****

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА  
(САМАРСКИЙ УНИВЕРСИТЕТ)»**

Институт информатики и кибернетики

Кафедра информационных систем и технологий

**ОТЧЁТ**

**по лабораторной работе**

Синтаксический анализ автоматных языков  
по дисциплине «Теория автоматов и формальных языков»

Выполнил Боряков Никита Сергеевич, группа 6202-020302D

Проверила Чигарина Елена Ивановна

Самара  
2022

ЗАДАНИЕ

Вариант 3

Написать программу синтаксического анализа автоматного языка оператора описания переменных языка Turbo Pascal, имеющего вид

**VAR** <список описаний>

<список описаний> :: = <описание>**;**|<список описаний><описание>**;**

<описание> :: = <список переменных>**:**<тип>

<список переменных> :: = <идентификатор>**,**<список переменных>|<идентификатор>

<тип> :: = <простой тип>|**ARRAY[**<диапазоны>**]OF**<простой тип>

<простой тип> :: = **BYTE**|**WORD**|**INTEGER**|**REAL**|**CHAR**|**DOUBLE**

<диапазоны> :: = <диапазон>|<диапазон>**,**<диапазон>

<диапазон> :: = <константа целая 1>**..**<константа целая 2>

<идентификатор> - идентификатор языка Turbo Pascal, начинается с буквы или подчеркивания, включает последовательность букв и цифр, не допускает пробелы и специальные символы, ввести ограничение на длину идентификатора, не более 8 символов, не является ключевым словом (VAR, ARRAY, OF, BYTE, WORD, INTEGER, REAL, CHAR, DOUBLE);

<константа целая> - целое число, возможно со знаком, в диапазоне  -32768 ÷ +32767. Проверять выполнение соотношения константа 1 меньше константы 2 в записи диапазона..

Семантика:

Сформировать и вывести таблицу имен переменных с указанием типа и объема памяти, требуемой для переменной. Учесть ограничения на идентификаторы и константы.

Сообщать об ошибках при анализе цепочек языка, указывая курсором позицию ошибки в цепочки и содержание ошибки.

Примеры правильных цепочек:

VAR  Aa,  B  :  REAL;  C  :  ARRAY  [0..79]  OF  CHAR;

VAR  Abc  :  ARRAY  [-25..30,10..15]  OF  INTEGER;

VAR  A,  K,  \_B\_M  :  BYTE;  C  :  WORD;  D17,  E\_7  :  CHAR;

