**Лабораторная работа №1**

**Тема**: Графическая поверхность. Карандаш и кисти. Графические примитивы: Линия. Ломаная линия. Прямоугольник.

**Цель работы:** Получения практических навыков при работе с графическими примитивами.

**Теоретический материал.**

Отображение графики для C++ в VS2010 обеспечивает компонент PictureBox (рис. 1.1). Подобно тому, как художник рисует на поверхности холста, программа "рисует" на *графической поверхности* компонента PictureBox.

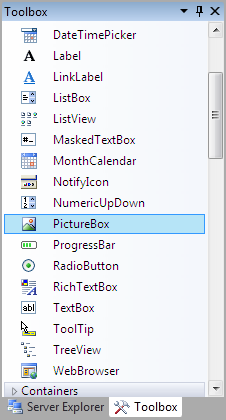


Рис. 1. 1 Компонент PictureBox

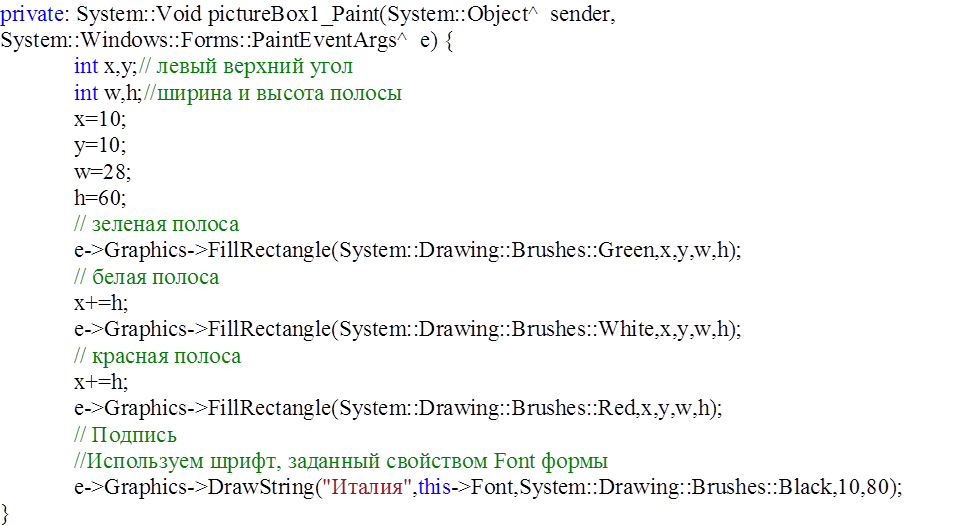
Графическая поверхность компонента PictureBox представляет собой объект Graphics, методы которого и обеспечивают вывод графики. Таким образом, для того чтобы на поверхности компонента PictureBox появилась линия, прямоугольник, окружность или загруженная из файла иллюстрация, необходимо вызвать соответствующий метод объекта Graphics.

Доступ к графической поверхности объекта (свойству Graphics) есть только у функции обработки события Paint. Поэтому сформировать (отобразить) графику может только она. Здесь следует вспомнить, что событие Paint воз-никает в начале работы программы, когда ее окно, а следовательно, и все компоненты появляются на экране, а также во время работы программы вся-кий раз, когда окно или его часть вновь появляется на экране, например, после того, как пользователь сдвинет другое окно, частично или полностью перекрывающее окно программы, или развернет свернутое окно.

Создается процедура обработки события Paint обычным образом: сначала надо выбрать компонент PictureBox, затем в окне **Properties** открыть вкладку **Events** и в поле события Paint ввести имя функции обработки события (или сделать двойной щелчок левой кнопкой мыши).

В качестве примера в листинге 1.1 приведена функция обработки события Paint компонента PictureBox, которая в поле компонента рисует итальянский флаг. Обратите внимание: доступ к объекту Graphics обеспечивает параметр e функции обработки события.

Листинг 1.1



**Графическая поверхность**

Графическая поверхность состоит из отдельных точек – пикселов. Положение точки на графической поверхности характеризуется горизонтальной (X) и вертикальной (Y) координатами (рис. 1.2). Координаты точек отсчитываются от левого верхнего угла и возрастают слева направо (координата X) и сверху вниз (координата Y). Левая верхняя точка графической поверхности имеет координаты (0, 0). Размер графической поверхности формы соответствует размеру *клиентской* области (т. е. без учета высоты области заголовка и ширины границ) формы (свойство ClientSize), а размер графической поверхности компонента PictureBox – размеру компонента.

