18;

else

{

SetError(Error.ProcedureExpected, \_ind);

state = State.Error;

}

}

break;

case State.A18:

{

if (\_str[\_ind] == 'E') state = State.A19;

else

{

SetError(Error.ProcedureExpected, \_ind);

state = State.Error;

}

}

break;

case State.A19:

{

if (\_str[\_ind] == ' ') state = State.B11;

else

{

SetError(Error.SpaceExpected, \_ind);

state = State.Error;

}

}

break;

case State.B11:

{

if (\_str[\_ind] == ' ') state = State.B11;

else if (\_str[\_ind] == '\_' || char.IsLetter(\_str[\_ind]))

{

c += \_str[\_ind];

\_identifierLength++;

state = State.B12;

}

else

{

SetError(Error.IdentifierFirstSymbolError, \_ind);

state = State.Error;

}

}

break;

case State.B12:

{

if (\_str[\_ind] == '\_' || char.IsLetterOrDigit(\_str[\_ind]))

{

c += \_str[\_ind];

\_identifierLength++;

if (\_identifierLength > 8)

{

SetError(Error.IdentifierLengthError, \_ind);

state = State.Error;

}

else state = State.B12;

}

else if (\_str[\_ind] == ' ')

{

if (ReservedWord(c))

{

SetError(Error.IdentifierIsReservedWordError, \_ind);

state = State.Error;

}

else

{

ListOfIdentifiers.Add(c);

c = "";

\_identifierLength = 0;

state = State.C11;

}

}

else if (\_str[\_ind] == '(')

{

if (ReservedWord(c))

{

SetError(Error.IdentifierIsReservedWordError, \_ind);

state = State.Error;

}

else

{

ListOfIdentifiers.Add(c);

c = "";

\_identifierLength = 0;

state = State.C12;

}

}

else if (\_str[\_ind] == ';')

{

if (ReservedWord(c))

{

SetError(Error.IdentifierIsReservedWordError, \_ind);

state = State.Error;

}

else

{

ListOfIdentifiers.Add(c);

c = "";

\_identifierLength = 0;

state = State.Final;

}

}

else

{

SetError(Error.OpenBracketOrSemicolonExpected, \_ind);//

state = State.Error;

}

}

break;

case State.C11:

{

if (\_str[\_ind] == ' ') state = State.C11;

else if (\_str[\_ind] == '(') state = State.C12;

else if (\_str[\_ind] == ';') state = State.Final;

else

{

SetError(Error.OpenBracketOrSemicolonExpected, \_ind);

state = State.Error;

}

}

break;

case State.C12:

{

if (\_str[\_ind] == ' ') state = State.C12;

else if (\_str[\_ind] == 'V')

{

c += \_str[\_ind];

\_identifierLength++;

state = State.D11;

}

else if (\_str[\_ind] == '\_' || char.IsLetter(\_str[\_ind]))

{

c += \_str[\_ind];

\_identifierLength++;

state = State.D15;

}

else

{

SetError(Error.IdentifierFirstSymbolError, \_ind);

state = State.Error;

}

}

break;

case State.D11:

{

if (\_str[\_ind] == 'A')

{

c += \_str[\_ind];

\_identifierLength++;

if (\_identifierLength > 8)

{

SetError(Error.IdentifierLengthError, \_ind);

state = State.Error;

}

else state = State.D12;

}

else if (\_str[\_ind] == '\_' || char.IsLetterOrDigit(\_str[\_ind]))

{

c += \_str[\_ind];

\_identifierLength++;

if (\_identifierLength > 8)

{

SetError(Error.IdentifierLengthError, \_ind);

state = State.Error;

}

else state = State.D15;

}

else

{

SetError(Error.IdentifierError, \_ind);

state = State.Error;

}

}

break;

case State.D12:

{

if (\_str[\_ind] == 'R')

{

c += \_str[\_ind];

\_identifierLength++;

if (\_identifierLength > 8)

{

SetError(Error.IdentifierLengthError, \_ind);

state = State.Error;

}

else state = State.D13;

}

else if (\_str[\_ind] == '\_' || char.IsLetterOrDigit(\_str[\_ind]))

{

c += \_str[\_ind];

\_identifierLength++;

if (\_identifierLength > 8)

{

SetError(Error.IdentifierLengthError, \_ind);

state = State.Error;

}

else state = State.D15;

}

else

{

SetError(Error.IdentifierError, \_ind);

state = State.Error;

}

}

break;

case State.D13:

{

if (\_str[\_ind] == ' ')

{

c = "";

\_identifierLength = 0;

state = State.D14;

}

else if (\_str[\_ind] == '\_' || char.IsLetterOrDigit(\_str[\_ind]))

{

c += \_str[\_ind];

\_identifierLength++;

if (\_identifierLength > 8)

