МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра автоматизированных систем управления



**КУРСОВАЯ РАБОТА**

**по дисциплине:** Теория формальных языков и компиляторов

**на тему:** **Комментарии языка PASCAL**

Выполнил:Проверил:

Студент гр. АВТ-912, АВТФ д.т.н., профессор

*Насибуллин Р.Р. Шорников Ю.В.*

«17» мая 2022 г.«\_\_\_» \_\_\_\_\_\_ 20\_\_ г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (подпись)

Новосибирск

2022

Оглавление

[Оглавление 2](#_Toc103843732)

[Постановка задачи 3](#_Toc103843733)

[Классификация грамматики 3](#_Toc103843734)

[Грамматика языка 3](#_Toc103843735)

[Метод анализа 4](#_Toc103843736)

[Тестовый пример 5](#_Toc103843737)

[Листинг 9](#_Toc103843738)

# Постановка задачи

Цель работы – константы с плавающей точкой языка С++.  
Порядок выполнения работы:

1. Разбор порождающей грамматики.
2. Определить тип грамматики по классификации Хомского.
3. Выбор метода анализа.
4. Реализация и тестирование.

При описании грамматики необходимо определить ее по классификации  
Хомского как автоматную или контекстно-свободную. Классификация  
грамматики влияет на выбор и реализацию метода синтаксического анализа:  
для автоматной – автоматный распознаватель, для контекстно-свободной –  
метод рекурсивного спуска.  
  
Для программной реализации был использован язык C# и .NET Framework.  
Данный выбор обусловлен тем, что .NET Framework позволяет создавать  
приложения для операционной системы Windows, одной из самых  
распространённых операционных систем. Для запуска приложения требуется  
операционная система Windows 7, 8, 10 с установленным .NET Framework  
актуальной версии.

# Классификация грамматики

Грамматика G[Z] по классификации Хомского относится к классу автоматных грамматик, так как все правила вывода относятся в виду:

A→ aB | a | ε.

# Грамматика языка

G[z]= {VT, VN, P, Z}

={double, float, 0,1,2,3,4,5,6,7,8,9,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,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,·, ;}

VN = {<ТИП>, <ПЕРЕМЕННАЯ>, <ЗНАЧЕНИЕ>}.

P = {

1. Z→<ТИП><переменная>’=’<ЗначЕНИЕ>’;’
2. <ТИП>→ (double | float)
3. <ПЕРЕМЕННАЯ>→Б{Б}
4. <ЗНАЧЕНИЕ>→ Ц{Ц}’.’ Ц{Ц}
5. Б → ‘a’ | ‘b’ | ‘c’... |’z’ | ‘A’ | ‘B’ | ‘C’... |’Z’
6. Ц → 0 |1 |… |9

}

# Метод анализа

Исходя из того, что грамматика является автоматной, анализ будет осуществляться при помощи графа конечного автомата:

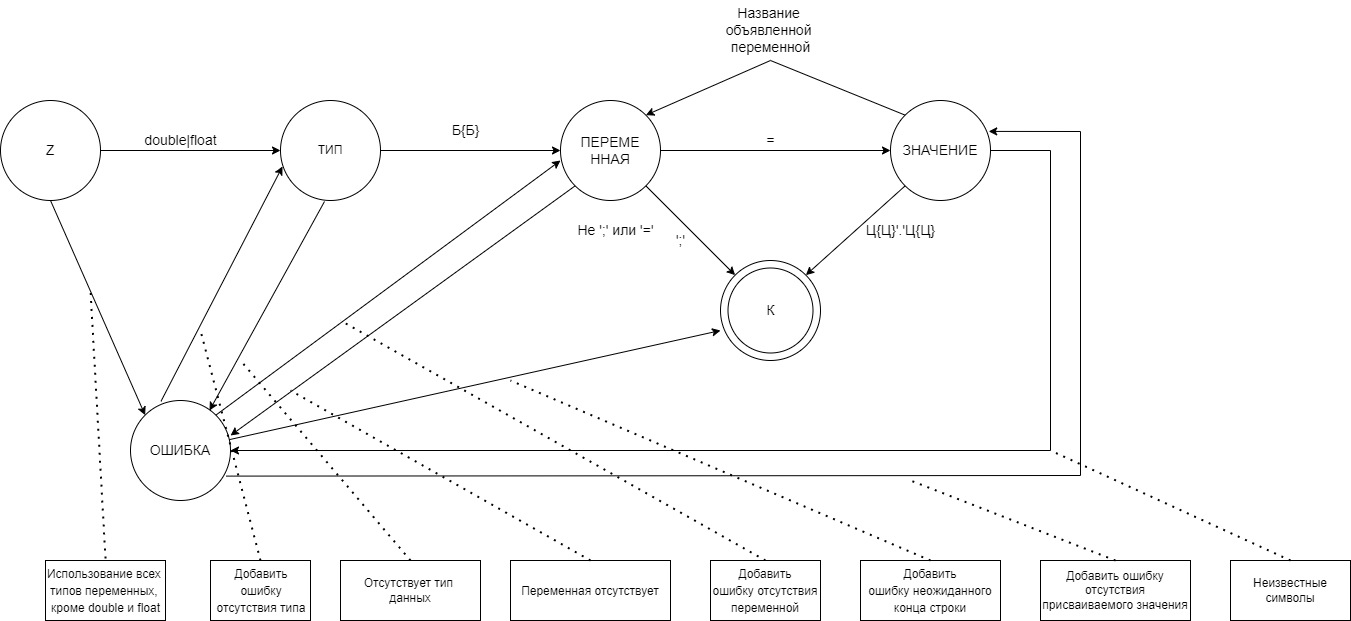


Рис. 1 Граф конечного автомата

Для диагностики и нейтрализации ошибок используется метод Айронса.

Основная идея – по контексту без возврата отбрасывать литеры, которые привели к тупиковой ситуации (когда продолжение анализа по грамматике невозможно), и продолжать разбор.

