МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

факультет програмної інженерії та бізнесу

кафедра інженерії програмного забезпечення

**Лабораторна робота № 2**

з дисципліни « Конструювання ПЗ »

*назва дисципліни*

на тему: «Разработка синтаксического анализатора  
 методом рекурсивного спуска»

Виконав: студент 3 курсу групи № 631п

напряму підготовки (спеціальності)

121. Інженерія програмного забезпечення

(шифр і назва напряму підготовки (спеціальності))

Кузьмич М. І.

(прізвище й ініціали студента)

Прийняв: к.т.н., доц. Волобуєва Л. О.

(посада, науковий ступінь, прізвище й ініціали)

Національна шкала:

Кількість балів:

Оцінка ECTS:

Харків – 2018

**Цель работы:** программирование синтаксического анализатора заданных конструкций методом рекурсивного спуска.

#### ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ

**Контекстно – свободные языки и грамматики**

К классу КС относятся грамматики, у которых не накладывается никаких ограничений на вид правых частей правил, а левая часть каждого правила – единственных нетерминалов.

Однозначной называется грамматика, в каждой сентенции которой соответствует единственное дерево вывода.

Рассмотрим однозначную грамматику , задающую синтаксис арифметических выражений

Построим два различных вывода цепочки a+b\*c в этой грамматике. В первом случае, если сентенциальная форма содержит более одного нетерминала, будем выполнять подстановку для самого левого нетерминала этой сентенциальной формы:

 Такой вывод называется **левосторонним**. Аналогично, вывод, в ходе которого замене всегда подвергается самый правый нетерминал сентенциальной формы над **правосторонним**.



Обе эти последовательности подстановок соответствуют одному и тому же дереву вывода, но разному порядку его построения.

**КС – грамматика называется однозначной**, если для каждого предложения языка порождаемого этой грамматикой, существует единственный левосторонний вывод.

**Синтаксический анализ КС-языков методом рекурсивного спуска.**

**Рекурсивный спуск** – это эффективный и простой нисходящий алгоритм распознавания. Он состоит в следующем.

Для каждого нетерминала грамматики (понятия, конструкции языка), т.е. для каждой синтаксической диаграммы, записывается отдельная распознающая процедура. При этом соблюдаются следующие соглашения:

1. Перед началом работы процедуры текущим является первый символ анализируемого понятия.
2. В процессе работы процедура считывает все символы входной цепочки, относящиеся к данному нетерминалу (выводимые из данного нетерминала) если сообщает об ошибке. Если правила для данного нетерминала содержат в правых частях другие нетерминалы, то процедура обращается к распознающим процедурам этих нетерминалов для анализа соответствующих частей входной цепочки.
3. По окончании работы процедуры, текущим становится первый символ, следующий во входной цепочке за данной конструкцией языка (символами, выводимыми из данного нетерминала).

Синтаксические диаграммы автоматного языка получают из диаграммы переходов ДКА устранив из диаграммы переходов обозначения состояний (заглавные буквы и кружки), те места, где были состояния автомата станут точками ветвления и соединения дуг.

Терминальные символы, отмечающие переходы – дуги, наоборот разместим в кружках на этих дугах.

Покажем такое построение на примере ДКА , распознающего язык идентификаторов (заменим по смыслу ).

|  |  |
| --- | --- |
| Диаграмма переходов ДКА | Синтаксическая диаграмма |

Распознавание начинается вызовом распознающей процедуры начального нетерминала. При этом текущим символом, как это следует из п. 1 должен быть первый символ входной цепочки. По завершении работы начальной процедуры текущим должен быть символ «конец текста».

Название «рекурсивный спуск» обусловлено тем, что при наличии в грамматике самовложения, вызовы распознающих процедур будут рекурсивными.

КС – грамматика, не содержащая самовложения, эквивалентна автоматной грамматике.

**Синтаксические диаграммы КС – языков**

Синтаксические диаграммы КС – языка могут быть построены по его грамматике на основании следующих правил:

1. Для каждого нетерминала грамматики строится отдельная диаграмма, обозначенная названием этого нетерминала.
2. Нетерминалы из правых частей правил изображаются прямоугольниками с названием нетерминала. Терминальные символы изображаются в кружках или овалах.
3. Для каждой правой части правила строится ветвь, представляющая собой последовательно соединенные прямоугольники и круги (овалы), следующие в том же порядке слева направо, что и соответствующие нетерминалы и терминалы правой части правила.
4. Ветви, соответствующие альтернативным правым частям правил для одного нетерминала, соединяются параллельно и образуют диаграмму для данного нетерминала.