ГРАФЫ СОСТОЯНИЙ

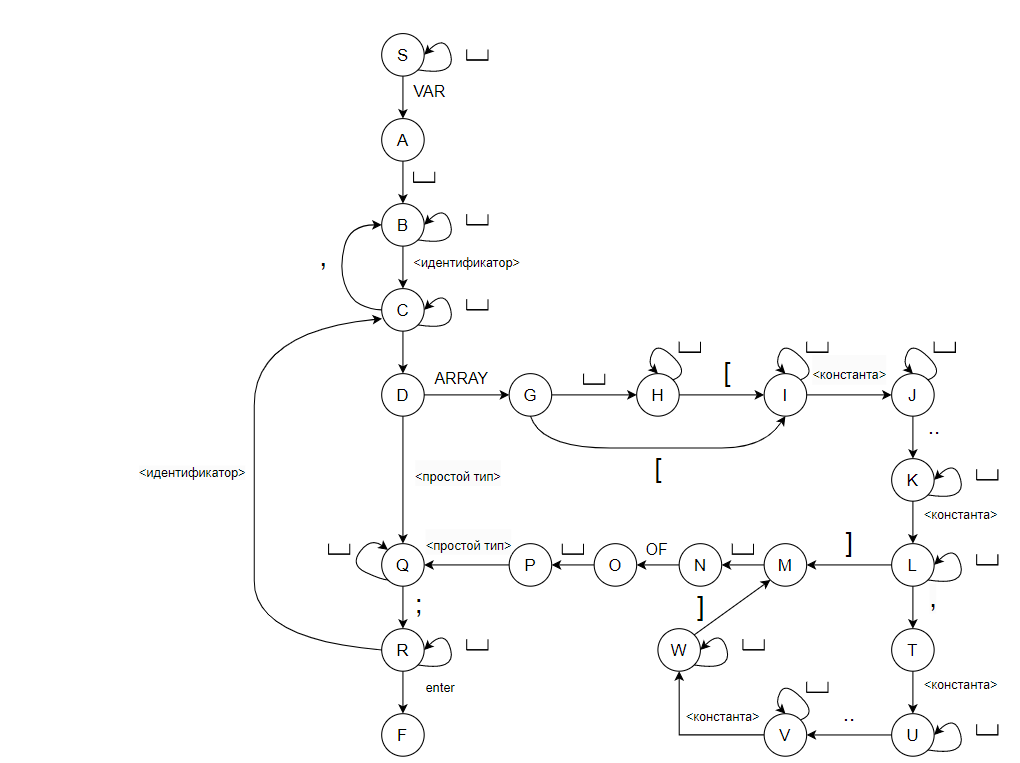


Рисунок 1 – Граф состояний анализатора

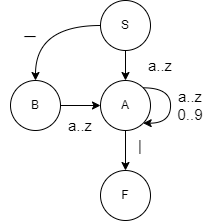


Рисунок 2 – Граф состояний анализатора идентификатора

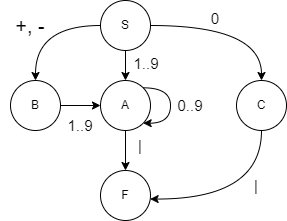


Рисунок 3 – Граф состояний анализатора константы

Изображение выглядит как вычерчивание линий

Автоматически созданное описание

Рисунок 4 – Граф состояний анализатора простого типа

ИНТЕРФЕЙС АНАЛИЗАТОРА

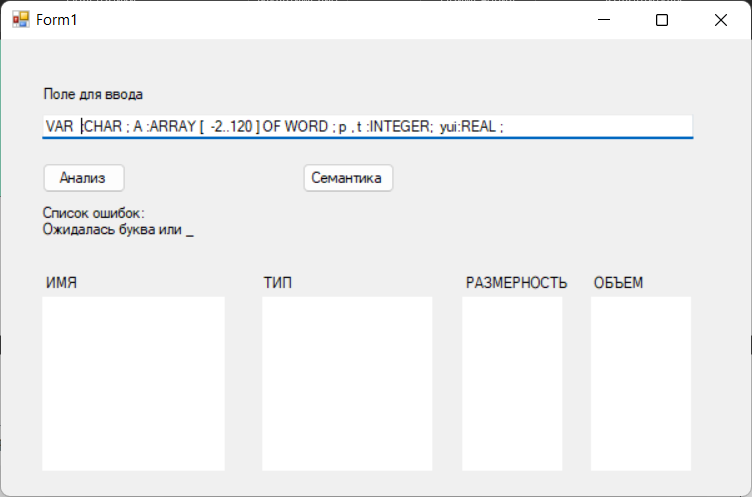


Рисунок 5 – Пример №1 синтаксическая ошибка

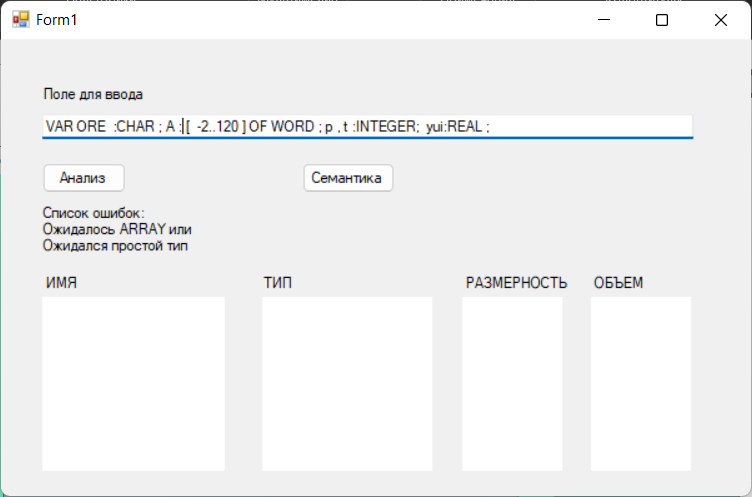


Рисунок 6 – Пример №2 синтаксическая ошибка

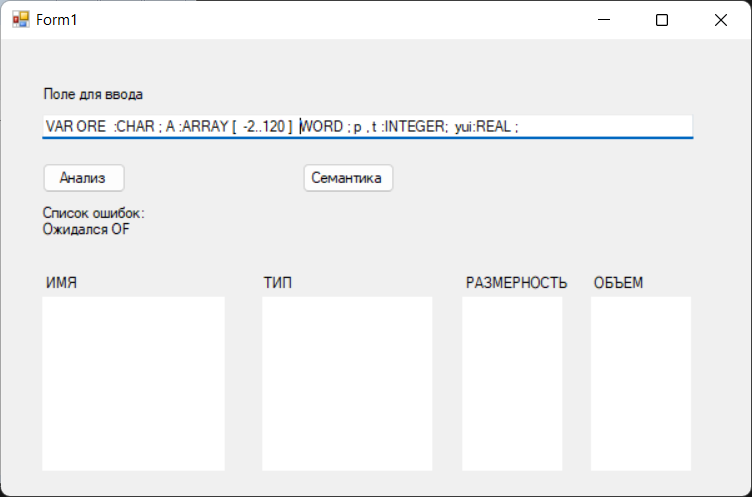


Рисунок 7 – Пример №3 синтаксическая ошибка

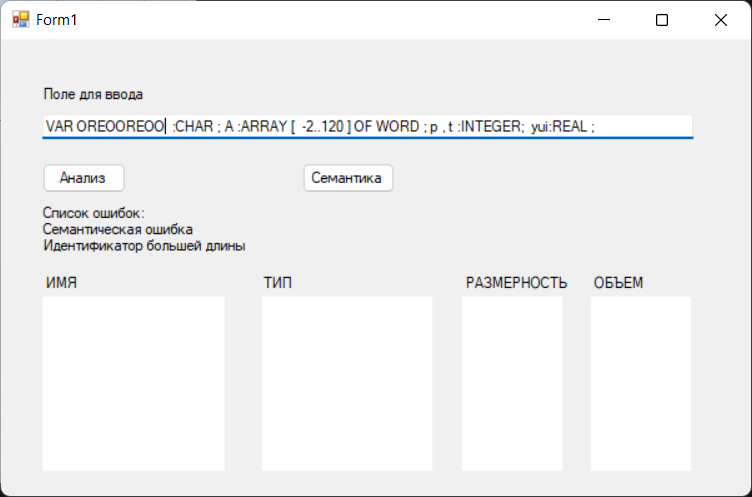


Рисунок 8 – Пример №1 семантическая ошибка

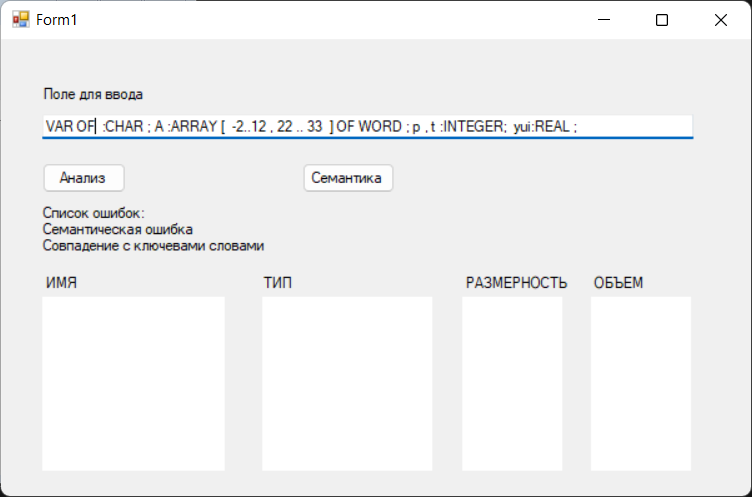