# Тестовый пример

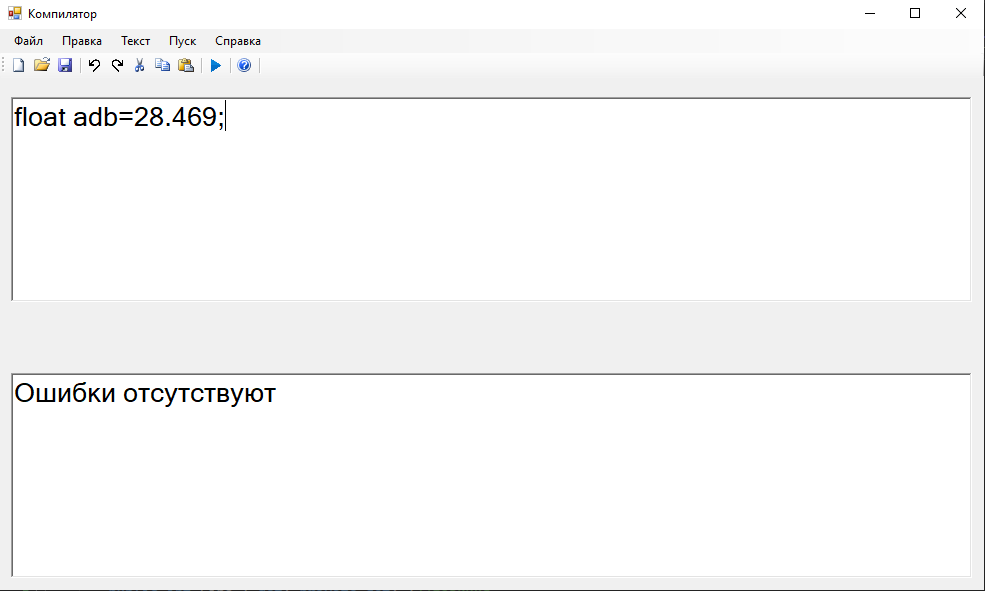


Рис. 2 Пример ввода без ошибок

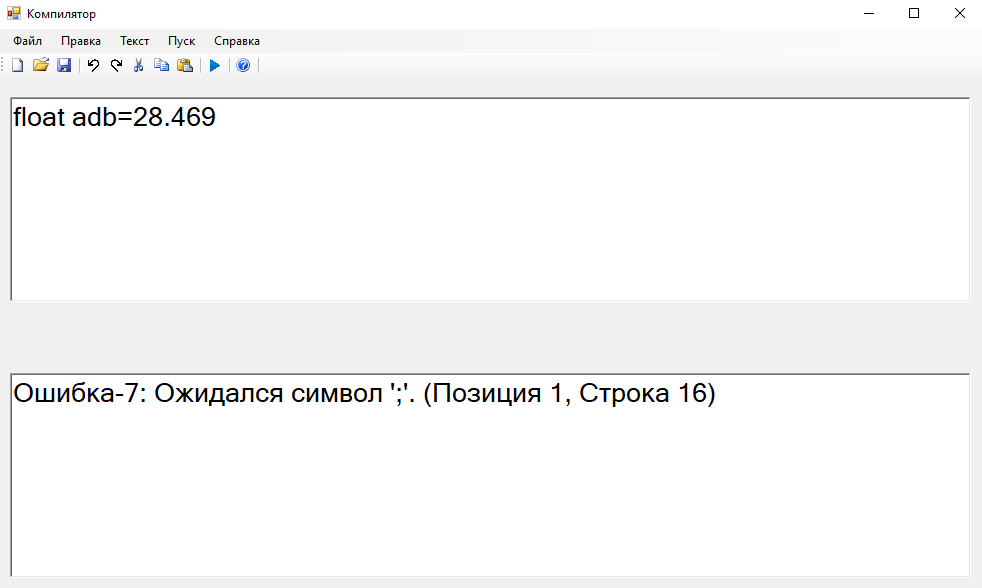


Рис. 3 Пример ввода с ошибкой

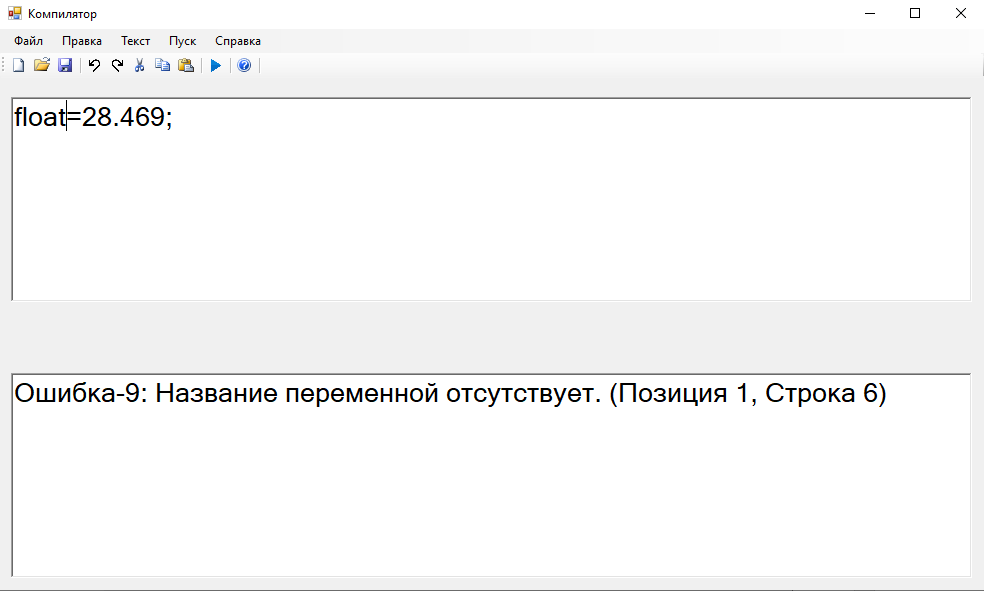


Рис. 4 Пример ввода с ошибкой

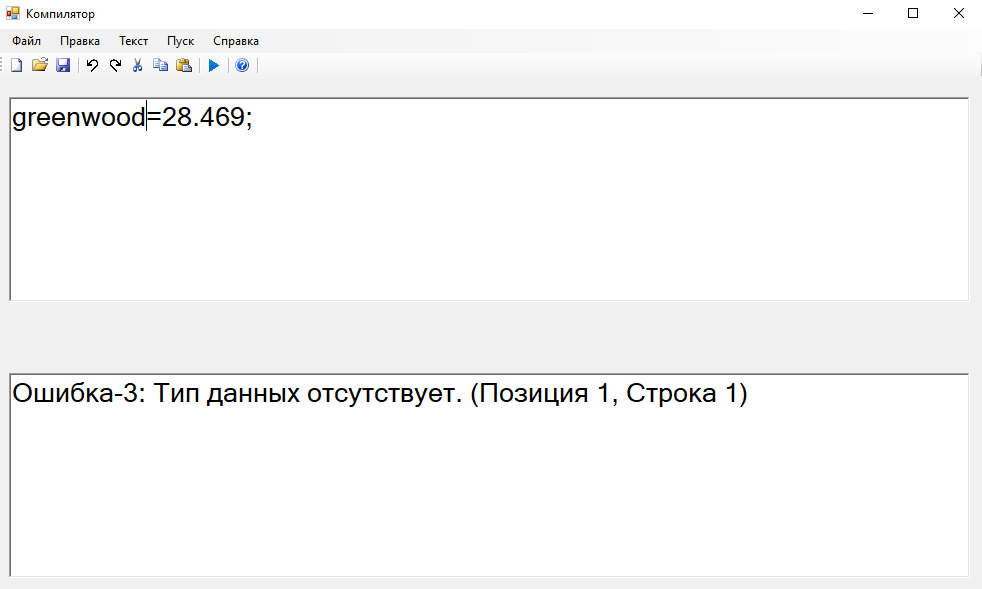


Рис. 5 Пример ввода с ошибкой

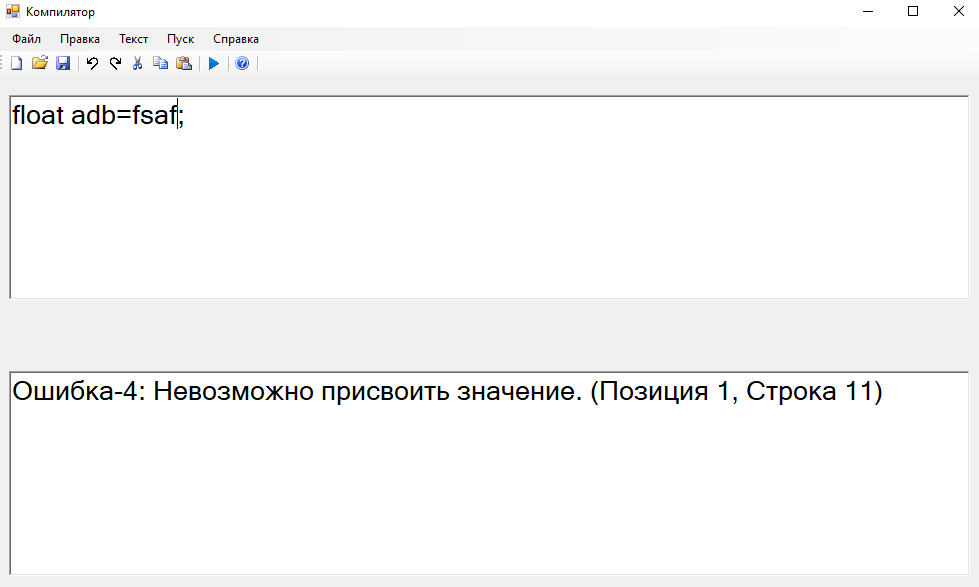


Рис. 6 Пример ввода с ошибкой

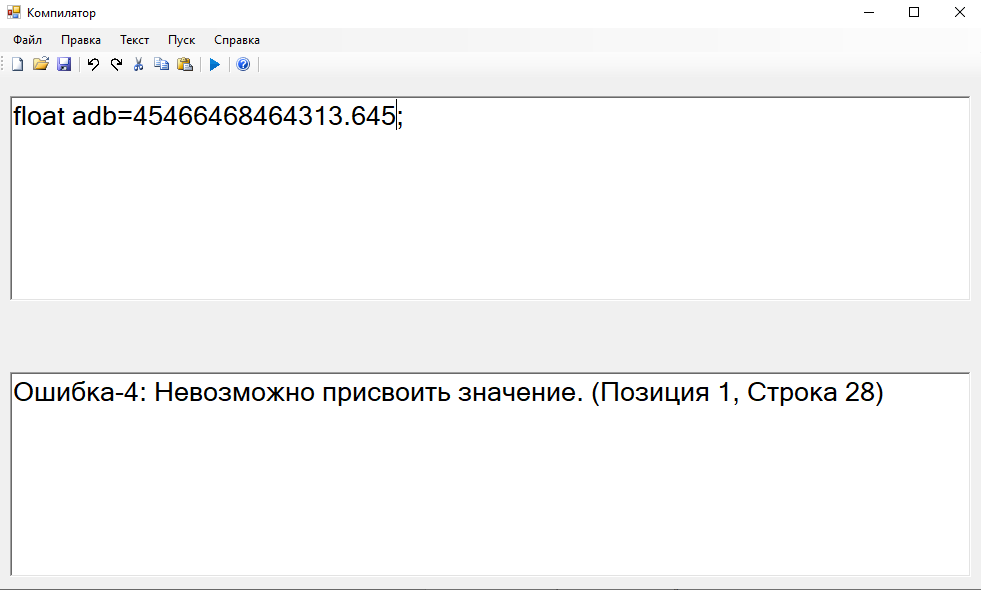


Рис. 7 Пример ввода с ошибкой

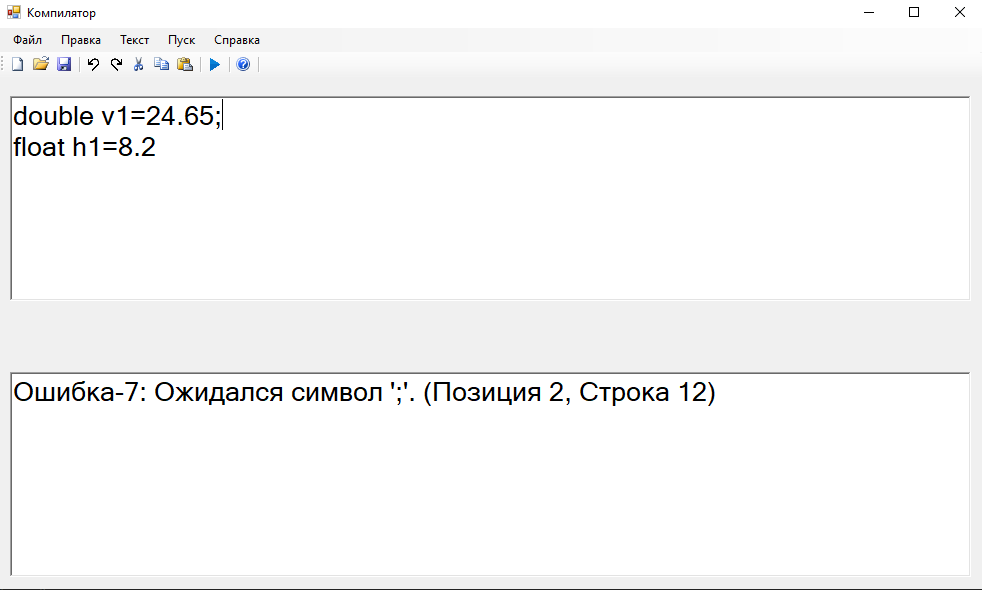


Рис. 8 Пример ввода с ошибкой

# Листинг

**Scanner.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp2

{

public class Error

{

private static readonly Dictionary<int, string> Errors = new Dictionary<int, string>

{

{1, "Неожиданный конец строки."},

{2, "Неизвестный тип данных."},

{3, "Тип данных отсутствует."},

{4, "Невозможно присвоить значение."},

{5, "Неправильное имя переменной."},