{

SetError(Error.IdentifierLengthError, \_ind);

state = State.Error;

}

else state = State.D15;

}

else

{

SetError(Error.IdentifierError, \_ind);

state = State.Error;

}

}

break;

case State.D14:

{

if (\_str[\_ind] == ' ') state = State.D14;

else if (\_str[\_ind] == '\_' || char.IsLetter(\_str[\_ind]))

{

c += \_str[\_ind];

\_identifierLength++;

if (\_identifierLength > 8)

{

SetError(Error.IdentifierLengthError, \_ind);

state = State.Error;

}

else state = State.D15;

}

else

{

SetError(Error.IdentifierError, \_ind);

state = State.Error;

}

}

break;

case State.D15:

{

if (\_str[\_ind] == ' ')

{

if (ListOfIdentifiers.Contains(c))

{

c = "";

\_identifierLength = 0;

SetError(Error.IdentifierDuplicationError, \_ind);

state = State.Error;

}

else if (ReservedWord(c))

{

SetError(Error.IdentifierIsReservedWordError, \_ind);

state = State.Error;

}

else

{

ListOfIdentifiers.Add(c);

\_identifierCount++;

c = "";

\_identifierLength = 0;

state = State.E11;

}

}

else if (\_str[\_ind] == ',')

{

if (ListOfIdentifiers.Contains(c))

{

c = "";

\_identifierLength = 0;

SetError(Error.IdentifierDuplicationError, \_ind);

state = State.Error;

}

else if (ReservedWord(c))

{

SetError(Error.IdentifierIsReservedWordError, \_ind);

state = State.Error;

}

else

{

ListOfIdentifiers.Add(c);

\_identifierCount++;

c = "";

\_identifierLength = 0;

state = State.D14;

}

}

else if (\_str[\_ind] == '\_' || char.IsLetterOrDigit(\_str[\_ind]))

{

c += \_str[\_ind];

\_identifierLength++;

if (\_identifierLength > 8)

{

SetError(Error.IdentifierLengthError, \_ind);

state = State.Error;

}

else state = State.D15;

}

else

{

SetError(Error.ListOfIdentifiersError, \_ind);

state = State.Error;

}

}

break;

case State.E11:

{

if (\_str[\_ind] == ' ') state = State.E11;

else if (\_str[\_ind] == ',') state = State.D14;

else if (\_str[\_ind] == ':') state = State.F1;

else

{

SetError(Error.ColonExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1:

{

if (\_str[\_ind] == ' ') state = State.F1;

else if (\_str[\_ind] == 'R')

{

c += \_str[\_ind];

state = State.F1A;//F1A,F1B,F1C - REAL

}

else if (\_str[\_ind] == 'I')

{

c += \_str[\_ind];

state = State.F1D;//F1D,F1E,F1F,F1G,F1H,F1I - INTEGER

}

else if (\_str[\_ind] == 'C')

{

c += \_str[\_ind];

state = State.F1J;//F1J,F1K,F1I - CHAR

}

else if (\_str[\_ind] == 'S')

{

c += \_str[\_ind];

state = State.F1L;//F1L,F1V,F1W,F1X,F1Y - STRING

}

else if (\_str[\_ind] == 'B')

{

c += \_str[\_ind];

state = State.F1Q;//F1Q,F1Q,F1Q,F1A,F1B,F1C - BOOLEAN //F1Q,F1R,F1P - BYTE

}

else if (\_str[\_ind] == 'D')

{

c += \_str[\_ind];

state = State.F1S;//F1S,F1T,F1U,F1O,F1P - DOUBLE

}

else

{

SetError(Error.TypeExpected, \_ind);

state = State.Error;

}

break;

}

case State.F1A:

{

if (\_str[\_ind] == 'E')

{

c += \_str[\_ind];

state = State.F1B;

}

else

{

SetError(Error.TypeExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1B:

{

if (\_str[\_ind] == 'A')

{

c += \_str[\_ind];

state = State.F1C;

}

else

{

SetError(Error.TypeExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1C:

{

if (\_str[\_ind] == 'L')

{

c += \_str[\_ind];

while (\_identifierCount > 0)

{

ListOfTypes.Add(c);

if (c == "REAL")

{

ListOfSizes.Add("4");

}

\_identifierCount--;

}

c = "";

state = State.G11;

}

else if (\_str[\_ind] == 'N')

{

c += \_str[\_ind];

while (\_identifierCount > 0)

{

ListOfTypes.Add(c);

if (c == "BOOLEAN")

{

ListOfSizes.Add("1");

}

\_identifierCount--;

}

c = "";

state = State.G11;

}

else

{

SetError(Error.TypeExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1D:

{

if (\_str[\_ind] == 'N')