Рисунок 9 – Пример №2 семантическая ошибка

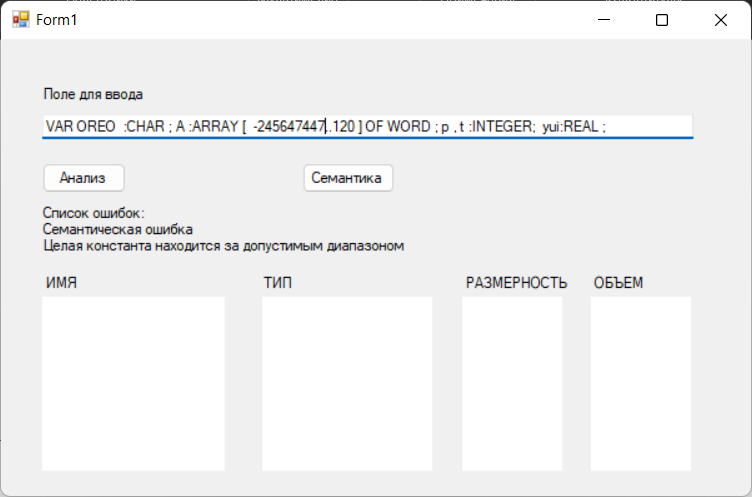


Рисунок 10 – Пример №3 семантическая ошибка

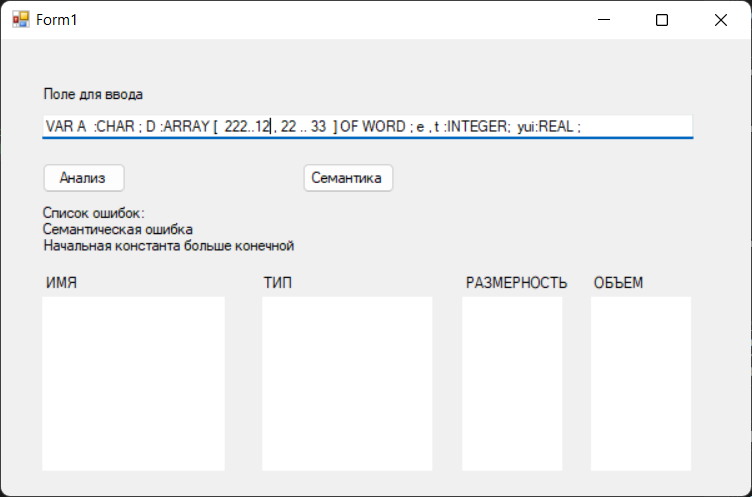


Рисунок 11 – Пример №4 семантическая ошибка

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 12 – Пример №1 вывод семантики

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 13 – Пример №2 вывод семантики

ЛИСТИНГ ОСНОВНЫХ МОДУЛЕЙ АНАЛИЗАТОРА

1) Класс «Анализатор состояний»:

Поля класса:

* cписок констант;
* список идентификаторов;
* список состояний;
* текущее состояние анализатора;
* исходная строка;
* индекс текущего анализируемого символа.

Методы:

* синтаксический анализ введённой строки;
* вывод сообщения об ошибке;
* проверка идентификатора на повторы;
* проверка на зарезервированные слова;
* проверка длины идентификатора;
* проверка констант на диапазон;
* вывод семантики.