Рассмотрим примеры построения диаграмм.

Правило  отображается .

Правило  порождает ветвь 

Если других правил для нетерминала А в грамматике нет, то диаграмма для этого нетерминала  будет отражаться так:

А:



Поскольку в правой части правила есть нетерминалы B и D, то должны быть правила, определяющие их.

Пусть правило для В имеет вид: 

В:



Можно заметить, что правила для В удовлетворяют ограничениям автоматных грамматик, но диаграмма содержит самовложение, что недопустимо для диаграмм автоматных грамматик. Для устранения этого несоответствия самовложение заменяют повторным входом в диаграмму:

В:



или так:

В:



В результате преобразования концевая (правая) рекурсия заменена циклом.

Пусть D определяется правилом .

D:

 или

D:



**Постановка задачи**

**Вариант №11**

Уравнение плоскости вида  с целыми коэффициентами.

**Ход выполнения работы**

Описание построенной грамматики

1. – 2 тип
2. – 3 тип
3. – 2 тип
4. – 2 тип
5. – 3 тип
6. – 3 тип

Общая грамматика 2-го типа.

**Изображение и описание синтаксических диаграмм**

На рисунках 1-6 изображены синтаксические диаграммы правил грамматики.

Рисунок 1 – Синтаксическая диаграмма нетерминала S



Рисунок 2 – Синтаксическая диаграмма нетерминала Z



Рисунок 3 – Синтаксическая диаграмма нетерминала K



Рисунок 4 – Синтаксическая диаграмма нетерминала H



Рисунок 5 – Синтаксическая диаграмма нетерминала C



Рисунок 6 – Синтаксическая диаграмма нетерминала C1

**Блок-схемы основных алгоритмов программы**



Рисунок 7 – Алгоритм правила S



Рисунок 8 – Алгоритм правила Z



Рисунок 9 – Алгоритм правила K



Рисунок 10 – Алгоритм правила H



