Изображение выглядит как цепь

Автоматически созданное описание

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-Петербургский государственный технологический институт (технический университет)»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| УГНС | | 09.00.00 | Информатика и вычислительная техника | | |
| Направление подготовки | | 09.03.01 | Информатика и вычислительная техника | | |
| Направленность (профиль) | |  | Автоматизированные системы обработки информации и управления | | |
| Форма обучения | |  | очная | | |
|  | |  |  | | |
| Факультет | |  | Информационных технологий и управления | | |
| Кафедра | |  | Систем автоматизированного проектирования и управления | | |
| Учебная дисциплина | |  | Разработка программных систем | | |
| Курс | II | | | Группа | 403 |

Отчёт по лабораторной работе № 3

Вариант № 9

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исполнитель: |  |  |  |  |
| обучающийся группы 403 |  |  |  | Шишко Даниил Юрьевич |
|  |  | (дата, подпись) |  |  |
|  |  |  |  |  |
| Проверили: |  |  |  | Корниенко Иван Григорьевич |
|  |  | (дата, подпись) |  | Федин Алексей Константинович |

**Оглавление**

[Постановка задачи 3](#_Toc96790610)

[Исходные данные 3](#_Toc96790611)

[Особые ситуации 3](#_Toc96790612)

[Математические методы и алгоритмы решения задач 3](#_Toc96790613)

[Форматы представления данных 3](#_Toc96790614)

[Структура программы 5](#_Toc96790615)

[Блок-схемы алгоритмов программы 6](#_Toc96790616)

[Описание хода выполнения работы 6](#_Toc96790617)

[Результат работы программы 7](#_Toc96790618)

[Исходный текст программы 8](#_Toc96790619)

[Документирование и комментирование исходного текста 18](#_Toc96790620)

# Постановка задачи

Необходимо написать приложение с использованием технологии WinForms для построения графика функции и вывода таблицы значений функции. Пользователь задает правую и левую границу, шаг, коэффициенты (при их наличии). При невозможности построить график функции в заданном интервале пользователю выдается предупреждение об этом с предложением сменить границы построения. Если график функции из-за коэффициентов вырождается в точку или не может быть построен пользователь также видит предупреждение.

# Исходные данные

В качестве исходных данных программа использует, вводимое пользователем константу ‘a’, левую границу графика и правую границу графика, также шаг, с периодичностью которого будет меняться ‘x’.

# Особые ситуации

Необходимо рассмотреть следующие особые ситуации.

* Не введены один или несколько изначальных данных.
* График вырождается в точку.
* Левая граница больше правой, или правая меньше левой.
* Значение правой границы при отрицательной константе больше константы ‘a’.
* Значение левой границы при положительной константе меньше ‘a’.

# Математические методы и алгоритмы решения задач

Находим y при известных x, a.

# Форматы представления данных

Программа использует следующие переменные

Таблица 1 – Переменные, используемые в программе

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| welcome | TextBox | Описание создателя |
| information | TextBox | Описание цели работы |
| dontShowAgreement | TextBox | Описание согласия на непоявление |
| close | Button | Закрыть форму |
| file | StreamWriter | Взаимодействие с файлом |
| Таблица 1 – Переменные, используемые в программе | | |
| multiply | int | Приближение изображения |
| error | string | Ошибка ввода |
| errorOfConst | string | Ошибка в константе |
| borderError | string | Ошибка в барьере |
| \_concetrarionService | IChirngauzSquareInfrastructure | Сервис для управления нахождениям точек функции |
| constA | string | Константа «а» |
| leftBorder | string | Правая граница |
| rightBorder | string | Левая граница |
| step | string | Шаг «х» |
| graphics | Graphics | Класс для рисования |
| points | List<ChirgauzSquareModel> | Значения функции в точках |
| somePointsUp | PointF[] | Значения функции при положительных y |
| somePointsDown | PointF[] | Значения функции при отрицательных y |
| centerX | int | Центр по «х» |
| centerY | int | Центр по «у» |
| iterator | int | Счётчик |
| pen | Pen | Ручка для рисования |
| drawFont | Font | Шрифт для рисования |
| drawBrush | SolidBrush | Кисть для рисования |
| yesOrNo | bool | Число из файла про соглашение |
| table | Table | Форма для таблицы |
| saveFileDialog | SaveFileDialog | Сохранение файла |
| filePath | string | Путь к файлу |
| openFileDialog | OpenFileDialog | Чтение файла |
| fileStream | Stream | Поток файла |
| package | ExcelPackage | Взаимодействие с Excel |
| sheet | ExcelWorksheet | Доступ к ячейкам таблицы |
| greeting | Greeting | Форма приветствия |
| tableWithData | PictureBox | Таблица значений функции |
| counter | int | Счётчик номеров строк |
| back | Buttom | Закрытие формы |
| save | Buttom | Сохранение таблицы |
| x | double | Икс функции |
| y | double | Игрик функции |
| numerator | double | Числитель |
| denominator | double | Знаменатель |