2) Класс «Простой тип»

Поля класса:

* текущее состояние анализатора;
* исходная строка;
* список констант;
* индекс текущего анализируемого символа.

Методы:

* получение значения;
* получение состояния;
* получение ошибки.

3) Класс «Константа»

Поля класса:

* текущее состояние анализатора;
* исходная строка;
* индекс текущего анализируемого символа.

Методы:

* получение значения;
* получение состояния;
* получение ошибки.

4) Класс «Идентификатор»:

Поля класса:

* текущее состояние анализатора;
* исходная строка;
* индекс текущего анализируемого символа.

Методы:

* получение значения;
* получение состояния;
* получение ошибки.

КОД ПРОГРАММЫ

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Boriakov\_labwork.Analyzers

{

public enum StateMain { S, A, B, C, D, G, H, I, J, K, L, M, N, O, P, Q, R, T, U, V, W, F, E }

internal class Mainanalyzer : Analyzerfields

{

public StateMain CurrentState { get; set; }

public int Index { get; set; }

public List<string> Table { get; set; }

public int Number\_names { get; set; }

public List<string> Name { get; set; }

public List<string> Type { get; set; }

public List<string> Dimension { get; set; }

public List<string> Volume { get; set; }

public Mainanalyzer(string str)

: base(str)

{

CurrentState = StateMain.S;

Index = 0;

}

public void MainAnalyze(ref string errorMessage)

{

string text = "";

string table\_text = "";

int mnogitel = 1;

int begunok = 0;

Table = new List<string>();

Name = new List<string>();

Type = new List<string>();

Dimension = new List<string>();

Volume = new List<string>();

List<string> wortList = new List<string>();

List<int> countList = new List<int>();

List<string> list = new List<string>()

{

"var",

"word",

"integer",

"real",

"char",

"double",

"array",

"..",

"of",

"byte"

};

string s1 = "var";

string s2 = "word";

string s3 = "integer";

string s4 = "real";

string s5 = "char";

string s6 = "double";

string s7 = "array";

string s8 = "..";

string s9 = "of";

string s10 = "byte";

while (CurrentState != StateMain.F && CurrentState != StateMain.E)

{

symbol = str[Index];

Index++;

switch (CurrentState)

{

// S

case StateMain.S:

if (symbol == 'v' && str[Index] == 'a' && str[Index + 1] == 'r')

{

Index = Index + 2;

CurrentState = StateMain.A;

}

else if (symbol == 'V' && str[Index] == 'A' && str[Index+1] == 'R')

{

Index = Index + 2;

CurrentState = StateMain.A;

}

// петля

else if (Char.IsWhiteSpace(symbol) == true)

{

CurrentState = StateMain.S;

}

else

{

CurrentState = StateMain.E;

errorMessage = "Ожидается VAR";

}

text = "";

break;

// A

case StateMain.A:

if (Char.IsWhiteSpace(symbol) == true)

{

CurrentState = StateMain.B;

}

else

{

CurrentState = StateMain.E;

errorMessage = "Ожидается пробел";

}

break;

// B

case StateMain.B:

string t\_errorMessage\_B = "";

// петля

if (Char.IsWhiteSpace(symbol) == true)

{

CurrentState = StateMain.B;

}

// индетификатор

else

{

bool key\_B = true;

// вырез идентификатора

while (key\_B == true)

{

if (symbol == ' ' || symbol == \_terminalSymbol || symbol == ':' || symbol == ',' )

{

break;

}

text += symbol;

Index++;

symbol = str[Index-1];

}

//

// проверка на дублирование

bool key\_E = true;

if (wortList.Contains(text))

{

errorMessage = "Семантическая ошибка\n" +

"Дублирование имен\n";

CurrentState = StateMain.E;

key\_E = false;

}

wortList.Add(text);

//

// заносим в LIST

table\_text += text + " ";

//

// заносим в NAME

Name.Add(text);

//

// проверка на совпадение и индетификатор

text = text.ToLower();

for (int i = 0; i < list.Count; i++)

{

if (string.Compare(text, list[i]) == 0)

{

errorMessage = "Семантическая ошибка\n" + "Совпадение с ключевами словами";

CurrentState = StateMain.E;

key\_E = false;

Index--;

break;

}

}

if (text.Length > 8)

{

errorMessage = "Семантическая ошибка\n" + "Идентификатор большей длины";

CurrentState = StateMain.E;

key\_E = false;

Index--;

}

//

if (key\_E)

{

Index--;

text += \_terminalSymbol;

Identifieranalyzer identifieranalyzer = new Identifieranalyzer(text);

identifieranalyzer.Analyze(ref t\_errorMessage\_B);

if (identifieranalyzer.CurrentState != State1.E)

{

CurrentState = StateMain.C;

begunok = 1;

}

else

{

errorMessage = t\_errorMessage\_B;

CurrentState = StateMain.E;

}

}

text = "";

}

break;

// C

case StateMain.C:

if (symbol == ':')

{

CurrentState = StateMain.D;

}

// петля

else if (Char.IsWhiteSpace(symbol) == true)

{

CurrentState = StateMain.C;

}

else if (symbol == ',')

{

// счетчик

Number\_names++;

//

CurrentState = StateMain.B;