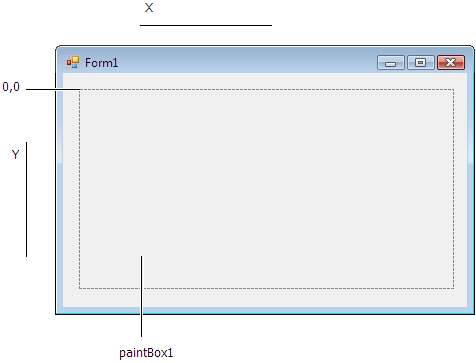


Рис. 1. 2 Координаты точек графической поверхности формы

**Карандаши и кисти**

Методы рисования графических примитивов (например, DrawLine – линия, DrawRectangle – прямоугольник, FillRectangle – область) используют *карандаши* и *кисти*. Карандаш (объект Pen) определяет вид линии, кисть (объект Brush) - вид закраски области. Например, метод

**DrawLine(System::Drawing::Pens::Black, 10,20, 100, 20);**

рисует из точки (10, 20) горизонтальную линию длиной 100 пикселов, ис-пользуя черный (Black) карандаш из стандартного набора карандашей (Pens), а метод

**FillRectangle(System::Drawing::Brushes::Green, 5, 10, 20, 20);**

при помощи стандартной кисти зеленого (Green) цвета рисует зеленый квадрат, левый верхний угол которого находится в точке (5, 10).

Карандаши и кисти определены в пространстве имен System::Drawing.

**Карандаш**

Карандаш определяет вид линии – цвет, толщину и стиль. В распоряжении программиста есть два набора карандашей: стандартный и системный. Также программист может создать собственный карандаш.

*Стандартный набор карандашей* – это цветные карандаши (всего их 141), которые рисуют непрерывную линию толщиной в *один пиксел*. Некоторые карандаши из стандартного набора приведены в табл. 1.1.

Табл.1.1 Некоторые карандаши из стандартного набора

|  |  |  |
| --- | --- | --- |
| |  | | --- | | **Карандаш** | | **Цвет** |
| Pens::Red | Красный |
| Pens::Orange | Оранжевый |
| Pens::Yellow | Желтый |
| Pens::Green | Зеленый |
| Pens::LightBlue | Голубой |
| Pens::Blue | Синий |
| Pens::Purple | Пурпурный |
| Pens::Black | Черный |
| Pens::LightGray | Серый |
| Pens::White | Белый |
| Pens::Transparent | Прозрачный |

*Системный набор карандашей* – это карандаши, цвет которых определяется текущей цветовой схемой операционной системы и совпадает с цветом какого-либо элемента интерфейса пользователя. Например, цвет карандаша SystemPens::ControlText совпадает с цветом, который в текущей цветовой схеме используется для отображения текста на элементах управления (ко-мандных кнопках и др.), а цвет карандаша SystemPens::WindowText – с цветом текста в окнах сообщений.

Карандаш из стандартного (Pens) и системного (SystemPens) наборов рисует непрерывную линию толщиной в один пиксел. Если надо нарисовать пунктирную линию или линию толщиной больше единицы, то следует использовать карандаш программиста.

*Карандаш программиста* – это объект типа Pen, свойства которого (табл. 1.2) определяют вид линии, рисуемой карандашом.

Табл.1.2 Свойства объекта Pen

|  |  |
| --- | --- |
| **Свойство** | **Описание** |
| Color | Цвет линии |
| Width | Толщина линии (задается в пикселах) |
| DashStyle | Стиль линии (DashStyle::Solid – сплошная; DashStyle::Dash – пунктирная, длинные штрихи; DashStyle::Dot – пунктирная, короткие штрихи; DashStyle::DashDot – пунктирная, чередование длинного и короткого штрихов; DashStyle::DashDotDot – пунктирная, чередование одного длинного и двух коротких штрихов; DashStyle::Custom – пунктирная линия, вид которой определяет свойство DashPattern) |
| DashPattern | Длина штрихов и промежутков пунктирной линии DashStyle::Custom |

Для того чтобы использовать карандаш программиста, его надо создать. Создает карандаш конструктор объекта Pen. Конструктор *перегружаемый*, т. е. для объекта Pen определено несколько конструкторов, которые различаются количеством параметров. Например, конструктор Pen(*Цвет*) создает карандаш указанного цвета толщиной в один пиксел, а Pen(*Цвет*, *Толщина*) – карандаш указанного цвета и толщины. В качестве параметра *Цвет* можно использовать константу типа Color (табл. 1.3). Другие константы типа Color (а их более 100) можно найти в справочной системе.

