CHAPTER 16

繪圖

16-1 繪圖的基本觀念

GDI+

相信很多人都有使用 PhotoImpact 、 PhotoShop 或 Windows 小畫家的經驗, 您要繪製直線、圓、扇形、多邊形、或輸入文字等, 都有一些工具列協助完成, 這些操作的背後都由許多函式堆積而成。在.NET Framework 中, 則引入了 GDI 的新一代技術, 稱爲 GDI+, 用來繪製圖形。

首先先了解什麼是 GDI 呢？ GDI 是從 Windows 95 到 Windows 2000 隨附的舊版繪圖裝置介面(Graphics Device Interface), 是屬於繪圖方面的 API (Application Programming Interface)。因爲應用程式不能直接控制硬體, 所以當我們要進行繪圖的動作時, 必須透過 GDI 才能完成。

那 GDI+ 又是什麼呢？ GDI+ 是 GDI 的後續產品, 是一種繪圖裝置介面, 可將應用程式和繪圖硬體分隔, 讓我們能夠撰寫與裝置無關的應用程式。它可以讓我們不需注意特定顯示裝置的詳細資料, 便可在螢幕或印表機顯示資訊。我們可以呼叫 GDI+ 類別所提供的方法, 然後這些方法會適當地呼叫特定的裝置驅動程式, 而完成繪圖。下表列出了一些實作 GDI+ 的命名空間, 這些命名空間包含許多基本與進階的繪圖類別, 供程式開發者來完成各種繪圖功能。本章在此僅介紹 System.Drawing 命名空間中一些常用的繪圖類別, 希望藉由本章的介紹, 讓讀者可以輕鬆地學習 C# 的繪圖功能。

命名空間	說明
System.Drawing	提供 GDI+ 基本繪圖功能。其中 Graphics 類別提供繪圖的方法給顯示裝置
System.Drawing.Drawing2D	提供進階的二維和向量圖形功能。例如,漸層筆刷、Matrix 類別 (用來定義幾何變換) 和 GraphicsPath 類別(表示一系列連接的直線和曲線)

System.Drawing.Imaging	提供進階的GDI+影像處理功能。Metafile類別提供記錄與儲存中繼檔(Metafile)的方法。 Encoder和Decoder類別讓使用者能夠擴充GDI+來支援任何的影像格式。 PropertyItem類別提供在影像檔中儲存與擷取中繼資料(Metadata)的方法
System.Drawing.Printing	提供列印相關的服務。例如,告知印表機如何列印文件
System.Drawing.Text	提供進階的GDI+印刷樣式功能。在這個命名空間中的類別允許使用者建立並使用字型的集合

System.Drawing 命名空間

繪圖最常用的類別都放在 System.Drawing 命名空間, 線上查詢 System.Drawing 命名空間, 此命名空間常用的類別如下：

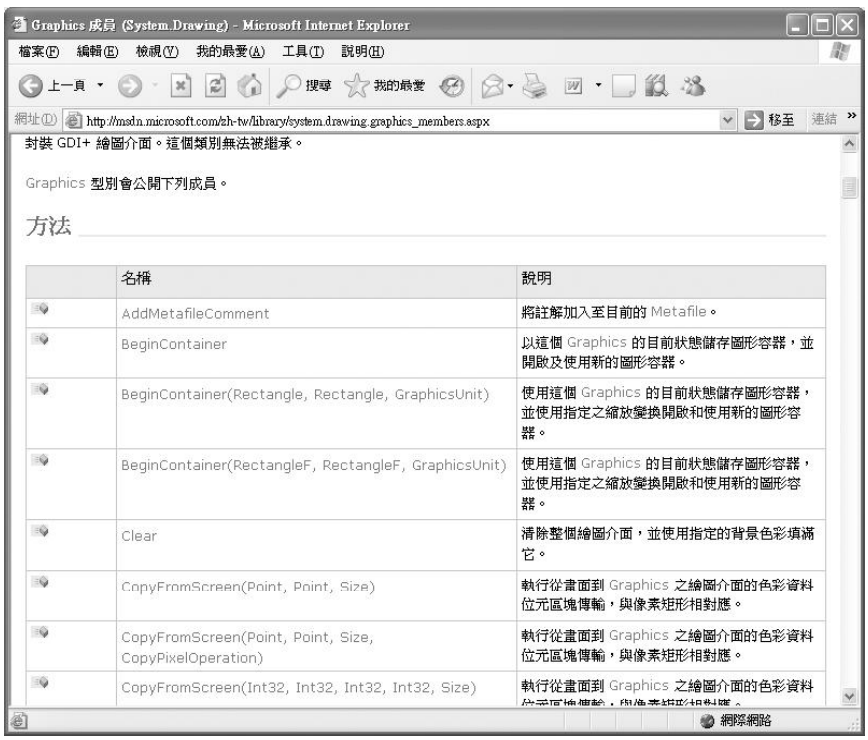
類別	說明
Bitmap	封裝GDI+點陣圖,這個點陣圖是由圖形影像的像素資料及其屬性所組成。Bitmap是用來處理像素資料所定義影像的物件
Brush	定義用於填滿圖形形狀內部的物件,例如矩形、橢圓形、派形、多邊形和路徑
Brushes	所有標準色彩的筆刷。這個類別無法被繼承
Font	定義文字的特定格式,包括字體、大小和樣式屬性(Attribute)。這個類別無法被繼承
Graphics	封裝GDI+繪圖介面。這個類別無法被繼承
Image	抽象基底類別。 Bitmap繼承Image類別, Metafile類別又繼承Bitmap類別
Pen	定義用來繪製直線與曲線的物件。這個類別無法被繼承
Pens	所有標準色彩的畫筆。這個類別無法被繼承
TextureBrush	TextureBrush類別的每一個屬性都是Brush物件,這個物件會使用影像來填滿形狀的內部。這個類別無法被繼承
SolidBrush	定義單一色彩的筆刷。筆刷是用來填滿圖形形狀,例如矩形、橢圓形、派形、多邊形和路徑。這個類別無法被繼承

System.Drawing 命名空間常用的結構如下：(輕量型的類別稱爲結構)

結構	說明
Color	表示 ARGB (Alpha、紅、綠、藍) 色彩
Point	表示整數 X 和 Y 座標的排序配對,這個配對會定義二維平面的點
PointF	表示浮點 X 和 Y 座標的排序配對,該配對會定義二維平面中的點
Rectangle	儲存四個爲一組的整數,表示矩形位置和大小。如需更多進階的區域函式, 請使用 Region 物件
Size	儲存已排序的整數配對,通常是矩形的寬度和高度
SizeF	儲存已排序的浮點數值 (Floating-Point Number) 配對, 通常是矩形的寬度和高度

Graphics

Graphics 是一個介面, 它提供了許多繪圖的方法, 如下圖所示：



因爲 Graphics 是一個繪圖介面, 我們無法直接樣例它, 而產生繪圖物件。當我們要使用繪圖功能時, 要如何產生一個繪圖物件呢？答案是使用 CreateGraphics 方法取得繪圖物件、或使用 Paint 事件, 分別說明如下：

CreateGraphics

使用 CreateGraphics 方法取得繪圖物件的語法如下：

```
Graphics      g;                                  // 宣告繪圖物件
g=          畫布物件.CreateGraphics();            // 取得畫布
```

以上敘述中的畫布物件可以是 Form1(this)、 Label 、 PictureBox1 等, 而這些控制項的區域即爲畫布的區域。例如, 以下敘述可將表單宣告爲畫布, 並於表單取得繪圖物件 gra 。

```
Graphics      g;                                  // 宣告繪圖物件
g=            this.CreateGraphics();              // 取得畫布
```

Paint 事件

Paint 事件的語法如下,

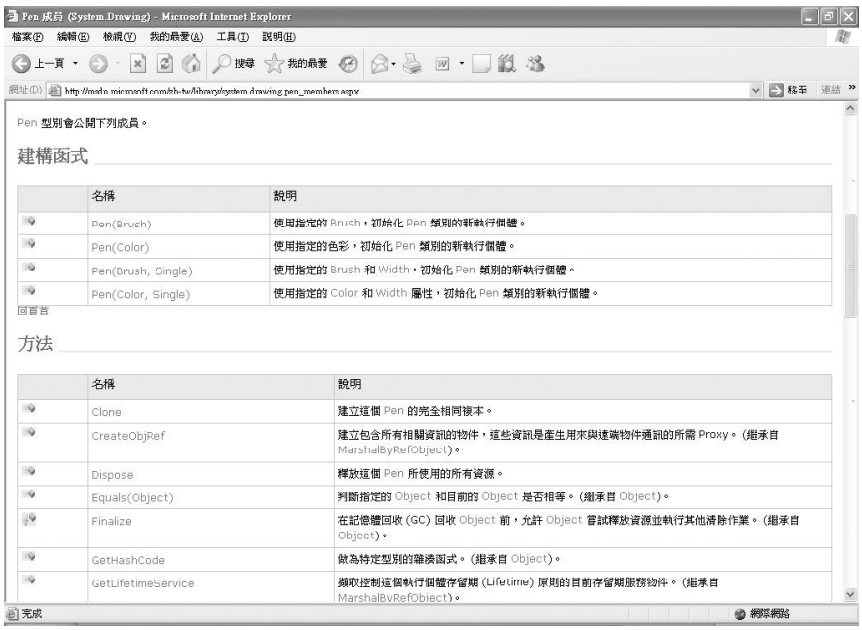
```
private      void      物件 _Paint(object      sender,      System.Windows.Forms.
PaintEventArgs       e){}
```

以上敘述的物件可以是 Form1 、 Label1 、 PictureBox1 等, 也就是 Form 、 Label 、 PictureBox 等控制項都可以拿來當畫布。其次再利用參數 e 取得繪圖物件。例如, 以下敘述可在表單利用參數 e 取得繪圖物件 g 。

```
private      void      Form1_Paint(object      sender,      System.Windows.Forms.
PaintEventArgs       e)
{
        Graphics  g;
          g=e.Graphics;
           g.DrawLine(drawPen,   10,   10,   300,   100);
}
```

Pen 類別

.NET 的繪圖至少必須藉助 Graphics 與 Pen 類別的協助, 其中 Graphics 提供許多方法, 而 Pen 類別就是畫筆了。我們可以藉由 Pen 類別提供的公用屬性, 來決定畫筆線條的顏色、粗細、與畫筆寬度。此類別提供的成員如下圖所示：



例如, 以下敘述可產生畫筆物件, 畫筆的線條顏色爲黑色, 線條粗細爲 3 。

```
Pen   p;
p   =   new   Pen(Color.Black,   3);
```

