Министерство образования Республики Беларусь

Белорусский Национальный Технический Университет

Факультет информационных технологий и робототехники

Кафедра «Программное обеспечение вычислительной техники

и автоматизированных систем»

**Отчёт**

по лабораторной работе № 4

по дисциплине ***«РАЗРАБОТКА ПРИЛОЖЕНИЙ В ВИЗУАЛЬНЫХ СРЕДАХ»***

тема: **«СОЗДАНИЕ ПРИЛОЖЕНИЯ, РЕАЛИЗУЮЩЕГО ПРИНЦИП**

**ПОЛИМОРФИЗМА»**

|  |  |  |
| --- | --- | --- |
| Исполнитель: |  | студент группы 10701217  Голованов Павел Андреевич |
| Преподаватель: |  | Гурский Николай Николаевич |

2019 учебный год

**ЛАБОРАТОРНАЯ РАБОТА № 4**

**СОЗДАНИЕ ПРИЛОЖЕНИЯ, РЕАЛИЗУЮЩЕГО ПРИНЦИП**

**ПОЛИМОРФИЗМА**

Цель работы:

Изучить правила создания дочерних классов.

Задание:

Разработать дочерний класс вычисления определенного интеграла с

различными подинтегральными функциями. Реализовать возможность

арифметических операций над интегралами с выбранными функциями.

Примерный вид приложения показан на рис.1.



Разработать приложение в соответствии с л.р. №1 и п.2.

Результаты выполнения задания:

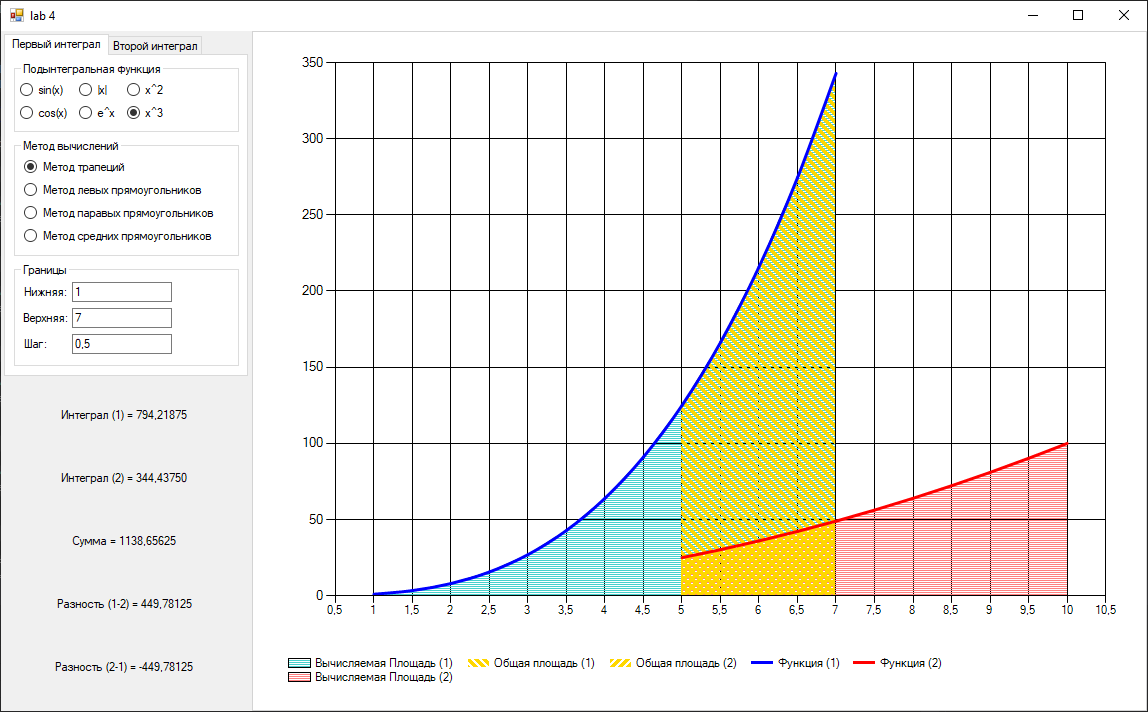


Рисунок 1 – Результат с разными степенями.