Табл.1.2 Константы типа Color

|  |  |  |
| --- | --- | --- |
| |  | | --- | | **Константа** | | **Цвет** |
| Color::Red | Красный |
| Color::Orange | Оранжевый |
| Color::Yellow | Желтый |
| Color::Green | Зеленый |
| Color::LightBlue | Голубой |
| Color::Blue | Синий |
| Color::Purple | Пурпурный |
| Color::Black | Черный |
| Color::LightGray | Серый |
| Color::White | Белый |
| Color::Transparent | Прозрачный |

Цвет, ширину линии и стиль карандаша, созданного программистом, можно изменить. Чтобы это сделать, надо изменить значение соответствующего свойства.

В листинге 1.2 приведена процедура, которая демонстрирует создание и ис-пользование карандаша программиста.

**Листинг 1.2 Создание карандаша**



Результат:

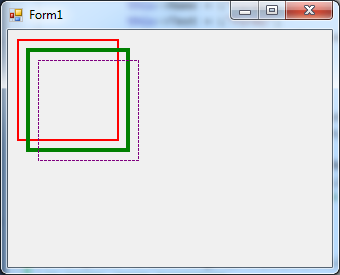


Рис. 1. 3 Результат листинга 1.2

Программист может создать карандаш, который рисует пунктирную линию, отличную от стандартной. Чтобы это сделать, надо в свойство DashPattern поместить ссылку на массив описания отрезка линии, а свойству DashStyle присвоить значение DashStyle::Custom.

Массив описания отрезка линии — это состоящий из двух элементов одномерный массив (типа float), который содержит информацию об отрезке пунктирной линии (цепочке "штрих — пропуск"). Первый элемент массива задает длину штриха, второй — пропуска. Приведенная в листинге 1.3 функция демонстрирует процесс создания карандаша, который рисует пунктирную линию, определенную программистом.

**Листинг 1.3 Линия определенная программистом**



Результат:

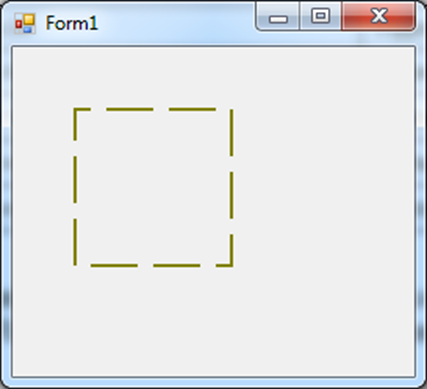


Рис. 1. 4 Результат листинга 1.3

**Кисть**

Кисти используются для закраски внутренних областей геометрических фигур. Например, инструкция

e->Graphics->FillRectangle(Brushes::DeepSkyBlue, x, y, w, h);

рисует закрашенный прямоугольник.

В приведенном примере Brushes::DeepSkyBlue – это стандартная кисть тем-но-небесного цвета. Параметры x, y определяют положение прямоугольника; w, h — его размер.

В распоряжении программиста есть три типа кистей: стандартные, штриховые и текстурные.

*Стандартная кисть* закрашивает область одним цветом (сплошная закраска). Всего есть 140 кистей, некоторые из них приведены в табл. 1.4. Полный список кистей можно найти в справочной системе.

Табл. 1.4 Некоторые кисти из стандартного набора

|  |  |
| --- | --- |
| **Кисть** | **Цвет** |
| Brushes::Red | Красный |
| Brushes::Orange | Оранжевый |
| Brushes::Yellow | Желтый |
| Brushes::Green | Зеленый |
| Brushes::LightBlue | Голубой |
| Brushes::Blue | Синий |
| Brushes::Purple | Пурпурный |
| Brushes::Black | Черный |
| Brushes::LightGray | Серый |
| Brushes::White | Белый |
| Brushes::Transparent | Прозрачный |

*Штриховая кисть* (HatchBrush) закрашивает область путем штриховки. Область может быть заштрихована горизонтальными, вертикальными или наклонными линиями разного стиля и толщины. В табл. 1.5 перечислены некоторые из возможных стилей штриховки. Полный список стилей штриховки можно найти в справочной системе.

