С. М. Станкевич

Технологии программирования

Лабораторная работа № 4

Программирование на языке C#. Построение графиков функций. Работа с графическими примитивами

С. М. Станкевич

Технологии программирования

Лабораторная работа № 4

Программирование на языке C#. Построение графиков функций. Работа с графическими примитивами

Учебно-методическое пособие для студентов, обучающихся по специальности «Прикладная информатика (программное обеспечение компьютерных систем)»

Витебск, 2020

Содержание

Введение

Целью лабораторной работы является создание проектов оконного приложения с помощью IDE Microsoft Visual Studio.

Задачи лабораторной работы:

– изучить приёмы работы и методы настройки интегрированной среды разработки приложений *Microsoft Visual Studio*;

– научится размещать элементы управления на форме и настраивать их внешний вид;

– научиться программировать элементы управления для организации работы приложения;

– научиться составлять каркас простейших консольных и оконных приложений в среде *Microsoft Visual Studio*;

– изучить возможности построения графиков с помощью элемента управления `Chart`;

– изучить возможности *Microsoft Visual Studio*по создание простейших графических изображений.

– написать и отладить программу:

1) построения графика заданной функции.
2) построения электрической схемы.

1. Построение графиков функций

1.1. Элемент управления `Chart`

Обычно результаты расчётов представляются в виде графиков и диаграмм. Библиотека *.NET Framework* имеет мощный элемент управления `Chart` для отображения графической информации (рис. 1).

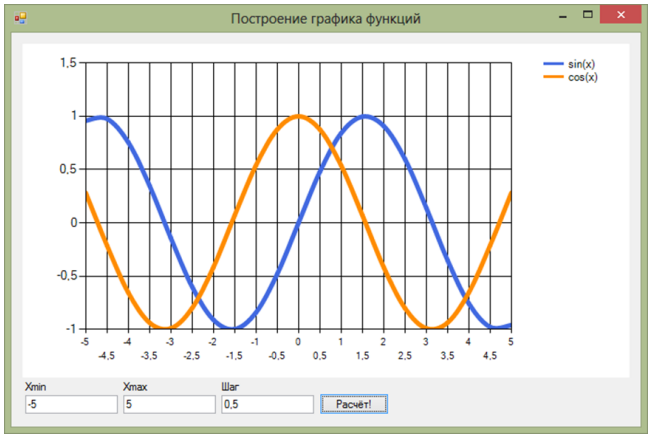

***Рис. 1** — Графическая информация, представленная с помощью элемента управления `Chart`*

Построение графика (диаграммы) производится после вычисления таблицы значений функции $y = f(x)$ на интервале $[x_{min}; x_{max}]$ с заданным шагом Δx. Полученная таблица передаётся в специальный массив `Points` объекта `Series` элемента управления `Chart` с помощью метода `DataBindXY()`. Элемент управления `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. В элементе управления `Chart` можно настроить толщину, стиль и цвет линий, параметры шрифта подписей, шаги разметки координатной сетки и многое другое. В процессе работы программы изменение параметров возможно через обращение к соответствующим свойствам элемента управления `Chart`. Так, например, свойство `AxisX` содержит значение максимального предела нижней оси графика и при его изменении во время работы программы автоматически изменяется изображение графика.

1.2. Пример написания программы

Пусть требуется составить программу, отображающую графики функций $y = \sin x$ и $y = \cos x$ на интервале $[x_{min}; x_{max}]$. В приложении следует предусмотреть возможность изменения разметки координатных осей, а также шага построения таблицы.

Прежде всего, следует поместить на форму сам элемент управления `Chart`. Он располагается в панели элементов в разделе `Данные`. Список графиков хранится в свойстве `Series`, который можно изменить, выбрав соответствующий пункт в окне свойств. Поскольку на одном поле требуется вывести два отдельных графика функций, нужно добавить ещё один элемент. Оба элемента, и существующий, и добавленный, нужно соответствующим образом настроить: изменить тип диаграммы `ChartType` на `Spline`. Здесь же можно изменить подписи к графикам с абстрактных `Series1` и `Series2` на `sin(x)` и `cos(x)` — за это отвечает свойство `Legend`. Наконец, с помощью свойства `BorderWidth` можно сделать линию графика потолще, а затем поменять цвет линии с помощью свойства `Color`.

