

Приднестровский государственный университет им. Т.Г. Шевченко

физико-математический факультет

кафедра прикладной математики и информатики

ЛАБОРАТОРНАЯ РАБОТА № 10

по дисциплине:
«Системы программирования»

Тема:
«Графика и визуализация»

РАЗРАБОТАЛИ:

ст. преподаватель кафедры ПМИИ
Великодный В.И.

ст. преподаватель кафедры ПМИИ
Калинкова Е.В.

Цель работы:

Изучить основные принципы и средства обработки растровых изображений с использованием языка C#. Изучить средства отображения графиков и диаграмм. Закрепить полученные знания при составлении программ при решении практических задач.

Теоретическая часть

Работа с изображениями

Класс Image

В платформе .NET и, в частности, в Windows Forms за представление изображений отвечает абстрактный класс `System.Drawing.Image`. Он соответствует как растровым (bitmap), так и векторным изображениям. Из-за того, что эти два вида изображений принципиально отличаются, класс `Image` содержит только самые основные свойства и методы.

Наиболее полезны свойства `Width` и `Height`, возвращающие ширину и высоту изображения в пикселях. Для загрузки изображения из файла и сохранения в файл используются:

- статический метод `Image.FromFile(имя_файла);`
- метод `Save(имя_файла).`

(Также поддерживается загрузка и сохранение изображений в потоки ввода/вывода.)

Изображение может быть любого из перечисленных форматов: BMP, JPEG, PNG, GIF, TIFF. Если файл не найден, выбрасывается исключение `System.IO.FileNotFoundException`. Формат открываемого файла определяется по расширению. При сохранении тип указывается при помощи перечисления `System.Drawing.Imaging.ImageFormat`.

Пример.

Откроем файл «image.jpeg» и сохраним его в формате PNG.

Возможно, для файла придётся указать полный путь. Например, @"d:\Images\image.jpeg" в Windows или @"/home/user/Images/image.jpeg" в Linux.

```
using System;
using System.IO;
using System.Drawing;
using System.Drawing.Imaging;

class Program
{
    public static void Main()
    {
        Image img;
        try
        {
            img = Image.FromFile("image.jpeg");
        }
        catch (FileNotFoundException)
        {
            Console.WriteLine("Файл не найден");
            return;
        }
        img.Save("image.png", ImageFormat.Png);
        Console.WriteLine("Файл сохранён!");
    }
}
```

```
}  
}
```

(Для запуска приложения потребуется подключить сборку `System.Drawing`, которая по умолчанию отключена у консольных программ.)

Растровые изображения

У класса `Image` есть два неабстрактных потомка:

- `System.Drawing.Bitmap` – представляет растровые изображения,
- `System.Drawing.Metafile` – представляет векторные изображения (то есть, хранит не информацию о пикселях, а последовательность действий для построения рисунка).

В рассмотренном ранее примере реальный тип хранимого экземпляра – `Bitmap`, так как JPEG – это растровый формат.

В дальнейшем для работы с изображениями будем пользоваться классом `Bitmap`. У него есть несколько конструкторов:

- `Bitmap(имя_файла)` – загружает изображение из указанного файла,
- `Bitmap(изображение)` – копирующий конструктор,
- `Bitmap(ширина, высота)` – создаёт пустое изображение с указанными размерами.

Самыми важными методами класса `Bitmap` являются:

- `GetPixel(x, y)` – получает цвет пикселя с координатами (x, y) ,
- `SetPixel(x, y, c)` – задаёт пикселю с координатами (x, y) цвет c .

Координаты отсчитываются от верхнего левого угла. При этом ось x направлена вправо, а ось y вниз.

Представление цвета

Цвет представлен структурой `System.Drawing.Color`. Она хранит информацию о цвете в формате ARGB. То есть цвет кодируется четырьмя числами: прозрачностью (A) и интенсивностями красной (R), зеленой (G) и синей (B) компонент. Каждое число имеет тип `int` и находится в диапазоне от 0 до 255. Прозрачность часто не указывают.

Для распространённых цветов в структуре предусмотрены константы с именами, соответствующими названиям цветов. Например:

- `Color.Aquamarine` – аквамарин,
- `Color.Black` – чёрный,
- `Color.LightGreen` – светло-зелёный,
- `Color.Red` – красный,
- `Color.White` – белый
- и многие другие.

Произвольный цвет можно задать статическим методом `Color.FromArgb`. Этот метод может принимать как три параметра (R, G и B), так и четыре (те же и прозрачность).

Например, так можно задать жёлтый цвет (смесь красного и зеленого):

```
Color.FromArgb(255, 255, 0);
```

Пример.

Создадим изображение 100×100 пикселей, закрасим его красным цветом и сохраним в файл «image.png».

```
Bitmap bmp = new Bitmap(100, 100);
for (int x = 0; x < bmp.Width; x++)
    for (int y = 0; y < bmp.Height; y++)
        bmp.SetPixel(x, y, Color.FromArgb(255, 0, 0));
bmp.Save(@"image.png");
```

Рисование на изображениях

Хотя теоретически любое изображение можно построить, задавая отдельные его пиксели, часто это неудобно. Даже для построения простой окружности придётся использовать достаточно сложные алгоритмы (например, алгоритм Брезенхэма). Поэтому в платформе .NET присутствует класс `System.Drawing.Graphics`, который позволяет рисовать на изображении различные графические примитивы.

Этот класс лишь предоставляет средства для рисования, поэтому при создании его экземпляра связывается с изображением, на котором нужно рисовать. Публичного конструктора у `Graphics` нет, для создания объекта можно воспользоваться статическим методом `FromImage`.

Экземпляр этого класса желательно удалить, когда он больше не нужен. Для того, чтобы гарантировать, что сборщик мусора его удалил можно воспользоваться командой `using`.

```
using (Graphics g = Graphics.FromImage(изображение))
{
    // рисование
}
```

Методы, начинающиеся с `Draw`, рисуют контуры, а начинающиеся с `Fill` – закрашенные фигуры.