{

c += \_str[\_ind];

state = State.F1E;

}

else

{

SetError(Error.IntegerExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1E:

{

if (\_str[\_ind] == 'T')

{

c += \_str[\_ind];

state = State.F1F;

}

else

{

SetError(Error.IntegerExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1F:

{

if (\_str[\_ind] == 'E')

{

c += \_str[\_ind];

state = State.F1G;

}

else

{

SetError(Error.IntegerExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1G:

{

if (\_str[\_ind] == 'G')

{

c += \_str[\_ind];

state = State.F1H;

}

else

{

SetError(Error.IntegerExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1H:

{

if (\_str[\_ind] == 'E')

{

c += \_str[\_ind];

state = State.F1I;

}

else

{

SetError(Error.IntegerExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1I:

{

if (\_str[\_ind] == 'R')

{

c += \_str[\_ind];

while (\_identifierCount > 0)

{

ListOfTypes.Add(c);

if (c == "INTEGER")

{

ListOfSizes.Add("4");

}

else if (c == "CHAR")

{

ListOfSizes.Add("1");

}

\_identifierCount--;

}

c = "";

state = State.G11;

}

else

{

SetError(Error.TypeExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1J:

{

if (\_str[\_ind] == 'H')

{

c += \_str[\_ind];

state = State.F1K;

}

else

{

SetError(Error.CharExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1K:

{

if (\_str[\_ind] == 'A')

{

c += \_str[\_ind];

state = State.F1I;

}

else

{

SetError(Error.CharExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1L:

{

if (\_str[\_ind] == 'T')

{

c += \_str[\_ind];

state = State.F1V;

}

else

{

SetError(Error.StringExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1O:

{

if (\_str[\_ind] == 'L')

{

c += \_str[\_ind];

state = State.F1P;

}

else

{

SetError(Error.DoubleExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1P:

{

if (\_str[\_ind] == 'E')

{

c += \_str[\_ind];

while (\_identifierCount > 0)

{

ListOfTypes.Add(c);

if (c == "BYTE")

{

ListOfSizes.Add("1");

}

else if (c == "DOUBLE")

{

ListOfSizes.Add("8");

}

\_identifierCount--;

}

c = "";

state = State.G11;

}

else

{

SetError(Error.TypeExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1Q:

{

if (\_str[\_ind] == 'Y')

{

c += \_str[\_ind]; state = State.F1R;

}

else if (\_str[\_ind] == 'O')

{

c += \_str[\_ind];

state = State.F1Q;

}

else if (\_str[\_ind] == 'L')

{

c += \_str[\_ind];

state = State.F1A;

}

else

{

SetError(Error.TypeExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1R:

{

if (\_str[\_ind] == 'T')

{

c += \_str[\_ind];

state = State.F1P;

}

else

{

SetError(Error.ByteExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1S:

{

if (\_str[\_ind] == 'O')

{

c += \_str[\_ind];

state = State.F1T;

}

else

{

SetError(Error.DoubleExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1T:

{

if (\_str[\_ind] == 'U')

{

c += \_str[\_ind];

state = State.F1U;

}

else

{

SetError(Error.DoubleExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1U:

{

if (\_str[\_ind] == 'B')

{

c += \_str[\_ind];

state = State.F1O;

}

else

{

SetError(Error.DoubleExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1V:

{

if (\_str[\_ind] == 'R')

{

c += \_str[\_ind];

state = State.F1W;

}

else

{

SetError(Error.StringExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1W:

{

if (\_str[\_ind] == 'I')

{

c += \_str[\_ind];

state = State.F1X;

}

else

{

SetError(Error.StringExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1X:

{

if (\_str[\_ind] == 'N')

{

c += \_str[\_ind];

state = State.F1Y;

}

else

{

SetError(Error.StringExpected, \_ind);

state = State.Error;

}

}

break;

case State.F1Y:

{

if (\_str[\_ind] == 'G')

{

c += \_str[\_ind];

while (\_identifierCount > 0)

{

ListOfTypes.Add(c);

if (c == "STRING")

{

ListOfSizes.Add("1");

}

\_identifierCount--;

}

c = "";

state = State.G11;

}

else

{

SetError(Error.StringExpected, \_ind);

state = State.Error;

}

}

break;

case State.G11:

{

if (\_str[\_ind] == ' ') state = State.G12;

else if (\_str[\_ind] == ')') state = State.G13;

else if (\_str[\_ind] == ';') state = State.C12;

else

{

SetError(Error.CloseBracketOrSemicolonExpected, \_ind);

state = State.Error;

}

}

break;

case State.G12:

{

if (\_str[\_ind] == ' ')

state = State.G12;

else if (\_str[\_ind] == ')')

state = State.G13;

else if (\_str[\_ind] == ';')

state = State.C12;

else

{

SetError(Error.CloseBracketOrSemicolonExpected, \_ind);

state = State.Error;