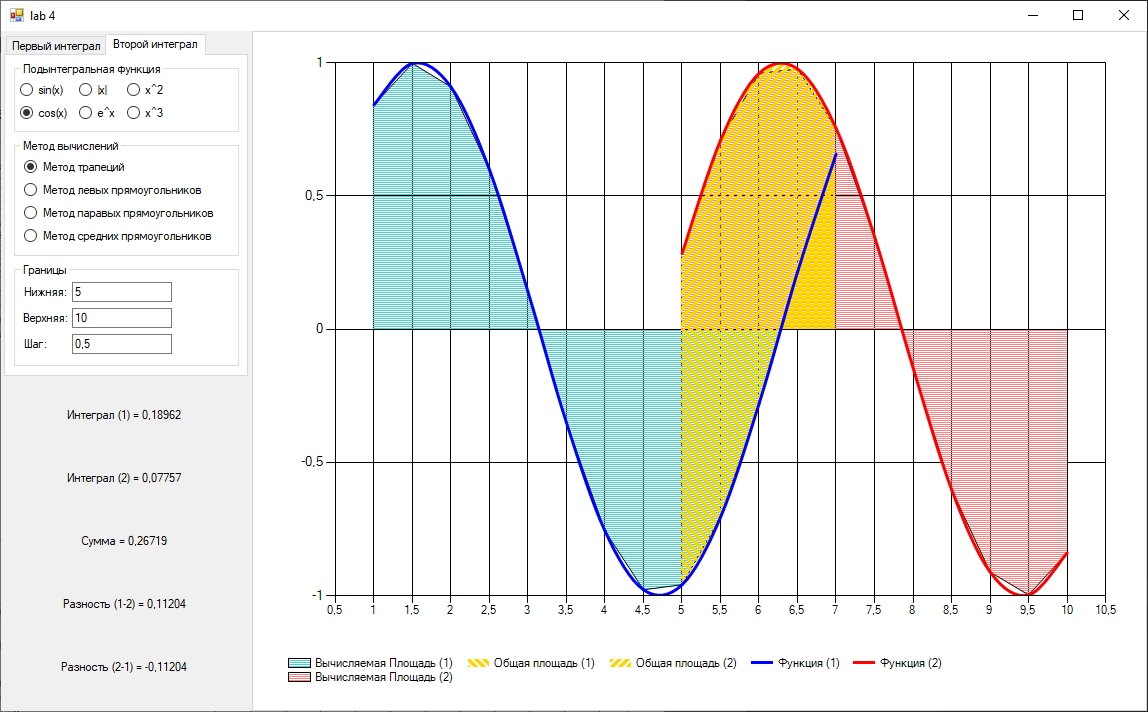


Рисунок 2 – Результат с синусом и косинусом.

*ПРИЛОЖЕНИЕ A*

Листинг исходных кодов программ

**Файл Lab4Form.cs**

using lab\_4.Integrals;

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 lab\_4

{

public partial class Lab4Form : Form

{

MegaIntegral integral1;

MegaIntegral integral2;

MegaIntegral integral1Area;

MegaIntegral integral2Area;

MutualAreaIntegals mutualAreaIntegals;

MegaIntegral.Func func1;

MegaIntegral.Func func2;

public Lab4Form()

{

InitializeComponent();

func1 = Math.Sin;

func2 = Math.Sin;

}

private void ReDrawGraph(object sender, EventArgs e)

{

bool bottom\_flag1 = double.TryParse(f1\_bottomBox.Text, out double bottom1);

bool top\_flag1 = double.TryParse(f1\_topBox.Text, out double top1);

bool step\_flag1 = double.TryParse(f1\_stepBox.Text, out double step1);

bool bottom\_flag2 = double.TryParse(f2\_bottomBox.Text, out double bottom2);

bool top\_flag2 = double.TryParse(f2\_topBox.Text, out double top2);

bool step\_flag2 = double.TryParse(f2\_stepBox.Text, out double step2);

if (bottom\_flag1 && top\_flag1 && step\_flag1 &&

bottom\_flag2 && top\_flag2 && step\_flag2)