Табл. 1.5 Некоторые стили штриховки областей

|  |  |
| --- | --- |
| **Стиль** | **Штриховка** |
| HatchStyle::LightHorizontal | Редкая горизонтальная |
| HatchStyle::Horizontal | Средняя горизонтальная |
| HatchStyle::NarrowHorizontal | Частая горизонтальная |
| HatchStyle::LightVertical | Редкая вертикальная |
| HatchStyle::Vertical | Средняя вертикальная |
| HatchStyle::NarrowVertical | Частая вертикальная |
| HatchStyle::LageGrid | Крупная сетка из горизонтальных и вертикальных линий |
| HatchStyle::SmallGrid | Мелкая сетка из горизонтальных и вертикальных линий |
| HatchStyle::DottedGrid | Сетка из горизонтальных и вертикальных линий, составленных из точек |
| HatchStyle::ForwardDiagonal | Диагональная штриховка "вперед" |
| HatchStyle::BackwardDiagonal | Диагональная штриховка "назад" |
| HatchStyle::Percent05 – HatchStyle::Percent90 | Точки (степень заполнения 5%, 10%, ..., 90%) |
| HatchStyle::HorizontalBrick | "Кирпичная стена" |
| HatchStyle::LargeCheckerBoard | "Шахматная доска" |
| HatchStyle::SolidDiamond | "Бриллиант" ("Шахматная доска", повернутая на 45) |
| HatchStyle::Sphere | "Пузырьки" |
| HatchStyle::ZigZag | "Зигзаг" |

В листинге 1.4 приведена функция, демонстрирующая процесс создания и использования штриховой кисти. При создании кисти конструктору передаются: константа HatchStyle, которая задает вид штриховки, и две константы типа Color, первая из которых определяет цвет штрихов, вторая – цвет фона.

**Листинг 1.4. Пример создания и использования штриховой кисти**

private: System::Void pictureBox1\_Paint(System::Object^ sender, System::Windows::Forms::PaintEventArgs^ e) {

// штриховка (HatchBrush-кисть)

HatchBrush^ hBrush =gcnew HatchBrush(HatchStyle::DottedGrid,Color::Black, Color::SkyBlue);

e->Graphics->FillRectangle(hBrush, 20, 20, 90, 60);

}

*Замечание*: для нормальной роботы программы необходимо в код добавить подключение пространства имен

**using namespace System::Drawing::Drawing2D;**

Результат:

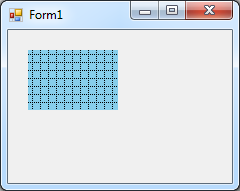


Рис. 1. 5 Результат работы листинга программы 1.4

*Текстурная кисть* (TextureBrush) представляет собой рисунок, который обычно загружается во время работы программы из файла (bmp, jpg или gif) или из ресурса. Закраска области текстурной кистью выполняется путем дублирования рисунка внутри области.

Листинг 1.5 демонстрирует процесс создания и использования текстурной кисти. Кисть создает конструктор, которому в качестве параметра передается текстура – имя графического файла.

Возможна градиентная, в простейшем случае двухцветная, закраска области, при которой цвет закраски области меняется от одного к другому вдоль линии градиента.

Чтобы закрасить область градиентом, надо создать *градиентную* кисть. При создании кисти конструктору передаются линия градиента и две константы, определяющие цвет градиента. Например:

myBrush = gcnew System::Drawing::Drawing2D::LinearGradientBrush

(Point(10,10),Point(110,10), Color::Red, Color::White);

После того как градиентная кисть будет создана, ее можно использовать для закраски области:

e->Graphics->FillRectangle(myBrush,10,10,100,40);

Следует обратить внимание, для равномерной закраски области координаты точек линии градиента должны находиться на границе закрашиваемой области или быть вне ее.

Иногда возникает необходимость, используя один и тот же градиент, закрасить несколько областей разного размера. Если попытаться закрасить область большего размера градиентной кистью меньшего размера, то область большего размера будет выглядеть так, как показано на рис. 1.6.

**Листинг 4.5. Пример создания и использования текстурной кисти**



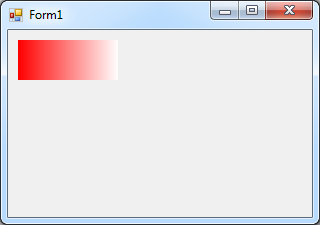


Рис. 1. 6 Размер градиентной кисти соответствует размеру области

Чтобы область была закрашена равномерно, для нее надо создать свою (но-вую) градиентную кисть или трансформировать существующую. Приведенная в листинге 1.6 функция показывает, как это сделать. Она же показывает, как можно изменить цвет градиентной кисти.