Таблица 3 – Пользовательские типы

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| ChirgauzSquareModel | Значение функции в точке |

В файле каждое сообщения хранятся в одной строке.

# Структура программы

Программа разделена на следующие модули:

1. Greeting.cs – приветствие;
2. MainForm.cs – главная форма;
3. Table.cs – таблица значений функции в точках;
4. IChirngauzSquare.cs – интерфейс для создания точек функции;
5. ModelChirgauzSquare.cs – точки функции;
6. ServicesChirngauzSquare.cs – реализация расчёта значений функции.

Таблица 4 – Функции, составляющие программу

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| GreetingInner | Создание элементов на форме приветствия |
| Close\_Click | Закрытие формы приветствия |
| CheckData | Проверка правильности введенных данных |
| Calculate\_Click | Расчёт точек функции |
| GetCalculations | Возвращает значения точек функции |
| informationToolStripMenuItem\_Click | Вызывает форму приветствия с условием |
| OpenGreetingsForm | Открывает форму приветствия |
| getInitialToolStripMenuItem\_Click | Считать начальные значения из файла |
| saveInitialToolStripMenuItem\_Click | Сохранить изначальные значения в файл |
| saveInExcelToolStripMenuItem\_Click | Сохранить результаты в Excel таблицу |
| Table\_Click | Открывает таблицу со значениями |
| CreateGrid | Создание таблицы |
| DataGridView\_MouseWheel | Прокрутка таблицы с помощью колёсика |
| FillDataGridWithData | Заполнить таблицу значениями |
| CreateButtons | Создание кнопок на форме |
| CloseClick | Закрыть форму |
| SaveClick | Сохранить значения в таблице |
| GetPoints | Получить значения графика функции |
| GetY | Получить значения точки y по x |

# Блок-схемы алгоритмов программы



Рисунок 1 – Блок-схема алгоритма решения

# Описание хода выполнения работы

1. В ходе работы было создано решение (Solution) в интегрированной среде разработки Microsoft Visual Studio 2015. В нём был создан проект.
2. Создано оформление главной формы, реализовано масштабирование графика.
3. Создана форма приветствия.
4. Создана форма таблицы значений функции, реализована прокрутка значений функции.
5. Исправлены возникающие ошибки при вводе некорректных данных.
6. PictureBox изменен на Chart и DataGridView вместо графика и таблциы соответственно.

# Результат работы программы

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3 – Результат работы программы

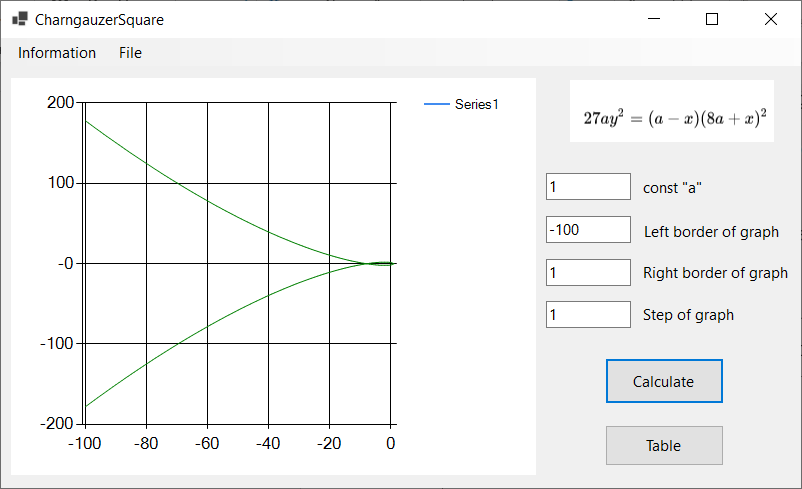


Рисунок 6 – Результат работы программы

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 7 – Результат работы программы