{

integral1 = new MegaIntegral(ref chart1, "Функция (1)", func1, bottom1, top1, step1);

integral1.DrawFunc();

integral2 = new MegaIntegral(ref chart1, "Функция (2)", func2, bottom2, top2, step2);

integral2.DrawFunc();

if (integral1Area == null)

{

integral1Area = new MegaIntegral(ref chart1, "Вычисляемая Площадь (1)", func1, bottom1, top1, step1);

}

else

{

integral1Area.Function = func1;

integral1Area.Bottom = bottom1;

integral1Area.Top = top1;

integral1Area.Step = step1;

}

integral1Area.DrawFunc();

if (integral2Area == null)

{

integral2Area = new MegaIntegral(ref chart1, "Вычисляемая Площадь (2)", func2, bottom2, top2, step2);

}

else

{

integral2Area.Function = func2;

integral2Area.Bottom = bottom2;

integral2Area.Top = top2;

integral2Area.Step = step2;

}

integral2Area.DrawFunc();

mutualAreaIntegals = new MutualAreaIntegals(ref chart1,

"Общая площадь (1)", func1,

"Общая площадь (2)", func2,

bottom1, top1, step1,

bottom2, top2, step2);

mutualAreaIntegals.DrawMutualArea();

label7.Text = string.Format("Интеграл (1) = {0:0.00000}", integral1Area.CalcArea());

label8.Text = string.Format("Интеграл (2) = {0:0.00000}", integral2Area.CalcArea());

label9.Text = string.Format("Сумма = {0:0.00000}", integral1Area.CalcArea() + integral2Area.CalcArea());

label10.Text = string.Format("Разность (1-2) = {0:0.00000}", integral1Area.CalcArea() - integral2Area.CalcArea());

label11.Text = string.Format("Разность (2-1) = {0:0.00000}", integral2Area.CalcArea() - integral1.CalcArea());

}

}

private double Pow2(double x) => Math.Pow(x, 2);

private double Pow3(double x) => Math.Pow(x, 3);

#region Выбор первой функции

private void F1sin\_radioButton\_CheckedChanged(object sender, EventArgs e)

{

func1 = Math.Sin;

ReDrawGraph(sender, e);

}

private void F1cos\_radioButton\_CheckedChanged(object sender, EventArgs e)

{

func1 = Math.Cos;

ReDrawGraph(sender, e);

}

private void F1abs\_radioButton\_CheckedChanged(object sender, EventArgs e)

{

func1 = Math.Abs;

ReDrawGraph(sender, e);

}

private void F1exp\_radioButton\_CheckedChanged(object sender, EventArgs e)

{

func1 = Math.Exp;

ReDrawGraph(sender, e);

}

private void F1pow2\_radioButton\_CheckedChanged(object sender, EventArgs e)

{

func1 = Pow2;

ReDrawGraph(sender, e);

}

private void F1pow3\_radioButton\_CheckedChanged(object sender, EventArgs e)

{

func1 = Pow3;

ReDrawGraph(sender, e);

}

#endregion

#region Выбор второй функции

private void F2sin\_radioButton\_CheckedChanged(object sender, EventArgs e)

{

func2 = Math.Sin;

ReDrawGraph(sender, e);

}

private void F2cos\_radioButton\_CheckedChanged(object sender, EventArgs e)

{

func2 = Math.Cos;

ReDrawGraph(sender, e);

}

private void F2abs\_radioButton\_CheckedChanged(object sender, EventArgs e)

{

func2 = Math.Abs;

ReDrawGraph(sender, e);

}

private void F2exp\_radioButton\_CheckedChanged(object sender, EventArgs e)

{

func2 = Math.Exp;

ReDrawGraph(sender, e);

}

private void F2pow2\_radioButton\_CheckedChanged(object sender, EventArgs e)

{

func2 = Pow2;

ReDrawGraph(sender, e);

}

private void F2pow3\_radioButton\_CheckedChanged(object sender, EventArgs e)

{

func2 = Pow3;

ReDrawGraph(sender, e);

}

#endregion

#region Выбор площади первой функции

private void F1trMod\_radioButton\_CheckedChanged(object sender, EventArgs e)

{

if (integral1Area != null)

{

integral1Area.SetModTRAPEZOID\_METHOD();