Для рисования нужно задать «карандаш» – экземпляр класса `System.Drawing.Pen`. Он содержит информацию о толщине линии и о цвете. Можно создать карандаш, указав эти параметры в конструкторе, а можно выбрать один из готовых карандашей среди констант класса `System.Drawing.Pens`.

```
Pen p1 = new Pen(Color.Red, 3); // Красный, толщина – 3 пикселя
Pen p2 = Pens.Blue; // Синий карандаш толщиной 1 пиксель
```

Для закрашивания фигуры нужно выбрать «кисть» типа `System.Drawing.Brush`. Это абстрактный класс, у которого есть несколько потомков. Например, `SolidBrush` (сплошная заливка) и `TextureBrush` (заливка узором). Параметры кисти задаются в конструкторе.

Как и для карандашей, есть класс, содержащий часто используемые кисти – `Brushes`.

```
Brush b1 = new SolidBrush(Color.Green); // Кисть для заливки зелёным цветом
Brush b2 = Brushes.Green; // То же самое
```

Как и объект `Graphics`, карандаши и кисти лучше создавать с помощью `using`.

Основные методы класса `Graphics`:

- `DrawArc`(карандаш, `x`, `y`, `w`, `h`, `a1`, `a2`) – дуга, вписанная в прямоугольник, заданный верхним левым углом (`x`, `y`), размерами `h` и `w`. Углы, задающие дугу – `a1` и `a2`.
- `DrawEllipse`(карандаш, `x`, `y`, `w`, `h`) – эллипс, заданный описанным прямоугольником.
- `DrawImage`(изображение, `x`, `y`) – вставляет в одно изображение другое по указанным координатам.

- DrawLine(карандаш, x1, y1, x2, y2) – линия, заданная координатами концов.
- DrawRectangle(карандаш, x, y, w, h) – прямоугольник.
- DrawString(строка, шрифт, кисть, x, y) – вставляет текст. Шрифт – экземпляр класса Font (например, new Font("Times New Roman", 12)).
- FillEllipse(кисть, x, y, w, h) – закрашенный эллипс.
- FillRectangle(кисть, x, y, w, h) – закрашенный прямоугольник.

Это лишь часть методов. Например, есть метод DrawCurve, позволяющий провести гладкую кривую через массив точек.

Кроме того, многие методы перегружены и могут принимать другой набор параметров.

Взаимодействие с Windows Forms

Windows Forms позволяет рисовать поверх любого компонента, но лучше для этого использовать компонент, специально предназначенный для хранения изображений – PictureBox. У этого компонента есть поле Image, которое хранит отображаемое изображение. Обратите внимание, что сразу после запуска значение свойства Image по умолчанию равно null, и чтобы отобразить какое-то изображение, необходимо его либо создать, либо загрузить из файла.

Имя файла лучше не задавать прямо в тексте программы, а запрашивать с помощью соответствующего диалога.

Важно отметить, что после изменения изображения в PictureBox оно не будет тут же перерисовано. Это связано с тем, что операционная система кэширует изображение компонентов для повышения производительности и перерисовывает их только по требованию. Чтобы перерисовать компонент, нужно указать операционной системе, что старое изображение больше не является правильным (valid). Для этого используется метод Invalidate().

Пример.

Пусть на форме расположена кнопка buttonDraw и изображение (компонент типа PictureBox) picture. Пусть также на форму добавлен диалог открытия файла с именем openImage.

Напишем программу, которая по нажатию на кнопку открывает изображение и увеличивает его яркость. Для этого все значения компонентов цвета пикселей умножаются на 1,5 с насыщением. То есть, если увеличенное значение превышает 255, то считается равным 255. Это нужно чтобы не выйти за пределы диапазона допустимых значений яркостей пикселей.

Приведем обработчик нажатия на кнопку.

```
// Умножение с насыщением
static int SatMul(int x, double k)
{
    x = (int)(x * k);
    return x > 255 ? 255 : x;
}

private void buttonDraw_Click(object sender, EventArgs e)
{
    DialogResult result = openImage.ShowDialog();
    if (result != DialogResult.OK)
        return;

    Bitmap bmp;
    try
    {
```

```

        bmp = new Bitmap(openImage.FileName);
    }
    catch (FileNotFoundException)
    {
        return;
    }

    for (int x = 0; x < bmp.Width; x++)
        for (int y = 0; y < bmp.Height; y++)
        {
            Color c = bmp.GetPixel(x, y);
            int R = SatMul(c.R, 1.5);
            int G = SatMul(c.G, 1.5);
            int B = SatMul(c.B, 1.5);
            bmp.SetPixel(x, y, Color.FromArgb(R, G, B));
        }

    picture.Image = bmp;
    picture.Invalidate();
}

```

Пример.

Напишем программу, по нажатию на кнопку создающую изображение размером 101×101 пиксель в виде серой сетки из квадратов и отображающую его в компоненте с именем picture.

Приведем обработчик нажатия на кнопку.

```

private void buttonDraw_Click(object sender, EventArgs e)
{
    Bitmap bmp = new Bitmap(101, 101);
    picture.Image = bmp;

    using (Graphics g = Graphics.FromImage(bmp))
    {
        for (int x = 0; x < 101; x += 10)
            g.DrawLine(Pens.Gray, x, 0, x, 101);
        for (int y = 0; y < 101; y += 10)
            g.DrawLine(Pens.Gray, 0, y, 101, y);
    }
    picture.Invalidate();
}

