**практическая РАБОТА 8. Программирование с использованием средств для отображения графической информации**

***Цель практической работы:*** изучить возможности построения графиков с помощью компонента отображения графической информации **Сhart**. Написать и отладить программу построения на экране графика заданной функции.

***8.1. Как строится график с помощью компонента Chart***

Обычно результаты расчетов представляются в виде графиков и диаграмм. Библиотека .NET Framework имеет мощный элемент управления Chart для отображения на экране графической информации (рис. 8.1).

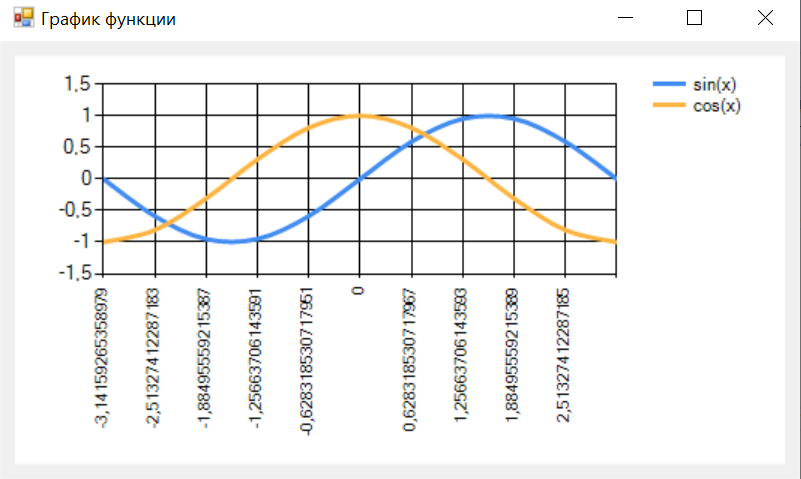


Рис 8.1. Окно программы с элементом управления.

Построение графика (диаграммы) производится после вычисления таблицы значений функции y=f(x) на интервале [Xmin, Xmax] с заданным шагом. Полученная таблица передается в специальный массив Points объекта Series компонента Сhart с помощью метода DataBindXY. Компонент Chart осуществляет всю работу по отображению графиков: строит и размечает оси, рисует координатную сетку, подписывает название осей и самого графика, отображает переданную таблицу в виде всевозможных графиков или диаграмм. При необходимости компоненту Сhart передаются данные о толщине, стиле и цвете линий, параметрах шрифта подписей, шагах разметки координатной сетки и другие настройки. В процессе работы программы изменение параметров возможно через обращение к соответствующим свойствам компонента Chart. Так, например, свойство AxisX содержит значение максимального предела нижней оси графика и при его изменении во время работы программы автоматически изменяется изображение графика.

***8.2. Пример написания программы***

Задание: составить программу, отображающую графики функций sin(x) и cos(x) на интервале [Xmin, Xmax]. Предусмотреть возможность изменения разметки координатных осей, а также шага построения таблицы.

Прежде всего, следует определить в коде класса все необходимые переменные и константы. Конечно, можно обойтись и без этого, вставляя значения в виде чисел прямо в формулы, но это, во-первых, снизит читабельность кода программы, а во-вторых, значительно усложнит изменение каких-либо параметров программы, например, интервала построения графика.

/// <summary>

/// Левая граница графика

/// </summary>

private double XMin = -Math.PI;

/// <summary>

/// Правая граница графика

/// </summary>

private double XMax = Math.PI;

/// <summary>

/// Шаг графика

/// </summary>

private double Step = (Math.PI \* 2) / 10;

// Массив значений X - общий для обоих графиков

private double[] x;

// Два массива Y - по одному для каждого графика

private double[] y1;

private double[] y2;

Также в коде класса следует описать глобальную переменную типа Chart, к которой мы будем обращаться из разных методов:

Chart chart;