或直接寫成

```
Pen   p   =   new   Pen(Color.Black,   3);
```

以上 Pen 類別的詳細說明請看 16-3 節。

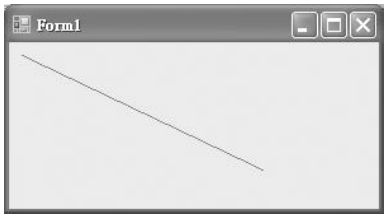
繪圖方法

Graphics 類別的常用繪圖方法有 DrawLine(直線)、 DrawRectangle(矩形)、 DrawEllipse(橢圓)、 DrawString(文字)。這些方法的詳細說明及範例請看 16-2 節。現在先使用 DrawLine(直線)繪製一條直線。例如, 以下敘述可繪製一條起點爲(10, 10), 終點爲(200, 100)的直線。

```
private     void     button1_Click(object     sender,     EventArgs     e)
{
      Graphics   g;    //g  是繪圖物件
```

```
            g=this.CreateGraphics()   ;// 設定表單爲繪圖區域
        Pen  p;
          p  =  new  Pen(Color.Red,  1);
           g.DrawLine(p,  10,  10,  200,  100);  // 在畫布畫直線
}
```

結果如右圖：



又例如, 以下敘述使用 Paint 事件繪製一條起點爲 (200, 10), 終點爲 (200, 100) 的直線。

```
private     void     Form1_Paint(object     sender,     PaintEventArgs     e)
{
        Graphics  g;
         g=  e.Graphics;
         Pen  p  =  new  Pen(Color.Green,  3);
          g.DrawLine(p,  200,  10,  200,  100);  // 在畫布畫直線
}
```

以上程式亦可簡化如下：

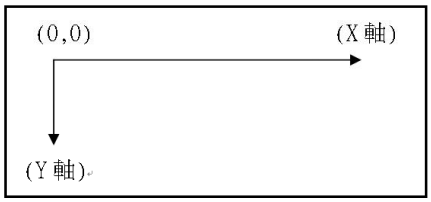
```
private     void     Form1_Paint(object     sender,     PaintEventArgs     e)
{
         Pen  p  =  new  Pen(Color.Green,  3);
           e.Graphics.DrawLine(p,  200,  10,  200,  100);  // 在畫布畫直線
}
```

其次, Paint 事件的執行時機如下：

(1) 新建的視窗。
(2) 視窗從隱藏還原爲可視。
(3) 視窗被改變大小。
(4) 視窗被別的視窗遮蓋再移開。
(5) 執行 Refresh()方法。

座標系統

GDI+ 座標原點 (0, 0) 定義在繪圖控制項的左上角, X 軸及 Y 軸分別向右及向下增加, 預設的單位長度是像素 (pixel), 如右圖所示：



若要改變單位長度, 則應探索 Graphics 類別的 PageUnit 屬性。例如, 以下敘述可將單位長度改爲英吋。

```
Graphics g= this.CreateGraphics();// 設定表單爲繪圖區域
g.PageUnit = GraphicsUnit.Inch;// 單位是 Inch
```

以下敘述可將單位長度改爲公釐。

```
Graphics g= this.CreateGraphics();// 設定表單爲繪圖區域
g.PageUnit = GraphicsUnit.Millimeter;// 單位是 mm
```

又例如, 以下敘述可繪製一條起點爲 (10, 10), 終點爲 (30, 10) 的直線。

```
private void button2_Click(object sender, EventArgs e)
{
    Graphics g= this.CreateGraphics();// 設定表單爲繪圖區域
    Pen p= new Pen(Color.Red, 1);
    g.PageUnit = GraphicsUnit.Millimeter;// 單位是 mm
    g.DrawLine(p, 10, 10, 30, 10);
}
```

以下敘述可清除圖形。

```
private void button3_Click(object sender, EventArgs e)
{
        Graphics g = this.CreateGraphics();// 設定表單爲繪圖區域
            g.Clear(this.BackColor);
}
```

範例 16-1a

示範以上繪圖功能 (請自行開啓檔案, 並執行程式)。

16-2 繪圖屬性與方法

本節將介紹一些 Graphics 類別的常用繪圖屬性與方法, 例如設定座標單位 (PageUnit)、繪製文字 (DrawString)、直線 (DrawLine)、矩形 (DrawRectangle)、橢圓 (DrawEllipse)、弧線 (DrawArc) 等方法, 如下圖, 請看以下說明。

	DrawArc	多載。繪製弧形，表示由一對座標、寬度和高度所指定的橢圓形的一部分。
	DrawBezier	多載。繪製由四個 Point 結構定義的貝茲曲線。
	DrawBeziers	多載。從 Point 結構陣列繪製一系列的貝茲曲線。
	DrawClosedCurve	多載。繪製由 Point 結構陣列定義的封閉的基本曲線。
	DrawCurve	多載。繪製穿過指定陣列的 Point 結構的基本曲線。
	DrawEllipse	多載。繪製由一對座標、高度和寬度所指定的邊框定義的橢圓形。
	DrawIcon	多載。在指定的座標處，繪製由指定之 Icon 所表示的影像。
	DrawIconUnstretched	繪製由指定之 Icon 表示的影像，但不縮放影像。
	DrawImage	多載。以原始大小，在指定之位置繪製指定的 Image。
	DrawImageUnscaled	多載。使用原始的實體大小將指定的影像繪製於座標對所指定的位置。
	DrawImageUnscaledAndClipped	必要時，繪製指定的影像，而不需要加以縮放或裁剪以容納在指定的矩形中。
	DrawLine	多載。繪製連接由座標對所指定的兩個點之直線。
	DrawLines	多載。繪製連接 Point 結構陣列的一系列直線線段。
	DrawPath	繪製 GraphicsPath。
	DrawPie	多載。繪製由橢圓形所定義的派形，該橢圓形是由座標對、寬度、高度和兩條放射線所指定。
	DrawPolygon	多載。繪製由 Point 結構陣列定義的多邊形。

繪圖屬性

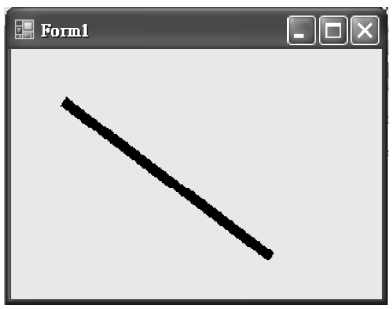
PageUnit

取得或設定此 Graphics 物件的座標單位, 其中座標單位必須是 GraphicsUnit 列舉型別的成員, 如下表所示。

成員名稱	說明
Display	指定 1/75 英吋做爲量測單位
Document	指定文件單位 (1/300 英吋) 做爲量測單位
Inch	指定英吋做爲量測單位
Millimeter	指定公釐做爲量測單位
Pixel	指定裝置像素做爲量測單位
Point	指定印表機的點 (1/72 英吋) 做爲測量單位
World	指定自然單位做爲量測

例如, 以下敘述可將座標單位設爲公釐 (mm) , 請讀者自行使用尺量線條的長度。

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
          Graphics g = this.CreateGraphics();
        Pen pen1 = new Pen(Color.Black, 2);
           g.PageUnit = GraphicsUnit.Millimeter;
         g.DrawLine(pen1, 10, 10, 50, 40);
}
```



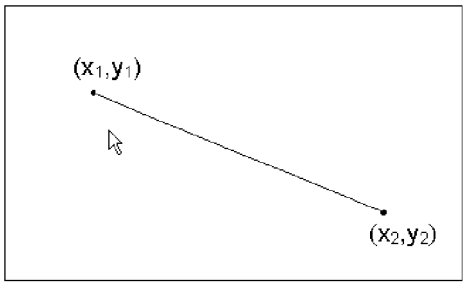
繪圖方法

DrawLine

繪製連接兩點的直線, 共有 4 種多載, 以下僅介紹最常用的一種多載, 其語法如下：

```
public    void    DrawLine(
    Pen   pen,
    int    x1,
    int    y1,
    int    x2,
    int   y2
) ;
```

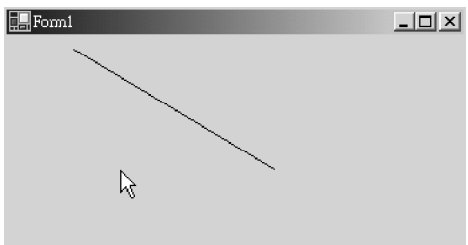
其中 pen 爲畫筆物件, (x1, y1) 爲起點的座標, (x2, y2) 爲終點的座標, 如右圖所示。



例如, 以下敘述可於表單上繪製一起點爲 (10, 10), 終點爲 (100, 100) 的直線。

```
Graphics    g    =    this.CreateGraphics();
Pen   drawPen   =   new   Pen(Color.Black,   1);
g.DrawLine(drawPen,    10,    10,    100,    100);
```

其結果如右圖：

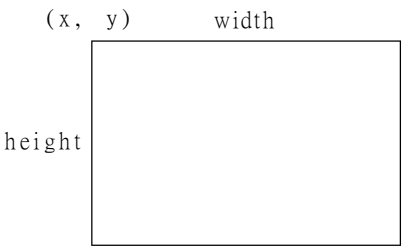


DrawRectangle

繪製矩形, 其語法如下：

```
public      void      DrawRectangle(
     Pen     pen,
     int     x,
     int     y,
     int       width,
     int       height
) ;
```

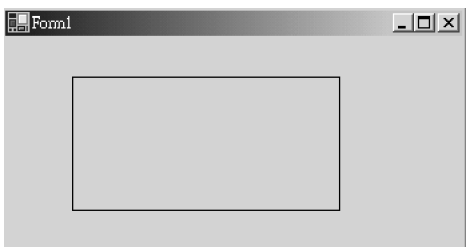
其中座標(x, y)爲矩形左上角點, width 爲矩形的寬度, height 爲矩形的高度, 如右圖所示。



例如, 以下敘述可於表單上繪出一個左上角位於 (50, 30), 寬度 200, 高度 100 的矩形。

```
Graphics      g      =      this.CreateGraphics();
Pen    drawPen    =    new    Pen(Color.Black,    1);
g.DrawRectangle(drawPen,      50,      30,      200,      100);
```

其結果如右圖：



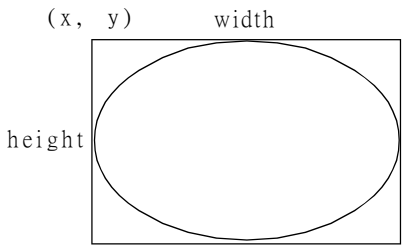
DrawEllipse

繪製橢圓, 其語法如下：

```
public      void      DrawEllipse(
     Pen     pen,
```

```
    int    x,
    int    y,
    int     width,
    int     height
) ;
```

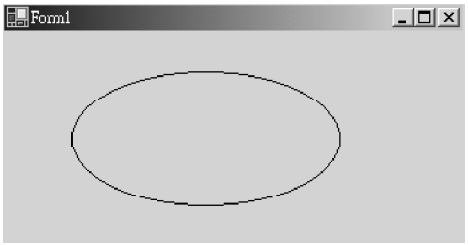
在指定左上角座標 (x, y) 寬度 (width) 及高度 (height) 的矩形內繪製橢圓形, 如右圖所示。



例如, 以下敘述可於表單上, 在一個左上角位於 (50, 30), 寬度爲 200, 高度爲 100 的矩形內繪出橢圓。