```

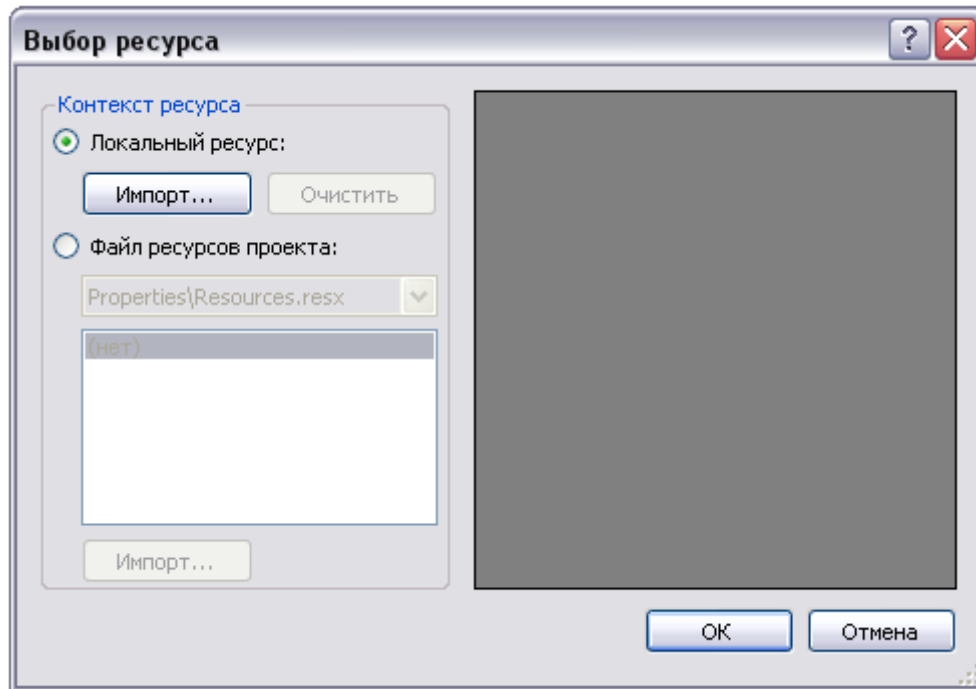
Элемент управления PictureBox

Элемент управления PictureBox предназначен для показа изображений. Он позволяет отобразить файлы в формате bmp, jpg, gif, а также метафайлы изображений и иконки. Для установки изображения в PictureBox можно использовать ряд свойств:

- **Image:** устанавливает объект типа Image
- **ImageLocation:** устанавливает путь к изображению на диске или в интернете
- **InitialImage:** некоторое начальное изображение, которое будет отображаться во время загрузки главного изображения, которое хранится в свойстве Image
- **ErrorImage:** изображение, которое отображается, если основное изображение не удалось загрузить в PictureBox

Чтобы установить изображение в Visual Studio, надо в панели Свойств PictureBox выбрать свойство Image. В этом случае нам откроется окно импорта изображения в проект, где

мы собственно и сможем выбрать нужное изображение на компьютере и установить его для PictureBox:



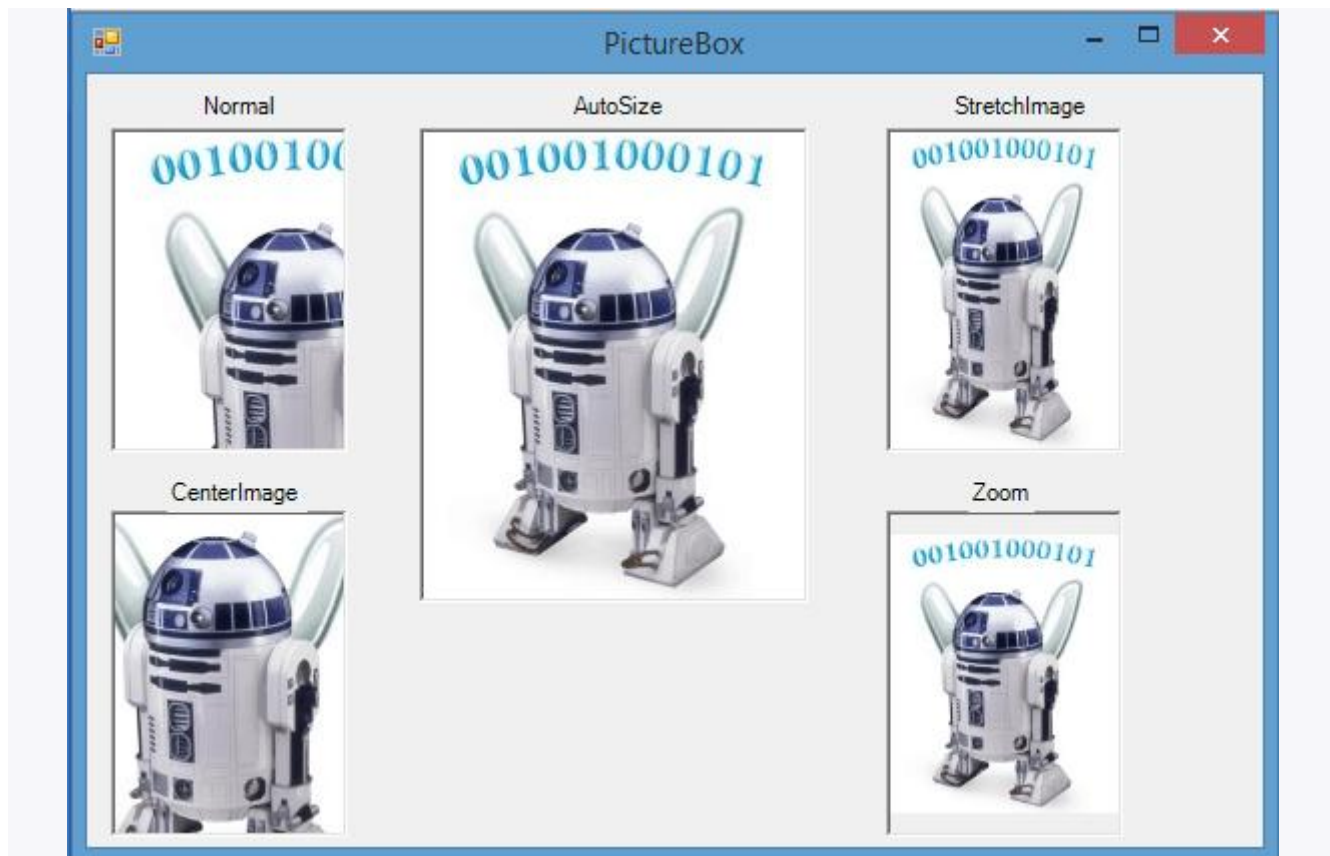
Либо можно загрузить изображение в коде:

```
pictureBox1.Image = Image.FromFile("D:\\Pictures\\12.jpg");
```