Поскольку данный класс не входит в пространства имен, подключаемые по умолчанию, следует выполнить дополнительные действия. Во-первых, в Обозревателе решений нужно щёлкнуть правой кнопкой по секции Ссылки и добавить ссылку на библиотеку визуализации (рис. 8.2):

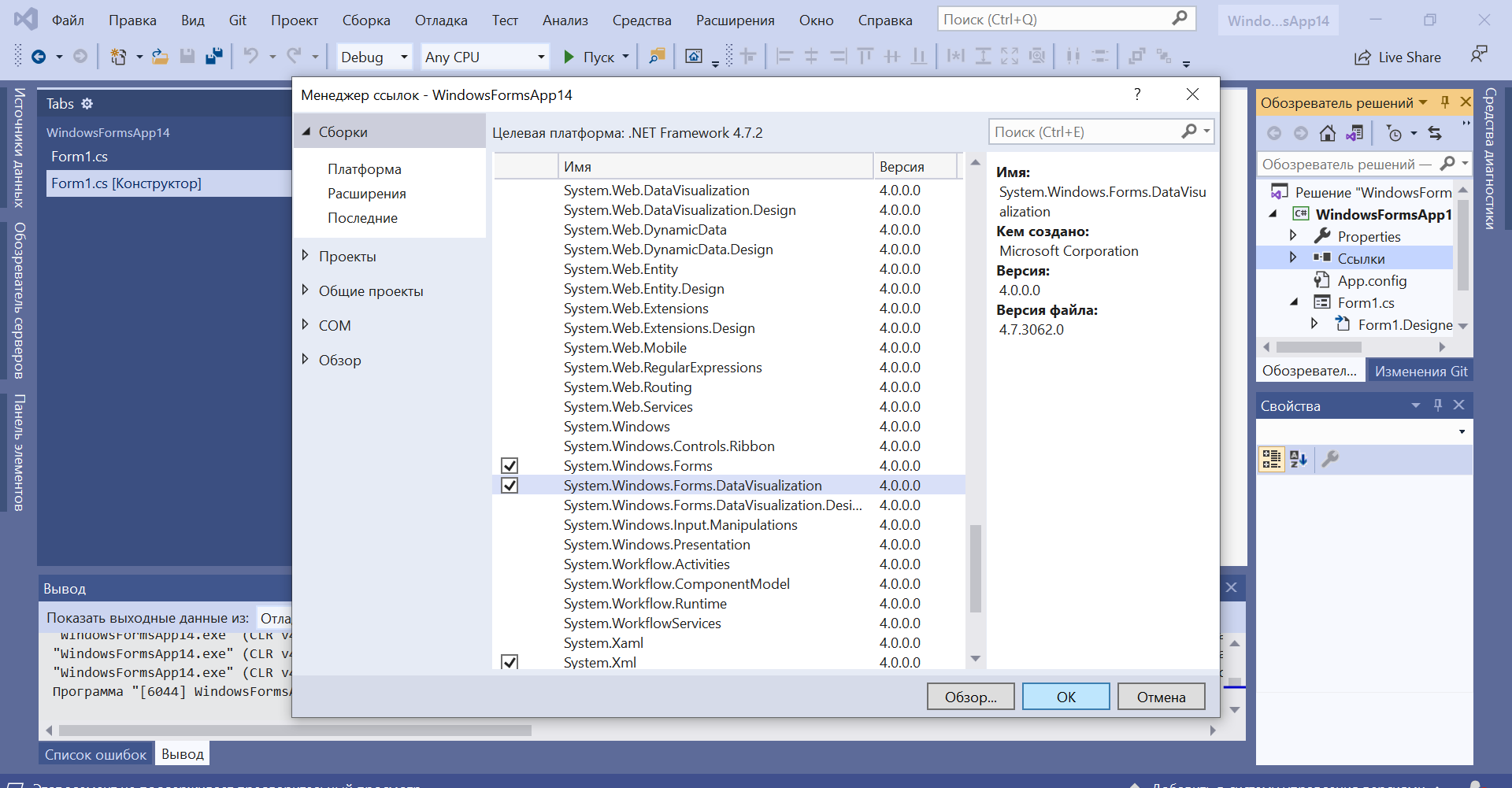


Рис. 8.2. Добавление ссылки на библиотеку визуализации.