ReDrawGraph(sender, e);

}

}

private void F1lrMod\_radioButton\_CheckedChanged(object sender, EventArgs e)

{

if (integral1Area != null)

{

integral1Area.SetModLEFT\_RECTANGLE\_METHOD();

ReDrawGraph(sender, e);

}

}

private void F1rrMod\_radioButton\_CheckedChanged(object sender, EventArgs e)

{

if (integral1Area != null)

{

integral1Area.SetModRIGHT\_RECTANGLE\_METHOD();

ReDrawGraph(sender, e);

}

}

private void F1mrMod\_radioButton\_CheckedChanged(object sender, EventArgs e)

{

if (integral1Area != null)

{

integral1Area.SetModMIDDLE\_RECTANGLE\_METHOD();

ReDrawGraph(sender, e);

}

}

#endregion

#region Выбор площади второй функции

private void F2trMod\_radioButton\_CheckedChanged(object sender, EventArgs e)

{

if (integral2Area != null)

{

integral2Area.SetModTRAPEZOID\_METHOD();

ReDrawGraph(sender, e);

}

}

private void F2lrMod\_radioButton\_CheckedChanged(object sender, EventArgs e)

{

if (integral2Area != null)

{

integral2Area.SetModLEFT\_RECTANGLE\_METHOD();

ReDrawGraph(sender, e);

}

}

private void F2rrMod\_radioButton\_CheckedChanged(object sender, EventArgs e)

{

if (integral2Area != null)

{

integral2Area.SetModRIGHT\_RECTANGLE\_METHOD();

ReDrawGraph(sender, e);

}

}

private void F2mrMod\_radioButton\_CheckedChanged(object sender, EventArgs e)

{

if (integral2Area != null)

{

integral2Area.SetModMIDDLE\_RECTANGLE\_METHOD();

ReDrawGraph(sender, e);

}

}

#endregion

}

}

**Файл Integrals.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms.DataVisualization.Charting;

namespace lab\_4

{

public class Integral

{

/// <summary>

/// Переччисление доступных режимов работы.

/// </summary>

private enum Mods { USUAL, LEFT\_RECTANGLE\_METHOD, MIDDLE\_RECTANGLE\_METHOD, RIGHT\_RECTANGLE\_METHOD, TRAPEZOID\_METHOD }

/// <summary>

/// Текущий режим рисованя.

/// </summary>

private Mods Mod = Mods.TRAPEZOID\_METHOD;

/// <summary>

/// Тип функции по которой рисуется график.

/// </summary>

/// <param name="x"></param>

/// <returns></returns>

//public delegate double Func(double x);

#region Выбор режима рисования

/// <summary>

/// Устанавливает отображение функции в обычный режим.

/// </summary>

public void SetModUSUAL() => Mod = Mods.USUAL;

/// <summary>

/// Устанавливает отображение функции в режим левых прямоугольников.

/// </summary>

public void SetModLEFT\_RECTANGLE\_METHOD() => Mod = Mods.LEFT\_RECTANGLE\_METHOD;

/// <summary>

/// Устанавливает отображение функции в режим средних прямоугольников.

/// </summary>

public void SetModMIDDLE\_RECTANGLE\_METHOD() => Mod = Mods.MIDDLE\_RECTANGLE\_METHOD;

/// <summary>

/// Устанавливает отображение функции в режим правых прямоугольников.

/// </summary>

public void SetModRIGHT\_RECTANGLE\_METHOD() => Mod = Mods.RIGHT\_RECTANGLE\_METHOD;

/// <summary>

/// Устанавливает отображение функции в трапеций.

/// </summary>

public void SetModTRAPEZOID\_METHOD() => Mod = Mods.TRAPEZOID\_METHOD;

#endregion

#region Поля класса

/// <summary>

/// Поле нижнего предела интеграла.

/// </summary>

private double bottom;

/// <summary>

/// Поле верхнего предела интеграла.

/// </summary>

private double top;

/// <summary>

/// Поле шага интеграла.

/// </summary>

private double step;

#endregion

#region Геттеры и сетторы класса

/// <summary>

/// Нижний предел интеграла.

/// </summary>

public double Bottom { get => bottom; set => bottom = value; }

/// <summary>

/// Верхний предел интеграла.