**Листинг 1.6. Пример создания и изменения градиентной кисти**

private: System::Void pictureBox1\_Paint(System::Object^ sender, System::Windows::Forms::PaintEventArgs^ e) {

Rectangle myRect = Rectangle(20,10,200,40);

LinearGradientBrush^ myBrush =

gcnew LinearGradientBrush(myRect,Color::Red,Color::White,0.0f,true);

e->Graphics->FillRectangle(myBrush, myRect);

// изменить цвет

array<Color>^ nc; // nc — NewColor

nc = gcnew array<Color>(2);

nc[0] = Color::Green;

nc[1] = Color::Yellow;

myBrush->LinearColors = nc;

// Так как будем закрашивать область другого размера,

// то надо изменить размер градиентной кисти

array<Point>^ transformArray ={Point(20,60),Point(420,60),Point(20,100)};

Matrix^ myMatrix = gcnew Matrix(myRect,transformArray);

myBrush->MultiplyTransform(myMatrix, MatrixOrder::Prepend);

// рисуем новой кистью

e->Graphics->FillRectangle(myBrush, 20, 60, 400, 40);

// восстановить размер кисти

myBrush->ResetTransform();

// рисуем область такого же размера, как и первую

e->Graphics->FillRectangle(myBrush, 20, 120, 200, 40);

}

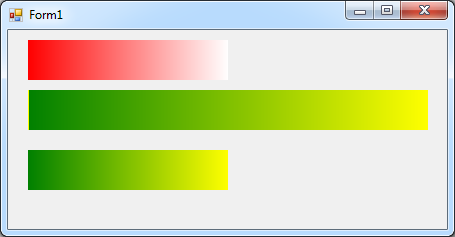


Рис. 1. 7 Результат создания и изменения градиентной кисти

**Графические примитивы**

Любая картинка, чертеж, схема представляет собой совокупность графических *примитивов*: точек, линий, окружностей, дуг, текста и др.

Вычерчивание графических примитивов на графической поверхности (Graphics) выполняют соответствующие методы (табл. 1.6).

Табл.1.6 Некоторые методы вычерчивания графических примитивов

|  |  |
| --- | --- |
| **Метод** | **Действие** |
| DrawLine(Pen, x1, y1, x2, y2), DrawLine(Pen, p1, p2) | Рисует линию. Параметр Pen определяет цвет, толщину и стиль линии; параметры x1, y1, x2, y2 или p1 и p2 — координаты точек начала и конца линии |
| DrawRectangle(Pen, x, y, w, h) | Рисует контур прямоугольника. Параметр Pen определяет цвет, толщину и стиль границы прямоугольника: параметры x, y– координаты левого верхнего угла; параметры w и h за-дают размер прямоугольника |
| FillRectangle(Brush, x, y, w, h) | Рисует закрашенный прямоугольник. Параметр Brush определяет цвет и стиль закраски прямоугольника; параметры x, y – координаты левого верхнего угла; параметры w и h задают размер прямоугольника |
| DrawEllipse(Pen, x, y, w, h) | Рисует эллипс (контур). Параметр Pen определяет цвет, толщину и стиль линии эллипса; параметры x, y, w, h – координаты левого верхнего угла и размер прямоугольника, внутри которого вычерчивается эллипс |
| FillEllipse(Brush, x, y, w, h) | Рисует закрашенный эллипс. Параметр Brush определяет цвет и стиль закраски внутренней области эллипса; параметры x, y, w, h – координаты левого верхнего угла и размер прямоугольника, внутри которого вычерчивается эллипс |
| DrawPolygon(Pen, P) | Рисует контур многоугольника. Параметр Pen определяет цвет, толщину и стиль линии границы многоугольника; параметр P (массив типа Point) — координаты углов многоугольника |
| FillPolygon(Brush, P) | Рисует закрашенный многоугольник. Пара-метр Brush определяет цвет и стиль закраски внутренней области многоугольника; пара-метр P (массив типа Point) – координаты углов многоугольника |
| DrawString(str, Font, Brush, x, y) | Выводит на графическую поверхность строку текста. Параметр Font определяет шрифт; Brush – цвет символов; x и y – точку, от ко-торой будет выведен текст |
| DrawImage(Image, x, y) | Выводит на графическую поверхность иллюстрацию. Параметр Image определяет иллюстрацию; x и y – координату левого верхнего угла области вывода иллюстрации |

Один и тот же элемент можно нарисовать при помощи *разных*, но имеющих *одинаковые* имена методов (вспомните: возможность объявления функций, имеющих одинаковые имена, но разные параметры, называется *перегрузкой).*

Например, прямоугольник можно нарисовать методом DrawRectangle, которому в качестве параметров передаются координаты левого верхнего угла и размеры прямоугольника:

e->Graphics->DrawRectangle(Pens::Black, x, x, w, h)

Эту же задачу может решить метод DrawRectangle, которому в качестве параметра передается структура типа Rectangle, поля которой определяют прямоугольник (положение и размер):

Rectangle aRect = Rectangle(20,100,50,50);

e->Graphics->DrawRectangle(Pens::Blue, aRect);

Существование нескольких методов, выполняющих одну и ту же задачу, позволяет программисту выбрать метод, наиболее подходящий для решения конкретной задачи.