```
g      =      this.CreateGraphics();
Pen   drawPen   =   new   Pen(Color.Black,   1);
g.DrawEllipse(drawPen,    50,    30,    200,    100);
```

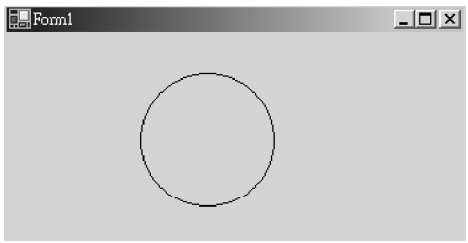
其結果如右圖：



此外, 若矩形的寬度和高度相等則可繪出一正圓形。例如, 以下敘述可於表單上, 在一個左上角位於 (50, 30), 寬度和高度皆爲 100 的矩形內繪出一正圓形。

```
g      =      this.CreateGraphics();
Pen   drawPen   =   new   Pen(Color.Black,   1);
g.DrawEllipse(drawPen,    50,    30,    100,    100);
```

其結果如下圖：



DrawArc

繪製弧線, 其語法如下：

```
public    void    DrawArc(
    Pen    pen,
    int    x,
    int    y,
    int      width,
    int       height,
    int         startAngle,
    int      sweepAngle
) ;
```

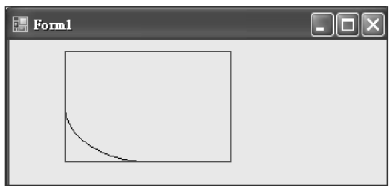
在一個指定左上角座標、寬度和高度的矩形內繪製弧線, 此弧線爲上述矩形所定義之橢圓的一部分。以上語法中各項參數的意義如下：

參數名稱	說明
Pen	Pen 物件, 決定弧線線條的色彩、寬度和樣式
x	矩形左上角的X軸座標
y	矩形左上角的Y軸座標
width	設定矩形的寬度
height	設定矩形的高度
startAngle	起始角。從 X 軸到弧線開始點的角度, 以度爲單位, 0 度在 3 點鐘方向, 角度爲正表順時針方向
sweepAngle	弧度角。從 startAngle 參數到弧線結束點的角度, 也就是弧線所掃過的角度。以度爲單位, 0 度在 3 點鐘方向, 角度爲正表順時針方向

例如, 以下敘述將於一個左上角位於 (50, 50), 寬度爲 100, 高度爲 50 的矩形內, 繪出一起始角爲 90 度, 弧度角爲 90 度的弧線。

```
g       =       this.CreateGraphics();
Pen    drawPen    =    new    Pen(Color.Black,    1);
g.DrawArc(drawPen,    50,    50,    200,    100,    90,    90);
```

其結果如右圖, 爲了方便讀者觀察起始角與弧度角, 筆者亦將對應的矩形繪出。



DrawPie

繪製扇形, 其語法如下：

```
public    void    DrawPie(
      Pen  pen,
      int  x,
      int  y,
       int   width,
        int   height,
         int   startAngle,
       int  sweepAngle
) ;
```

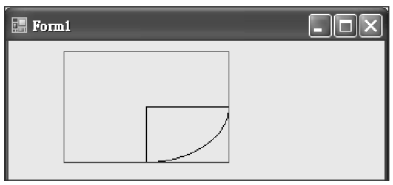
在一個指定左上角座標、寬度和高度的矩形內繪製扇形, 此扇形面積爲上述矩形所定義之橢圓形面積的一部分。以上語法中各項參數的意義如下：

參數名稱	說明
pen	Pen 物件, 決定扇形外圍線條的色彩、寬度和樣式
x	矩形左上角的X軸座標
y	矩形左上角的Y軸座標
width	設定矩形的寬度
height	設定矩形的高度
startAngle	起始角。從X軸到弧線開始點的角度,以度爲單位,0度在3點鐘方向,角度爲正表順時針方向
sweepAngle	弧度角。從startAngle參數到弧線結束點的角度,也就是弧線所掃過的角度。以度爲單位,0度在3點鐘方向,角度爲正表順時針方向

例如, 以下敘述將於一個左上角位於 (50, 50), 寬度爲 100, 高度爲 50 的矩形內, 繪出一起始角爲 0 度, 弧度角爲 90 度的扇形。

```
g      =      this.CreateGraphics();
Pen   drawPen   =   new   Pen(Color.Black,   1);
g.DrawPie(drawPen,   50,   50,   100,   50,   0,   90);
```

其結果如右圖, 爲了方便讀者觀察起始角與弧度角, 筆者亦將對應的矩形繪出。



DrawLines

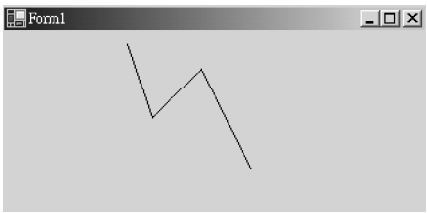
繪製連續線段, 其語法如下：

```
public    void    DrawLines(
     Pen  pen,
       Point[]  points
) ;
```

繪製連結 Point 結構陣列的連續線段。例如, 以下敘述可繪出一條起點爲 (100, 10), 終點爲 (200, 110), 並通過 (120, 70) 及 (160, 30) 兩點的連續線段。

```
g      =      this.CreateGraphics();
Pen   drawPen   =   new   Pen(Color.Black,   1);
Point   p1,   p2,   p3,   p4;
p1   =   new   Point(100,   10);
p2   =   new   Point(120,   70);
p3   =   new   Point(160,   30);
p4   =   new   Point(200,   110);
Point[]   points   =   {p1,   p2,   p3,   p4};
g.DrawLines(drawPen,          points);
```

其結果如右圖：



DrawPolygon

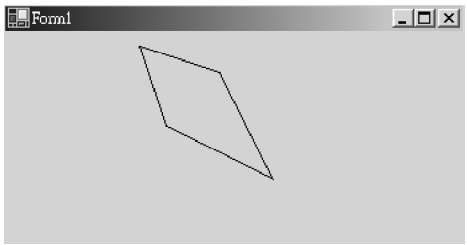
繪製封閉多邊形, 其語法如下：

```
public    void    DrawPolygon(
     Pen  pen,
       Point[]  points
) ;
```

繪製連結指定 Point 結構陣列的封閉多邊形。例如, 以下敘述可繪出一個封閉多邊形, 其起點爲 (100, 10), 終點爲 (160, 30), 並通過 (120, 70) 及 (200, 110) 兩點, 最後此方法會在起點與終點之間補上一條直線。

```
g = this.CreateGraphics();
Pen drawPen = new Pen(Color.Black, 1);
Point p1, p2, p3, p4;
p1 = new Point(100, 10);
p2 = new Point(120, 70);
p3 = new Point(200, 110);
p4 = new Point(160, 30);
Point[] points = {p1, p2, p3, p4};
g.DrawPolygon(drawPen, points);
```

其結果如右圖：



DrawString

繪製文字, 其語法如下：

```
public void DrawString(
    string s,
    Font font,
    Brush brush,
    float x,
    float y
) ;
```

使用指定的 Brush 和 Font 物件, 將文字字串繪製於指定位置 (x, y), 其中有關 Brush 和 Font 類別在 16-3 節有進一步說明, 在此可先略過。例如, 以下敘述可於座標 (100, 50) 的位置繪製文字。

```
g = this.CreateGraphics();
Font drawFont = new Font("Arial", 16);
SolidBrush drawBrush = new SolidBrush(Color.Black);
g.DrawString(" 歡迎光臨 ", drawFont, drawBrush, 100, 50);
```

其結果如右圖：

FillRectangle

繪製一個填滿顏色的矩形, 其語法如下：

```
public      void      FillRectangle(
    Brush     brush,
    int    x,
    int    y,
    int     width,
    int     height
) ;
```

其中 brush 爲筆刷物件, 有關 Brush 類別在下一節中有進一步的說明, (x, y) 爲矩形的左上角點, width 爲矩形的寬度, height 爲矩形的高度。例如, 以下敘述可於表單上繪出一個左上角位於 (50, 30), 寬度爲 200, 高度爲 100, 且以紅色填滿的矩形。

```
g      =      this.CreateGraphics();
SolidBrush    sb    =    new    SolidBrush(Color.Red);
g.FillRectangle(sb,    50,    30,    200,    100);
```

其結果如右圖。

FillEllipse

繪製一個填滿顏色的橢圓, 其語法如下：

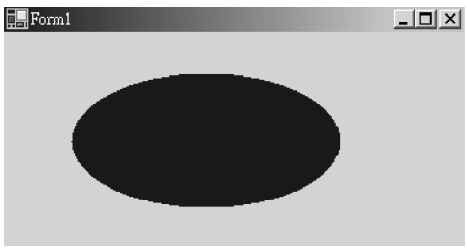
```
public      void      FillEllipse(
    Brush     brush,
```

```
    int    x,
    int    y,
    int     width,
    int     height
) ;
```

在指定左上角座標 (x, y), 寬度 (width) 及高度 (height) 的矩形內繪製一個填滿顏色的橢圓。例如, 以下敘述可繪出一個填滿藍色的橢圓。

```
g       =       this.CreateGraphics();
SolidBrush     sb     =     new     SolidBrush(Color.Blue);
g.FillEllipse(sb,     50,     30,     200,     100);
```

其結果如右圖。



FillPie

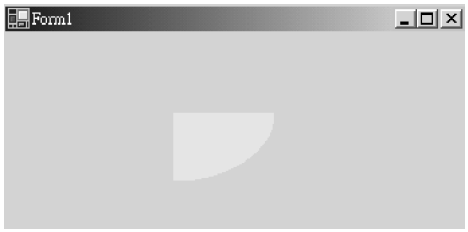
繪製一個填滿顏色的扇形, 其語法如下：

```
public     void     FillPie(
    Brush     brush,
    int    x,
    int    y,
    int     width,
    int      height,
    int       startAngle,
    int     sweepAngle
) ;
```

以上語法中各項參數的意義, 除 brush 物件外, 其它各項參數意義同 DrawPie 方法。例如, 以下敘述可繪出一個填滿黃色的扇形。

```
g       =       this.CreateGraphics();
SolidBrush     sb     =     new     SolidBrush(Color.Yellow);
g.FillPie(sb,    50,    10,    150,    100,    0,    90);
```

其結果如右圖。



FillPolygon

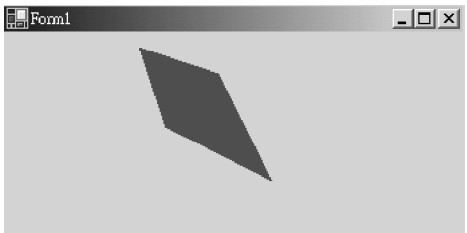
繪製一個填滿顏色的封閉多邊形, 其語法如下：

```
public      void      FillPolygon(
       Brush   brush,
        Point[]   points
) ;
```

將指定 Point 結構陣列所定義多邊形內部填滿顏色。例如, 以下敘述可繪出一個填滿綠色的多邊形。

```
g      =      this.CreateGraphics();
SolidBrush     sb     =     new     SolidBrush(Color.Green);
Point   p1,   p2,   p3,   p4;
p1   =   new   Point(100,   10);
p2   =   new   Point(120,   70);
p3   =   new   Point(200,   110);
p4   =   new   Point(160,   30);
Point[]   points   =   {p1,   p2,   p3,   p4};
g.FillPolygon(sb,            points);
```

其結果如右圖。



Clear

清除整個繪圖介面, 並使用指定的背景顏色填滿它。其語法如下：