# Исходный текст программы

[ Начало программы ---]

[Greetings]

namespace ChirngauzerSquareGraph

{

public partial class Greetings : Form

{

public Greetings()

{

InitializeComponent();

this.StartPosition = FormStartPosition.CenterParent;

GreetingInner();

}

CheckBox dontShowAgain;

private void GreetingInner()

{

this.Size = new Size(500, 350);

this.MinimumSize = this.Size;

this.MaximumSize = this.Size;

this.TopMost = true;

TextBox welcome = new TextBox();

welcome.ReadOnly = true;

welcome.Text = "Created by: Shishko Daniil Yrevich";

welcome.Size = new Size(this.Width / 2, 20);

welcome.Location = new Point(this.Width / 10, 5);

welcome.BorderStyle = BorderStyle.None;

this.Controls.Add(welcome);

TextBox information = new TextBox();

information.ReadOnly = true;

information.Text = "Option: 9" + Environment.NewLine + "Aim: Create Win “+

“Forms application to " +

"plotting a function graph and outputting a table of function values.”+ “User sets " +

"rigth and left borders, step, coefficients (if presents). If unable to " +

"plot a finction graph in seted gap user get warning " +

"about this to change borders. If function graph in cause of" +

"coefficients becomes a dot or can't be ploted user also sees warning.";

information.Multiline = true;

information.TextAlign = HorizontalAlignment.Left;

information.Size = new Size(this.Width / 2 + 150, this.Height / 2);

information.Location = new Point(welcome.Location.X - 5, welcome.Height + 10);

information.BorderStyle = BorderStyle.None;

this.Controls.Add(information);

dontShowAgain = new CheckBox();

dontShowAgain.Enabled = true;

dontShowAgain.Size = new Size(20, welcome.Height);

dontShowAgain.Location = new Point(welcome.Location.X, information.Height + 20

+ welcome.Height);

dontShowAgain.Appearance = Appearance.Normal;

using (StreamReader file = new StreamReader("Agreement.txt"))

{

if (int.Parse(file.ReadLine()) == 1)

dontShowAgain.Checked = true;

else

dontShowAgain.Checked = false;

}

this.Controls.Add(dontShowAgain);

TextBox dontShowAgreement = new TextBox();

dontShowAgreement.ReadOnly = true;

dontShowAgreement.TextAlign = HorizontalAlignment.Left;

dontShowAgreement.Text = "Check if you don't want see this again";

dontShowAgreement.Size = new Size(welcome.Width + 20, 0);

dontShowAgreement.Location = new Point(welcome.Location.X + 30,

dontShowAgain.Location.Y + 3);

dontShowAgreement.BorderStyle = BorderStyle.None;

this.Controls.Add(dontShowAgreement);

Button close = new Button();

close.Text = "OK";

close.Location = new Point(dontShowAgain.Location.X +

dontShowAgreement.Size.Width + dontShowAgain.Size.Width + 20,

dontShowAgain.Location.Y);

close.Size = new Size(40, 30);

this.Controls.Add(close);

close.Click += Close\_Click;

}

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

{

if (dontShowAgain.Checked)

{

using (StreamWriter file = new StreamWriter("Agreement.txt", false))

{

file.WriteLine(1);

}

}

if (!dontShowAgain.Checked)

{

using (StreamWriter file = new StreamWriter("Agreement.txt", false))

{

file.WriteLine(0);

}

}

this.Close();

}

}

}

[Greetings]

[MainForm]

using System.Windows.Forms.DataVisualization.Charting;

using OfficeOpenXml;

using Services;

namespace ChirngauzerSquareGraph

{

public partial class MainForm : Form

{

private readonly string error = "Wrong data";

private readonly string errorOfConst = "Graph is dot";

private readonly string borderError = "Wrong border edges";

private readonly ICharngauzerSquare \_concentrationService;

Chart chart;

public MainForm(ICharngauzerSquare concentrationService)

{

\_concentrationService = concentrationService ?? throw new

ArgumentNullException(nameof(concentrationService));

InitializeComponent();

ConstA.Text = "1";