Далее приведен текст обработчика нажатия кнопки Расчёт! (см. рис. 1), который выполняет все требуемые настройки и расчёты и отображает графики функций:

```
private void btnCalc_Click(object sender, EventArgs e) {
  // Считывание из полей формы требуемые значения
  double Xmin = double.Parse(textBoxXmin.Text);
  double Xmax = double.Parse(textBoxXmax.Text);
  double Step = double.Parse(textBoxStep.Text);

  // Вычисление количества точек графика
  int count = (int)Math.Ceiling((Xmax - Xmin) / Step) + 1;

  // Массив значений X - общий для обоих графиков
  double[] x = new double[count];

  // Два массива Y - по одному для каждого графика
  double[] y1 = new double[count];
  double[] y2 = new double[count];

  // Вычисление координат точек для графиков функции
  for (int i = 0; i < count; i++) {
    // Вычисление значение X
    x[i] = Xmin + Step * i;
    // Вычисление значение функций в точке X
    y1[i] = Math.Sin(x[i]);
    y2[i] = Math.Cos(x[i]);
  }

  // Настройка оси абсцисс графика
  chart1.ChartAreas[0].AxisX.Minimum = Xmin;
  chart1.ChartAreas[0].AxisX.Maximum = Xmax;
```

```
    // Вычисление шага сетки
    chart1.ChartAreas[0].AxisX.MajorGrid.Interval = Step;

    // Добавление вычисленных значений в графики
    chart1.Series[0].Points.DataBindXY(x, y1);
    chart1.Series[1].Points.DataBindXY(x, y2);
}
```

2. Компьютерная графика

2.1. Событие `Paint`

Для форм в C# предусмотрен 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 передаётся событие `Paint`. Все, что требуется от программиста, это создать обработчик данного события (рис. 2), наполнив его необходимой функциональностью.

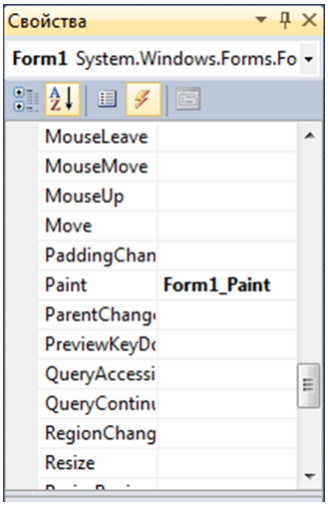

Рис. 2 — *Создание обработчика события* `Paint`

2.2. Объект `Graphics` для рисования

Для рисования линий и фигур, отображение текста, вывода изображений и т. д. нужно использовать объект `Graphics`. Этот объект предоставляет поверхность рисования и используется для создания графических изображений.

Работа с графикой представляется в два этапа:

– создание или получение объекта `Graphics`;

– использование объекта `Graphics` для рисования.

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;
    // Далее вставляется код рисования
    . . .
}
```

2.3. 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 незакрашенных фигур (таких, например, как прямоугольные рамки), а вторые — для рисования закрашенных геометрических фигур. Ниже рассматривается применение наиболее часто используемых методов, более полную информацию можно найти в документации.

Метод **DrawLine()** рисует линию, соединяющую две точки с заданными координатами. У метода есть несколько перегруженных версий:

```
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 задаёт инструмент для рисования линии — *перо*. Перья создаются как объекты класса **Pen**. Например, чёрное перо толщиной 2px создаст следующая инструкция:

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

При создании пера можно 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 запятой.

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

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

Чтобы понять принципы работы метода **DrawLine()**, создайте приложение и разместите в нём 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, 200, 4 * i + 20);
}
```

Результат выполнения этого фрагмента — 50 горизонтальных линий — показан на рис. 3.

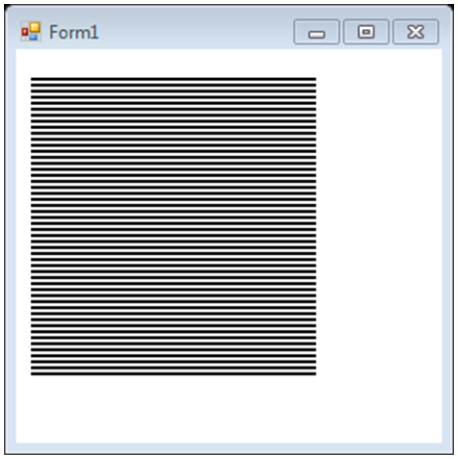

Рис. 3 — *Рисование линий с помощью метода* `DrawLine()`

Метод `DrawLines()` позволяет соединить между собой несколько точек. Координаты этих точек по горизонтальной и вертикальной осям 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. Код будет выглядеть следующим образом:

```
Point[] points = new Point[50];
Pen pen = new Pen(Color.Black, 2);

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;
    xPos = (i % 2 == 0) ? 10 : 400;
    points[i] = new Point(xPos, 10 * i);
  }
}
```

Результат выполнения этого фрагмента показан на рис. 4.

Для прорисовки прямоугольников можно использовать метод `DrawRectangle()`:

```
DrawRectangle(Pen, int, int, int, int);
```

В качестве первого параметра передаётся перо класса `Pen`. Остальные параметры задают расположение и размеры прямоугольника.

Для прорисовки многоугольников можно использовать следующий метод:

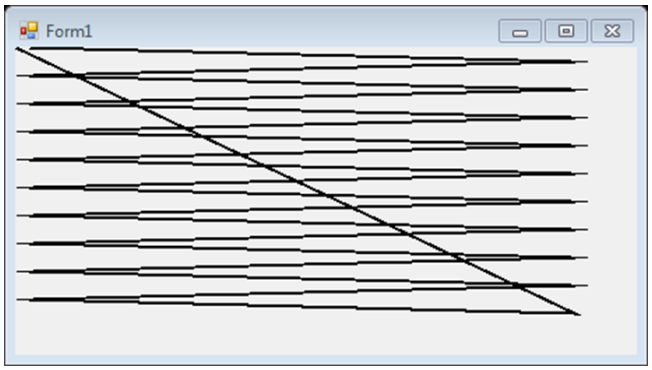

Рис. 4 — *Пример использования массива точек — построение замкнутой линии*

```
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 задаётся 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. 5).

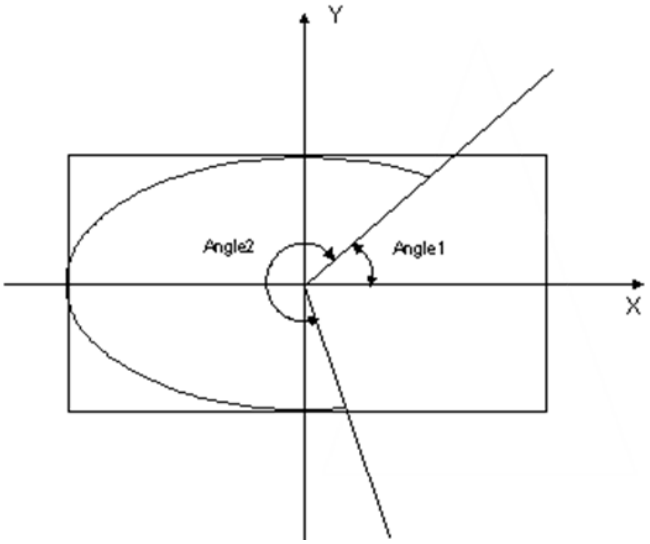

Рис. 5 — *Углы и прямоугольник, задающие сегмент эллипса*

В классе `Graphics` определен ряд методов, предназначенных для рисования закрашенных фигур. Такие методы в названии имеют префикс `Fill*`, например: закрашенный прямоугольник `FillRectangle()`, множество закрашенных прямоугольников `FillRectangles()`, закрашенный многоугольник `FillPolygon()`, закрашенный эллипс `FillEllipse()`, закрашенный сегмент эллипса `FillPie()`, закрашенный сплайн `FillClosedCurve()`, закрашенная область типа *Region* `FillRegion()`.

Есть два отличия методов с префиксом `Fill*` от одноимённых методов с префиксом `Draw*`. Прежде всего, методы с префиксом `Fill*` рисуют закрашенные фигуры, а методы с префиксом `Draw*` — незакрашенные. Кроме этого, в качестве первого параметра методам с префиксом `Fill*` передаётся не перо класса `Pen`, а кисть класса `SolidBrush`. Например, фрагмент кода для вывода закрашенного прямоугольника может выглядеть так:

```
SolidBrush Brsh = new SolidBrush(Color.DeepPink);
g.FillRectangle(Brsh, 0, 0, 100, 100);
```

3. Задания для самостоятельного выполнения

3.1. Задание № 1

Указания по выполнению. Создайте приложение, выводящее на форму графики функций для соответствующего варианта из лабораторной работы № 2 (рис. 6). Откорректируйте элементы управления на форме для правильного и красивого отображения графической и текстовой информации.