Кроме того, следует подключить соответствующее пространство имен:

using System.Windows.Forms.DataVisualization.Charting;

Далее следует определить метод, который будет рассчитывать количество шагов и вычислять значения функций в каждой точке, внося вычисленные значения в массивы *x*, *y1* и *y2*:

/// <summary>

/// Расчёт значений графика

/// </summary>

private void CalcFunction()

{

// Количество точек графика

int count = (int)Math.Ceiling((XMax - XMin) / Step)

+ 1;

// Создаём массивы нужных размеров

x = new double[count];

y1 = new double[count];

y2 = new double[count];

// Расчитываем точки для графиков функции

for (int i = 0; i < count; i++)

{

// Вычисляем значение X

x[i] = XMin + Step \* i;

// Вычисляем значение функций в точке X

y1[i] = Math.Sin(x[i]);

y2[i] = Math.Cos(x[i]);

}

}

После расчёта значений нужно отобразить графики на форме с помощью элемента Chart. Элемент управления Chart можно выбрать с помощью панели элементов, а можно создать прямо в коде программы. Вторым шагом следует создать область отображения графика и настроить внешний вид осей:

/// <summary>

/// Создаём элемент управления Chart и настраиваем его

/// </summary>

private void CreateChart()

{

// Создаём новый элемент управления Chart

chart = new Chart();

// Помещаем его на форму

chart.Parent = this;

// Задаём размеры элемента

chart.SetBounds(10, 10, ClientSize.Width - 20,

ClientSize.Height - 20);

// Создаём новую область для построения графика

ChartArea area = new ChartArea();

// Даём ей имя (чтобы потом добавлять графики)

area.Name = "myGraph";

// Задаём левую и правую границы оси X

area.AxisX.Minimum = XMin;

area.AxisX.Maximum = XMax;

// Определяем шаг сетки

area.AxisX.MajorGrid.Interval = Step;

// Добавляем область в диаграмму

chart.ChartAreas.Add(area);

// Создаём объект для первого графика

Series series1 = new Series();

// Ссылаемся на область для построения графика

series1.ChartArea = "myGraph";

// Задаём тип графика - сплайны

series1.ChartType = SeriesChartType.Spline;

// Указываем ширину линии графика

series1.BorderWidth = 3;

// Название графика для отображения в легенде

series1.LegendText = "sin(x)";

// Добавляем в список графиков диаграммы

chart.Series.Add(series1);

// Аналогичные действия для второго графика

Series series2 = new Series();

series2.ChartArea = "myGraph";

series2.ChartType = SeriesChartType.Spline;

series2.BorderWidth = 3;

series2.LegendText = "cos(x)";

chart.Series.Add(series2);

// Создаём легенду, которая будет показывать названия

Legend legend = new Legend();

chart.Legends.Add(legend);

}

Наконец, все эти методы следует откуда-то вызвать. Чтобы графики появлялись сразу после запуска программы, надо вызывать их в обработчике события Load формы:

private void Form1\_Load(object sender, EventArgs e)

{

// Создаём элемент управления

CreateChart();

// Расчитываем значения точек графиков функций

CalcFunction();

// Добавляем вычисленные значения в графики

chart.Series[0].Points.DataBindXY(x, y1);

chart.Series[1].Points.DataBindXY(x, y2);

}

***8.3. Выполнение индивидуального задания***

Постройте графики функций для соответствующих вариантов из лабораторной работы №2. Таблицу данных получить путём изменения параметра X с шагом h. Самостоятельно выбрать удобные параметры настройки.

**Практическая РАБОТА 9. Программирование ГРАФИКИ**

***Цель практической работы:*** изучить возможности Visual Studio по создание простейших графических изображений. Написать и отладить программу построения на экране различных графических примитивов.