```
public     void     Clear(
       Color   color
) ;
```

其中 color 爲繪圖介面的背景顏色, 有關 Color 結構在 16-3 節有進一步的說明。例如, 以下敘述可清除表單之繪圖介面, 並以表單的背景顏色將其填滿。

```
g.Clear(this.BackColor);
```

繪點

Graphics 介面目前並沒有繪點的方法, 那要如何繪製點呢？變通的辦法答案是繪製一條很短的直線, 如以下敘述：

```
private void button1_Click(object sender, EventArgs e)
{
        Graphics g = this.CreateGraphics();// 設定表單爲繪圖區域
      Pen p = new Pen(Color.Red, 2);
        g.DrawLine(p, 100, 100, 101, 100);
}
```

圓心繪圓

Graphics 介面並沒有提供以圓心與半徑繪圓的方法, 若要以圓心與半徑繪圓, 則必須自行撰寫方法如下：

```
namespace WindowsFormsApplication1
{
      public partial class Form1 : Form
  {
            private void button2_Click(object sender, EventArgs e)
    {
          // 設定表單爲繪圖區域
                Graphics g = this.CreateGraphics();
              Gra gra1 = new Gra();
                gra1.drawCircle(g, 100, 100, 40);
    }
  }
     class Gra
  {
              public void drawCircle(Graphics g, int x, int y, int r){
                int x1, y1, weight, height;
                Pen p=new Pen(Color.Red, 2);
              x1 = x - r;
              y1 = y - r;
```

```
                    weight = 2 * r;
                    height = 2 * r;
                          g.DrawEllipse(p, x1, y1, weight, height);
    }
          // 繪點的另一方法
                    public void_drawPoint(Graphics g, int x, int y)
    {
                      Pen p = new Pen(Color.Red, 2);
                        g.DrawEllipse(p, x, y, 1, 1);
    }
  }
}
```

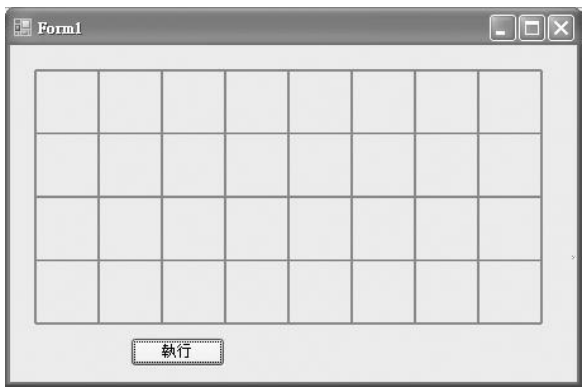
範例 16-2a

示範以上繪圖方法。(請自行開啓檔案)

範例 16-2b

請寫一程式, 完成以下棋盤。

執行結果



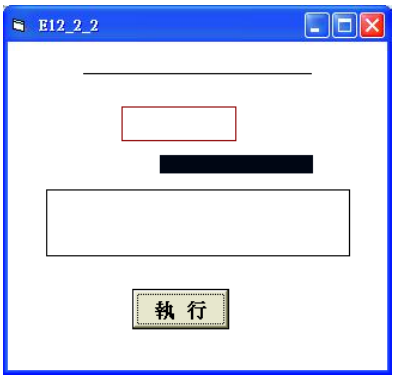
程式列印

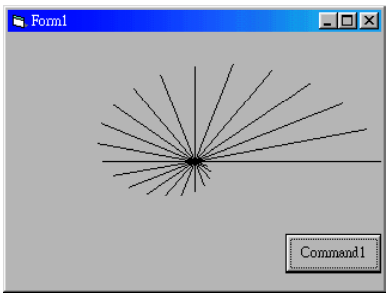
```
private      void      button1_Click(object      sender,      EventArgs      e)
{
          Graphics   g   =   this.CreateGraphics();// 設定表單爲繪圖區域
        Pen   p   =   new   Pen(Color.Red,   2);
    // 些許偏移量, 以免與表單邊框重疊
        int   x0=20;
        int   y0=20;
     // 繪製垂直線
```

```
            for  (int  i=0  ;i<=8;i++)
                g.DrawLine(p, 50 * i + x0, 0 + y0, 50 * i + x0, 200 + y0);
        // 繪製水平線
            for  (int  i  =  0;  i  <=  4;  i++)
                 g.DrawLine(p, x0, i * 50 + y0, 400 + x0, i * 50 + y0);
}
```

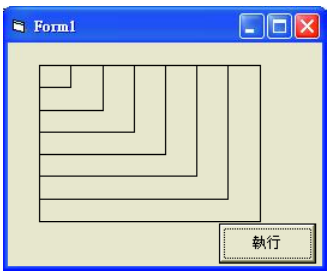
自我練習

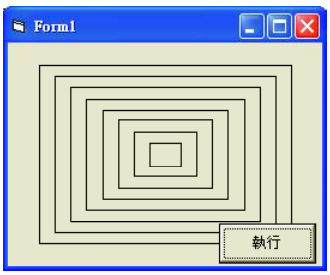
1. 請實際觀察羽球場、排球、棒球場或網球場, 並寫程式, 繪出其場地標線圖。
2. 請實際觀察棒球場, 並於表單繪出其場地標線圖、防守九人名單。
3. 請寫一程式, 完成如下圖左。
4. 請寫一程式, 完成如下圖右。



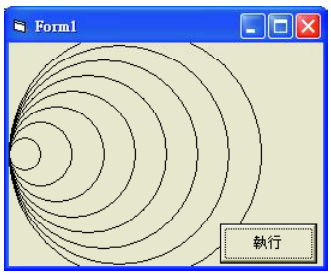


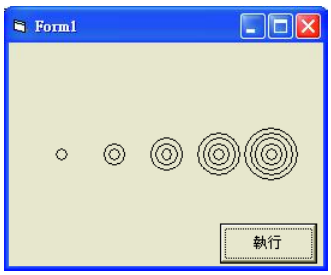
5. 請寫一程式, 繪出如下圖左：
6. 請寫一程式, 繪出如下圖右：





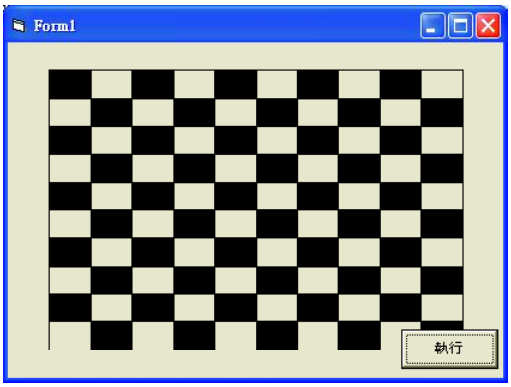
7. 請畫出下圖左的圖形。
8. 請畫出下圖右的圖形。

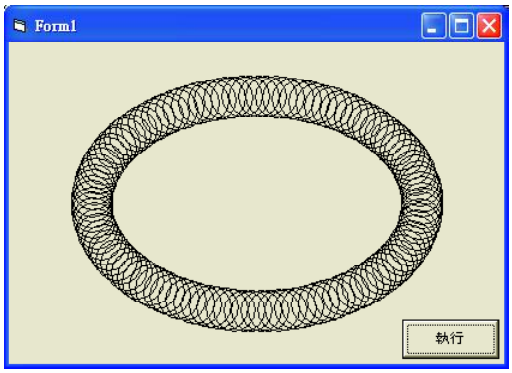




9. 請寫一程式, 繪出如下圖左的結果。

10. 請寫一程式, 繪出如下圖右的結果。





16-3 繪圖相關類別

本單元將介紹的是一些與繪圖相關的類別或結構, 它們分別是 Pen 、Brush 、 Color 及 Font 類別, 雖然這些類別並不具有直接繪圖的能力, 但卻是繪圖方法必備的參數。以下將針對這四個繪圖相關類別作進一步的介紹。

Pen 類別

Pen 類別可協助設定畫筆的寬度、顏色、填滿的樣式、線條樣式以及線條端點樣式, 請看以下說明。

建構子

Pen 的建構子如下圖所示：

	名稱	說明
	Pen(Brush)	使用指定的 Brush，初始化 Pen 類別的新執行個體。
	Pen(Color)	使用指定的色彩，初始化 Pen 類別的新執行個體。
	Pen(Brush, Single)	使用指定的 Brush 和 Width，初始化 Pen 類別的新執行個體。
	Pen(Color, Single)	使用指定的 Color 和 Width 屬性，初始化 Pen 類別的新執行個體。

以下僅介紹第四種建構子, 此建構子語法如下：

```
Pen  (Color  color,  float  Width)
```

以指定畫筆顏色與寬度, 初始化 Pen 類別的執行個體。例如, 以下敘述可建立 Pen 類別的執行個體, 其畫筆顏色爲紅色且筆寬爲 3 。

```
Pen  drawPen  =  new  Pen(Color.Red,  3);
```

實例屬性

以下將介紹 Pen 類別的常用實例屬性。

- Width

  取得或設定畫筆寬度。例如, 以下敘述可將 drawPen 畫筆物件的筆寬設定爲 5 。

```
    drawPen.Width  =  5;
```

- Color

  取得或設定畫筆顏色。例如, 以下敘述可將 drawPen 畫筆物件的畫筆顏色設定藍色。

```
      drawPen.Color  =  Color.Blue;
```

- DashStyle

  取得或設定線條的樣式, 此樣式必須是 DashStyle 列舉型別的成員, 如下表所示。

成員名稱	說明
Custom	使用者定義的自訂破折號樣式
Dash	破折線
DashDot	破折線-虛線的重複線條
DashDotDot	破折線-虛線-虛線的重複線條
Dot	虛線
Solid	實線

例如, 以下敘述可將 drawPen 畫筆物件的線條樣式設定爲虛線。

```
drawPen.DashStyle = System.Drawing.Drawing2D.DashStyle.Dot;
```

此外, DashStyle 列舉型別屬於 System 、 Drawing 及 Drawing2D 命名空間, 讀者可於程式開頭使用先行宣告, 以避免鍵入全名。

```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
```

● StartCap 與 EndCap

取得或設定線條的端點樣式, 此樣式必須是 LineCap 列舉型別的成員, 如下表所示。

成員名稱	說明
ArrowAnchor	箭頭形狀的端點
Custom	使用者自訂的端點
DiamondAnchor	菱形端點
Round	測試後無作用
RoundAnchor	圓形端點
Square	測試後無作用
SquareAnchor	四方形端點
Triangle	測試後無作用

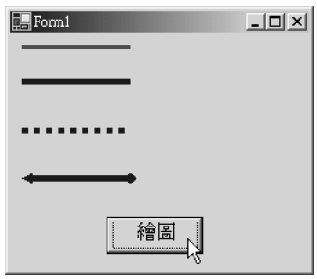
例如, 以下敘述可將 drawPen 畫筆物件的線條端點設爲箭頭及菱形。

```
drawPen.StartCap =
        System.Drawing.Drawing2D.LineCap.ArrowAnchor;
drawPen.EndCap =
        System.Drawing.Drawing2D.LineCap.DiamondAnchor;
```

範例 16-3a

示範以上 Pen 類別的用法。

執行結果



程式列印