}

else

{

CurrentState = StateMain.E;

errorMessage = "Ожидалось : или ,";

}

break;

// D

case StateMain.D:

string t\_errorMessage\_D = "";

// ARRAY

if (symbol == 'A' && str[Index] == 'R' && str[Index + 1] == 'R' && str[Index + 2] == 'A' && str[Index + 3] == 'Y')

{

Index = Index + 4;

// LABEL

table\_text += " " + "ARRAY ";

//

begunok = 2;

CurrentState = StateMain.G;

}

//

else if (symbol == 'a' && str[Index] == 'r' && str[Index + 1] == 'r' && str[Index + 2] == 'a' && str[Index + 3] == 'y')

{

Index = Index + 4;

// LABEL

table\_text += " " + "ARRAY ";

//

begunok = 2;

CurrentState = StateMain.G;

}

//

else

{

bool key\_D = true;

// вырез простого типа

while (key\_D == true)

{

if (symbol == ' ' || symbol == \_terminalSymbol || symbol == ';')

{

break;

}

text += symbol;

Index++;

symbol = str[Index - 1];

}

//

Index--;

// множитель

switch (text)

{

case ("BYTE"):

mnogitel = 1;

break;

case ("WORD"):

mnogitel = 2;

break;

case ("INTEGER"):

mnogitel = 4;

break;

case ("CHAR"):

mnogitel = 1;

break;

case ("REAL"):

mnogitel = 8;

break;

default:

break;

}

// LABEL

table\_text += " " + text + " " + mnogitel;

//

for (int i = 0; i <= Number\_names; i++)

{

Type.Add(text);

Dimension.Add("---");

Volume.Add(mnogitel.ToString());

}

// простой тип

text += \_terminalSymbol;

Simpletypeanalyzer simpletypeanalyzer\_D = new Simpletypeanalyzer(text);

simpletypeanalyzer\_D.Analyze(ref t\_errorMessage\_D);

if (simpletypeanalyzer\_D.CurrentState != StateSimple.E)

{

CurrentState = StateMain.Q;

}

else

{

CurrentState = StateMain.E;

errorMessage = "Ожидалось ARRAY или\n" + t\_errorMessage\_D;

}

//

Number\_names = 0;

text = "";

}

break;

// G

case StateMain.G:

if (symbol == '[')

{

CurrentState = StateMain.I;

}

else if (Char.IsWhiteSpace(symbol) == true)

{

CurrentState = StateMain.H;

}

else

{

CurrentState = StateMain.E;

errorMessage = "Ожидалcя пробел или [";

}

break;

// H

case StateMain.H:

if (symbol == '[')

{

CurrentState = StateMain.I;

}

// петля

else if (Char.IsWhiteSpace(symbol) == true)

{

CurrentState = StateMain.H;

}

else

{

CurrentState = StateMain.E;

errorMessage = "Ожидалcя пробел или [";

}

break;

// I

case StateMain.I:

// петля

if (Char.IsWhiteSpace(symbol) == true)

{

CurrentState = StateMain.I;

}

//

else

{

string t\_errorMessage\_I = "";

bool key\_I = true;

// вырез константы

while (key\_I == true)

{

if (symbol == ' ' || symbol == \_terminalSymbol || symbol == ']' || symbol == '.')

{

break;

}

text += symbol;

Index++;

symbol = str[Index - 1];

}

//

Index--;

string text\_copy\_I = text;

// константа 1

text += \_terminalSymbol;

Constantanalyzer constantanalyzer = new Constantanalyzer(text);

constantanalyzer.Analyze(ref t\_errorMessage\_I);

if (constantanalyzer.CurrentState != State.E)

{

CurrentState = StateMain.J;

int r1 = text\_copy\_I.IndexOf('+');

// заносим константу в список для дальнейших проверок и операций

if (r1 == -1)

{

countList.Add(int.Parse(text\_copy\_I));

}

else

{

text\_copy\_I.Remove(0);

countList.Add(int.Parse(text\_copy\_I));

}

// проверка на диапазоном

if (countList[countList.Count - 1] < -32768 || countList[countList.Count - 1] > 32767)

{

CurrentState = StateMain.E;

errorMessage = "Семантическая ошибка\n" + "Целая константа находится за допустимым диапазоном";

text\_copy\_I = "0";

}

//

}

else

{

CurrentState = StateMain.E;

errorMessage = t\_errorMessage\_I;

}

//

text = "";

}

break;

// J

case StateMain.J:

if (symbol == '.' && str[Index] == '.')