/// </summary>

public double Top { get => top; set => top = value; }

/// <summary>

/// Поле шага интеграла.

/// </summary>

public double Step { get => step; set => step = value; }

/// <summary>

/// Указатель на редактируемый chart.

/// </summary>

private Chart chart;

public Chart Chart { get => chart; set => chart = value; }

/// <summary>

/// Имя серии в chart.

/// </summary>

public string NameOfSeries { get; set; }

/// <summary>

/// Указатель на функцию.

/// </summary>

//public Func Function { get; set; }

#endregion

public Integral(ref Chart chartRef)

{

chart = chartRef;

}

public Integral(ref Chart chartRef, string nameOfSeries, /\*Func Function,\*/ double bottom, double top, double step)

{

chart = chartRef;

chart.Series[nameOfSeries].Points.Clear();

NameOfSeries = nameOfSeries;

this.bottom = bottom;

this.top = top;

this.step = step;

//this.Function = Function;

}

public virtual double StaticFunction(double x) => Math.Sin(x);

#region Вычисление точек

private List<DataPoint> GetFuncDots()

{

List<DataPoint> points = new List<DataPoint>();

switch (Mod)

{

case Mods.LEFT\_RECTANGLE\_METHOD:

{

for (double x = bottom; x <= top; x += step)

{

double y = StaticFunction(x);

points.Add(new DataPoint(x, y));

if (x + step <= top)

{

points.Add(new DataPoint(x + step, y));

}

}

break;

}

case Mods.MIDDLE\_RECTANGLE\_METHOD:

{

for (double x = bottom; x <= top; x += step)

{

double y = StaticFunction(x + step / 2);

points.Add(new DataPoint(x, y));

if (x + step <= top)

{

points.Add(new DataPoint(x + step, y));

}

}

break;

}

case Mods.RIGHT\_RECTANGLE\_METHOD:

{

for (double x = bottom; x <= top; x += step)

{

double y = StaticFunction(x + step);

points.Add(new DataPoint(x, y));

if (x + step <= top)

{

points.Add(new DataPoint(x + step, y));

}

}

break;

}

case Mods.TRAPEZOID\_METHOD:

{

for (double x = bottom; x <= top; x += step)

{

double y = StaticFunction(x);

points.Add(new DataPoint(x, y));

}

break;

}

default:

break;

}

return points;

}

#endregion

#region Рисование функции

/// <summary>

/// Отрисовывает Функцию на заданной серии

/// </summary>

/// <returns></returns>

public bool DrawFunc()

{

if (bottom < top && step != 0) //защита от зацикливания

{

chart.Series[NameOfSeries].Points.Clear();

List<DataPoint> points = GetFuncDots();

foreach (DataPoint point in points)

{

chart.Series[NameOfSeries].Points.AddXY(point.XValue, point.YValues[0]);

}

return true;

}

else

{

return false;

}

}

#endregion

#region Вычисление площади

public double CalcArea()

{

double area = 0;

switch (Mod)

{

case Mods.LEFT\_RECTANGLE\_METHOD:

{

for (double x = bottom; x <= top; x += step)

{

double y = Math.Abs(StaticFunction(x));

area += y \* step;

}

break;

}

case Mods.MIDDLE\_RECTANGLE\_METHOD:

{

for (double x = bottom; x <= top; x += step)

{

double y = StaticFunction(x + step / 2);

area += y \* step;

}

break;

}

case Mods.RIGHT\_RECTANGLE\_METHOD:

{

for (double x = bottom; x <= top; x += step)

{

double y = StaticFunction(x + step);

area += y \* step;

}

break;

}

case Mods.TRAPEZOID\_METHOD:

{

for (double x = bottom; x <= top; x += step)

{

double y = StaticFunction(x);

area += step \* (StaticFunction(x) + StaticFunction(x + step)) / 2;

}

break;

}

}

return area;

}

#endregion

}

}

**Файл MegaIntegral.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms.DataVisualization.Charting;

namespace lab\_4.Integrals

{

public class MegaIntegral : Integral

{

/// <summary>

/// Тип функция по которой считается интеграл.