LeftBorder.Text = "-100";

RightBorder.Text = "1";

Step.Text = "1";

Bitmap bmp = new Bitmap(@"D:\4

семестр\РПС\ChirngauzerSquareGraph\ChirngauzerSquareGraph\Equation.png");

PictureOfGraph.Image = bmp;

if (OpenGreetingsForm() == 0)

{

Greetings greeting = new Greetings();

greeting.StartPosition = FormStartPosition.CenterParent;

greeting.Show();

}

}

private void CheckData()

{

WrongData.Clear();

string constA = "", leftBorder = "", rightBorder = "", step = "";

if (ConstA.Text == "" || !double.TryParse(ConstA.Text, out double

uselessResult))

{

WrongData.SetError(ConstA, error);

}

if (ConstA.Text != "" && double.TryParse(ConstA.Text, out uselessResult))

{

if (double.Parse(ConstA.Text) == 0)

{

WrongData.SetError(ConstA, errorOfConst);

}

else

{

constA = ConstA.Text;

}

}

if (LeftBorder.Text == "" || !double.TryParse(LeftBorder.Text, out

uselessResult))

WrongData.SetError(LeftBorder, error);

else

{

leftBorder = LeftBorder.Text;

}

if (RightBorder.Text == "" || !double.TryParse(RightBorder.Text, out

uselessResult))

WrongData.SetError(RightBorder, error);

else

{

rightBorder = RightBorder.Text;

}

if (Step.Text == "" | !double.TryParse(Step.Text, out uselessResult) |

(int)uselessResult <= 0)

WrongData.SetError(Step, error);

else

{

step = Step.Text;

}

if (ConstA.Text != "" && ConstA.Text != "0" && WrongData.GetError(LeftBorder)

== "" && WrongData.GetError(RightBorder) == "")

{

if (double.Parse(constA) > 0)

if (double.Parse(rightBorder) > double.Parse(constA))

WrongData.SetError(RightBorder, error);

if (double.Parse(constA) < 0)

if (double.Parse(leftBorder) < double.Parse(constA))

WrongData.SetError(LeftBorder, error);

}

if (RightBorder.Text != "" && double.TryParse(RightBorder.Text, out

uselessResult))

{

if (LeftBorder.Text != "" && double.TryParse(LeftBorder.Text, out

uselessResult))

{

if (double.Parse(LeftBorder.Text) > double.Parse(RightBorder.Text))

{

WrongData.SetError(LeftBorder, borderError);

WrongData.SetError(RightBorder, borderError);

}

}

}

}

private List<ModelCharngauzerSquare> GetCalculations()

{

CheckData();

if (WrongData.GetError(LeftBorder) == "" && WrongData.GetError(RightBorder) ==

""

&& WrongData.GetError(ConstA) == "" && WrongData.GetError(Step) == "")

return \_concentrationService.GetPoints(double.Parse(ConstA.Text),

double.Parse(LeftBorder.Text), double.Parse(RightBorder.Text),

double.Parse(Step.Text));

return null;

}

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

{

CheckData();

if (WrongData.GetError(LeftBorder) == "" && WrongData.GetError(RightBorder) ==

""

&& WrongData.GetError(ConstA) == "" && WrongData.GetError(Step) == "")

{

var points = GetCalculations();

PointF[] somePointsUp = new PointF[points.Count / 2];

PointF[] somePointsDown = new PointF[points.Count / 2];

int centerX = 0;

int centerY = 0;

int iterator = 0;

for (int i = 0; i < points.Count; i += 2)

{

somePointsUp[iterator] = new PointF((float)(points[i].x + centerX),

((float)points[i].y + centerY));

somePointsDown[iterator] = new PointF((float)(points[i + 1].x + centerX),

(float)(points[i + 1].y + centerY));

iterator++;

}

if (chart != null)

chart.Dispose();

CreateChart();

var seriesUp = new Series

{

Name = "ChirngauzSquareUp",

Color = Color.Green,

IsVisibleInLegend = false,

IsXValueIndexed = true,

ChartType = SeriesChartType.Line,

LegendText = "True",

};

var seriesDown = new Series

{

Name = "ChirngauzSquareDown",

Color = Color.Green,

IsVisibleInLegend = false,

IsXValueIndexed = true,

ChartType = SeriesChartType.Line

};

chart.Series.Add(seriesUp);

chart.Series.Add(seriesDown);

chart.Series.Add(new Series { ChartType = SeriesChartType.Line });