```
private void button1_Click(object sender, System.EventArgs e)
{
    Graphics g = this.CreateGraphics();
    Pen drawPen = new Pen(Color.Red, 3);
    g.DrawLine(drawPen, 10, 10, 100, 10);

    drawPen.Width = 5;
    drawPen.Color = Color.Blue;
    g.DrawLine(drawPen, 10, 40, 100, 40);

    drawPen.DashStyle =
        System.Drawing.Drawing2D.DashStyle.Dot;
    g.DrawLine(drawPen, 10, 80, 100, 80);

    drawPen.DashStyle =
        System.Drawing.Drawing2D.DashStyle.Solid;
    drawPen.StartCap =
        System.Drawing.Drawing2D.LineCap.ArrowAnchor;
    drawPen.EndCap =
        System.Drawing.Drawing2D.LineCap.DiamondAnchor;
    g.DrawLine(drawPen, 10, 120, 100, 120);
}
```

Brush 類別

我們曾經提過, Graphics 類別像是一塊畫布, Pen 類別像是一支畫筆, 但是這支畫筆只具有畫直線及外框(例如, 橢圓形及扇形)的能力, 若要對某一塊區域進行填色的動作, Pen 類別就沒有辦法做到了, 而 Brush 類別就是用來對各種封閉圖形填色的工具。其次, Brush 是抽象基礎類別, 我們無法直接樣例其執行個體。針對各種需要, GDI+ 提供了五種 Brush 的

衍生類別,分別是 SolidBrush(單色)、 TextureBrush(材質)、 HatchBrush 、athGradientBrush 以及 LinearGradientBrush(漸層)等, 以下僅針對 SolidBrush(單色)與 TextureBrush(材質)類別作進一步的介紹。

SolidBrush 類別

此類別是 Brush 類別中最基本的一種, 用來將某一區域填入單一顏色。其建構子如下：

```
SolidBrush      (Color      color)
```

以指定色彩初始化 SolidBrush 的執行個體。例如, 以下敘述可在一個左上角位於 (50, 30), 寬度 100, 高度 50 的矩形內填入紅色。

```
Graphics      g      =      this.CreateGraphics();
SolidBrush     sb     =     new      SolidBrush(Color.Red);
g.FillRectangle(sb,      50,      30,      100,      50);
```

TextureBrush 類別

TextureBrush 類別是將圖檔填入某一區域, 圖檔必須藉由 Image 類別來載入。 TextureBrush 類別的建構子如下：

```
TextureBrush      (Image      bitmap)
```

以指定圖檔來初始化 TextureBrush 的執行個體。例如, 以下敘述可在一個左上角位於 (40, 0), 寬度爲 300, 高度爲 200 的矩形內填入檔名爲 "c:\CsBook\a01.bmp" 的圖檔。

```
Graphics      g      =      this.CreateGraphics();
TextureBrush     tb     =      new      TextureBrush(Image.FromFile(
                                        "c:\\CsBook\\a01.bmp"));
g.FillRectangle(tb,      40,      0,      300,      200);
```

範例 16-3b

示範以上 Brush 類別的用法。

補充說明

請讀者自行開啓檔案 e16-3b, 執行程式並瀏覽其結果。

程式列印

```
using      System;
using        System.Drawing;
using           System.Collections;
using         System.ComponentModel;
using         System.Windows.Forms;
using        System.Data;

namespace     e16_3b
{
          public    class    Form1    :    System.Windows.Forms.Form
  {
          private     Graphics     g;

          private       void       btnSolidBrush_Click(object       sender,
                                                  System.EventArgs  e)
          {
               SolidBrush     sb     =     new     SolidBrush(Color.Red);
               g.FillRectangle(sb,     50,     30,     100,     50);
          }

          private       void       btnTextureBrush_Click(object       sender,
                                                    System.EventArgs  e)
          {
               TextureBrush    tb    =    new    TextureBrush(
                                 Image.FromFile("c:\\CsBook\\a01.bmp."));
               g.FillRectangle(tb,     40,     0,     300,     200);
          }

          private     void      Form1_Load(object      sender,
                                           System.EventArgs  e)
          {
               g      =      this.CreateGraphics();
          }

          private       void       btnClear_Click(object       sender,
                                                  System.EventArgs  e)
          {
               g.Clear(this.BackColor);
          }
  }
}
```

Color 結構

靜態屬性

Color 結構定義了許多常用的色彩屬性, 例如, Black 、Blue 、 Red 及 Yellow 等, 由於數目相當多, 在此並不一一介紹, 讀者可自行線上查詢。右圖是筆者線上查詢的結果。

Beige	取得系統定義的色彩。
Bisque	取得系統定義的色彩。
Black	取得系統定義的色彩。
BlanchedAlmond	取得系統定義的色彩。
Blue	取得系統定義的色彩。
BlueViolet	取得系統定義的色彩。
Brown	取得系統定義的色彩。
BurlyWood	取得系統定義的色彩。
CadetBlue	取得系統定義的色彩。
Chartreuse	取得系統定義的色彩。

此外, 由於這些屬性都是靜態屬性, 所以可用 " 結構名稱.屬性 " 的方式來取得屬性值。例如, 以下敘述可將畫筆物件設爲紅色, 並繪出一條起點爲 (10, 30), 終點爲 (200, 30), 寬度爲 20 的直線。

```
Graphics    g   =   e.Graphics;
Pen   drawPen   =   new   Pen(Color.Red,   20);
g.DrawLine(drawPen,   10,   30,   200,   30);
```

靜態方法

- FromArgb

  由四個 8 位元 ARGB 元件 (Alpha (色彩的透明度) 、紅 (R)、綠 (G)和藍 (B)) 值建立 Color 結構。共有 4 種多載, 以下僅介紹常用的兩種, 分別說明如下：

- public static Color FromArgb(int red, int green, int blue);

  從指定的 8 位元色彩值 (紅、綠和藍) 建立 Color 結構, 有效值爲 0-255 之間, 在此多載中 Alpha (色彩明度) 值預設爲 255 (完全不透明)。例如, 以下敘述可將 drawPen 畫筆物件設定爲紅色, 並繪出一條起點爲 (10, 60), 終點爲 (200, 60) 的直線。

```
drawPen.Color = Color.FromArgb(255, 0, 0);
g.DrawLine(drawPen, 10, 60, 200, 60);
```

  又例如, 以下敘述可將 drawPen 畫筆物件設定爲綠色, 並繪出一條起點爲 (10, 90), 終點爲 (200, 90) 的直線。

```
drawPen.Color = Color.FromArgb(0, 255, 0);
g.DrawLine(drawPen, 10, 90, 200, 90);
```

- public static Color FromArgb(int alpha, int red, int green, int blue);

  從四個 ARGB 元件 (Alpha 、紅、綠和藍) 值建立 Color 結構, 有效值在 0-255 之間。其中 Alpha 值表示色彩透明度, 即色彩與背景色彩混合的程度, 0 表示完全透明的色彩, 255 表示完全不透明的色彩。例如, 以下敘述可將 drawPen 畫筆物件設定爲透明度爲 120 的紅色, 並繪出一起點爲 (10, 120), 終點爲 (200, 120) 的直線 (請與前一多載所繪出之紅色直線作顏色上比較)。

```
drawPen.Color = Color.FromArgb(120, 255, 0, 0);
g.DrawLine(drawPen, 10, 120, 200, 120);
```

  又例如, 以下敘述可將 drawPen 畫筆物件設定爲透明度爲 120 的綠色, 並繪出一起點爲 (10, 150), 終點爲 (200, 150) 寬度爲 20 的直線 (請與前一多載所繪出之綠色直線作顏色比較)。

```
drawPen.Color = Color.FromArgb(120, 0, 255, 0);
g.DrawLine(drawPen, 10, 150, 200, 150);
```

範例 16-3c

示範以上 Color 結構的用法。

補充說明

請讀者自行開啓檔案 e16-3c, 執行程式並瀏覽其結果。

程式列印

```
private void Form1_Paint(object sender,
    System.Windows.Forms.PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Pen drawPen = new Pen(Color.Red, 20);
    g.DrawLine(drawPen, 10, 30, 200, 30);

    drawPen.Color = Color.FromArgb(255, 0, 0);
    g.DrawLine(drawPen, 10, 60, 200, 60);

    drawPen.Color = Color.FromArgb(0, 255, 0);
    g.DrawLine(drawPen, 10, 90, 200, 90);

    drawPen.Color = Color.FromArgb(120, 255, 0, 0);
    g.DrawLine(drawPen, 10, 120, 200, 120);

    drawPen.Color = Color.FromArgb(120, 0, 255, 0);
    g.DrawLine(drawPen, 10, 150, 200, 150);
}
```

Font 類別

建構子

Font 類別常用的建構子如下：(線上查詢時, 請點選 System.Drawing 命名空間的 Font 類別)

```
Font (String familyName, float emSize, FontStyle Style)
```

familyName 是字型名稱, emSize 是字型大小, Style 是字型樣式。其中字型樣式必須是 FontStyle 列舉型別的成員, 如下表所示。

成員名稱	說明
Bold	粗體文字
Italic	斜體文字
Regular	一般文字
Strikeout	中間有線條經過的文字
Underline	加上底線的文字

例如, 以下敘述可建立一個 f 物件, 其字型是名稱是 " 新細明體 ", 字型大小是 14, 字型樣式是斜體。

```
Font f = new Font(" 新細明體 ", 14, FontStyle.Italic);
```

其次, 字型樣式可與 or (|) 運算子相結合, 同時設定多個字型樣式。例如, 以下敘述可建立一個 f1 物件, 其字型名稱是 " 新細明體 ", 字型大小是 14, 字型樣式是粗斜體。

```
Font f1 = new Font(" 新細明體 ", 14,FontStyle.Italic
                                | FontStyle.Bold);
```

範例 16-3d

示範 Font 類別。

執行結果

程式列印

```
private void Form1_Paint(object sender,
    System.Windows.Forms.PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Font f = new Font(" 新細明體 ", 14, FontStyle.Italic);
```

```
        SolidBrush sb = new SolidBrush(Color.Red);
        g.DrawString("歡迎光臨", f, sb, 10, 10);

        Font f1 = new Font("新細明體", 14, FontStyle.Italic
                                | FontStyle.Bold);
        g.DrawString("歡迎光臨", f1, sb, 10, 50);
}
```

16-4 實例探討(一)

範例 16-4a

小畫家之研究。

前面已經研究各種繪圖方法, 筆者觀察小畫家的畫直線方式如下：

1. 點選直線圖項。
2. 移至直線起始點。
3. 拖曳直線 (拖曳的過程均不斷呈現繪圖結果)
4. 直到使用者放開滑鼠, 結束直線的繪製, 此時電腦呈現唯一一條直線。

經由以上的觀察, 乃激起筆者實作以上程式的動機, 以下則是筆者的解題過程。

解題步驟

1. 當使用者按下滑鼠左鍵時, 建立一個繪圖物件, 記錄直線的繪圖起點, 並設定 md(mousedown) 旗號爲真。