В качестве параметров методов вычерчивания графических примитивов часто используется структура Point. Ее поля X и Y определяют положение (координаты) точки графической поверхности. Например:

**Point p1 = Point(10,10);**

**Point p2 = Point(100,10);**

*// рисуем линию из p1 в p2*

**e->Graphics->DrawLine(Pens::Green, p1, p2);**

**Линия**

Метод DrawLine рисует прямую линию. В инструкции вызова метода следует указать карандаш, которым надо нарисовать линию, и координаты точек начала и конца линии:

DrawLine(aPen, x1, y1, x2, y2)

или

DrawLine(aPen, p1, p2)

Параметр aPen (типа Brush) задает карандаш, которым рисуется линия, x1 и y1 или p1 — точку начала линии, а x2 и y2 или p2 — точку конца линии. Параметры x1, y1, x2 и y2 должны быть одного типа (Integer или Single). Тип параметров p1 и p2 — Point.

Например, инструкция

e->Graphics->DrawLine(Pens::Green,10,10,300,10

рисует зеленую линию толщиной в один пиксел из точки (10, 10) в точку (300, 10).

Эту же линию можно нарисовать и так:

Point p1 = Point(10,10);

Point p2 = Point(300,10);

e->Graphics->DrawLine(Pens::Green, p1, p2);

Программа "График" (ее окно приведено на рис. 1.8, а текст — в листинге 1.7) показывает использование метода DrawLine. Данные, отображаемые в окне программы, загружаются из текстового файла.

Загружает данные из файла конструктор формы (файл данных должен находиться в папке приложения, при запуске программы в режиме отладки из среды разработки — в папке Debug). Он же выполняет обработку массива данных – ищет минимальный и максимальный элементы ряда. Конструктор также, в случае успешной загрузки данных, задает функцию обработки собы-тия Paint формы. Непосредственно вывод графика на поверхность формы выполняет функция drawDiagram. Ее вызывает функция обработки события Paint формы, а также косвенно (путем вызова метода Refresh) — функция обработки события Resize формы.



Рис. 1. 8 Программа "График"

**Листинг 1.7. График**

public ref class Form1 : public System::Windows::Forms::Form

{

public:

Form1(void)

{

InitializeComponent();

// чтение данных из файла в массив

System::IO::StreamReader^ sr; // поток для чтения

try

{

sr = gcnew System::IO::StreamReader(

Application::StartupPath + "\\usd.dat");\*/

// создаем массив

d = gcnew array<double>(10);

// читаем данные из файла

int i = 0;

String^ t = sr->ReadLine();

while ((t != String::Empty) && (i < d->Length))

for(int i=0;i<10;i++)

{

// записываем считанное число в массив

d[i++] = Convert::ToDouble(i);

//t = sr->ReadLine();

}

// закрываем поток

sr->Close();

// определить минимальное и максимальное значения ряда

min = d[0];

max = d[0];

for (int i = 1; i < d->Length; i++)

{

if (d[i] > max) max = d[i];

if (d[i] < min) min = d[i];

}

// Данные загружены.

// Задаем функцию обработки события Paint

this->Paint += gcnew System::Windows::Forms::PaintEventHandler(this,&Form1::drawDiagram);

}

// Обработка исключений: файл данных не найден

catch (System::IO::FileNotFoundException^ ex)

{

MessageBox::Show(ex->Message + "\n",

"График",

MessageBoxButtons::OK,

MessageBoxIcon::Error);

}

}

protected:

~Form1()

{

if (components)

{

delete components;

}

}

private:

System::ComponentModel::Container ^components;

array<double>^ d; // ряд данных

double max; // максимальный элемент массива

double min; // минимальный элемент массива

#pragma region Windows Form Designer generated code

void InitializeComponent(void)

{

this->SuspendLayout();

//

// Form1

//

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(284, 262);

this->Name = L"Form1";

this->Text = L"Form1";

this->Resize += gcnew System::EventHandler(this, &Form1::Form1\_Resize);

this->ResumeLayout(false);

}

#pragma endregion

// строим график

void drawDiagram(System::Object^ sender,

System::Windows::Forms::PaintEventArgs^ e)

{

// графическая поверхность

System::Drawing::Graphics^ g = e->Graphics;

// шрифт подписей данных

System::Drawing::Font^ dFont =gcnew System::Drawing::Font("Tahoma", 9);

// шрифт заголовка

System::Drawing::Font^ hFont =

gcnew System::Drawing::Font("Tahoma", 13, FontStyle::Regular);

String^ header = "Изменение курса доллара";

// ширина области отображения текста

int wh =(int)g->MeasureString(header, hFont).Width;

int x = (this->ClientSize.Width - wh) / 2;

//g->DrawString(header,hFont, Brushes::DarkGreen, x, 5);

/\* Область построения графика.