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

{

seriesUp.Points.AddXY(somePointsUp[i].X, somePointsUp[i].Y);

seriesDown.Points.AddXY(somePointsDown[i].X, somePointsDown[i].Y);

}

chart.Invalidate();

}

}

private void CreateChart()

{

ChartArea chartArea1 = new ChartArea();

Legend legend1 = new Legend();

chart = new System.Windows.Forms.DataVisualization.Charting.Chart();

chartArea1.Name = "ChartArea";

chart.ChartAreas.Add(chartArea1);

legend1.Name = "Graph";

chart.Legends.Add(legend1);

chart.Location = new Point(10, 40);

chart.Name = "chart";

chart.Size = new Size(((this.Width \* 2) / 3) - 20, this.Height - 100);

chart.TabIndex = 0;

chart.Text = "chart";

chart.Series.Clear();

this.Controls.Add(chart);

}

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

{

Greetings greeting = new Greetings();

greeting.StartPosition = FormStartPosition.CenterParent;

greeting.Show();

}

private int OpenGreetingsForm()

{

FileStream createFile = null;

StreamReader file = null;

try

{

file = new StreamReader("Agreement.txt");

}

catch

{

createFile = File.Create("Agreement.txt");

createFile.Close();

file = new StreamReader("Agreement.txt");

}

int Agreement = 0;

if (file != null)

{

string yesOrNo = file.ReadLine();

if (yesOrNo != null)

Agreement = int.Parse(yesOrNo);

else

Agreement = 0;

}

file.Close();

return Agreement;

}

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

{

using (OpenFileDialog openFileDialog = new OpenFileDialog())

{

openFileDialog.InitialDirectory = "d:\\4 семестр\\РПС";

openFileDialog.Filter = "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*";

openFileDialog.FilterIndex = 2;

openFileDialog.RestoreDirectory = true;

if (openFileDialog.ShowDialog() == DialogResult.OK)

{

var filePath = openFileDialog.FileName;

var fileStream = openFileDialog.OpenFile();

using (StreamReader reader = new StreamReader(fileStream))

{

ConstA.Text = reader.ReadLine();

LeftBorder.Text = reader.ReadLine();

RightBorder.Text = reader.ReadLine();

Step.Text = reader.ReadLine();

}

}

}

}

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

{

CheckData();

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.InitialDirectory = "d:\\4 семестр\\РПС";

saveFileDialog.Filter = "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*";

if (WrongData.GetError(LeftBorder) == "" && WrongData.GetError(RightBorder) ==

""

&& WrongData.GetError(ConstA) == "" && WrongData.GetError(Step) == "")

{

if (saveFileDialog.ShowDialog() == DialogResult.OK)

{

var filePath = saveFileDialog.FileName;

using (StreamWriter file = new StreamWriter(filePath, false))

{

if (int.Parse(ConstA.Text) == 0)

{

WrongData.SetError(ConstA, errorOfConst);

}

else

{

file.WriteLine(double.Parse(ConstA.Text));

file.WriteLine(double.Parse(LeftBorder.Text));

file.WriteLine(double.Parse(RightBorder.Text));

file.WriteLine(double.Parse(Step.Text));

}

}

}

}

}

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

{

CheckData();

if (WrongData.GetError(LeftBorder) == "" && WrongData.GetError(RightBorder) ==

""

&& WrongData.GetError(ConstA) == "" && WrongData.GetError(Step) == "")

{

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.InitialDirectory = "d:\\4 семестр\\РПС";

saveFileDialog.Filter = "Excel files (\*.xlsx)|\*.xlsx|All files

(\*.xlsx)|\*.xlsx";

saveFileDialog.FilterIndex = 2;

saveFileDialog.RestoreDirectory = true;

var filePath = "";

if (saveFileDialog.ShowDialog() == DialogResult.OK)

{

filePath = saveFileDialog.FileName;