Размер изображения

Для установки изображения в PictureBox используется свойство `SizeMode`, которое принимает следующие значения:

- **Normal**: изображение позиционируется в левом верхнем углу PictureBox, и размер изображения не изменяется. Если PictureBox больше размеров изображения, то справа и снизу появляются пустоты, если меньше – изображение обрезается
- **StretchImage**: изображение растягивается или сжимается таким образом, чтобы вписаться по всей ширине и высоте элемента PictureBox
- **AutoSize**: элемент PictureBox автоматически растягивается, подстраиваясь под размеры изображения
- **CenterImage**: если PictureBox меньше изображения, то изображение обрезается по краям и выводится только его центральная часть. Если же PictureBox больше изображения, то оно позиционируется по центру.
- **Zoom**: изображение подстраивается под размеры PictureBox, сохраняя при этом пропорции



Элемент управления Chart

Элемент управления  Chart позволяет строить различные диаграммы и графики, которые выглядят очень эффектно.

Элемент управления Chart является контейнером объектов Series – серий данных, характеризующихся различными стилями отображения. Каждый элемент управления может включать несколько серий. Если вы хотите отображать график, то каждая серия будет соответствовать одной кривой на графике.

Элемент управления Chart имеет множество свойств, методов, событий.

Некоторые свойства элемента управления Chart:

Имя свойство	Описание
BackColor	Получает или задает цвет фона для объекта Chart
BackGradientStyle	Получает или задает ориентацию градиента фона для элемента управления Chart. Также определяет, используется ли градиент.
BackImage	Получает или задает фоновое изображение для элемента управления Chart.
BorderlineColor	Получает или задает цвет линии границы.
BorderlineDashStyle	Получает или задает стиль линии границы.
BorderlineWidth	Получает или задает толщину линии границы.
Legends	Получает или задает свойства легенды: Alignment – получает или задает тип выравнивания легенды; BackColor – получает или задает цвет фона легенды; BorderColor – получает или задает цвет границы легенды;

	Font – получает или задает свойств шрифта легенды.
Series	Получает или задает свойства диаграммы: BorderWidth – получает или задает ширину границы точки данных; ChartType – получает или задает тип диаграммы для ряда; Color – получает или задает цвет точки данных; LegendText – получает или задает текст элемента в легенде; MarkerBorderColor – получает или задает цвет границы маркера; MarkerColor – получает или задает цвет маркера; MarkerStyle – получает или задает стиль маркера; Name – получает или задает уникальное имя объекта Series.
Title	Получает или задает свойства заголовка диаграммы.

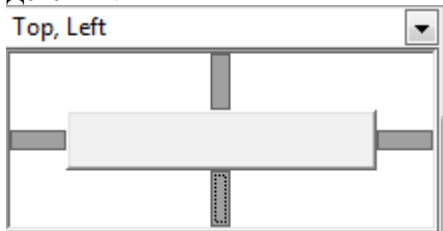
Практическая часть


Упражнение 1

Разработаем приложение, которое позволит открывать и сохранять графический файл и поворачивать графическое изображение на 90°.

Реализация проекта

1. Создайте проект с графическим интерфейсом.
2. Поместите на форму элемент управления PictureBox и установите следующие значения свойств объектов:

Класс объектов	Обозначение объекта по умолчанию	Свойство	Значение
Form	Form1	Text	Работа с изображением
PictureBox	PictureBox1	Name	picture
		SizeMode	Zoom
		Anchor	Top, Bottom, Left, Right Действия: Top, Left
			

3. Поместите на форму элемент управления  MenuStrip (меню). В режиме дизайнера добавьте пункты меню первого и второго уровней (рис. 1, 2).

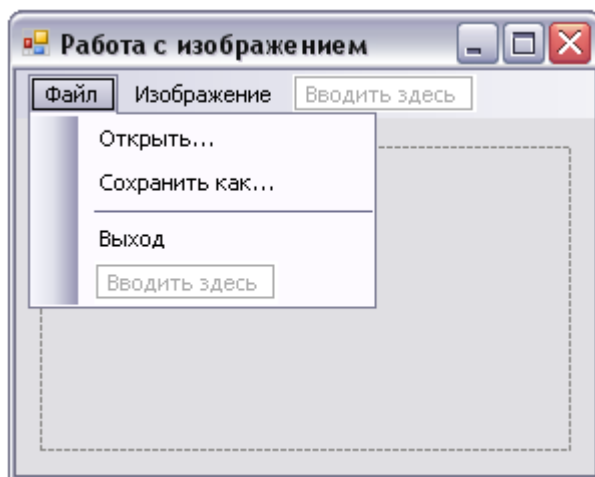


Рис. 1

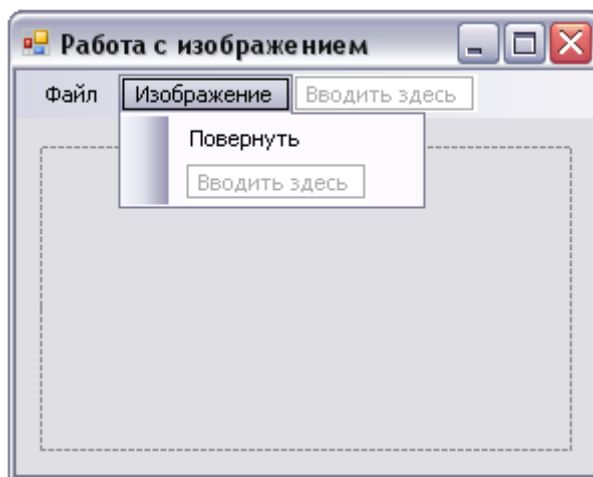


Рис. 2

4. Измените свойство Name созданных пунктов меню:

- открытьToolStripMenuItem - openToolStripMenuItem,
- сохранитьКакToolStripMenuItem - saveasToolStripMenuItem,
- выходToolStripMenuItem - exitToolStripMenuItem,
- повернутьToolStripMenuItem - rotateToolStripMenuItem.