Отступы: сверху — 100;

снизу — 20;

слева — 20;

справа — 20.

ClientSize — размер внутренней области окна

График строим в отклонениях от минимального значения

ряда данных так, чтобы он занимал всю область построения

\*/

// рисуем график

int x1, y1, x2, y2;

// расстояние между точками графика (шаг по Х)

int sw = (int)((this->ClientSize.Width - 40) /(d->Length - 1));

// первая точка

x1 = 20;

y1 = this->ClientSize.Height - 20 -(int)((this->ClientSize.Height - 100) \*(d[0] - min) / (max - min));

// маркер первой точки

g->DrawRectangle(Pens::Black, x1 - 2, y1 - 2, 4, 4);

// подпись численного значения первой точки

g->DrawString(Convert::ToString(d[0]),dFont, Brushes::Black, x1 - 10, y1 - 20);

// остальные точки

for (int i = 1; i < d->Length; i++)

{

x2 = 8 + i \* sw;

y2 = this->ClientSize.Height - 20 -(int)((this->ClientSize.Height - 100) \*(d[i] - min) / (max - min));

// маркер точки

g->DrawRectangle(Pens::Black, x2 - 2, y2 - 2, 4, 4);

// соединим текущую точку с предыдущей

g->DrawLine(Pens::Black, x1, y1, x2, y2);

// подпись численного значения

g->DrawString(Convert::ToString(d[i]),

dFont, Brushes::Black, x2 - 10, y2 - 20);

x1 = x2;

y1 = y2;

}

}

private: System::Void Form1\_Resize(System::Object^ sender, System::EventArgs^ e) {

this->Refresh();

}

};

}

Замечание: Листинг программы имеет ошибки, их необходимо исправить. Кроме того, добавить файл, содержащий необходимую информацию.

**Ломаная линия**

Метод DrawLines рисует ломаную линию. В качестве параметров методу передается карандаш (Pen) и массив типа Point, элементы которого содержат координаты узловых точек линии. Метод рисует ломаную линию, последовательно соединяя точки, координаты которых находятся в массиве: первую со второй, вторую с третьей, третью с четвертой и т. д. Например, следующий фрагмент кода рисует ломаную линию, состоящую из четырех звеньев.

**array**<Point>^ p; *// массив точек*

p = gcnew array<Point>(5);

p[0].X = 10; p[0].Y =50;

p[1].X = 20; p[1].Y =20;

p[2].X = 30; p[2].Y =50;

p[3].X = 40; p[3].Y =20;

p[4].X = 50; p[4].Y =50;

e->Graphics->DrawLines(Pens::Green,p);

Метод DrawLines можно использовать для вычерчивания замкнутых контуров. Для этого первый и последний элементы массива должны содержать координаты одной и той же точки.

**Прямоугольник**

Метод DrawRectangle чертит прямоугольник (рис. 4.5). В качестве параметров метода надо указать карандаш, координаты левого верхнего угла и размер прямоугольника:

DrawRectangle(aPen, x, y, w, h);

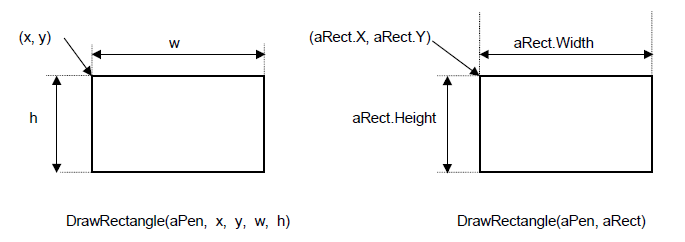


Рис. 1. 9 Метод DrawRectangle рисует прямоугольник

Вместо четырех параметров, определяющих прямоугольник, можно указать структуру типа Rectangle:

DrawRectangle(aPen, aRect);

Поля X и Y структуры aRect задают координаты левого верхнего угла прямо-угольника, а Width и Height — размер (ширину и высоту).

Вид линии границы прямоугольника (цвет, стиль и ширину) определяет параметр aPen, в качестве которого можно использовать один из стандартных карандашей или карандаш, созданный программистом.