}

ExcelPackage.LicenseContext = LicenseContext.NonCommercial;

var package = new ExcelPackage();

var sheet = package.Workbook.Worksheets.Add("GraphPoint");

sheet.Cells[1, 1].Value = "a - "; sheet.Cells[1, 2].Value =

double.Parse(ConstA.Text);

sheet.Cells[2, 1].Value = "Left border - "; sheet.Cells[2, 2].Value =

double.Parse(LeftBorder.Text);

sheet.Column(1).Width = 14;

sheet.Column(2).Width = 14;

sheet.Cells[3, 1].Value = "Right border - "; sheet.Cells[3, 2].Value =

double.Parse(RightBorder.Text);

sheet.Cells[4, 1].Value = "Step - "; sheet.Cells[4, 2].Value =

double.Parse(Step.Text);

sheet.Cells[5, 1, 5, 2].LoadFromArrays(new object[][] { new[] { "X", "Y" }

});

sheet.Cells[5, 1].Style.HorizontalAlignment =

OfficeOpenXml.Style.ExcelHorizontalAlignment.Center;

sheet.Cells[5, 2].Style.HorizontalAlignment =

OfficeOpenXml.Style.ExcelHorizontalAlignment.Center;

var points = GetCalculations();

int iterator = 0;

for (int i = 0; i < points.Count; i+=2)

{

sheet.Cells[iterator + 6, 1].Value = points[i].x;

sheet.Cells[iterator + 6, 2].Value = points[i].y;

iterator++;

}

iterator = 0;

for(int i = 1; i < points.Count; i += 2)

{

sheet.Cells[iterator + 6, 3].Value = points[i].y;

iterator++;

}

var chart = (ExcelLineChart)graph.Drawings.AddChart("Square",

eChartType.Line);

chart.Legend.Position = eLegendPosition.Right;

chart.Legend.Add();

chart.SetPosition(1, 0, 1, 0);

chart.SetSize(1000, 600);

int count = points.Count / 2;

ExcelRangeBase excelFirstRangeBase = sheet.Cells["B6:B" + count.ToString()];

ExcelRangeBase excelSecondRangeBase = sheet.Cells["C6:C" +

count.ToString()];

chart.Series.Add(excelFirstRangeBase);

chart.Series.Add(excelSecondRangeBase);

File.WriteAllBytes(filePath, package.GetAsByteArray()); }

}

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

{

Table table = new Table(GetCalculations());

table.Show();

}

}

}

[MainForm]

[Table]

using Services;

namespace ChirngauzerSquareGraph

{

public partial class Table : Form

{

private DataGridView dataGridView = null;

public Table(List<ModelCharngauzerSquare> data)

{

InitializeComponent();

CreateGrid();

FillDataGridWithData(data);

CreateButtons();

}

private void CreateGrid()

{

this.Size = new Size(500, 350);

this.MinimumSize = this.Size;

this.MaximumSize = this.Size;

dataGridView = new DataGridView();

dataGridView.AllowUserToAddRows = false;

dataGridView.AllowUserToDeleteRows = false;

dataGridView.AllowUserToResizeRows = false;

dataGridView.AllowUserToOrderColumns = false;

dataGridView.AllowUserToResizeColumns = false;

dataGridView.RowHeadersVisible = false;

dataGridView.ReadOnly = true;

dataGridView.Location = new Point(10, 10);

dataGridView.Size = new Size(((this.Width \* 2) / 3) - 39, this.Height - 80);

var count = new DataGridViewTextBoxColumn();

count.Name = "Count";

count.HeaderText = "";

count.Width = 60;

dataGridView.Columns.Add(count);

var columnX = new DataGridViewTextBoxColumn();

columnX.Name = "X";

columnX.HeaderText = "X";

columnX.Width = 105;

dataGridView.Columns.Add(columnX);

var columnY = new DataGridViewTextBoxColumn();

columnY.Name = "Y";

columnY.Width = 105;

columnY.HeaderText = "Y";

dataGridView.Columns.Add(columnY);

this.Controls.Add(dataGridView);

dataGridView.MouseWheel += new MouseEventHandler(DataGridView\_MouseWheel);

}

private void DataGridView\_MouseWheel(object sender, MouseEventArgs e)

{

int currentIndex = this.dataGridView.FirstDisplayedScrollingRowIndex;

int scrollLines = SystemInformation.MouseWheelScrollLines;

if (e.Delta > 0)

{

this.dataGridView.FirstDisplayedScrollingRowIndex

= Math.Max(0, currentIndex - scrollLines);