5. Используя свойство ShortcutKeys, задайте клавиши быстрого доступа для пунктов меню (рис. 3)

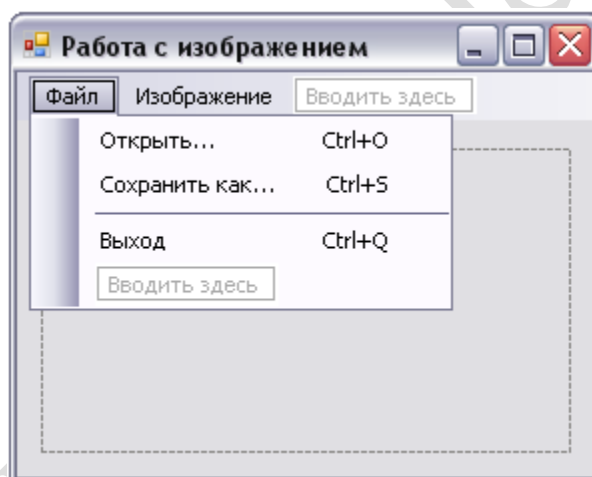




Рис. 3

6. Поместите на форму элементы управления  OpenFileDialog и  SaveFileDialog. Установите следующие значения свойств объектов:

Класс объектов	Обозначение объекта по умолчанию	Свойство	Значение
OpenFileDialog	openFileDialog1	Name	openImage
SaveFileDialog	saveFileDialog1	Name	saveImage
		Filter	Image Files (*.BMP) *.BMP Image Files (*.JPG) *.JPG Image Files (*.GIF) *.GIF Image Files (*.PNG) *.PNG All files (*.*) *.*

7. В окне исполнительного кода добавьте инструкцию using для пространства имен System.IO и System.Drawing.Imaging:

```
using System.IO;
using System.Drawing.Imaging;
```

8. Создайте обработчики события Click для пунктов меню Файл и Изображение:

```
private void openToolStripMenuItem_Click(object sender, EventArgs e)
{
```

```

DialogResult result = openImage.ShowDialog();
if (result != DialogResult.OK) //если в окне не была нажата кнопка "OK"
    return;

try
{
    picture.Image = new Bitmap(openImage.FileName);
    picture.Invalidate();
}
catch (FileNotFoundException)
{
    MessageBox.Show("Файл не найден", "Ошибка", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
}
}

private void saveasToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (picture.Image == null) //если в picture отсутствует изображение
        return;

    DialogResult result = saveImage.ShowDialog();
    if (result != DialogResult.OK)
        return;

    try
    {
        picture.Image.Save(saveImage.FileName, ImageFormat.Jpeg);
    }
    catch
    {
        MessageBox.Show("Невозможно сохранить изображение", "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    Application.Exit();
}

private void rotateToolStripMenuItem_Click(object sender, EventArgs e)
{
    Bitmap bmp = picture.Image as Bitmap;
    if (bmp == null)
        return;

    Bitmap bmp_rotated = new Bitmap(bmp.Height, bmp.Width);
    for (int x = 0; x < bmp.Width; x++)
        for (int y = 0; y < bmp.Height; y++)
        {
            Color c = bmp.GetPixel(x, y);
            bmp_rotated.SetPixel(bmp_rotated.Width - y - 1, x, c);
        }
    picture.Image = bmp_rotated;
    picture.Invalidate();
}

```

9. Запустите проект на выполнение и протестируйте его (Рис. 4, 5).

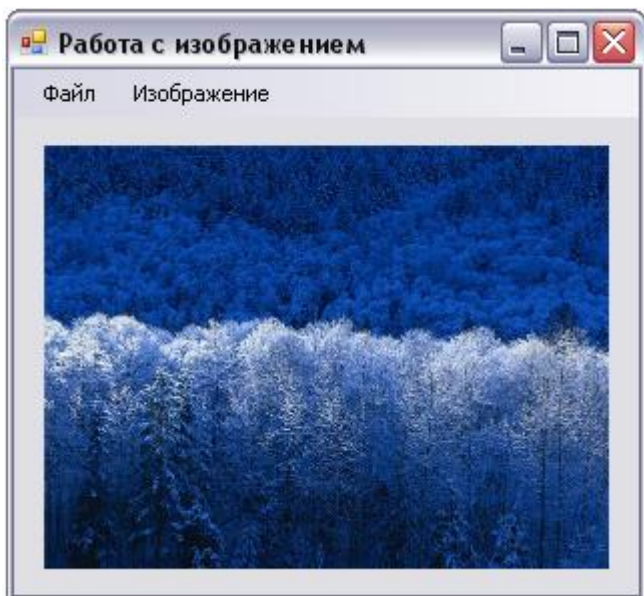


Рис. 4

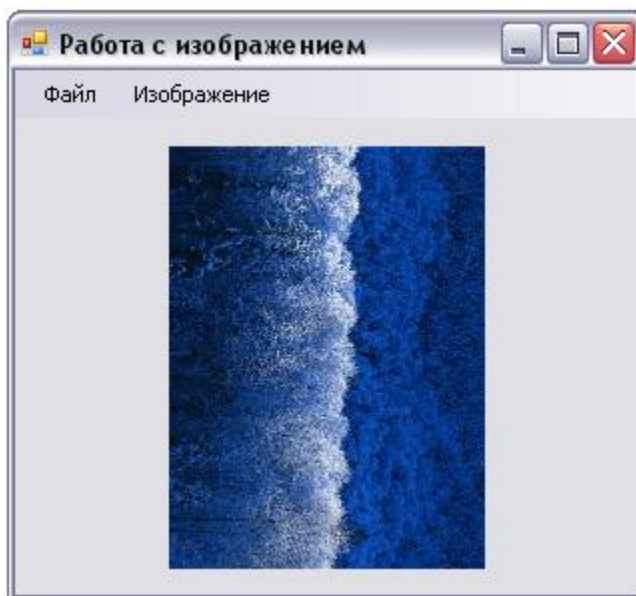


Рис. 5

Упражнение 2

Создадим приложение для вывода совмещенных графиков функций: $y = x^2 - 2$ и $y = 2x + 4$ в диапазоне $[-4; 4]$.