{6, "Неизвестный символ."},

{7,"Ожидался символ ';'." },

{8,"Ожидался символ '='." },

{9, "Название переменной отсутствует." },

{10, "Присваиваемое значение отсутствует." }

};

public int Code { get; private set; }//код ошибки

public int Line { get; private set; }//строка

public int Column { get; private set; }//позиция

public Error(int code, int line, int column)

{

Code = code;

Line = line;

Column = column;

}

public string FormattedError()

{

if ((Code == 1))

{

return string.Format("Ошибка-{0}: {1} (Строка {2})", Code, Errors[Code], Line);

}

else

{

return string.Format("Ошибка-{0}: {1} (Строка {2}, Позиция {3})", Code, Errors[Code], Line, Column);

}

}

}

public class Token

{

public int Code { get; set; }

public int Line { get; set; }

public int Column { get; set; }

public Token(int code, int line, int column)

{

Code = code;

Line = line;

Column = column;

}

}

public class Scanner

{

public List<List<Token>> ScannedText { get; private set; }//список токенов

public Scanner(string[] source)

{

Errors = new List<Error>();

ScannedText = Scan(source);

}

public List<Error> Errors { get; set; }//список ошибок

private List<List<Token>> Scan(string[] source)//функция сканирования

{

var scannedText = new List<List<Token>>();//переменная, которая хранит просканированный текст

foreach (var line in source)

{

var scanned = new List<Token>();//переменная, которая хранит просканированную строку

for (var i = 0; i < line.Length; i++)

{

if (i + 5 < line.Length)//double

if (line[i] == 'd' && line[i + 1] == 'o' && line[i + 2] == 'u' && line[i + 3] == 'b' && line[i + 4] == 'l' && line[i + 5] == 'e')

{

scanned.Add(new Token(4, Array.IndexOf(source, line) + 1, i + 1));

i += 5;

continue;

}

if (i + 4 < line.Length)//float

if (line[i] == 'f' && line[i + 1] == 'l' && line[i + 2] == 'o' && line[i + 3] == 'a' && line[i + 4] == 't')

{

scanned.Add(new Token(5, Array.IndexOf(source, line) + 1, i + 1));

i += 4;

continue;

}

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

{

bool checknul = true;

List<char> a = new List<char>();

string b = "";

bool fraction = false;

while (i < line.Length)

{

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

{

if (line[i] == '0' && checknul)

{

i++;

continue;

}

else

{

if (!fraction)

a.Add(line[i]);

i++;

checknul = false;

continue;

}

}

if (line[i] == '.' || line[i] == ',')

{

i++;

fraction = true;

continue;

}

if (checknul)

a.Add('0');

break;

}

i--;

for (int j = 0; j < a.Count; j++)

b += a[j].ToString();

int n = NumResolution(b, fraction);

scanned.Add(new Token(n, Array.IndexOf(source, line) + 1, i + 1));

continue;