***9.1. Сообщение WM\_PAINT***

Прежде чем приступить к описанию способов рисования в окнах, применяемых приложениями .NET Frameworks, расскажем о том, как это делают «классические» приложения Microsoft Windows.

ОС Microsoft Windows следит за перемещением и изменением размера окон и при необходимости извещает приложения, о том, что им следует перерисовать содержимое окна. Для извещения в очередь приложения записывается сообщение с идентификатором **WM\_PAINT**. Получив такое сообщение, функция окна должна выполнить перерисовку всего окна или его части, в зависимости от дополнительных данных, полученных вместе с сообщением **WM\_PAINT**.

Для облегчения работы по отображению содержимого окна весь вывод в окно обычно выполняют в одном месте приложения — при обработке сообщения **WM\_PAINT** в функции окна. Приложение должно быть сделано таким образом, чтобы в любой момент времени при поступлении сообщения **WM\_PAINT** функция окна могла перерисовать все окно или любую его часть, заданную своими координатами.

Последнее нетрудно сделать, если приложение будет хранить где-нибудь в памяти свое текущее состояние, пользуясь которым функция окна сможет перерисовать окно в любой момент времени.

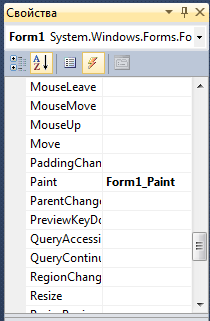
Здесь не имеется в виду, что приложение должно хранить образ окна в виде графического изображения и восстанавливать его при необходимости, хотя это и можно сделать. Приложение должно хранить информацию, на основании которой оно может в любой момент времени перерисовать окно.

Сообщение **WM\_PAINT** передается функции окна, если стала видна область окна, скрытая раньше другими окнами, если пользователь изменил размер окна или выполнил операцию прокрутки изображения в окне. Приложение может передать функции окна сообщение **WM\_PAINT** явным образом, вызывая функции программного интерфейса Win32 API, такие как **UpdateWindow**, **InvalidateRect** или **InvalidateRgn**.

Иногда ОС Microsoft Windows может сама восстановить содержимое окна, не посылая сообщение **WM\_PAINT**. Например, при перемещении курсора мыши или значка свернутого приложения ОС восстанавливает содержимое окна. Если же ОС не может восстановить окно, функция окна получает от ОС сообщение **WM\_PAINT** и перерисовывает окно самостоятельно.

***9.2. Событие Paint***

Для форм класса **System.Windows.Forms** предусмотрен удобный объектно-ориентированный способ, позволяющий приложению при необходимости перерисовывать окно формы в любой момент времени. Когда вся клиентская область окна формы или часть этой области требует перерисовки, форме передается событие **Paint**. Все, что требуется от программиста, это создать обработчик данного события (рис. 9.1.), наполнив его необходимой функциональностью.



*Рис. 9.1. Создание обработчика события Paint*

***9.3. Объект Graphics для рисования***

Перед тем как рисовать линии и фигуры, отображать текст, выводить изображения и управлять ими в GDI необходимо создать объект **Graphics**. Объект Graphics представляет поверхность рисования GDI и используется для создания графических изображений. Ниже представлены два этапа работы с графикой.

1. Создание объекта Graphics.
2. Использование объекта Graphics для рисования линий и фигур, отображения текста или изображения и управления ими.

Существует несколько способов создания объектов **Graphics**. Одним из самых используемых является получение ссылки на объект **Graphics** через объект **PaintEventArgs** при обработке события **Paint** формы или элемента управления:

private void Form1\_Paint(object sender, PaintEventArgs e)

{

Graphics g = e.Graphics; // Объявляется объект Graphics

// Далее вставляется код рисования

}

***9.4. Методы и свойства класса Graphics***

Имена большого количества методов, определенных в классе Graphics, начинается с префикса Draw\* и Fill\*. Первые из них предназначены для рисования текста, линий и не закрашенных фигур (таких, например, как прямоугольные рамки), а вторые — для рисования закрашенных геометрических фигур. Мы рассмотрим применение только самых важных из этих методов, а полную информацию Вы найдете в документации.