}

}

break;

case State.G13:

{

if (\_str[\_ind] == ' ')

state = State.G13;

else if (\_str[\_ind] == ';')

state = State.Final;

else

{

SetError(Error.SemicolonExpected, \_ind);

state = State.Error;

}

}

break;

//FUNCTION

case State.A21:

{

if (\_str[\_ind] == 'U') state = State.A22;

else

{

SetError(Error.FunctionExpected, \_ind);

state = State.Error;

}

}

break;

case State.A22:

{

if (\_str[\_ind] == 'N') state = State.A23;

else

{

SetError(Error.FunctionExpected, \_ind);

state = State.Error;

}

}

break;

case State.A23:

{

if (\_str[\_ind] == 'C') state = State.A24;

else

{

SetError(Error.FunctionExpected, \_ind);

state = State.Error;

}

}

break;

case State.A24:

{

if (\_str[\_ind] == 'T') state = State.A25;

else

{

SetError(Error.FunctionExpected, \_ind);

state = State.Error;

}

}

break;

case State.A25:

{

if (\_str[\_ind] == 'I') state = State.A26;

else

{

SetError(Error.FunctionExpected, \_ind);

state = State.Error;

}

}

break;

case State.A26:

{

if (\_str[\_ind] == 'O') state = State.A27;

else

{

SetError(Error.FunctionExpected, \_ind);

state = State.Error;

}

}

break;

case State.A27:

{

if (\_str[\_ind] == 'N') state = State.A28;

else

{

SetError(Error.FunctionExpected, \_ind);

state = State.Error;

}

}

break;

case State.A28:

{

if (\_str[\_ind] == ' ') state = State.B21;

else

{

SetError(Error.SpaceExpected, \_ind);

state = State.Error;

}

}

break;

case State.B21:

{

if (\_str[\_ind] == ' ') state = State.B21;

else if (\_str[\_ind] == '\_' || char.IsLetter(\_str[\_ind]))

{

c += \_str[\_ind];

\_identifierLength++;

state = State.B22;

}

else

{

SetError(Error.IdentifierFirstSymbolError, \_ind);

state = State.Error;

}

}

break;

case State.B22:

{

if (\_str[\_ind] == '\_' || char.IsLetterOrDigit(\_str[\_ind]))

{

c += \_str[\_ind];

\_identifierLength++;

if (\_identifierLength > 8)

{

SetError(Error.IdentifierLengthError, \_ind);

state = State.Error;

}

else state = State.B22;

}

else if (\_str[\_ind] == ' ')

{

if (ReservedWord(c))

{

SetError(Error.IdentifierIsReservedWordError, \_ind);

state = State.Error;

}

else

{

ListOfIdentifiers.Add(c);

c = "";

\_identifierLength = 0;

state = State.C21;

}

}

else if (\_str[\_ind] == '(')

{

if (ReservedWord(c))

{

SetError(Error.IdentifierIsReservedWordError, \_ind);

state = State.Error;

}

else

{

ListOfIdentifiers.Add(c);

c = "";

\_identifierLength = 0;

state = State.C22;

}

}

else if (\_str[\_ind] == ':')

{

if (ReservedWord(c))

{

SetError(Error.IdentifierIsReservedWordError, \_ind);

state = State.Error;

}

else

{

ListOfIdentifiers.Add(c);

c = "";

\_identifierLength = 0;

state = State.F3;

}

}

else

{

SetError(Error.IdentifierError, \_ind);

state = State.Error;

}

}

break;

case State.C21:

{

if (\_str[\_ind] == ' ') state = State.C21;

else if (\_str[\_ind] == '(') state = State.C22;

else if (\_str[\_ind] == ':') state = State.F3;

else

{

SetError(Error.OpenBracketOrColonExpected, \_ind);

state = State.Error;

}

}

break;

case State.C22:

{

if (\_str[\_ind] == ' ') state = State.C22;

else if (\_str[\_ind] == 'V')

{

c += \_str[\_ind];

\_identifierLength++;

state = State.D21;

}

else if (\_str[\_ind] == '\_' || char.IsLetter(\_str[\_ind]))

{

c += \_str[\_ind];

\_identifierLength++;

state = State.D25;

}

else

{

SetError(Error.IdentifierFirstSymbolError, \_ind);

state = State.Error;

}

}

break;

case State.D21:

{

if (\_str[\_ind] == 'A')

{

c += \_str[\_ind];

\_identifierLength++;

if (\_identifierLength > 8)

{

SetError(Error.IdentifierLengthError, \_ind);

state = State.Error;

}

else state = State.D22;