Рисунок 11 – Алгоритм правила C



Рисунок 12 – Алгоритм правила C1

**Экранные формы работы программы**

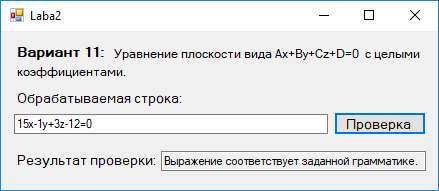


Рисунок 16 – Экранная форма №1

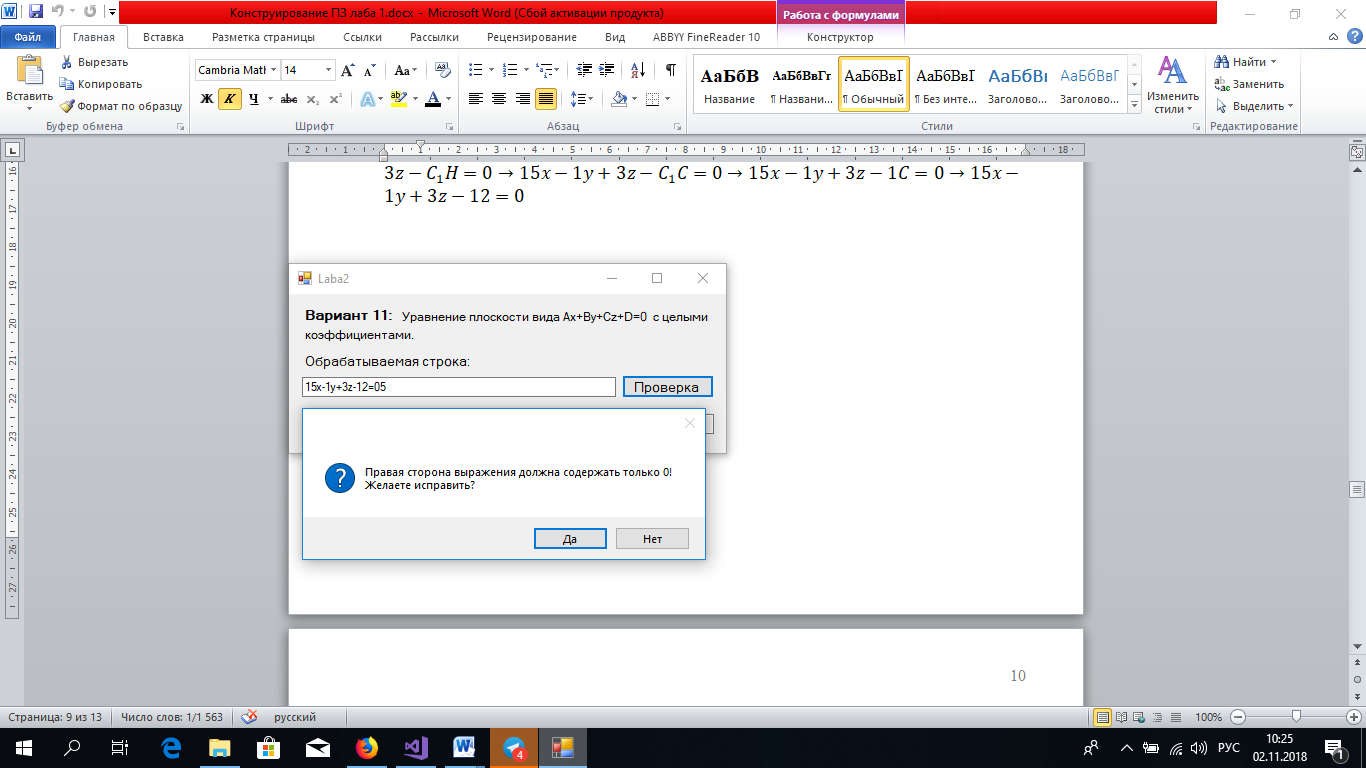


Рисунок 17 – Экранная форма №2

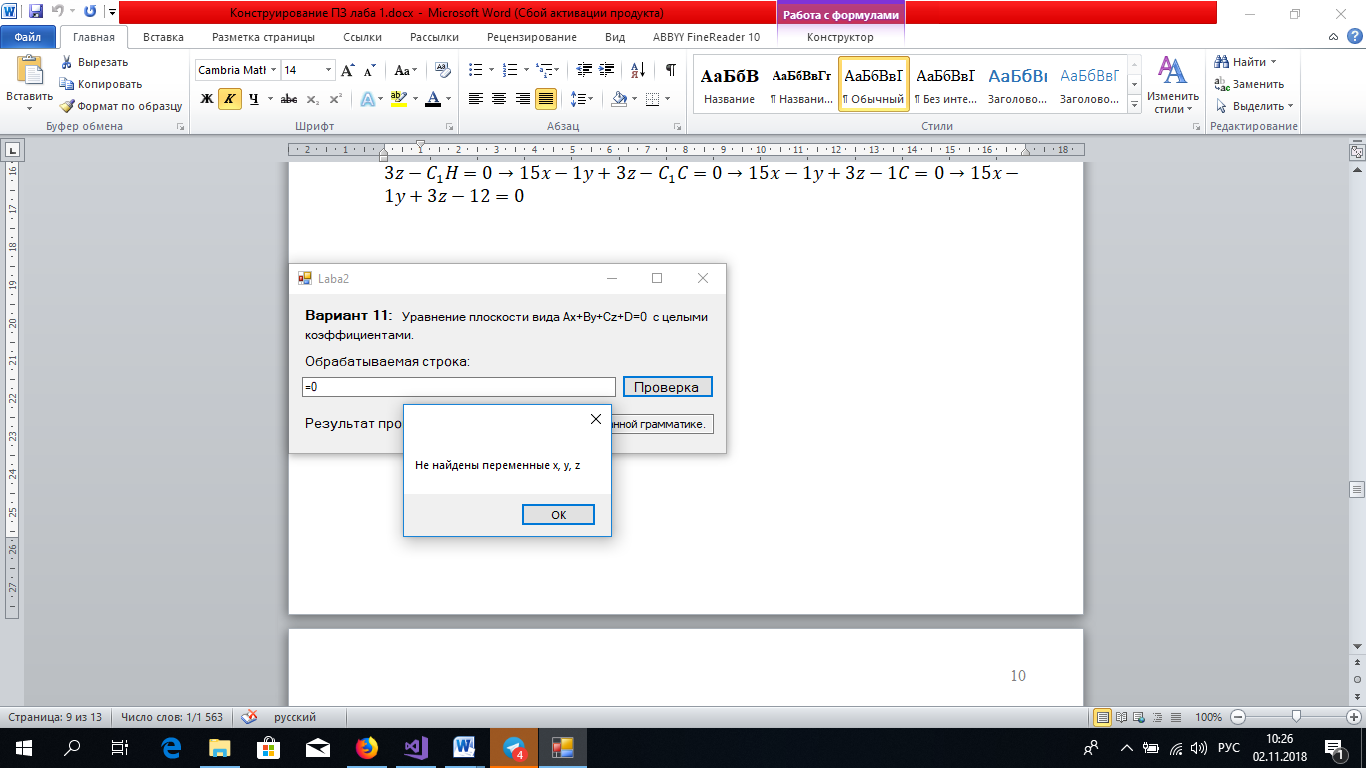


Рисунок 18 – Экранная форма №3

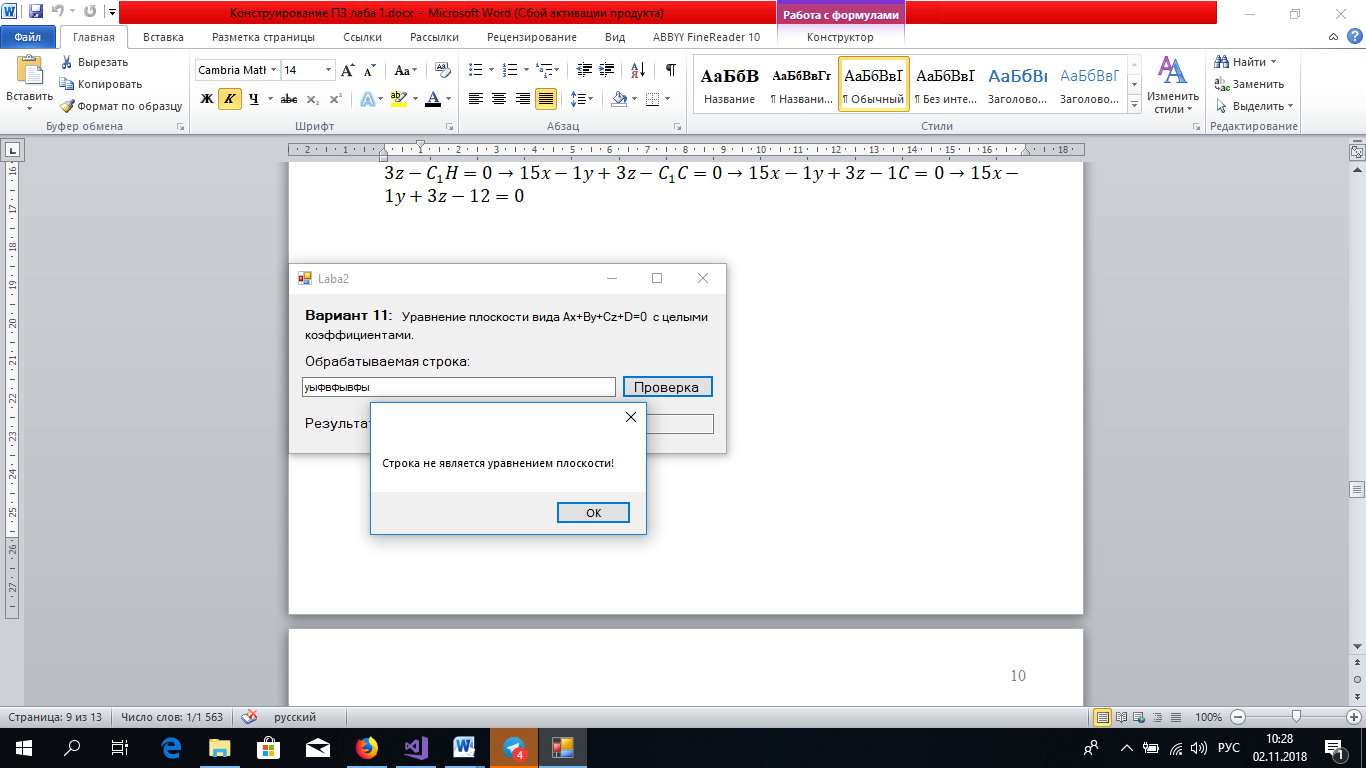


Рисунок 19 – Экранная форма №4

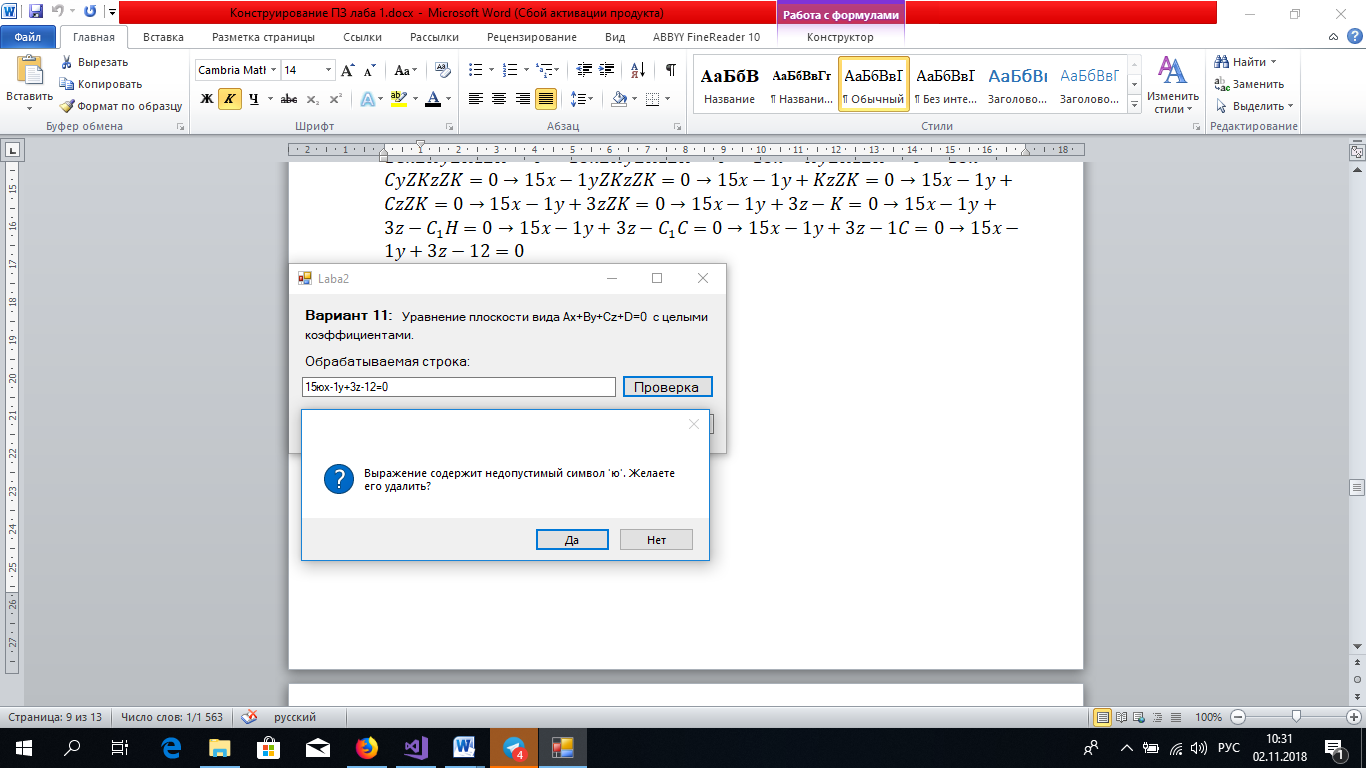


Рисунок 20 – Экранная форма №5

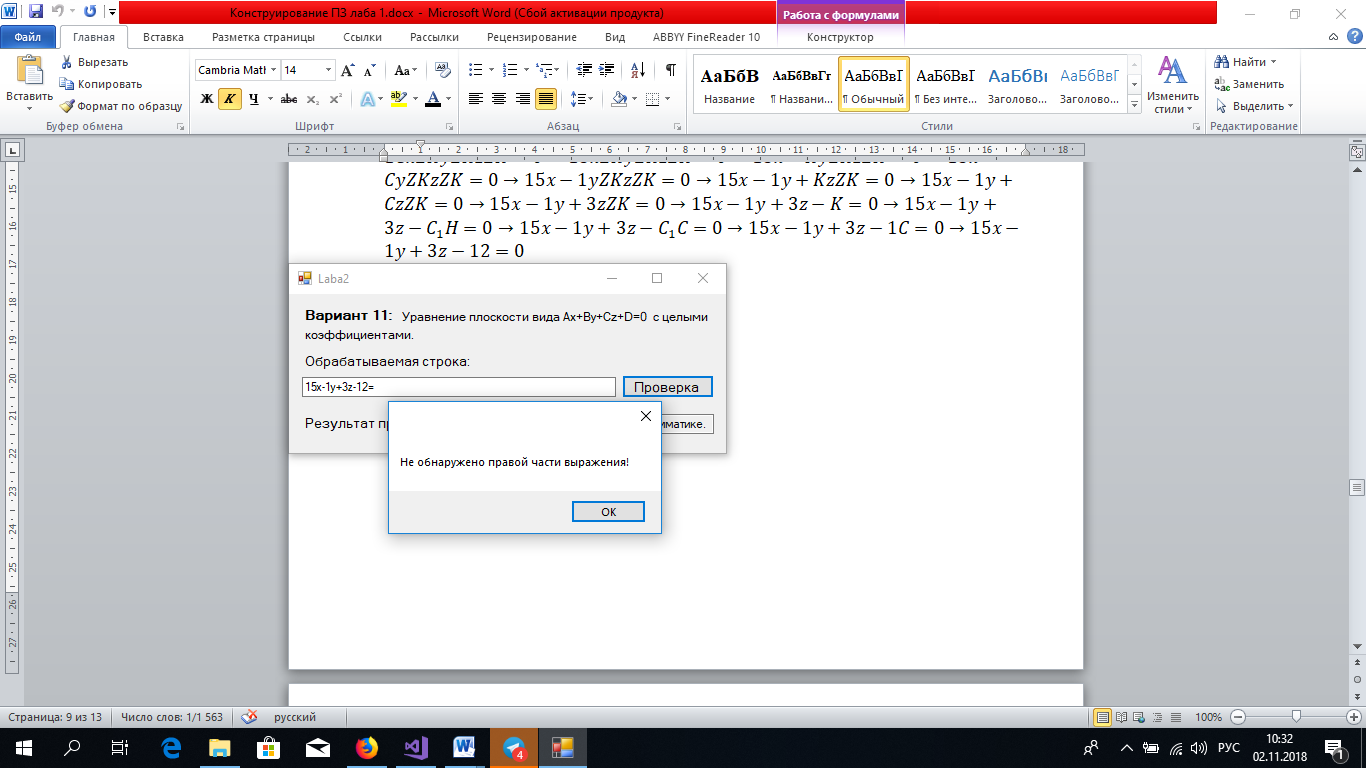


Рисунок 21 – Экранная форма №6

**Вывод**

В ходе выполнения данной лабораторной работы была разработана программа для синтаксического анализа заданного по варианту выражения. Основные алгоритмы программы были реализованы с помощью метода рекурсивного спуска, но также в программе имеются и сторонние методы, которые выполняют дополнительные проверки.