Реализация проекта

1. Создайте новый проект.
2. Поместите на форму кнопку и элемент управления  Chart (рис. 6).

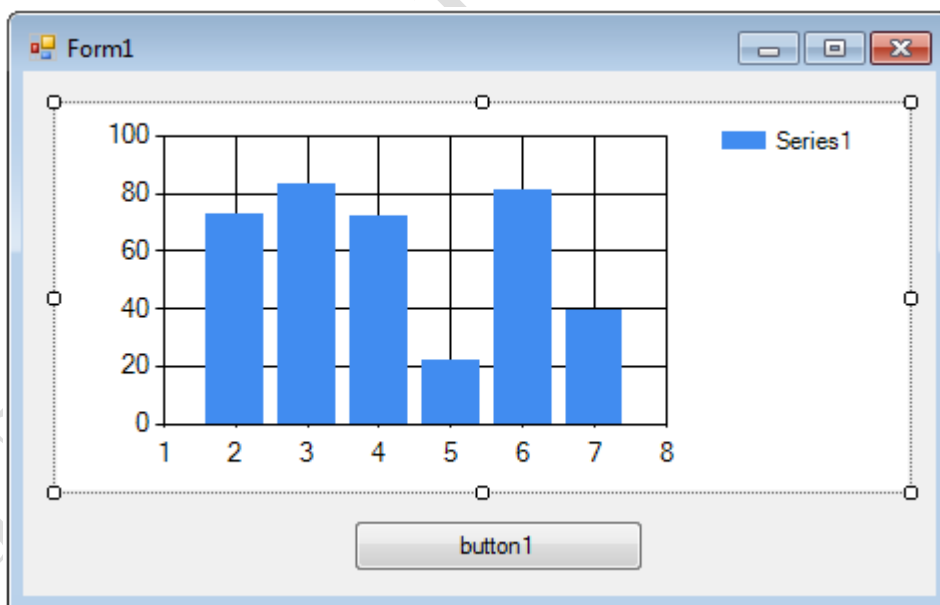


Рис. 6

3. Измените некоторые свойства элемента управления Chart.

а) фон диаграммы и ее границу

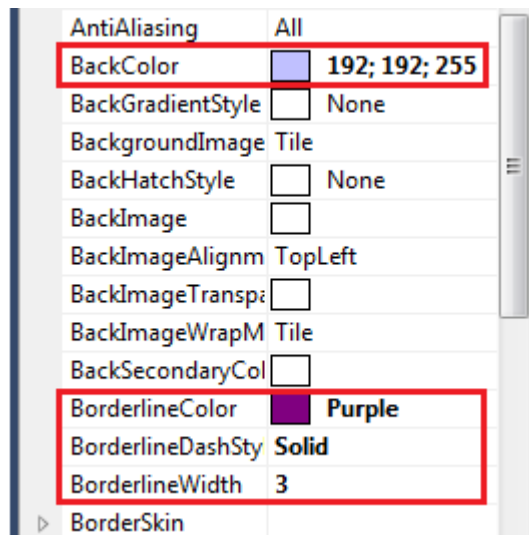


Рис. 7

б) тип диаграммы и свойства точек данных

Выберите свойство Series (рис. 8) и измените некоторые свойства в открывшемся диалоговом окне Series Collection Editor (рис. 9, 10).