Метод FillRectangle рисует закрашенный прямоугольник. В качестве пара-метров методу надо передать кисть, координаты левого верхнего угла и размер прямоугольника:

FillRectangle(aBrush, x, y, w, h);

Вместо x, y, w и h можно указать структуру типа Rectangle:

FillRectangle(aBrush, aRect);

Параметр aBrush, в качестве которого можно использовать стандартную или созданную программистом штриховую (HatchBrush), градиентную (LineadGradientBrush) или текстурную (TextureBrush) кисть, определяет цвет и стиль закраски области.

Далее приведен фрагмент кода (процедуры обработки события Paint), который демонстрирует использование методов DrawRectangle и FillRectangle.

Rectangle aRect; *// положение и размер прямоугольника*

*// Зеленый прямоугольник размером 60х30,*

*// левый верхний угол которого в точке (10, 10)*

aRect = Rectangle(10,10,60,30); *// положение и размер*

e->Graphics->FillRectangle(Brushes::ForestGreen, aRect);

*// Желтый прямоугольник с черной границей размером 60х30,*

*// левый верхний угол которого в точке (100, 10)*

*aRect.X = 100;*

*e->Graphics->FillRectangle(Brushes::Gold, aRect); // прямоугольник*

*e->Graphics->DrawRectangle(Pens::Black, aRect); // граница*

**Точка**

Казалось бы, чего проще — нарисовать на графической поверхности точку.Но у объекта Graphics нет метода, который позволяет это сделать. Метод SetPixel есть у объекта Bitmap. Поэтому если действительно необходимо сформировать картинку из точек, придется создать объект Bitmap, сформиро вать на его поверхности изображение, а затем это изображение при помощи метода DrawImage вывести на графическую поверхность. Но можно поступить проще — вместо точки вывести квадрат размером в один пиксел.

Например, инструкция

e->Graphics->FillRectangle(Brushes::Red,x,y,1,1);

рисует на графической поверхности красную точку.

**Многоугольник**

Метод DrawPolygon чертит многоугольник (контур). Инструкция вызова метода в общем виде выглядит так:

DrawPolygon(aPen, p)

Параметр p — массив типа Point, определяет координаты вершин много-угольника. Метод DrawPolygon чертит многоугольник, соединяя прямыми линиями точки, координаты которых находятся в массиве: первую со второй, вторую с третьей и т. д. Последняя точка соединяется с первой. Вид границы многоугольника определяет параметр aPen, в качестве которого можно использовать стандартный или созданный программистом карандаш.

Закрашенный многоугольник рисует метод FillPolygon. Инструкция вызова метода в общем виде выглядит так:

FillPolygon(aBrush, p)

Параметр aBrush, в качестве которого можно использовать стандартную или созданную программистом штриховую (HatchBrush), градиентную (LineadGradientBrush) или текстурную (TextureBrush) кисть, определяет цвет и стиль закраски внутренней области многоугольника.

Далее приведен фрагмент кода, который демонстрирует использование мето дов DrawPolygon и FillPolygon - рисует корону.

**array**<Point>^ p;

p = **gcnew array**<Point>(5);

p[0].X = 10; p[0].Y =30;

p[1].X = 10; p[1].Y =10;

p[2].X = 30; p[2].Y =20;

p[3].X = 50; p[3].Y =10;

p[4].X = 50; p[4].Y =30;

e->Graphics->FillPolygon(Brushes::Gold, p);

e->Graphics->DrawPolygon(Pens::Black,p);

**Задания.**

1. Набрать коды всех листингов, находящихся в теоретической части работы, уметь комментировать их.
2. Построить фигуру, соответствующую Вашему варианту:

|  |  |
| --- | --- |
| Вариант | Фигура |
|  | Ромб |
|  | Стрелка влево |
|  | Фигура в виде буквы W |
|  | Стрелка вниз |
|  | Фигура в виде буквы Н |
|  | Фигура в виде буквы F |
|  | Правильный пятиугольник |
|  | Правильный восьмиугольник |
|  | Фигура в виде буквы П |
|  | Правильный семиугольник |
|  | Фигура в виде буквы Ш |
|  | Трапеция |
|  | Стрелка вправо |
|  | Параллелограмм |
|  | Фигура в виде буквы И |
|  | Правильный шестиугольник |
|  | Стрелка вверх |
|  | Фигура в виде 8 |
|  | Фигура в виде 4 |
|  | Фигура в виде 9 |
|  | Квадрат в квадрате |
|  | Песочные часы |
|  | Правильный пятиугольник |
|  | Фигура в виде цифры 5 |
|  | Фигура в виде цифры 3 |