{

Index = Index + 1;

CurrentState = StateMain.K;

}

else if (Char.IsWhiteSpace(symbol) == true)

{

CurrentState = StateMain.J;

}

else

{

CurrentState = StateMain.E;

errorMessage = "Ожидалось ..";

}

text = "";

break;

// K

case StateMain.K:

// петля

if (Char.IsWhiteSpace(symbol) == true)

{

CurrentState = StateMain.K;

}

else

{

// константа 2

string t\_errorMessage\_K = "";

bool key\_K = true;

// вырез константы

while (key\_K == true)

{

if (symbol == ' ' || symbol == \_terminalSymbol || symbol == ']' || symbol == '.' || symbol == ',')

{

break;

}

text += symbol;

Index++;

symbol = str[Index - 1];

}

//

Index--;

string text\_copy\_K = text;

text += \_terminalSymbol;

Constantanalyzer constantanalyzer = new Constantanalyzer(text);

constantanalyzer.Analyze(ref t\_errorMessage\_K);

if (constantanalyzer.CurrentState != State.E)

{

CurrentState = StateMain.L;

// проверки констант

int r1 = text\_copy\_K.IndexOf('+');

if (r1 == -1)

{

countList.Add(int.Parse(text\_copy\_K));

}

else

{

text\_copy\_K.Remove(0);

countList.Add(int.Parse(text\_copy\_K));

}

if (countList[countList.Count - 1] < -32768 || countList[countList.Count - 1] > 32767)

{

CurrentState = StateMain.E;

errorMessage = "Семантическая ошибка\n" + "Целые константы находятся за допустимым диапазоном";

}

if (countList[countList.Count - 2] > countList[countList.Count - 1])

{

CurrentState = StateMain.E;

errorMessage = "Семантическая ошибка\n" + "Начальная константа больше конечной";

}

//

}

else

{

CurrentState = StateMain.E;

errorMessage = t\_errorMessage\_K;

}

text = "";

}

break;

// L --> U

case StateMain.L:

if (symbol == ']')

{

CurrentState = StateMain.M;

}

// петля

else if(Char.IsWhiteSpace(symbol) == true)

{

CurrentState = StateMain.L;

}

else if (symbol == ',')

{

CurrentState = StateMain.T;

}

else

{

CurrentState = StateMain.E;

errorMessage = "Ожидалcя пробел или ] или ,";

}

break;

// M

case StateMain.M:

if (Char.IsWhiteSpace(symbol) == true)

{

CurrentState = StateMain.N;

}

else

{

CurrentState = StateMain.E;

errorMessage = "Ожидался пробел";

}

break;

// N

case StateMain.N:

if (symbol == 'O' && str[Index] == 'F')

{

Index = Index + 1;

CurrentState = StateMain.O;

}

else if (symbol == 'o' && str[Index] == 'f')

{

Index = Index + 1;

CurrentState = StateMain.O;

}

else

{

CurrentState = StateMain.E;

errorMessage = "Ожидался OF";

}

text = "";

break;

// O

case StateMain.O:

if (Char.IsWhiteSpace(symbol) == true)

{

CurrentState = StateMain.P;

}

else

{

CurrentState = StateMain.E;

errorMessage = "Ожидался пробел";

}

break;

// P

case StateMain.P:

string t\_errorMessage\_P = "";

bool key = true;

// вырез простого типа

while (key == true)

{

if (symbol == ' ' || symbol == \_terminalSymbol || symbol == ';')

{

break;

}

text += symbol;

Index++;

symbol = str[Index - 1];

}

//

Index--;

// заносим в LIST

table\_text += text + " ";

//

for (int i = 0; i <= Number\_names; i++)

{

}

// множитель

switch (text)

{

case ("BYTE"):

mnogitel = 1;

break;

case ("WORD"):

mnogitel = 2;

break;

case ("INTEGER"):

mnogitel = 4;

break;

case ("CHAR"):

mnogitel = 1;

break;

case ("REAL"):

mnogitel = 8;

break;

default:

break;

}

//

// рассчёт памяти

int summa = 0;

if (begunok == 10)

{

summa = ((countList[countList.Count - 1] - countList[countList.Count - 2]) \* (countList[countList.Count - 3] - countList[countList.Count - 4]) \* mnogitel);

table\_text += summa.ToString();

}

else

{

begunok = 4;

summa = (countList[countList.Count - 1] - countList[countList.Count - 2]) \* mnogitel;

table\_text += summa.ToString();

}

//

// занос в LABEL

Table.Add(table\_text);

//

for (int i = 0; i <= Number\_names; i++)

{

Type.Add("ARRAY OF " + text);

if (begunok == 10)

{

Dimension.Add("2");

}

else

{

Dimension.Add("1");

}

Volume.Add(summa.ToString());

}

// простой тип

text += \_terminalSymbol;

Simpletypeanalyzer simpletypeanalyzer = new Simpletypeanalyzer(text);

simpletypeanalyzer.Analyze(ref t\_errorMessage\_P);

if (simpletypeanalyzer.CurrentState != StateSimple.E)

{

CurrentState = StateMain.Q;

}

else

{

CurrentState = StateMain.E;

errorMessage = t\_errorMessage\_P;

}

//

Number\_names = 0;

table\_text = "";

text = "";

break;

// Q

case StateMain.Q:

if (symbol == ';')

{

CurrentState = StateMain.R;

if (begunok == 1)

{

// занос в LABEL

Table.Add(table\_text);

//

}

}

// петля

else if (Char.IsWhiteSpace(symbol) == true)

{

CurrentState = StateMain.Q;

}

else

{

CurrentState = StateMain.E;

errorMessage = "Ожидалcя пробел или ;";

}

table\_text = "";

break;

// R

case StateMain.R:

string t\_errorMessage\_R = "";

if (symbol == \_terminalSymbol)

{

CurrentState = StateMain.F;

}

// петля

else if (Char.IsWhiteSpace(symbol) == true)