/// </summary>

/// <param name="x"></param>

/// <returns></returns>

public delegate double Func(double x);

/// <summary>

/// Функция по которой рисуется интеграл.

/// </summary>

public Func Function { get; set; }

public MegaIntegral(ref Chart chartRef) : base(ref chartRef)

{

}

public MegaIntegral(ref Chart chartRef, string nameOfSeries,

Func Function, double bottom, double top, double step) : base

(ref chartRef, nameOfSeries, bottom, top, step)

{

this.Function = Function;

}

public override double StaticFunction(double x) => Function(x);

}

}

**Файл MutualAreaIntegals.cs**

using lab\_4.Integrals;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms.DataVisualization.Charting;

namespace lab\_4

{

public class MutualAreaIntegals : MegaIntegral

{

public MutualAreaIntegals(ref Chart chart,

string ser1name, Func func1,

string ser2name, Func func2,

double f1bottom, double f1top, double f1step,

double f2bottom, double f2top, double f2step) : base(ref chart)

{

if (f1bottom > f2bottom)

{

Bottom = f1bottom;

}

else

{

Bottom = f2bottom;

}

if (f1top < f2top)

{

Top = f1top;

}

else

{

Top = f2top;

}

SerName1 = ser1name;

SerName2 = ser2name;

Func1 = func1;

Func2 = func2;

Step1 = f1step;

Step2 = f2step;

}

/// <summary>

/// Шаг первой функции.

/// </summary>

public double Step1 { get; set; }

/// <summary>

/// Шаг второй функции.

/// </summary>

public double Step2 { get; set; }

/// <summary>

/// Имя серии на которой будет отрисовываться первая функция.

/// </summary>

public string SerName1 { get; set; }

/// <summary>

/// Имя серии на которой будет отрисовываться вторая функция.

/// </summary>

public string SerName2 { get; set; }

/// <summary>

/// Функция по которой будет отрисовываться общая площадь первой функции.

/// </summary>

public Func Func1 { get; set; }

/// <summary>

/// Функция по которой будет отрисовываться общая площадь второй функции.

/// </summary>

public Func Func2 { get; set; }

/// <summary>

/// Рисует общую площадь двух функций.

/// </summary>

public void DrawMutualArea()

{

NameOfSeries = SerName1;

Function = Func1;

Step = Step1;

DrawFunc();

NameOfSeries = SerName2;

Function = Func2;

Step = Step2;

DrawFunc();

}

/// <summary>

/// Возвращает общуюю площадь методом левых прямоугольников.

/// </summary>

/// <returns></returns>

public double GetLRSquare()

{

double answer = 0;

SetModLEFT\_RECTANGLE\_METHOD();

NameOfSeries = SerName1;

Function = Func1;

Step = Step1;

answer += CalcArea();

NameOfSeries = SerName2;

Function = Func2;

Step = Step2;

answer += CalcArea();

return answer;

}

/// <summary>

/// Возвращает общуюю площадь методом средних прямоугольников.

/// </summary>

/// <returns></returns>

public double GetMRSquare()

{

SetModMIDDLE\_RECTANGLE\_METHOD();

double answer = 0;

NameOfSeries = SerName1;

Function = Func1;

Step = Step1;

answer += CalcArea();

NameOfSeries = SerName2;

Function = Func2;

Step = Step2;

answer += CalcArea();

return answer;

}

/// <summary>

/// Возвращает общуюю площадь методом правых прямоугольников.

/// </summary>

/// <returns></returns>

public double GetRRSquare()

{

SetModRIGHT\_RECTANGLE\_METHOD();

double answer = 0;

NameOfSeries = SerName1;

Function = Func1;

Step = Step1;

answer += CalcArea();

NameOfSeries = SerName2;

Function = Func2;

Step = Step2;

answer += CalcArea();

return answer;

}

/// <summary>

/// Возвращает общуюю площадь методом трапеций.

/// </summary>

/// <returns></returns>

public double GetTRSquare()

{

SetModTRAPEZOID\_METHOD();

double answer = 0;

NameOfSeries = SerName1;

Function = Func1;

Step = Step1;

answer += CalcArea();

NameOfSeries = SerName2;

Function = Func2;

Step = Step2;

answer += CalcArea();

return answer;

}

}

}