***Приложение А***

Исходный код программы

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace ConstruirovanieLaba2

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

bool b;

char sym;

int num = 0;

bool Double\_var(ref string strin, char symb, TextBox tb)

{

if (strin.Count(x => x == symb) != 1)

{

DialogResult result;

result = MessageBox.Show($"Строка содержит более одного экземпляра переменной {symb}! Желаете исправить?", "", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (result == DialogResult.No)

{

Label\_text(strin, symb);

return false;

}

else

{

int index = strin.IndexOf(symb);

strin = strin.Replace(strin[strin.IndexOf(symb)].ToString(), "");

tb.Text = strin.Insert(index, symb.ToString());

strin = tb.Text;

return true;

}

}

return true;

}

int Position(string str, char sym)

{

int cnt = 0;

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

{

if (str[i] == sym) break;

cnt++;

}

return cnt;

}

void Label\_text(string s, char sym)

{

int pos = Position(s, sym);

label6.Visible = true;

label6.Text = "Ошибка на позиции: " + (pos + 1).ToString();

}

private void btn\_check\_Click(object sender, EventArgs e)

{

b = First\_Rule\_S(inp\_str\_text);

if (b) result.Text = "Выражение соответствует заданной грамматике.";

else result.Text = "Выражение неверно!";

}

public bool Сoefficient(ref string s, char symb)

{

string str = "Правила грамматики предпологают наличие коэффициента перед переменной " + symb.ToString() + ".";

if (!Char.IsDigit(s[0]) && s.IndexOf(symb) == 0)

{

MessageBox.Show(str);

sym = s[0];