}

else if (\_str[\_ind] == '\_' || char.IsLetterOrDigit(\_str[\_ind]))

{

c += \_str[\_ind];

\_identifierLength++;

if (\_identifierLength > 8)

{

SetError(Error.IdentifierLengthError, \_ind);

state = State.Error;

}

else state = State.D25;

}

else

{

SetError(Error.IdentifierError, \_ind);

state = State.Error;

}

}

break;

case State.D22:

{

if (\_str[\_ind] == 'R')

{

c += \_str[\_ind];

\_identifierLength++;

if (\_identifierLength > 8)

{

SetError(Error.IdentifierLengthError, \_ind);

state = State.Error;

}

else state = State.D23;

}

else if (\_str[\_ind] == '\_' || char.IsLetterOrDigit(\_str[\_ind]))

{

c += \_str[\_ind];

\_identifierLength++;

if (\_identifierLength > 8)

{

SetError(Error.IdentifierLengthError, \_ind);

state = State.Error;

}

else state = State.D25;

}

else

{

SetError(Error.IdentifierError, \_ind);

state = State.Error;

}

}

break;

case State.D23:

{

if (\_str[\_ind] == ' ') state = State.D24;

else if (\_str[\_ind] == '\_' || char.IsLetterOrDigit(\_str[\_ind])) state = State.D25;

else

{

SetError(Error.IdentifierError, \_ind);

state = State.Error;

}

}

break;

case State.D24:

{

if (\_str[\_ind] == ' ')

{

c = "";

\_identifierLength = 0;

state = State.D24;

}

else if (\_str[\_ind] == '\_' || char.IsLetter(\_str[\_ind]))

{

c += \_str[\_ind];

\_identifierLength++;

if (\_identifierLength > 8)

{

SetError(Error.IdentifierLengthError, \_ind);

state = State.Error;

}

else state = State.D25;

}

else

{

SetError(Error.IdentifierError, \_ind);

state = State.Error;

}

}

break;

case State.D25:

{

if (\_str[\_ind] == ' ')

{

if (ListOfIdentifiers.Contains(c))

{

c = "";

\_identifierLength = 0;

SetError(Error.IdentifierDuplicationError, \_ind);

state = State.Error;

}

else if (ReservedWord(c))

{

SetError(Error.IdentifierIsReservedWordError, \_ind);

state = State.Error;

}

else

{

ListOfIdentifiers.Add(c);

\_identifierCount++;

c = "";

\_identifierLength = 0;

state = State.E21;

}

}

else if (\_str[\_ind] == ',')

{

if (ListOfIdentifiers.Contains(c))

{

c = "";

\_identifierLength = 0;

SetError(Error.IdentifierDuplicationError, \_ind);

state = State.Error;

}

else if (ReservedWord(c))

{

SetError(Error.IdentifierIsReservedWordError, \_ind);

state = State.Error;

}

else

{

ListOfIdentifiers.Add(c);

\_identifierCount++;

c = "";

\_identifierLength = 0;

state = State.D24;

}

}

else if (\_str[\_ind] == '\_' || char.IsLetterOrDigit(\_str[\_ind]))

{

c += \_str[\_ind];

\_identifierLength++;

if (\_identifierLength > 8)

{

SetError(Error.IdentifierLengthError, \_ind);

state = State.Error;

}

else state = State.D25;

}

else

{

SetError(Error.ListOfIdentifiersError, \_ind);

state = State.Error;

}

}

break;

case State.E21:

{

if (\_str[\_ind] == ' ') state = State.E21;

else if (\_str[\_ind] == ',') state = State.D24;

else if (\_str[\_ind] == ':') state = State.F2;

else

{

SetError(Error.ColonExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2:

{

if (\_str[\_ind] == ' ') state = State.F2;

else if (\_str[\_ind] == 'R')

{

c += \_str[\_ind];

state = State.F2A; //F2A,F2B,F2C - REAL

}

else if (\_str[\_ind] == 'I')

{

c += \_str[\_ind];

state = State.F2D; //DEFGHI - INTEGER

}

else if (\_str[\_ind] == 'C')

{

c += \_str[\_ind];

state = State.F2J; //JKI - CHAR

}

else if (\_str[\_ind] == 'S')

{

c += \_str[\_ind];

state = State.F2L; //LVWXY - STRING

}

else if (\_str[\_ind] == 'B')

{

c += \_str[\_ind];

state = State.F2Q; //QQQABC - BOOLEAN //QRP - BYTE

}

else if (\_str[\_ind] == 'D')

{

c += \_str[\_ind];

state = State.F2S; //STUOPR - DOUBLE

}

else

{

SetError(Error.TypeExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2A:

{

if (\_str[\_ind] == 'E')

{

c += \_str[\_ind];

state = State.F2B;