}

if (line[i] >= 'a' & line[i] <= 'z' || line[i] >= 'A' & line[i] <= 'Z')

{

scanned.Add(new Token(11, Array.IndexOf(source, line) + 1, i + 1));

i++;

while (i < line.Length)

{

if (line[i] >= 'a' & line[i] <= 'z' || line[i] >= 'A' & line[i] <= 'Z' || line[i] >= '0' & line[i] <= '9')

{

i++;

continue;

}

break;

}

i--;

continue;

}

switch (line[i])

{

case ';':

scanned.Add(new Token(12, Array.IndexOf(source, line) + 1, i + 1));

break;

case '=':

scanned.Add(new Token(13, Array.IndexOf(source, line) + 1, i + 1));

break;

case '-':

scanned.Add(new Token(16, Array.IndexOf(source, line) + 1, i + 1));

break;

case ' ':

break;

default:

Errors.Add(new Error(6, Array.IndexOf(source, line) + 1, i + 1));

break;

}

}

scannedText.Add(scanned);

}

return scannedText;

}

public int NumResolution(string str, bool fraction)//функция сравнения чисел на привышение

{

int result = 0;

if (fraction)

{

result = String.Compare(str, "999999999");

if (result < 0 && str.Length < 10) return 9;

result = String.Compare(str, "999999999");

if (result < 0 && str.Length < 10) return 18;

}

return 19;

}

}

public class Syntax

{

public List<Error> Errors { get; set; }// список ошибок

List<Token> line;//список токенов

int iterator, lineNum;

public Syntax(List<List<Token>> scanned)

{

Errors = new List<Error>();

StartSyntax(scanned);

}

private void StartSyntax(List<List<Token>> scanned)//функция определения переменных

{

if (Errors.Count != 0)

return;

foreach (var Line in scanned)

{

line = Line;

iterator = 0;

lineNum = scanned.IndexOf(Line) + 1;

logexp();

}

}

private void logexp()//функция начала разбора

{

if (iterator >= line.Count)

{

return;

}

if (line[iterator].Code != 4 && line[iterator].Code != 5)

{

if (line[iterator].Code == 11)

Errors.Add(new Error(3, line[iterator].Line, line[iterator].Column));

}

if (iterator + 1 < line.Count)

if (line[iterator + 1].Code == 5)

Next();

if (line[iterator].Code == 11 && line[iterator + 1].Code == 11)

Errors.Add(new Error(2, line[iterator].Line, line[iterator].Column));

Next();

assign();

}

private void Next()//переход на след символ

{

if (iterator >= line.Count - 1)

{

return;

}

else

iterator++;//передвижение каретки

}

private void assign()

{

if (Errors.Count == 0)

{

if (line[iterator].Code != 11)

{

Errors.Add(new Error(9, line[iterator].Line, line[iterator].Column));

}

else Next();

}

if (line[iterator].Code != 13)

{

Errors.Add(new Error(8, line[iterator].Line, line[iterator].Column)); Next();

}

if (line[iterator].Code == 12)

{

Errors.Add(new Error(10, line[iterator].Line, line[iterator].Column)); Next();

}

else Next();

switch (line[0].Code)

{

case 4:

{

if (line[iterator].Code == 16) Next();

if (line[iterator].Code == 9 || line[iterator].Code == 18) Next();

else { Errors.Add(new Error(4, line[iterator].Line, line[iterator].Column)); Next(); }

break;

}

case 5:

{

if (line[iterator].Code == 16) Next();

if (line[iterator].Code == 9) Next();

else { Errors.Add(new Error(4, line[iterator].Line, line[iterator].Column)); Next(); }

break;

}

default: Next(); break;

}

if (line[iterator].Code != 12) Errors.Add(new Error(7, line[iterator].Line, line[iterator].Column));

}

}

}