```
private void pic_MouseDown(object sender, MouseEventArgs e)
{
          g = pic.CreateGraphics();
        point1 = new Point(e.X, e.Y);
      md = true;
}
```

2. 當使用者移動滑鼠時, 此時有三種情況, 第一是還沒有按下滑鼠左鍵；第二是已經按下滑鼠左鍵, 且是第一次移動滑鼠；第三種是已經按下滑鼠左鍵, 且是已經移動過滑鼠。首先, 若是僅移動滑鼠, 但還沒按下滑鼠, 當然是不予理會。可能是使用者還在尋找繪圖起點。其次, 若是已經按下滑鼠左鍵, 且是第一次移動滑鼠, 則應畫出此直線。此直線的起點是剛剛按下滑鼠左鍵時所記錄的 point1, 終點則是目前滑鼠的所在位置, 且設定滑鼠已經移動的旗號 mm (mousemove) 爲真。以上說明的程式如下：

```
private void pic_MouseMove(object sender, MouseEventArgs e)
{
    Pen penb=new Pen(Color.Black ,1);
    Pen penw=new Pen(Color.White ,1);
    Point point2=new Point(e.X,e.Y);
    if ((md) & (!mm))
    {
            g.DrawLine(penb, point1, point2);
          pointold=point2;
    }
    mm=true;
```

第三種是已經按下滑鼠左鍵, 且是已經移動過滑鼠。這一種情況, 表示使用者還在尋找適當的結束點, 此時要不斷的將剛剛的直線以白色的筆擦掉, 並畫一條新的直線, 且將此次畫線的結束點記錄下來, 以便將來可以擦掉。以上說明所需程式如下：

```
  if ((md) & (mm))
  {
          g.DrawLine(penw, point1, pointold);
          g.DrawLine(penb, point1, point2);
        pointold = point2;
  }
```

3. 當使用者放開滑鼠左鍵時, 應將 md(mousedown) 旗號設爲 false 。以上說明程式如下：

```
private void pic_MouseUp(object sender, MouseEventArgs e)
{
      md = false;
}
```

4．爲使一些變數流通各事件, 設定全域變數如下：

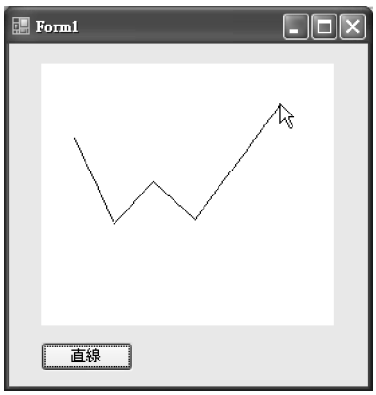
```
private     Graphics     g;
private      Point      point1;
private      Point      pointold;
Boolean     md;
Boolean     mm;
```

5．整體程式列印如下：

```
public   partial   class   Form1   :   Form
{
         private   Graphics   g;
          private   Point   point1;
           private   Point   pointold;
       Boolean   md;
      Boolean   mm;
    private     void     pic_MouseDown(object     sender,     MouseEventArgs     e)
    {
        g      =      pic.CreateGraphics();
        point1    =    new    Point(e.X,    e.Y);
        md   =   true;
    }

    private     void     pic_MouseMove(object     sender,     MouseEventArgs     e)
    {
        Pen     penb=new     Pen(Color.Black     ,1);
        Pen     penw=new     Pen(Color.White     ,1);
        Point       point2=new       Point(e.X,e.Y);
        if   ((md)   &   (!mm))
   {
            g.DrawLine(penb,       point1,       point2);
            pointold=point2;
        }
        mm=true;
        if   ((md)   &   (mm))
   {
            g.DrawLine(penw,       point1,       pointold);
            g.DrawLine(penb,       point1,       point2);
            pointold     =     point2;
        }
    }
    private     void     pic_MouseUp(object     sender,     MouseEventArgs     e)
    {
            md   =   false;
    }
}
```

執行結果

右圖是筆者所繪製的連續直線, 操作方式同小畫家, 請讀者自行開啓檔案練習。



補充說明

1. 本例並未於直線按鈕撰寫任何程式, 那是因爲本例僅處理直線, 如果像小畫家那樣, 必須同時處理矩形、圓形、任意多邊行等圖形, 那就必須在此設定旗號, 以作爲處理各項方法的依據。

```
private void btnLine_Click(object sender, EventArgs e)
{
}
```

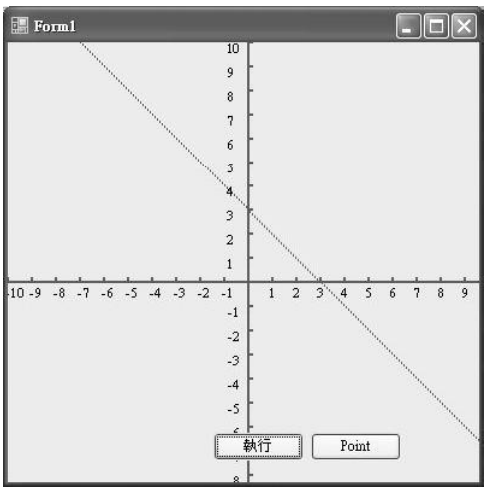
自我練習

1. 請參考小畫家的畫矩形方式, 並寫程式完成。
2. 請參考小畫家的畫圓形方式, 並寫程式完成。
3. 請參考小畫家的繪圖方式, 安排三個按鈕, 分別是直線、矩形與圓等, 並寫程式完成其功能。

範例 16-4b

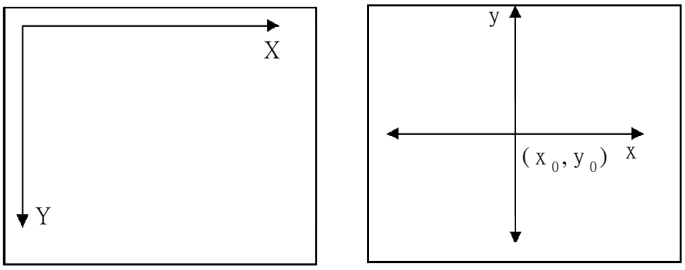
幾何圖形之研究。請寫一程式, 繪出 x+y=3 的圖形。

執行結果



解題步驟

1. C# 預設座標原點於繪圖物件的左上角, x 軸向右爲正, y 軸向下爲正, 如下圖左。此與數學的座標習慣不同, 因爲數學的座標系統, 預設原點於中心點, x 軸向右爲正, y 軸向上爲正, 如下圖右, 所以必須調整原點位置。本例設定畫布長與寬都分別爲 400,並設原點於中心點, 以上設定的程式如下, TranslateTransform(x0, y0) 即是將座標原點移至繪圖物件表單的中心。




```
Graphics g = this.CreateGraphics();
Pen myPen = new Pen(Color.Red, 1);
this.Width = 400;
this.Height = 400;
int x0 = this.Width / 2;
int y0 = this.Height / 2;
g.TranslateTransform(x0, y0);
```

2. 數學習慣向上爲正, 但是 C# 卻是向下爲正, 所以 y 座標必須進行方向調整, 也就是乘以 -1 。所以本例設定如下：

```
const int dy = -1;
```

3. 設定顯示比例。本例因爲考慮數學繪圖習慣, x 軸僅考慮 -10 到 10, 但是我們的畫布卻是 -200 到 200, 所以必須設定顯示比率爲 20 倍, 才能將結果填滿整個畫布, 本例設定如下：

```
const int sx = 20;
const int sy = 20;
```

4. 繪製座標軸。本例程式如下：

```
g.DrawLine(myPen, -200, 0, 200,0);
g.DrawLine(myPen, 0, 200, 0, -200);
```

6. 以上全部程式列印如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics g = this.CreateGraphics();
    Pen p1 = new Pen(Color.Red, 2);
    this.Width = 400;
    this.Height = 400;
    int x0 = this.Width / 2;
    int y0 = this.Height / 2;
    g.TranslateTransform(x0, y0);
    const short sx = 20;// 水平放大倍數
    const short sy = 20;
    const short dy = -1;
    // 繪出座標軸
    g.DrawLine(p1, -10*sx, 0, 10*sx, 0);
    g.DrawLine(p1, 0, 10*sy*dy, 0, -10*sy*dy);
    // 繪出函數圖
    int x1, y1;
    for (double i=-10;i<=10;i=i+0.1){
        x1=(int)(i*sx);
        y1=(int)((3-i)*sy*dy);
        g.DrawLine(p1, x1, y1, x1+1, y1);
    }
    Font f1 = new Font("細明體", 8, FontStyle.Regular);
```

```
    // 繪出水平刻度與刻度值
    for (int i = -10; i <= 10; i++)
    {
        if (i == 0)
            continue;
        x1 = (i * sx);
        y1 = (int)(0.2*sy*dy);
        g.DrawLine(p1, x1, 0, x1 , y1);
        g.DrawString(i.ToString(), f1, Brushes.Black, x1-5, -y1);
    }
    // 繪出垂直刻度與刻度值
    for (int i = -10; i <= 10; i++)
    {
        if (i==0)
            continue;
        x1 = (int)(0.2*sx);
        int x2 = (int)(-1 * sx);
        y1 = i * sy * dy;
        g.DrawLine(p1, 0, y1, x1, y1);
        g.DrawString(i.ToString(), f1, Brushes.Black, x2, y1);
        }
}
```

7. 以下程式是筆者以 Point 結構重做以上程式的思考方式。首先宣告一個 points 結構陣列, 敘述如下：(結構陣列與類別陣列相同, 請看 9-6 節的物件陣列)

```
Point[] points = new Point[21];
```

其次, 新增陣列元素, 其敘述如下：

```
for(int i=-10;i<=10;i++)
    points[i+10]=new Point(i*s,-1*(3-i)*s);
```

最後以 DrawLines 方法繪圖, 其敘述如下：

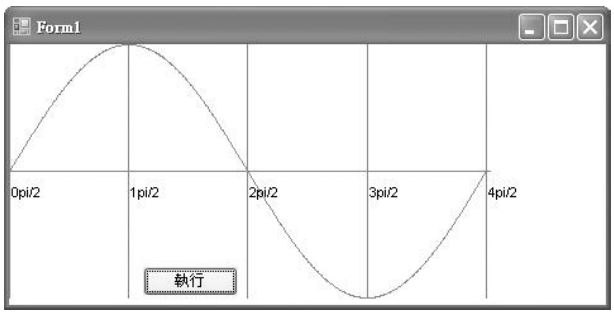
```
g.DrawLines(p1,points);
```

以上說明的全部程式如下：

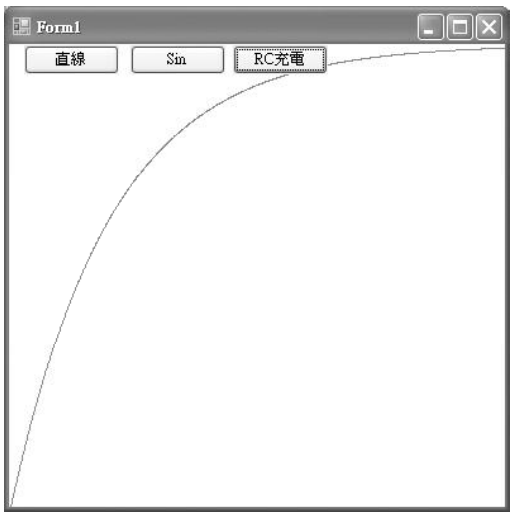
```
private void button2_Click(object sender, EventArgs e)
{
    Graphics g = this.CreateGraphics();
    Pen p1 = new Pen(Color.Red, 2);
    this.Width = 400;
    this.Height = 400;
    int x0 = this.Width / 2;
    int y0 = this.Height / 2;
    g.TranslateTransform(x0, y0);
    const short sx = 20;// 水平放大倍數
    const short sy = 20;
    const short dy = -1;
    // 繪出座標軸
    g.DrawLine(p1, -10 * sx, 0, 10 * sx, 0);
    g.DrawLine(p1, 0, 10 * sy * dy, 0, -10 * sy * dy);

    Point[] points = new Point[21];
    for(int i=-10;i<=10;i++)
        points[i+10]=new Point(i*sx, (3-i)*sy*dy);
    g.DrawLines(p1, points);
}
```