}

else

{

SetError(Error.TypeExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2B:

{

if (\_str[\_ind] == 'A')

{

c += \_str[\_ind];

state = State.F2C;

}

else

{

SetError(Error.TypeExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2C:

{

if (\_str[\_ind] == 'L')

{

c += \_str[\_ind];

while(\_identifierCount > 0)

{

ListOfTypes.Add(c);

if (c == "REAL")

{

ListOfSizes.Add("4");

}

\_identifierCount--;

}

c = "";

state = State.G21;

}

else if (\_str[\_ind] == 'N')

{

c += \_str[\_ind];

while (\_identifierCount > 0)

{

ListOfTypes.Add(c);

if (c == "BOOLEAN")

{

ListOfSizes.Add("1");

}

\_identifierCount--;

}

c = "";

state = State.G21;

}

else

{

SetError(Error.TypeExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2D:

{

if (\_str[\_ind] == 'N')

{

c += \_str[\_ind];

state = State.F2E;

}

else

{

SetError(Error.IntegerExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2E:

{

if (\_str[\_ind] == 'T')

{

c += \_str[\_ind];

state = State.F2F;

}

else

{

SetError(Error.IntegerExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2F:

{

if (\_str[\_ind] == 'E')

{

c += \_str[\_ind];

state = State.F2G;

}

else

{

SetError(Error.IntegerExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2G:

{

if (\_str[\_ind] == 'G')

{

c += \_str[\_ind];

state = State.F2H;

}

else

{

SetError(Error.IntegerExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2H:

{

if (\_str[\_ind] == 'E')

{

c += \_str[\_ind];

state = State.F2I;

}

else

{

SetError(Error.IntegerExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2I:

{

if (\_str[\_ind] == 'R')

{

c += \_str[\_ind];

while (\_identifierCount > 0)

{

ListOfTypes.Add(c);

if (c == "INTEGER")

{

ListOfSizes.Add("4");

}

else if (c == "CHAR")

{

ListOfSizes.Add("1");

}

\_identifierCount--;

}

c = "";

state = State.G21;

}

else

{

SetError(Error.TypeExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2J:

{

if (\_str[\_ind] == 'H')

{

c += \_str[\_ind];

state = State.F2K;

}

else

{

SetError(Error.CharExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2K:

{

if (\_str[\_ind] == 'A')

{

c += \_str[\_ind];

state = State.F2I;

}

else

{

SetError(Error.CharExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2L:

{

if (\_str[\_ind] == 'T')

{

c += \_str[\_ind];

state = State.F2V;

}

else

{

SetError(Error.StringExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2O:

{

if (\_str[\_ind] == 'L')

{

c += \_str[\_ind];

state = State.F2P;

}

else

{

SetError(Error.DoubleExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2P:

{

if (\_str[\_ind] == 'E')

{

c += \_str[\_ind];

while (\_identifierCount > 0)

{

ListOfTypes.Add(c);

if (c == "BYTE")

{

ListOfSizes.Add("1");

}

else if (c == "DOUBLE")

{

ListOfSizes.Add("8");

}

\_identifierCount--;

}

c = "";

state = State.G21;

}

else

{

SetError(Error.TypeExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2Q:

{

if (\_str[\_ind] == 'Y')

{

c += \_str[\_ind];

state = State.F2R;

}

else if (\_str[\_ind] == 'O')

{

c += \_str[\_ind];

state = State.F2Q;

}

else if (\_str[\_ind] == 'L')

{

c += \_str[\_ind];

state = State.F2A;

}

else

{

SetError(Error.TypeExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2R:

{

if (\_str[\_ind] == 'T')

{

c += \_str[\_ind];

state = State.F2P;

}

else

{

SetError(Error.ByteExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2S:

{

if (\_str[\_ind] == 'O')

{

c += \_str[\_ind];

state = State.F2T;

}

else

{

SetError(Error.DoubleExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2T:

{

if (\_str[\_ind] == 'U')

{

c += \_str[\_ind];

state = State.F2U;

}

else

{

SetError(Error.DoubleExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2U:

{

if (\_str[\_ind] == 'B')

{

c += \_str[\_ind];

state = State.F2O;

}

else

{

SetError(Error.DoubleExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2V:

{

if (\_str[\_ind] == 'R')

{

c += \_str[\_ind];

state = State.F2W;

}

else

{

SetError(Error.StringExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2W:

{

if (\_str[\_ind] == 'I')

{

c += \_str[\_ind];

state = State.F2X;

}

else

{

SetError(Error.StringExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2X:

{

if (\_str[\_ind] == 'N')

{

c += \_str[\_ind];

state = State.F2Y;

}

else

{

SetError(Error.StringExpected, \_ind);

state = State.Error;