Label\_text(s, sym);

return false;

}

if (!Char.IsDigit(s[s.IndexOf(symb) - 1]))

{

if ((s[s.IndexOf(symb) - 1] == '+' || s[s.IndexOf(symb) - 1] == '-'))

{

MessageBox.Show(str);

sym = s[s.IndexOf(symb)];

Label\_text(s, sym);

return false;

}

else

{

DialogResult result;

result = MessageBox.Show($"Перед переменной {symb} содержится недопустимый символ '{s[s.IndexOf(symb) - 1]}'. Желаете его удалить?", "", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (result == DialogResult.No) { sym = s[s.IndexOf(symb) - 1]; Label\_text(s, sym); return false; }

else

{

s = s.Replace(s[s.IndexOf(symb) - 1].ToString(), "");

inp\_str\_text.Text = s;

return true;

}

}

}

return true;

}

public bool First\_Rule\_S(TextBox tb)

{

string s = tb.Text.Replace(" ", "");

if (s.Length == 0)

{

MessageBox.Show("Входная строка не может быть пустой!");

return false;

}

if (s.Count(x => x == '=') == 0)

{

MessageBox.Show("Строка не является уравнением плоскости!");

sym = s[0];

Label\_text(s, sym);

return false;

}

if (s.Count(x => x == '=') == 1 && s.Length == 1)

{

MessageBox.Show("Отсутствуют левая и правая части уравнения.");

sym = s[0];

Label\_text(s, sym);

return false;

}

if (s.Count(x => x == '=') != 1)

{

MessageBox.Show("Выражение неверно, т.к. содержится больше одного знака равно!");

return false;

}

int index = s.IndexOf('=') + 1;

if (s[s.Length - 1].ToString() == "=")

{

MessageBox.Show("Не обнаружено правой части выражения!");

Label\_text(s, sym);

return false;

}

if (s[index].ToString() != "0")

{

DialogResult result;

result = MessageBox.Show("Правая сторона выражения должна содержать только 0! Желаете исправить?", "", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (result == DialogResult.No)

{

sym = s[s.IndexOf('=') + 1];

Label\_text(s, sym);

return false;

}

else

{

s = s.Substring(0, index);

s += "0";

tb.Text = s;

}

}

int cnt = 0;

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

{

cnt++;

}

if (cnt > 1)

{

DialogResult result;

result = MessageBox.Show("Правая сторона выражения должна содержать только 0! Желаете исправить?", "", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (result == DialogResult.No)

{

Label\_text(s, sym);

return false;

}

else

{

s = s.Substring(0, index);

s += "0";

tb.Text = s;

}

}

char[] variables = new char[0];

variables = NewMethod(s, variables, 'x');

variables = NewMethod(s, variables, 'y');

variables = NewMethod(s, variables, 'z');

string var\_array = "";

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

{

if (variables.Length == 1) var\_array += variables[i].ToString();

else var\_array += variables[i].ToString() + ", ";

}

if (variables.Length == 1)

{

MessageBox.Show("Не найдена переменная " + var\_array);

return false;

}

else if (variables.Length > 1)

{

var\_array = var\_array.Substring(0, var\_array.Length - 2);

MessageBox.Show("Не найдены переменные " + var\_array);

return false;

}

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

{

if ((s[i].ToString() == "+" || s[i].ToString() == "-") && (s[i + 1].ToString() == "+" || s[i + 1].ToString() == "-"))

{

DialogResult result;

result = MessageBox.Show("Строка содержит символы '" + s[i].ToString() + "', '" + s[i + 1].ToString() + "', идущие друг за другом. Желаете заменить на '" + s[i].ToString() + "'?", "", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (result == DialogResult.No)

{

sym = s[i + 1];

Label\_text(s, sym);

break;

}

else

{

tb.Text = s.Remove(i + 1, 1);

s = tb.Text;

break;

}

}

}

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

{

if ((s[i].ToString() == "+" || s[i].ToString() == "-") && (s[i + 1].ToString() == "+" || s[i + 1].ToString() == "-"))

{

Label\_text(s, sym);

return false;

}

}

if (s[index - 2].ToString() == "-" || s[index - 2].ToString() == "+")

{

MessageBox.Show("Грамматика предпологает наличие коэффициентов после знаков '+' или '-'");

Label\_text(s, sym);

return false;