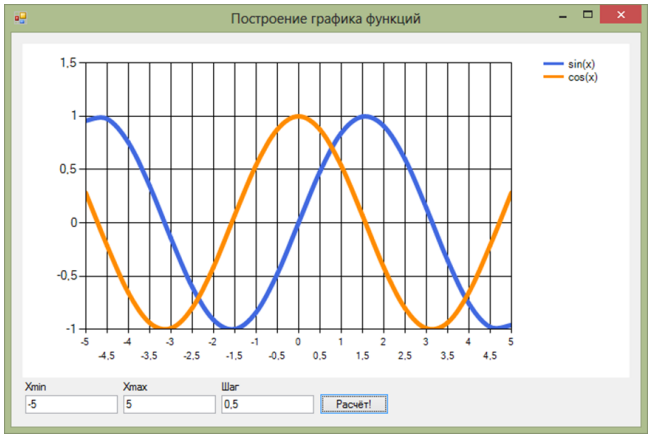

Рис. 6 — *Примерный вид окна приложения для построения графиков функции*

3.2. Задание № 2

Указания по выполнению. Изучите с помощью документации на сайте *Microsoft Docs* методы и свойства классов `Graphics`, `Color`, `Pen` и `SolidBrush`. Создайте приложение, выводящее на форму рисунок — электрическую схему (рис. 7), состоящую из различных объектов (линий, окружностей, прямоугольников и пр.). Варианты схем расположены в СДО.

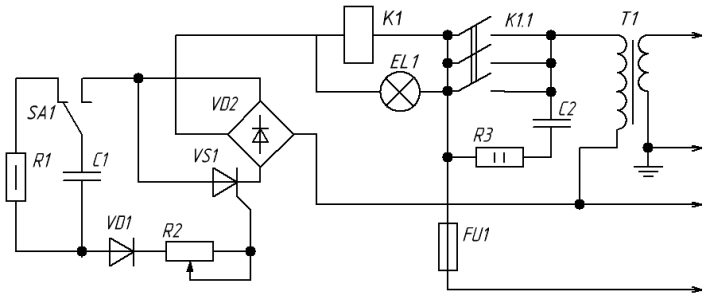

Рис. 7 — *Пример электрической схемы для рисования*

Список использованных источников

1. Троелсен Э. C# и платформа .NET / [пер. с англ. Р. Михеев]. — Санкт-Петербург [и др.]: Питер, 2007. — 796 с.
2. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft.NET Framework 2.0 на языке C#: [пер. с англ.]. — Санкт-Петербург [и др.]: Питер, 2007. — 636 с.
3. Павловская, Т. А. C#. Программирование на языке высокого уровня. Учебник для вузов. — СПб.: Питер, 2007. — 432 с.
4. Марченко А. Л. Основы программирования на C# 2.0: учебное пособие. — Москва: Интернет-Университет Информационных Технологий: БИНОМ. Лаборатория знаний, 2007. — 551 с.
5. Нортроп Т. Основы разработки приложений на платформе Microsoft .NET Framework: учебный курс Microsoft / [пер. с англ. под общ. ред. А. Е. Соловченко]. — Санкт-Петербург [и др.]: Питер, 2007. – 842 с.
6. Дёмин А. Ю. Лабораторный практикум по информатике: учебное пособие / А. Ю. Дёмин, В. А. Дорофеев; Томский политехнический университет. — Томск: Изд-во Томского политехнического университета, 2014. — 134 с.
7. Потапова, Л. Е. Алгоритмизация и программирование на языке C#: метод. рекомендации к выполнению лабораторных работ / Л. Е. Потапова, Т. Г. Алейникова. — Витебск: ВГУ имени П. М. Машерова, 2014. — 50 с.
8. Документация по C# [Электронный ресурс] // Microsoft Docs. — https://docs.microsoft.com/ru-ru/dotnet/csharp/.
9. Windows Forms [Электронный ресурс] // Microsoft Docs. — https://docs.microsoft.com/ru-ru/dotnet/framework/winforms/.
10. Практическое руководство. Создание объектов Graphics для рисования [Электронный ресурс] // Microsoft Docs. — https://docs.microsoft.com/ru-ru/dotnet/framework/winforms/advanced/how-to-create-graphics-objects-for-drawing