{

CurrentState = StateMain.R;

}

// индетификатор

else

{

bool key\_B = true;

// вырез индентификатора

while (key\_B == true)

{

if (symbol == ' ' || symbol == \_terminalSymbol || symbol == ':' || symbol == ',')

{

break;

}

text += symbol;

Index++;

symbol = str[Index - 1];

}

//

// проверка

bool key\_E\_R = true;

if (wortList.Contains(text))

{

errorMessage = "Семантическая ошибка\n" + "Дублирование имен\n";

CurrentState = StateMain.E;

key\_E\_R = false;

Index--;

}

//

// VAR s,q:ARRAY [12..23] OF word; e:ARRAY [12..23] OF word;

wortList.Add(text);

// заносим в LIST

table\_text += text + " ";

//

Name.Add(text);

text = text.ToLower();

for (int i = 0; i < list.Count; i++)

{

if (string.Compare(text, list[i]) == 0)

{

errorMessage += "Семантическая ошибка\n" + "Совпадение с ключевами словами\n";

CurrentState = StateMain.E;

key\_E\_R = false;

Index--;

break;

}

}

if (text.Length > 8)

{

errorMessage += "Семантическая ошибка\n" + "Идентификатор большей длины\n";

CurrentState = StateMain.E;

key\_E\_R = false;

Index--;

}

if (key\_E\_R == true)

{

Index--;

text += \_terminalSymbol;

Identifieranalyzer identifieranalyzer = new Identifieranalyzer(text);

identifieranalyzer.Analyze(ref t\_errorMessage\_R);

if (identifieranalyzer.CurrentState != State1.E)

{

CurrentState = StateMain.C;

}

else

{

errorMessage = t\_errorMessage\_R;

CurrentState = StateMain.E;

}

}

//

text = "";

}

break;

case StateMain.T:

// петля

if (Char.IsWhiteSpace(symbol) == true)

{

CurrentState = StateMain.T;

}

//

else

{

string t\_errorMessage\_T = "";

bool key\_T = true;

// вырез константы

while (key\_T == true)

{

if (symbol == ' ' || symbol == \_terminalSymbol || symbol == ']' || symbol == '.')

{

break;

}

text += symbol;

Index++;

symbol = str[Index - 1];

}

//

Index--;

string text\_copy\_I = text;

// константа 3

text += \_terminalSymbol;

Constantanalyzer constantanalyzer = new Constantanalyzer(text);

constantanalyzer.Analyze(ref t\_errorMessage\_T);

if (constantanalyzer.CurrentState != State.E)

{

CurrentState = StateMain.U;

int r1 = text\_copy\_I.IndexOf('+');

// заносим константу в список для дальнейших проверок и операций

if (r1 == -1)

{

countList.Add(int.Parse(text\_copy\_I));

}

else

{

text\_copy\_I.Remove(0);

countList.Add(int.Parse(text\_copy\_I));

}

// проверка на диапазоном

if (countList[countList.Count - 1] < -32768 || countList[countList.Count - 1] > 32767)

{

CurrentState = StateMain.E;

errorMessage = "Семантическая ошибка\n" + "Целая константа находится за допустимым диапазоном";

t\_errorMessage\_T = "0";

}

//

}

else

{

CurrentState = StateMain.E;

errorMessage = t\_errorMessage\_T;

}

//

text = "";

}

break;

case StateMain.U:

if (symbol == '.' && str[Index] == '.')

{

Index = Index + 1;

CurrentState = StateMain.V;

}

else if (Char.IsWhiteSpace(symbol) == true)

{

CurrentState = StateMain.U;

}

else

{

CurrentState = StateMain.E;

errorMessage = "Ожидалось ..";

}

text = "";

break;

case StateMain.V:

// петля

if (Char.IsWhiteSpace(symbol) == true)

{

CurrentState = StateMain.V;

}

else

{

// константа 2

string t\_errorMessage\_V = "";

bool key\_V = true;

// вырез константы

while (key\_V == true)

{

if (symbol == ' ' || symbol == \_terminalSymbol || symbol == ']' || symbol == '.')

{

break;

}

text += symbol;

Index++;

symbol = str[Index - 1];

}

//

Index--;

string text\_copy\_K = text;

text += \_terminalSymbol;

Constantanalyzer constantanalyzer = new Constantanalyzer(text);

constantanalyzer.Analyze(ref t\_errorMessage\_V);

if (constantanalyzer.CurrentState != State.E)

{

CurrentState = StateMain.W;

// проверки констант

int r1 = text\_copy\_K.IndexOf('+');

if (r1 == -1)

{

countList.Add(int.Parse(text\_copy\_K));

}

else

{

text\_copy\_K.Remove(0);

countList.Add(int.Parse(text\_copy\_K));

}

if (countList[countList.Count - 1] < -32768 || countList[countList.Count - 1] > 32767)

{

CurrentState = StateMain.E;

errorMessage = "Семантическая ошибка\n" + "Целые константы находятся за допустимым диапазоном";

}

if (countList[countList.Count - 2] > countList[countList.Count - 1])

{

CurrentState = StateMain.E;

errorMessage = "Семантическая ошибка\n" + "Начальная константа больше конечной";