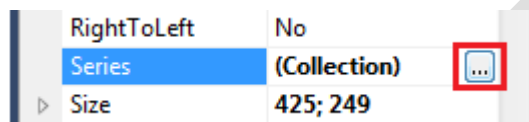


Рис. 8

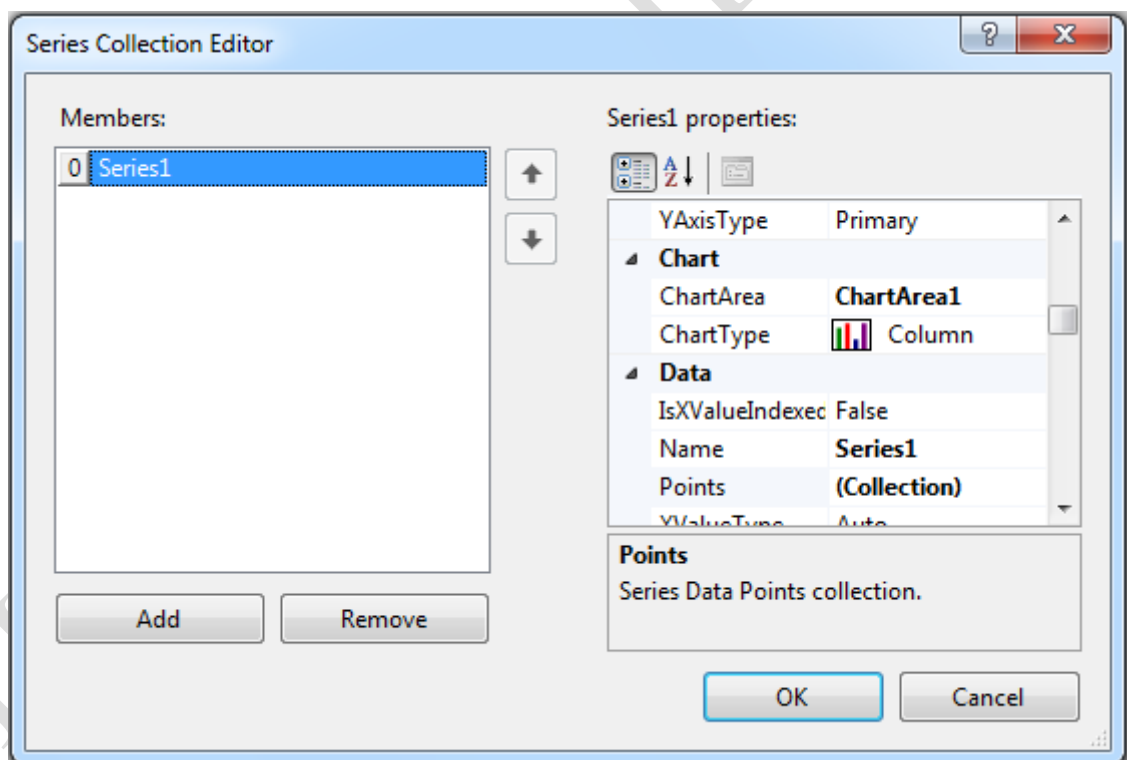


Рис. 9

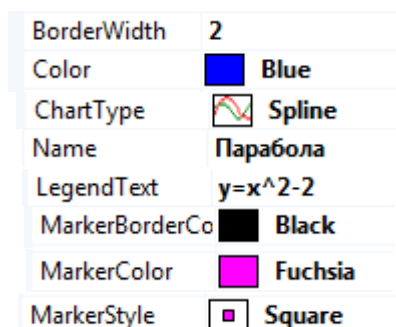


Рис. 10

В диалоговом окне Series Collection Editor нажмите кнопку Add. Добавится новая серия для графика (рис. 11). Измените некоторые свойства для второго графика (рис. 12).

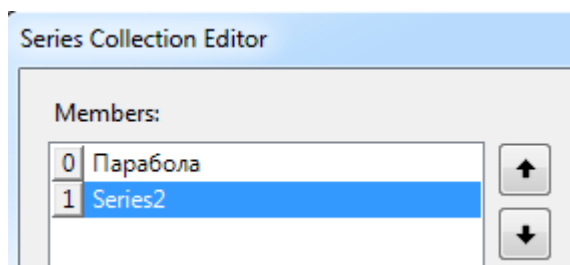


Рис. 11






BorderWidth	2
Color	 Green
ChartType	 Line
Name	Прямая
LegendText	$y=2x+4$
MarkerBorderColor	 Black
MarkerColor	 Yellow
MarkerStyle	 Circle

Рис. 12

в) легенду

Выберите свойство Legends (рис. 13) и измените некоторые свойства легенды (рис. 14).

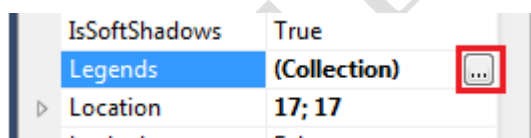


Рис. 13


BorderColor	 Black
Alignment	Center

Рис. 14

г) заголовок диаграммы

Выберите свойство Title (рис. 15), в открывшемся диалоговом окне нажмите кнопку Add и введите заголовок диаграммы (рис. 16).

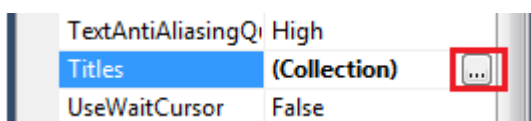


Рис. 15

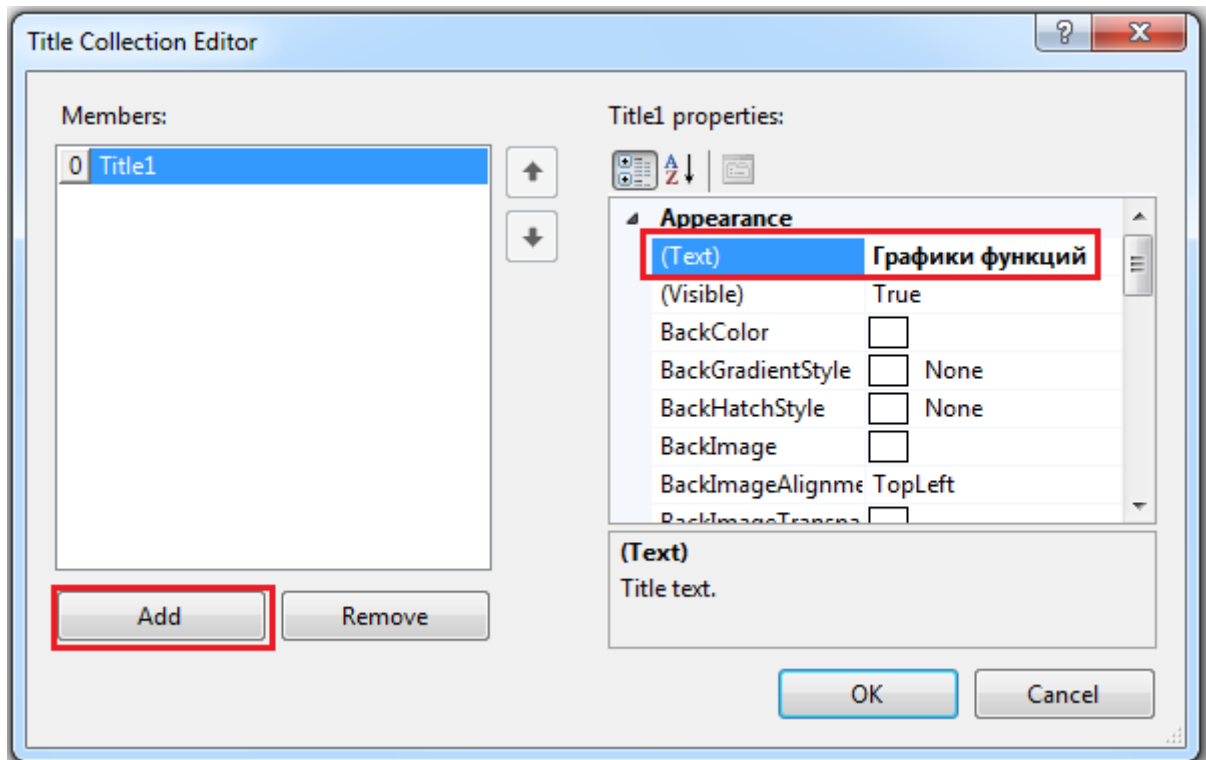


Рис. 16

4. Измените заголовок формы и текст на кнопке (рис. 17).