}

else if (e.Delta < 0)

{

this.dataGridView.FirstDisplayedScrollingRowIndex

= currentIndex + scrollLines;

}

}

private void FillDataGridWithData(List<ModelCharngauzerSquare> data)

{

PointF[] somePointsUp = new PointF[data.Count / 2];

PointF[] somePointsDown = new PointF[data.Count / 2];

int centerX = 0;

int centerY = 0;

int iterator = 0;

for (int i = 0; i < data.Count; i += 2)

{

somePointsUp[iterator] = new PointF((float)(data[i].x + centerX),

((float)data[i].y + centerY));

somePointsDown[iterator] = new PointF((float)(data[i + 1].x + centerX),

(float)(data[i + 1].y + centerY));

iterator++;

}

iterator = 0;

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

{

dataGridView.Rows.Add(iterator.ToString(), Math.Round(somePointsUp[i].X, 4),

Math.Round(somePointsUp[i].Y, 4));

iterator++;

dataGridView.Rows.Add(iterator.ToString(), Math.Round(somePointsDown[i].X,

4), Math.Round(somePointsDown[i].Y, 4));

iterator++;

}

}

private void CreateButtons()

{

Button save = new Button();

save.Location = new Point((this.Width - dataGridView.Width) / 4 +

dataGridView.Width, this.Height / 4);

save.Size = new Size(60, 40);

save.Text = "Save";

save.Click += SaveClick;

this.Controls.Add(save);

Button close = new Button();

close.Location = new Point(save.Location.X, save.Location.Y + save.Height +

20);

close.Size = save.Size;

close.Text = "Return";

close.Click += CloseClick;

this.Controls.Add(close);

}

private void CloseClick(object sender, EventArgs e)

{

this.Close();

}

private void SaveClick(object sender, EventArgs e)

{

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.InitialDirectory = @"d:\4 семестр\РПС";

saveFileDialog.Filter = "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*";

if (saveFileDialog.ShowDialog() == DialogResult.OK)

{

var filePath = saveFileDialog.FileName;

StreamWriter file = new StreamWriter(filePath, false);

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

{

DataGridViewRow row = dataGridView.Rows[i];

file.WriteLine(row.Cells[1].Value + " " + row.Cells[2].Value);

}

file.Close();

}

}

}

}

[Table]

[ICharngauzerSquare]

using System.Collections.Generic;

namespace Services

{

public interface ICharngauzerSquare

{

public List<ModelCharngauzerSquare> GetPoints(double a, double leftBorder, double rigthBorder, double step);

}

}

[ICharngauzerSquare]

[ModelCharngauzerSquare]

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Services

{

public class ModelCharngauzerSquare

{

public double x { get; set; }

public double y { get; set; }

public ModelCharngauzerSquare(double innerX, double innerY)

{

x = innerX;

y = innerY;

}

}

}

[ModelCharngauzerSquare]

[ServicesCharngauzerSquare]

using System;

using System.Collections.Generic;

namespace Services

{

public class ServicesCharngauzerSquare : ICharngauzerSquare

{

public List<ModelCharngauzerSquare> GetPoints(double a, double leftBorder,

double rigthBorder, double step)

{

List<ModelCharngauzerSquare> points = new

List<ModelCharngauzerSquare>();

for (double x = leftBorder; x <= rigthBorder; x += step)

{

double y = GetY(x, a);

points.Add(new ModelCharngauzerSquare((double)x, y));

points.Add(new ModelCharngauzerSquare((double)x, -y));

}

return points;

}

private double GetY(double x, double a)

{

if (a > 0)

if (x > a)

throw new ArgumentOutOfRangeException("If 'a' greater than 0,”+

“'x' must be less than 'a'");

if (a < 0)

if (x < a)

throw new ArgumentOutOfRangeException("If 'a' less than 0, 'x'”+

“must be greater than 'a'");

if (x + 0.1 > a && x - 0.1 < a)

return 0;

double y, numerator, denominator;

denominator = 27 \* a;

double leftTemp = a - x;

double rightTemp = (8 \* a) + x;

numerator = leftTemp \* Math.Pow(rightTemp, 2);

y = Math.Sqrt(numerator / denominator);

return y;

}

}

}

[ServicesCharngauzerSquare]

[ --- Конец программы]

# Документирование и комментирование исходного текста