}

}

break;

case State.F2Y:

{

if (\_str[\_ind] == 'G')

{

c += \_str[\_ind];

while (\_identifierCount > 0)

{

ListOfTypes.Add(c);

if (c == "STRING")

{

ListOfSizes.Add("1");

}

\_identifierCount--;

}

c = "";

state = State.G21;

}

else

{

SetError(Error.StringExpected, \_ind);

state = State.Error;

}

}

break;

case State.G21:

{

if (\_str[\_ind] == ' ') state = State.G22;

else if (\_str[\_ind] == ')') state = State.G23;

else if (\_str[\_ind] == ';') state = State.C22;

else

{

SetError(Error.CloseBracketOrSemicolonExpected, \_ind);

state = State.Error;

}

}

break;

case State.G22:

{

if (\_str[\_ind] == ' ')

state = State.G22;

else if (\_str[\_ind] == ')')

state = State.G23;

else if (\_str[\_ind] == ';')

state = State.C22;

else

{

SetError(Error.CloseBracketOrSemicolonExpected, \_ind);

state = State.Error;

}

}

break;

case State.G23:

{

if (\_str[\_ind] == ' ')

state = State.G23;

else if (\_str[\_ind] == ':')

state = State.F3;

else

{

SetError(Error.ColonExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3:

{

if (\_str[\_ind] == ' ') state = State.F3;

else if (\_str[\_ind] == 'R')

{

c += \_str[\_ind];

state = State.F3A;//F3A,F3B,F3C - REAL

}

else if (\_str[\_ind] == 'I')

{

c += \_str[\_ind];

state = State.F3D;//F3D,F3E,F3F,F3G,F3H,F3I - INTEGER

}

else if (\_str[\_ind] == 'C')

{

c += \_str[\_ind];

state = State.F3J;//F3J,F3K,F3I - CHAR

}

else if (\_str[\_ind] == 'S')

{

c += \_str[\_ind];

state = State.F3L;//F3L,F3V,F3W,F3X,F3Y - STRING

}

else if (\_str[\_ind] == 'B')

{

c += \_str[\_ind];

state = State.F3Q;//F3Q,F3Q,F3Q,F3A,F3B,F3C - BOOLEAN //F3Q,F3R,F3P - BYTE

}

else if (\_str[\_ind] == 'D')

{

c += \_str[\_ind];

state = State.F3S;//F3S,F3T,F3U,F3O,F3P - DOUBLE

}

else

{

SetError(Error.TypeExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3A:

{

if (\_str[\_ind] == 'E')

{

c += \_str[\_ind];

state = State.F3B;

}

else

{

SetError(Error.TypeExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3B:

{

if (\_str[\_ind] == 'A')

{

c += \_str[\_ind];

state = State.F3C;

}

else

{

SetError(Error.TypeExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3C:

{

if (\_str[\_ind] == 'L')

{

c += \_str[\_ind];

ListOfTypes.Insert(0, c);

if (c == "REAL")

{

ListOfSizes.Insert(0, "4");

}

c = "";

state = State.G31;

}

else if (\_str[\_ind] == 'N')

{

c += \_str[\_ind];

ListOfTypes.Insert(0, c);

if (c == "BOOLEAN")

{

ListOfSizes.Insert(0, "1");

}

c = "";

state = State.G31;

}

else

{

SetError(Error.TypeExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3D:

{

if (\_str[\_ind] == 'N')

{

c += \_str[\_ind];

state = State.F3E;

}

else

{

SetError(Error.IntegerExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3E:

{

if (\_str[\_ind] == 'T')

{

c += \_str[\_ind];

state = State.F3F;

}

else

{

SetError(Error.IntegerExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3F:

{

if (\_str[\_ind] == 'E')

{

c += \_str[\_ind];

state = State.F3G;

}

else

{

SetError(Error.IntegerExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3G:

{

if (\_str[\_ind] == 'G')

{

c += \_str[\_ind];

state = State.F3H;

}

else

{

SetError(Error.IntegerExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3H:

{

if (\_str[\_ind] == 'E')

{

c += \_str[\_ind];

state = State.F3I;

}

else

{

SetError(Error.IntegerExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3I:

{

if (\_str[\_ind] == 'R')

{

c += \_str[\_ind];

ListOfTypes.Insert(0, c);

if (c == "INTEGER")

{

ListOfSizes.Insert(0, "4");

}

else if (c == "CHAR")

{

ListOfSizes.Insert(0, "1");

}

c = "";

state = State.G31;

}

else

{

SetError(Error.TypeExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3J:

{

if (\_str[\_ind] == 'H')

{

c += \_str[\_ind];

state = State.F3K;

}

else

{

SetError(Error.CharExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3K:

{

if (\_str[\_ind] == 'A')