Метод **DrawLine** рисует линию, соединяющую две точки с заданными координатами. Ниже мы привели прототипы различных перегруженных версий этого метода:

**public void DrawLine(Pen, Point, Point);**

**public void DrawLine(Pen, PointF PointF;**

**public void DrawLine(Pen, int, int, int, int);**

**public void DrawLine(Pen, float, float, float, float);**

Первый параметр задает инструмент для рисования линии — перо. Перья создаются как объекты класса **Pen**, например:

**Pen p = new Pen(Brushes.Black,2);**

Здесь мы создали черное перо толщиной 2 пиксела. Создавая перо, Вы можете выбрать его цвет, толщину и тип линии, а также другие атрибуты.

Остальные параметры перегруженных методов DrawLine задают координаты соединяемых точек. Эти координаты могут быть заданы как объекты класса **Point** и **PointF**, а также в виде целых чисел и чисел с плавающей десятичной точкой.

В классах **Point** и **PointF** определены свойства **X** и **Y**, задающие, соответственно, координаты точки по горизонтальной и вертикальной оси. При этом в классе **Point** эти свойства имеют целочисленные значения, а в классе **PointF** — значения с плавающей десятичной точкой.

Третий и четвертый вариант метода **DrawLine** позволяет задавать координаты соединяемых точек в виде двух пар чисел. Первая пара определяет координаты первой точки по горизонтальной и вертикальной оси, а вторая — координаты второй точки по этим же осям. Разница между третьим и четвертым методом заключается в использовании координат различных типов (целочисленных **int** и с плавающей десятичной точкой **float**).

Чтобы испытать метод **DrawLine** в работе, создайте приложение DrawLineApp (аналогично тому, как Вы создавали предыдущее приложение). В этом приложении создайте следующий обработчик события **Paint**:

private void Form1\_Paint(object sender, PaintEventArgs e)

{

Graphics g = e.Graphics;

g.Clear(Color.White);

for (int i = 0; i < 50; i++)