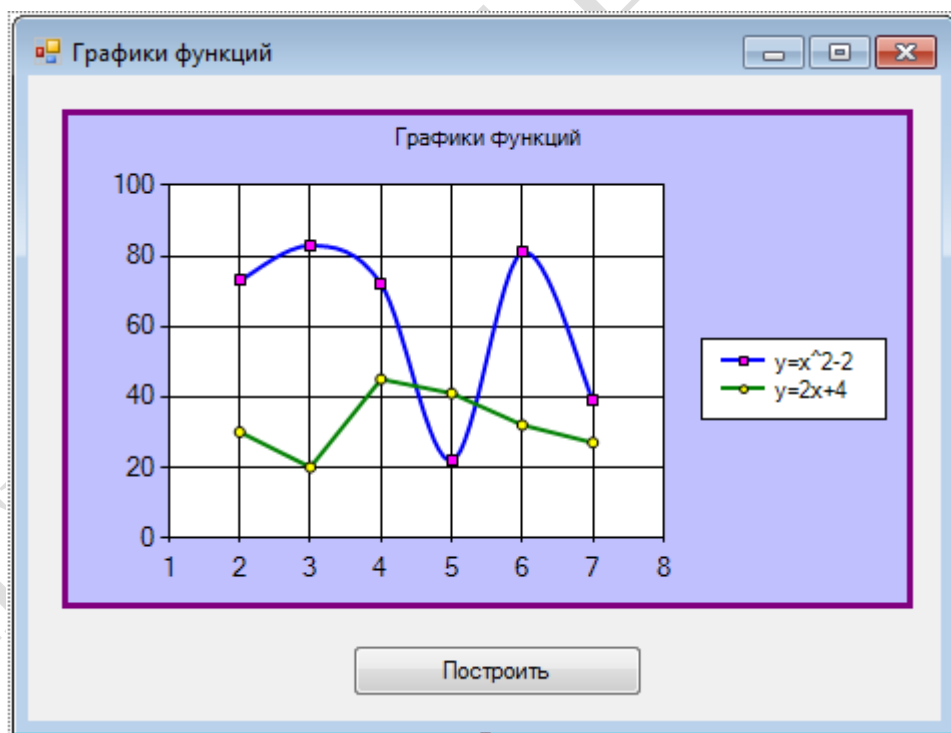


Рис. 17

5. В окне исполнительного кода измените обработчик события Click для кнопки:


```
private void button1_Click(object sender, EventArgs e)
{
    int x, y, a = -4, b = 4;
    chart1.Series["Парабола"].Points.Clear();
    chart1.Series["Прямая"].Points.Clear();
    for (x = a; x <= b; x++)
    {
        y = x * x - 2;
        chart1.Series["Парабола"].Points.AddXY(x, y);
    }
    for (x = a; x <= b; x++)
    {
        y = 2 * x + 4;
        chart1.Series["Прямая"].Points.AddXY(x, y);
    }
}
```

6. Запустите проект на выполнение и протестируйте его (Рис. 18).

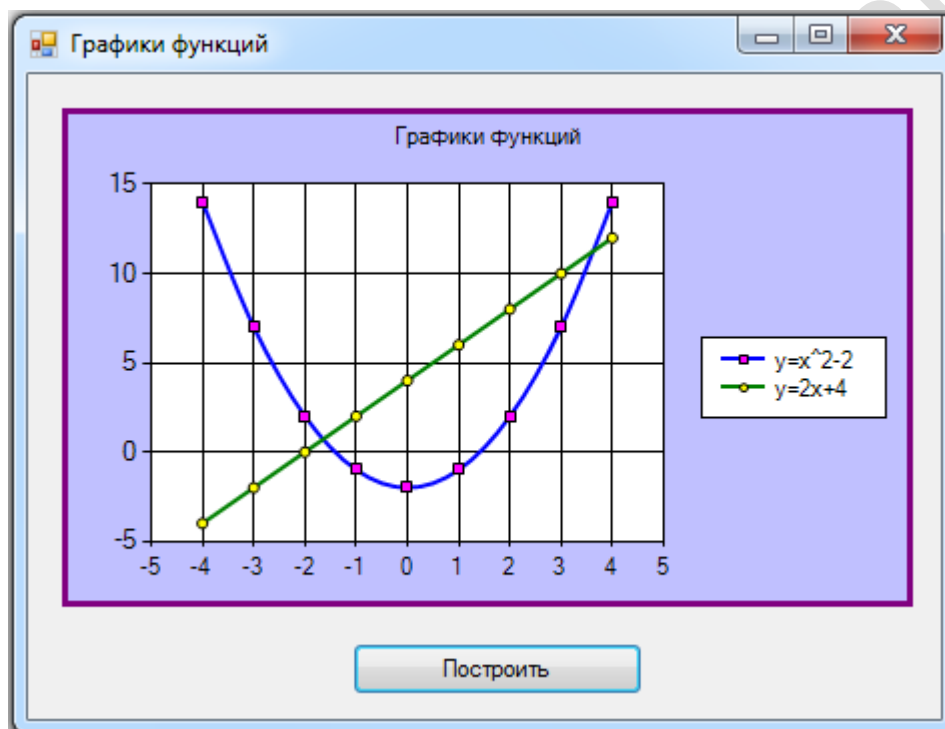


Рис. 18

Задания для самостоятельной работы

Разработайте приложения для решения задач из сборника задач по программированию согласно вашему варианту.