{

c += \_str[\_ind];

state = State.F3I;

}

else

{

SetError(Error.CharExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3L:

{

if (\_str[\_ind] == 'T')

{

c += \_str[\_ind];

state = State.F3V;

}

else

{

SetError(Error.StringExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3O:

{

if (\_str[\_ind] == 'L')

{

c += \_str[\_ind];

state = State.F3P;

}

else

{

SetError(Error.DoubleExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3P:

{

if (\_str[\_ind] == 'E')

{

c += \_str[\_ind];

ListOfTypes.Insert(0, c);

if (c == "BYTE")

{

ListOfSizes.Insert(0, "1");

}

else if (c == "DOUBLE")

{

ListOfSizes.Insert(0, "8");

}

c = "";

state = State.G31;

}

else

{

SetError(Error.TypeExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3Q:

{

if (\_str[\_ind] == 'Y')

{

c += \_str[\_ind];

state = State.F3R;

}

else if (\_str[\_ind] == 'O')

{

c += \_str[\_ind];

state = State.F3Q;

}

else if (\_str[\_ind] == 'L')

{

c += \_str[\_ind];

state = State.F3A;

}

else

{

SetError(Error.TypeExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3R:

{

if (\_str[\_ind] == 'T')

{

c += \_str[\_ind];

state = State.F3P;

}

else

{

SetError(Error.ByteExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3S:

{

if (\_str[\_ind] == 'O')

{

c += \_str[\_ind];

state = State.F3T;

}

else

{

SetError(Error.DoubleExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3T:

{

if (\_str[\_ind] == 'U')

{

c += \_str[\_ind];

state = State.F3U;

}

else

{

SetError(Error.DoubleExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3U:

{

if (\_str[\_ind] == 'B')

{

c += \_str[\_ind];

state = State.F3O;

}

else

{

SetError(Error.DoubleExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3V:

{

if (\_str[\_ind] == 'R')

{

c += \_str[\_ind];

state = State.F3W;

}

else

{

SetError(Error.StringExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3W:

{

if (\_str[\_ind] == 'I')

{

c += \_str[\_ind];

state = State.F3X;

}

else

{

SetError(Error.StringExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3X:

{

if (\_str[\_ind] == 'N')

{

c += \_str[\_ind];

state = State.F3Y;

}

else

{

SetError(Error.StringExpected, \_ind);

state = State.Error;

}

}

break;

case State.F3Y:

{

if (\_str[\_ind] == 'G')

{

c += \_str[\_ind];

if (c == "STRING")

{

ListOfSizes.Insert(0, "1");

}

c = "";

state = State.G31;

}

else

{

SetError(Error.StringExpected, \_ind);

state = State.Error;

}

}

break;

case State.G31:

{

if (\_str[\_ind] == ' ') state = State.G31;

else if (\_str[\_ind] == ';') state = State.Final;

else

{

SetError(Error.SemicolonExpected, \_ind);

state = State.Error;

}

}

break;

default:

{

SetError(Error.UnknownError, \_ind);

state = State.Error;

}

break;

}

}

\_ind++;

}

if (state == State.Error)

{

tmpPos = \_ind;

return false;

}

return true;

}

}

}

*Класс пользовательского интерфейса*

using System;

using System.Windows.Forms;

namespace Analyzer

{

public partial class MainForm : Form

{

private const string NewLineString = "\n";

public MainForm()

{

InitializeComponent();

}

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

{

richTextBoxIdentifiers.Clear();

richTextBoxTypes.Clear();

richTextBoxSizes.Clear();

richTextBoxError.Clear();

var stringInput = textBoxInput.Text.ToUpper();

var test = Analyzer.Test(stringInput);

foreach (var item in Analyzer.ListOfIdentifiers)

{

richTextBoxIdentifiers.Text += item;

richTextBoxIdentifiers.Text += NewLineString;

}

Analyzer.ListOfIdentifiers.Clear();

foreach (var item in Analyzer.ListOfTypes)

{

richTextBoxTypes.Text += item;

richTextBoxTypes.Text += NewLineString;

}

Analyzer.ListOfTypes.Clear();

foreach (var item in Analyzer.ListOfSizes)

{

richTextBoxSizes.Text += item;

richTextBoxSizes.Text += NewLineString;

}

Analyzer.ListOfSizes.Clear();

if (test.ErrorPosition == 0)

richTextBoxError.Text = test.ErrorMessage;

else

{

richTextBoxError.Text = (test.ErrorPosition + 1).ToString() + NewLineString + test.ErrorMessage;

textBoxInput.Focus();

textBoxInput.Select(test.ErrorPosition, 0);

textBoxInput.ScrollToCaret();

}

}

}

}

**Результат выполнения программы:**