自我練習

1. 請寫一程式, 繪出 $y=x^2$
2. 請寫一程式, 繪出 $x^2 + y^2 = 9$ 。
3. 請寫一程式, 繪出 $y=x^3 + 3x^2 -1$ 之圖形。
4. 請寫一程式, 繪出 $x^2 / 25 + y^2 /16=1$ 。
5. 繪出 sin θ 的圖形, 執行結果如下：



6. 於 RC 充電電路, E=10V, R=100k , C=100uF , $V_c(t)=E(1-e^{-(t/RC)})$, 請繪出其波形。執行結果如右圖：



演算說明：RC 充電的時間爲 5RC, 也就是 5 倍 RC 就已經充滿電, 所以應該繪出時間 0 到 5*RC, 且最高充電至外加電壓 E 。

7. 請寫一程式, 可以輸入兩點座標, 並且標示座標軸、此兩點座標、與經過此兩點座標的直線。

範例 16-4c

暴力法繪圖之研究。

您我從國中起, 數學老師就說, 一切 x, y 屬於實數, 滿足 x+y=3 所成的集合, 將會是一條直線, 學習電腦至今, 您是否可用電腦證明。

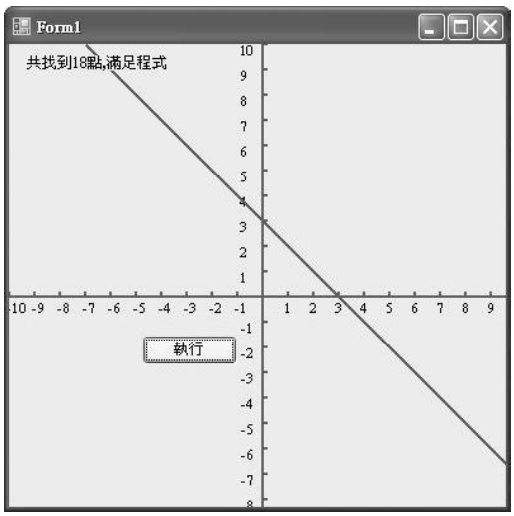
解題步驟

所謂暴力法, 就是將滿足條件的點通通帶入檢驗, 所以我們使用雙迴圈, x 從 -10 到 10, y 亦從 -10 到 10,一一代入, 所以程式如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics g = this.CreateGraphics();
    Pen p1 = new Pen(Color.Red, 2);
    this.Width = 400;
    this.Height = 400;
    int x0 = this.Width / 2;
    int y0 = this.Height / 2;
    g.TranslateTransform(x0, y0);
```

```
    const short sx = 20;// 水平放大倍數
    const short sy = 20;// 垂直放大倍數
    const short dy = -1;
    int x1, y1;
    // 繪出座標軸
    g.DrawLine(p1, -10 * sx, 0, 10 * sx, 0);
    g.DrawLine(p1, 0, 10 * sy * dy, 0, -10 * sy * dy);
    Font f1 = new Font(" 細明體 ", 8, FontStyle.Regular);
    // 繪出水平刻度與刻度值
    for (int i = -10; i <= 10; i++)
    {
        if (i == 0)
            continue;
        x1 = (i * sx);
        y1 = (int)(0.2 * sy * dy);
        g.DrawLine(p1, x1, 0, x1, y1);
        g.DrawString(i.ToString(), f1, Brushes.Black, x1 - 5, -y1);
    }
    // 繪出垂直刻度與刻度值
    for (int i = -10; i <= 10; i++)
    {
        if (i == 0)
            continue;
        x1 = (int)(0.2 * sx);
        int x2 = (int)(-1 * sx);
        y1 = i * sy * dy;
        g.DrawLine(p1, 0, y1, x1, y1);
        g.DrawString(i.ToString(), f1, Brushes.Black, x2, y1);
    }

    Point[] points = new Point[18];
    int c=0;
    for (int x = -10; x <= 10; x++)
        for(int y =-10; y<= 10 ;y++)
            if ((x + y) == 3)
            {
                points[c] = new Point(x * sx, (3 - x) * sy * dy);
                c++;
            }


    g.DrawLines(p1, points);
    label1.Text = " 共找到 "+c.ToString() +" 點, 滿足程式 ";

}
```

執行結果



補充說明

1. 筆者的電腦通常要按兩次 " 執行 " 按鈕才能得到全圖。

2. 以上是以整數考慮, 若以實數考慮, 也就是說 x, y 每次遞增 0.1f, 那滿足條件 (Math.abs(x+y-3) <=0.1f) 的點將會更多。

自我練習

1. 請以暴力法繪出 $x^2+y^2=36$ 之圖形。

2. 請以暴力法繪出 $\sqrt{(x-3)^2+y^2}+\sqrt{(x+3)^2+y^2}=10$ 之圖形。

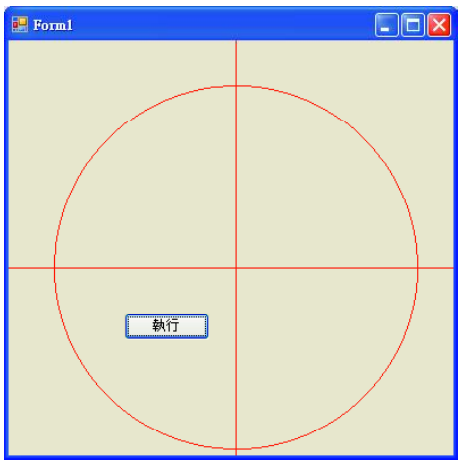
3. 請繪出以下兩函式的圖形、並標示交點座標。 y=3-x, $y=x^2$ 。

範例 16-4d

極座標繪圖。

請寫一程式, 以極座標的方式繪一圓。本例假設繪出 $x^2+y^2=64$ 。

執行結果



解題步驟

1. 將圓化成參數式。本例參數式如下：

$$x = r\cos\theta = 8\cos\theta$$
$$y = r\sin\theta = 8\sin\theta$$

2. 全部程式如下：

```
private void button1_Click(object sender, EventArgs e)
{
        Graphics g = this.CreateGraphics();
      Pen p1 = new Pen(Color.Red, 2);
      this.Width = 400;
       this.Height = 400;
      int x0 = this.Width / 2;
      int y0 = this.Height / 2;
          g.TranslateTransform(x0, y0);
     const short sx = 20;// 水平放大倍數
     const short sy = 20;
     const short dy = -1;
   // 繪出座標軸
      g.DrawLine(p1, -10 * sx, 0, 10 * sx, 0);
     g.DrawLine(p1, 0, 10 * sy * dy, 0, -10 * sy * dy);
   // 繪出函數圖
     double pi = Math.PI;
     for (float i =0; i <= 2*pi; i = i + 0.01F){
              float x1 = (float)(8 * Math.Cos(i));
```

```
                float y1 = (float)(8 * Math.Sin(i));
            float x2 = x1 * sx;
            float y2 = y1 * sy * dy;
              g.DrawLine(p1, x2, y2, x2 + 1, y2);
    }

}
```

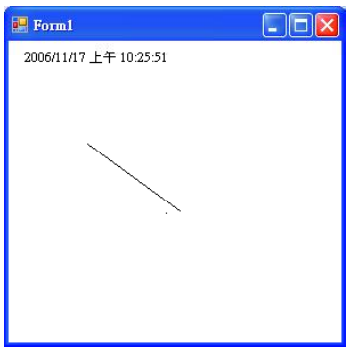
自我練習

1. 請繪出 r=8*cos(5 θ) 的圖形。

範例 16-4e

請寫一程式, 可以繪出動態移動的秒針。

執行結果



解題步驟

1. 座標系統的設定：常用的座標系統是原點在中間, X 軸向右爲正, Y 軸向上爲正, 但是 C# 繪圖系統原點是預設在表單 (Form1) 左上角, X軸向右爲正, Y 軸向下爲正。本專題爲了將座標系統原點設於表單的中心點, 所以使用 TranslateTransform 方法如下：

```
g       =       this.CreateGraphics();
this.Width     =     300;
this.Height     =     300;
int   x0   =   this.Width   /   2;
int   y0   =   this.Height   /   2;
g.TranslateTransform(x0,            y0);
```

則不管使用者如何改變表單的大小, 其原點均在表單的中心點。

2. 極座標與直角座標的轉換：時針的數字是由 0 到 11 表示, 分針與秒針則是由 0 到 59 的數字表示, 此方式與極座標 (r, θ) 相似。因爲極座標是由 (r, θ) 所組成, r 表示長度, θ 表示角度。但是, C ＃的繪圖系統是直角座標系 (x, y) , 其轉換關係如下：

```
x = rcos θ
y = rsin θ
```

3. 分針、秒針位置與極座標的轉換：分針與秒針的數字是由 0 到 59 所組成, 極座標的角度是 0 到 359 所組成, 所以時間 t 與角度 θ 的關係是：

```
θ = 2 π ／ 60 × t
```

其次, 極座標 θ 定義逆時針爲正, 但是時鐘是定義順時針爲正, 相差一個負號, 所以上式應調整爲：

```
θ = －2 π ／ 60 × t
```

最後, 極座標的 0° 是平貼在向右的X軸上, 但是秒針的 0° 是平貼在向上的Y軸上, 兩者相差 π ／ 2 。所以上式應調整如下：

```
θ = (-2 * π * t / 60) + π / 2
```

4. 秒針的繪製。秒針的繪製採用極座標繪製如下：

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g;
    const int r = 5;        // 秒針半徑
    const int sx = 15;      // 水平放大倍數
    const int sy = 15;
    const int dy = -1;
    Pen p1 = new Pen(Color.Black, 1);
    g = this.CreateGraphics();
    this.Width = 300;
    this.Height = 300;
    int x0 = this.Width / 2;
```

```
    int y0 = this.Height / 2;
    g.TranslateTransform(x0, y0);
    int ds = DateTime.Now .Second;
    label1.Text = DateTime.Now.ToString();
    float dsf = (float)((-2 * Math.PI * ds / 60) + Math.PI / 2);
    float x2 = (float)(r * Math.Cos(dsf));
    float y2 = (float)(r * Math.Sin(dsf));
    g.DrawLine(p1, 0, 0, x2 * sx, y2 * sy*dy);

}
```

5. 動畫的製作。秒針每秒移動一次, 所以可以每秒執行一次 Refresh()方法, 清除畫布內容, 並執行 Paint 事件。

```
private void timer1_Tick(object sender, EventArgs e)
{
        this.Refresh();// 畫布內容清除, 並執行 Paint 事件
}
```