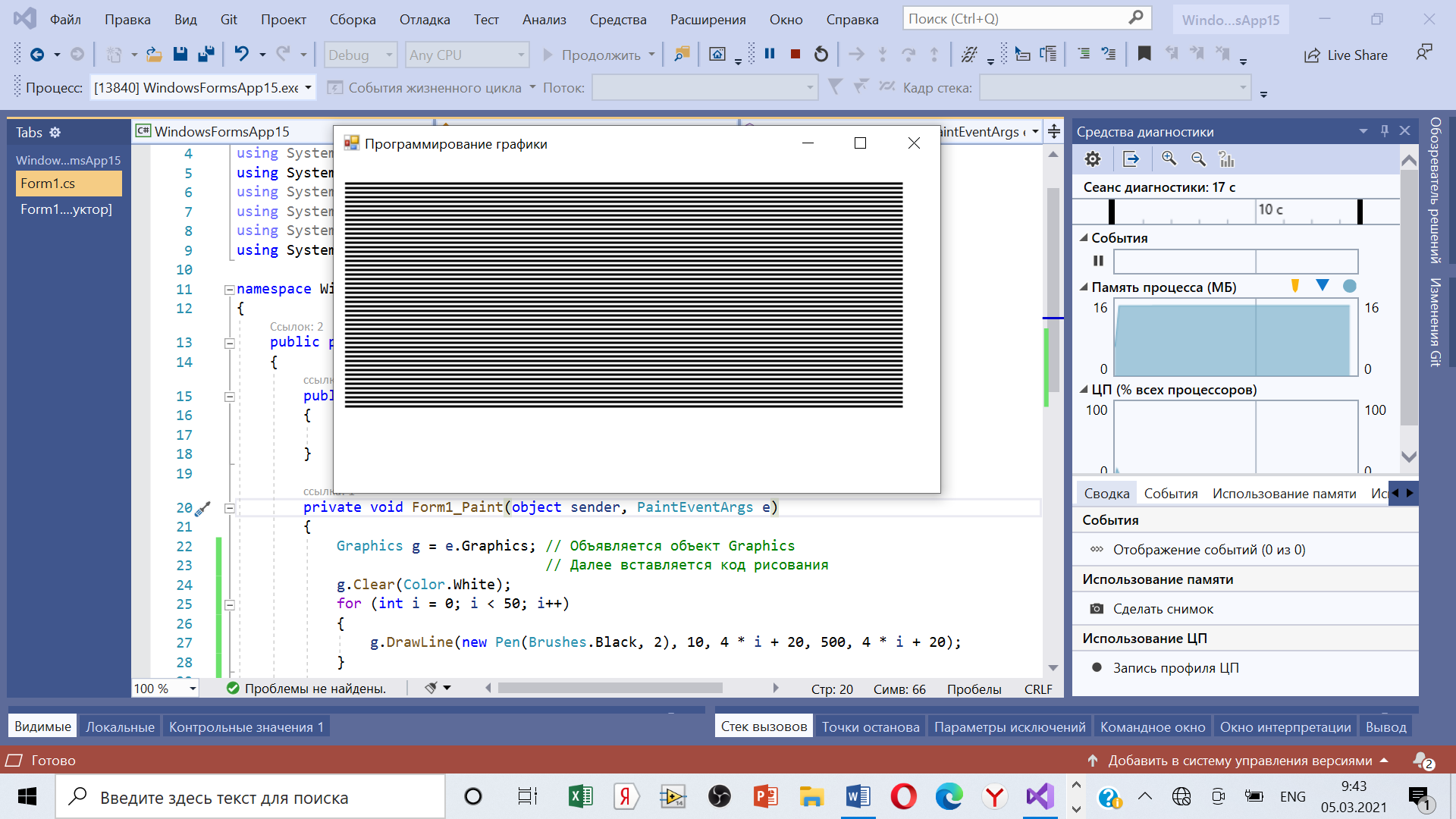
{

g.DrawLine(new Pen(Brushes.Black, 2), 10, 4 \* i + 20, 500, 4 \* i + 20);

}

}

Здесь мы вызываем метод **DrawLine** в цикле, рисуя 50 горизонтальных линий (рис. 9.2.).



*Рис. 9.2. Пример использования DrawLine*

Вызвав один раз метод DrawLines, можно нарисовать сразу несколько прямых линий, соединенных между собой. Иными словами, метод **DrawLines** позволяет соединить между собой несколько точек. Координаты этих точек по горизонтальной и вертикальной оси передаются методу через массив класса **Point** или **PointF**:

**public void DrawLines(Pen, Point[]);**

**public void DrawLines(Pen, PointF[];**

Для демонстрации возможностей метода **DrawLines** создайте приложение. Создайте кисть **pen** для рисования линий:

**Pen pen = new Pen(Color.Black, 2);**

а также массив точек **points**, которые нужно соединить линиями:

**Point[] points = new Point[50];**

**for(int i=0; i < 20; i++)**

**{**

**int xPos;**

**if(i%2 == 0)**

**{**

**xPos=10;**

**}**

**else**

**{**

**xPos=400;**

**}**

**points[i] = new Point(xPos, 10 \* i);**

**}**

Код будет выглядеть следующим образом:

public partial class Form1 : Form

{

Point[] points = new Point[50];

Pen pen = new Pen(Color.Black, 2);

public Form1()

{

InitializeComponent();

}

private void Form1\_Paint(object sender, PaintEventArgs e)

{

Graphics g = e.Graphics;

g.DrawLines(pen, points);

}

private void Form1\_Load(object sender, EventArgs e)

{

for (int i = 0; i < 20; i++)

{

int xPos;

if (i % 2 == 0)

{

xPos = 10;