}

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

{

if ((s[i].ToString() == "x" || s[i].ToString() == "y" || s[i].ToString() == "z") && (s[i + 1].ToString() != "+" && s[i + 1].ToString() != "-"))

{

if ((!Char.IsDigit(s[i + 1]))) break;

MessageBox.Show("После переменной " + s[i].ToString() + " не было обнаружено знака '+' или '-'.");

Label\_text(s, sym);

return false;

}

}

bool[] b = new bool[3];

b[0] = Сoefficient(ref s, 'x');

b[1] = Сoefficient(ref s, 'y');

b[2] = Сoefficient(ref s, 'z');

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

{

if (b[i] == false) return false;

}

int kol = 0;

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

{

if (s[i].ToString() == "x" || s[i].ToString() == "y" || s[i].ToString() == "z")

{

if (i == 1 && s[0].ToString() == "0") kol++;

else if (s[i - 1].ToString() == "0" && (s[i - 2].ToString() == "+" || s[i - 2].ToString() == "-"))

{

kol++;

}

}

}

if (kol > 2)

{

MessageBox.Show("Коэффициент хотя бы одной переменной не должен быть равен 0.");

return false;

}

if (!Double\_var(ref s, 'x', tb)) return false;

if (!Double\_var(ref s, 'y', tb)) return false;

if (!Double\_var(ref s, 'z', tb)) return false;

string[] str = s.Split('x', 'y', 'z', '=');

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

{

for (int j = 0; j < str[i].Length; j++)

{

if (str[i][j] >= '0' && str[i][j] <= '9' || str[i][j] == '+' || str[i][j] == '-')

{

Second\_Rule\_Z(str[i]);

if (num == 3)

{

DialogResult result;

result = MessageBox.Show("Коэффициенты переменных не должны начинать с нуля!Желаете исправить?", "", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (result == DialogResult.No)

{

Label\_text(s, sym);

num = 0;

return false;

}

else

{

if (str[0][j].ToString() == "0")

{

tb.Text = s.Replace(str[i][j].ToString(), "") + "0";

s = tb.Text;

num = 0;

return true;

}

else

{

tb.Text = s.Replace(str[i][j + 1].ToString(), "") + "0";

s = tb.Text;

num = 0;

return true;

}

}

}

if ((!(Second\_Rule\_Z(str[i]))) && Char.IsDigit(sym)) return false;

}

else

{

DialogResult result;

result = MessageBox.Show($"Выражение содержит недопустимый символ '{str[i][j]}'. Желаете его удалить?", "", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (result == DialogResult.No) { sym = str[i][j]; Label\_text(s, sym); return false; }

else

{

tb.Text = s.Replace(str[i][j].ToString(), "");

s = tb.Text;

}

}

}

}

label6.Visible = false;

return true;

}

private static char[] NewMethod(string s, char[] variables, char symb)

{

if (s.Count(x => x == symb) == 0)

{

Array.Resize(ref variables, variables.Length + 1);

variables[variables.Length - 1] = symb;

}

return variables;

}

public bool Second\_Rule\_Z(string str)

{

string[] new\_s = str.Split('+', '-');

if (new\_s.Length == 1 && new\_s[0] == "") return true;

for (int i = 0; i < new\_s.Length; i++)

{

for (int j = 0; j < new\_s[i].Length; j++)

{

if (new\_s[i][j] >= '0' && new\_s[i][j] <= '9') return Third\_Rule\_K(new\_s[i]);

else return false;

}

}

return false;

}

public bool Third\_Rule\_K(string str)

{

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

{

if (str.Length == 1) return Fifth\_Rule\_C(str[0]);

else if (str.Length > 1)

{

return Sixth\_Rule\_C1(str[0]) && Fourth\_Rule\_H(str.ToString().Substring(1));

}

}

return false;

}

public bool Fourth\_Rule\_H(string str)

{

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

{

if (str.Length == 1) return Fifth\_Rule\_C(str[0]);

else if (str.Length > 1)

{

return Fifth\_Rule\_C(str[0]) && Fourth\_Rule\_H(str.ToString().Substring(1));

}

}

return false;

}

public bool Fifth\_Rule\_C(char str)

{

if (str >= '0' && str <= '9') return true;

else

{

sym = str;

return false;

}

}

public bool Sixth\_Rule\_C1(char str)

{

if (str >= '1' && str <= '9') return true;

else

{

num = 3;

sym = str;

return false;

}

}

}

}