}

//

}

else

{

CurrentState = StateMain.E;

errorMessage = t\_errorMessage\_V;

}

text = "";

}

break;

case StateMain.W:

if (symbol == ']')

{

CurrentState = StateMain.M;

begunok = 10;

}

// петля

else if (Char.IsWhiteSpace(symbol) == true)

{

CurrentState = StateMain.W;

}

else

{

CurrentState = StateMain.E;

errorMessage = "Ожидалcя пробел или ]";

}

break;

case StateMain.F:

break;

case StateMain.E:

break;

default:

break;

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Boriakov\_labwork.Analyzers

{

public enum State { S, A, B, C, F, E }

internal class Constantanalyzer : Analyzerfields

{

public State CurrentState { get; set; }

public int Index { get; set; }

public Constantanalyzer(string str)

: base(str)

{

CurrentState = State.S;

Index = 0;

}

public void Analyze(ref string errorMessage)

{

while (CurrentState != State.F && CurrentState != State.E)

{

symbol = str[Index];

Index++;

switch (CurrentState)

{

case State.S:

if (symbol == '+' || symbol == '-')

{

CurrentState = State.A;

}

else if (Char.IsDigit(symbol) == true && symbol != '0')

{

CurrentState = State.B;

}

else if (symbol == '0')

{

CurrentState = State.C;

}

else

{

CurrentState = State.E;

errorMessage = "Ожидалась цифра или знак +/-";

}

break;

case State.A:

if (Char.IsDigit(symbol) == true && symbol != '0')

{

CurrentState = State.B;

}

else

{

CurrentState = State.E;

errorMessage = "Ожидалась любая цифра кроме нуля";

}

break;

case State.B:

if (Char.IsDigit(symbol) == true)

{

CurrentState = State.B;

}

else if (symbol == \_terminalSymbol)

{

CurrentState = State.F;

}

else

{

CurrentState = State.E;

errorMessage = "Ожидалась цифра";

}

break;

case State.C:

if (symbol == \_terminalSymbol)

{

CurrentState = State.F;

}

else

{

CurrentState = State.E;

errorMessage = "Ожидался конец цепочки";

}

break;

case State.F:

break;

case State.E:

break;

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Boriakov\_labwork.Analyzers

{

public enum StateSimple { S, A, F, E }

internal class Simpletypeanalyzer : Analyzerfields

{

public StateSimple CurrentState { get; set; }

public int Index { get; set; }

public Simpletypeanalyzer(string str)

: base(str)

{

CurrentState = StateSimple.S;

Index = 0;

}

public void Analyze(ref string errorMessage)

{

string text = "";

string s1 = "byte";

string s2 = "word";

string s3 = "integer";

string s4 = "real";

string s5 = "char";

string s6 = "double";

s1 = s1.ToUpper();

s2 = s2.ToUpper();

s3 = s3.ToUpper();

s6 = s6.ToUpper();

s4 = s4.ToUpper();

s5 = s5.ToUpper();

while (str[Index] != '?')

{

text += str[Index];

Index++;

}

if (Index == 0)

{

CurrentState = StateSimple.E;

errorMessage = "Ожидался простой тип";

}

while (CurrentState != StateSimple.F && CurrentState != StateSimple.E)

{

symbol = str[Index - 1];

Index++;

switch (CurrentState)

{

case StateSimple.S:

if (string.Compare(text, s1) == 0 || string.Compare(text, s2) == 0 || string.Compare(text, s3) == 0 || string.Compare(text, s4) == 0 || string.Compare(text, s5) == 0 || string.Compare(text, s6) == 0)

{

CurrentState = StateSimple.A;

}

else

{

CurrentState = StateSimple.E;

errorMessage = "Ожидался простой тип";

}

break;

case StateSimple.A:

if (symbol == \_terminalSymbol)

{

CurrentState = StateSimple.F;

}

else

{

CurrentState = StateSimple.E;

errorMessage = "Ожидался простой тип";

}

break;

case StateSimple.F:

break;

case StateSimple.E:

break;

default:

break;

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Boriakov\_labwork.Analyzers

{

public enum State1 { S, A, F, E }

internal class Identifieranalyzer : Analyzerfields

{

public State1 CurrentState { get; set; }

public int Index { get; set; }

public Identifieranalyzer(string str)

: base(str)

{

CurrentState = State1.S;

Index = 0;

}

public void Analyze(ref string errorMessage)

{

while (CurrentState != State1.F && CurrentState != State1.E)

{

symbol = str[Index];

Index++;

switch (CurrentState)

{

case State1.S:

if (Char.IsLetter(symbol) == true || symbol == '\_')

{

CurrentState = State1.A;

}

else

{

CurrentState = State1.E;

errorMessage = "Ожидалась буква или \_";

}

break;

case State1.A:

if (Char.IsLetter(symbol) == true || Char.IsDigit(symbol) == true && Char.IsPunctuation(symbol) == false)

{

CurrentState = State1.A;

}

else if (symbol == \_terminalSymbol)

{

CurrentState = State1.F;

}

else

{

CurrentState = State1.E;

errorMessage = "Ожидался индетификатор в нормальной форме";

}

break;

case State1.F:

break;

case State1.E:

break;

}

}

}

}

}