}

else

{

xPos = 400;

}

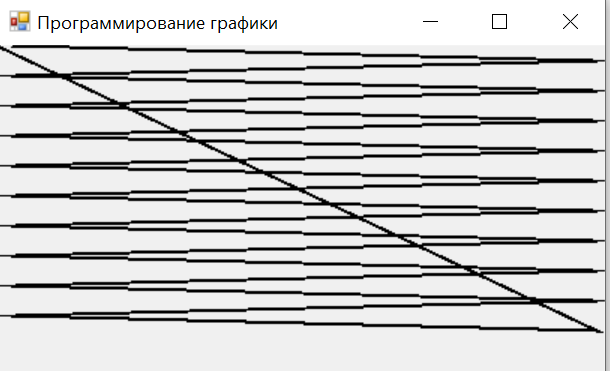
points[i] = new Point(xPos, 10 \* i);

}

}

}

Результат приведен на рис. 9.3.

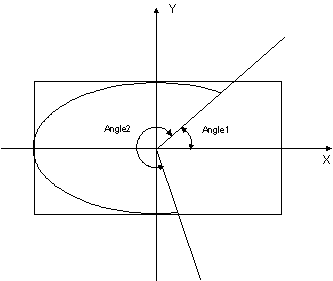


*Рис. 9.3. Пример использования массива точек*

Для прорисовки прямоугольников можно использовать метод **DrawRectangle(Pen, int, int, int, int);** В качестве первого параметра передается перо класса Pen. Остальные параметры задают расположение и размеры прямоугольника.

Для прорисовки многоугольников можно использовать метод **DrawPolygon(Pen, Point[]);**

Метод **DrawEllipse** рисует эллипс, вписанный в прямоугольную область, расположение и размеры которой передаются ему в качестве параметров. При помощи метода **DrawArc** программа может нарисовать сегмент эллипса. Сегмент задается при помощи координат прямоугольной области, в которую вписан эллипс, а также двух углов, отсчитываемых в направлении против часовой стрелки. Первый угол Angle1 задает расположение одного конца сегмента, а второй Angle2 — расположение другого конца сегмента (рис. 9.4.).



*Рис. 9.4. Углы и прямоугольник, задающие сегмент эллипса*

В классе **Graphics** определен ряд методов, предназначенных для рисования закрашенных фигур. Имена некоторых из этих методов, имеющих префикс Fill:

**FillRectangle** (рисование закрашенного прямоугольника), **FillRectangles** (рисование множества закрашенных многоугольников), **FillPolygon** (рисование закрашенного многоугольника), **FillEllipse** (рисование закрашенного эллипса) **FillPie** (рисование закрашенного сегмента эллипса) **FillClosedCurve** (рисование закрашенного сплайна) **FillRegion** (рисование закрашенной области типа **Region**).

Есть два отличия методов с префиксом Fill от одноименных методов с префиксом Draw. Прежде всего, методы с префиксом Fill рисуют закрашенные фигуры, а методы с префиксом Draw — не закрашенные. Кроме этого, в качестве первого параметра методам с префиксом Fill передается не перо класса **Pen**, а кисть класса **Brush**.

Как видите платформа .Net содержит большое число классов со многими методами и свойствами. Нет смысла описывать все классы, методы в каком- либо учебнике или в данном пособии, поскольку по любому методу или классу можно получить MSDN справку набрав наименование метода в среде Visual Studio и нажав на нем клавишу F1. Также, при наборе метода в редакторе кода среда показывает краткую справку о передаваемых параметрах.

***9.5. Выполнение индивидуального задания***

Изучите с помощью справки MSDN методы и свойства классов **Graphics**, **Color**, **Pen** и **SolidBrush**. Создайте собственное приложение выводящее на форму рисунок, состоящий из различных объектов (линий, многоугольников, эллипсов, прямоугольников и пр.), не закрашенных и закрашенных полностью. Используйте разные цвета и стили линий (сплошные, штриховые, штрих пунктирные).