自我練習

1. 同上範例, 但加上分針、時針、刻度及時鐘的邊框。

   提示：

   時針位置與極座標的轉換：前面的分針與秒針都是 0 到 59, 但是時針的數字是 0 到 12, 且時針是每 12 分鐘走分針的一格。所以時針 h 與角度 θ 的關係是：

```
θ = (-pi * (h * 30 + m * 6 / 12) / 180) + pi / 2
```

2. 請設計出軌跡爲 r=8*cos(5 θ) 的動畫。

3. 請設計一動畫, 其軌跡爲圓。

16-5 Image 與 Bitmap 類別

前面的 Graphics 介面提供一些繪圖方法, 讓我們使用這些方法完成一張圖畫。現在影像已經完成, 若要進行影像處理, 例如存檔、取檔、取得某一像素的顏色值, 或直接改變某一像素的顏色, 則必須將此影像交由 Image 或 Bitmap 類別處理。

Image 類別

Image 類別是提供功能給 Bitmap 和 Metafile 子代類別的抽象基底類別。本章先說明 Image 成員如下：

建構子

Image 是一個抽象基底類別, 所以沒有任何建構函式。

方法

此類別的常用方法如下：

- FromFile

FromFile 是一個靜態方法, 其功能是從指定的檔案載入 Image 物件。其使用語法如下：

```
public   static   Image   FromFile   (
         string   filename
)
```

例如, 以下敘述可將檔案 gwosheng.bmp 載入 image1 物件。

```
String    filename1    =@"c:\Csbook\gwosheng.bmp";    // 同    c:
\\Csbook\\gwosheng.bmp;
Image   image1   ;
image1      =      Image.FromFile(filename1);
```

上式的檔案名稱使用小老鼠 @ 當前導字元的優點是可排除一些逸出字元, 以上敘述同：

```
string      filename1      =      "c:\\Csbook\\gwosheng.bmp";
```

● Save

儲存 Image 至指定的檔案或資料流, 其使用語法如下：

```
public    void    Save(
          string  filename
)
```

例如, 以下敘述可將 Image 物件存檔。

```
String        filename2       =@"c:\Csbook\gwosheng2.bmp";
image1.Save(filename2)          ;
```

Image vs Graphics

Image 物件與 Graphics 物件的溝通, 通常透過 Graphics 物件的 DrawImage 方法。例如, 以下敘述可將 Image 物件顯示在表單的座標 (0, 0) 位置。

```
Graphics     g1     =     this.CreateGraphics();
g1.DrawImage(image1,     0,     0);
```

以下敘述可將 Image 物件顯示在 PictureBox1 的座標 (0, 0) 位置, 並將影像調整成 PictureBox1 的寬與高。

```
Graphics      g2      =      this.PictureBox1.CreateGraphics();
g2.DrawImage(image1,     0,     0,     PictureBox1.Width,     PictureBox1.
Height);
```

影像顯示於表單或控制項後, 即可使用 Graphics 的所有繪圖方法, 例如, 以下敘述, 可將原影像於座標 (30, 60) 的地方加上文字。

```
Font   font1   =   new   Font(" 標楷體 ",   14);
SolidBrush    brus1    =    new        SolidBrush(Color.Black);
g2.DrawString(" 洪國勝 ",    font1,    brus1,    30,    60);
```

範例 16-5a

示範以上方法。

執行結果

程式列印

```
private void Button1_Click(object sender, EventArgs e)
{
    // 載入影像
    //string filename1 =@"c:\vbbook\gwosheng.bmp";
    string filename1 = "c:\\Csbook\\gwosheng.bmp";
    Image image1 ;
    image1 = Image.FromFile(filename1);
    // 輸出影像於表單
    Graphics g1 = this.CreateGraphics();
    g1.DrawImage(image1, 0, 0);
    // 輸出影像於控制項
    Graphics g2 = this.PictureBox1.CreateGraphics();
    g2.DrawImage(image1, 0, 0, PictureBox1.Width, PictureBox1.Height);
    // 使用繪圖方法
    Font font1 = new Font("標楷體", 14);
    SolidBrush brus1 = new SolidBrush(Color.Black);
    g2.DrawString("洪國勝", font1, brus1, 30, 60);
    // 儲存影像
    String filename2 =@"c:\Csbook\gwosheng2.bmp";
    image1.Save(filename2) ;
}
```

Bitmap 類別

剛才的 Image 類別是一個抽象基底類別, 僅提供影像處理的部分方法、如存檔、取檔等功能, 若要初始化其執行個體、或進一步處理影像, 例如, 取得某一像素的顏色, 改變某一像素的顏色等, 則應使用其衍生類別, 例如 Bitmap 類別。 Bitmap 類別常用成員說明如下：

建構函式

Bitmap 常用的建構函式如下：

Bitmap(Int32, Int32)	使用指定的大小, 初始化 Bitmap 類別的新執行個體
Bitmap(String)	從指定的檔案, 初始化 Bitmap 類別的新執行個體
Bitmap(Image)	從指定的現有影像初始化 Bitmap 類別的新執行個體

以下說明此 3 種多載如下：

● Bitmap(Int32, Int32)

建構一個指定寬度與高度的空白畫布。其語法如下：

```
public      Bitmap(
        int  width,
        int  height
)
```

例如, 以下敘述可取得一個新的空白畫布, 其大小爲 100*200 像素。

```
Bitmap  b1  =  new  Bitmap(100,  200);
```

以下敘述可取得一個新的空白畫布, 其寬度是表單的寬度與高度。

```
Bitmap    b2    =    new    Bitmap(this.Width,    this.Height);
```

以下敘述可取得一個新的空白畫布, 其寬度是 textBox1 的寬度與高度。

```
Bitmap    b3    =    new    Bitmap(this.PictureBox1.Width,    this.
PictureBox1.Height);
```

以上 b1 、 b2 、 b3 物件即可使用 Bitmap 與 Image 類別的所有方法。

● Bitmap(String)

從指定的檔案, 初始化 Bitmap 類別的新執行個體。其語法如下：

```
public      Bitmap(
          string   filename
)
```

例如, 以下敘述可使用 gwosheng.bmp 初始化 Bitmap 類別的執行個體

```
String      filename      =      @"c:\vbbook\gwosheng.bmp";
bitmap1     =     new     Bitmap(filename);
```

● Bitmap(Image)

從指定的 Image 實體, 初始化 Bitmap 執行個體, 其用法如下：

```
public      Bitmap(
         Image   original
)
```

以下敘述可從 Image 實體, 初始化 Bitmap 執行個體。

```
string      filename1       =       "c:\\Csbook\\gwosheng.bmp";
Image     image1;
image1       =       Image.FromFile(filename1);
Bitmap    b4    =    new    Bitmap(image1);
```

方法成員

Bitmap 常用方法成員如下：

● GetPixel

取得這個 Bitmap 中指定像素的色彩。其語法如下：

```
public      Color_GetPixel      (
        int  x,
        int  y
)
```

例如, 以下敘述可取得座標 (50, 50) 的顏色值。

```
String      filename      =      @"c:\vbbook\gwosheng.bmp";
// 以指定的影像檔例  Bitmap  物件
bitmap1     =     new     Bitmap(filename);
// 輸出影像於控制項
this.PictureBox1.Image        =        bitmap1;
Color      pixelcolor      =      bitmap1.GetPixel(50,      50);
Label2.Text          =          pixelcolor.ToString();
```

- SetPixel

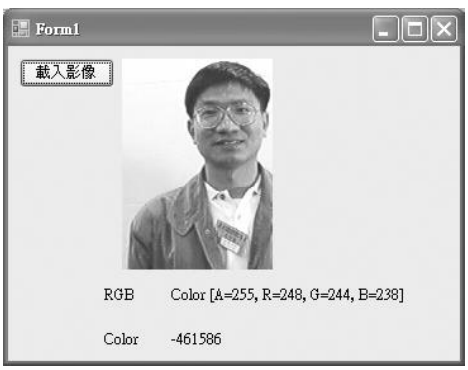
設定此 Bitmap 中指定像素的色彩。其語法如下：

```
public    void    SetPixel    (
        int  x,
        int  y,
         Color  color
)
```

範例 16-5b

請寫一程式, 可以取得滑鼠座標所在顏色。

執行結果



程式列印

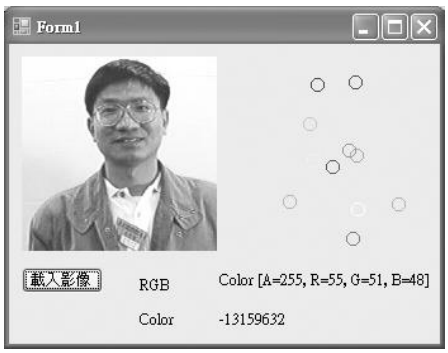
```
public    partial    class    Form1    :    Form
{
         Bitmap  bitmap1;
```

```
    // 使用 drawImage 載入影像於指定畫布
    private void Button1_Click(object sender, EventArgs e)
    {
        String filename = @"c:\Csbook\gwosheng.bmp";
        bitmap1 = new Bitmap(filename);
        Graphics g1 = this.PictureBox1.CreateGraphics();
        g1.DrawImage(bitmap1, 0, 0);
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        String filename = @"c:\vbbook\gwosheng.bmp";
        // 以指定的影像樣例 Bitmap 物件
        bitmap1 = new Bitmap(filename);
        // 輸出影像於控制項
        //this.PictureBox1.Image = bitmap1
    }

    private void PictureBox1_MouseDown(object sender, MouseEventArgs e)
    {
        Color pixelColor = bitmap1.GetPixel(e.X, e.Y);
        Pen pen1 = new Pen(pixelColor);
        Graphics g2 = this.PictureBox2.CreateGraphics();
        g2.DrawEllipse(pen1, e.X, e.Y, 10, 10);
        Label2.Text = pixelColor.ToString();
        Label4.Text = pixelColor.ToArgb().ToString();
    }

    private void PictureBox1_MouseMove(object sender, MouseEventArgs e)
    {
        Color pixelColor = bitmap1.GetPixel(e.X, e.Y);
        Label2.Text = pixelColor.ToString();
        Label4.Text = pixelColor.ToArgb().ToString();
    }
}
```

範例 16-5c

同上範例, 當滑鼠按一下時, 請使用滑鼠指向的顏色, 於 PictureBox2 繪製一圓。

執行結果



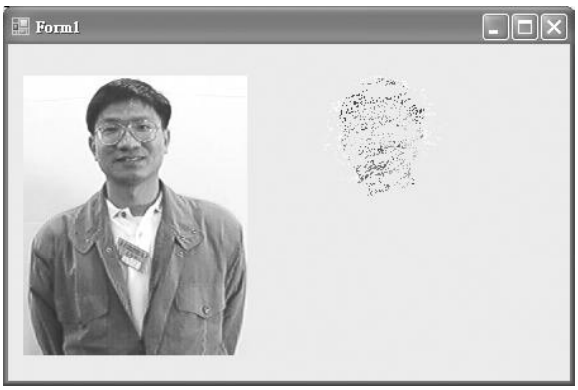
程式列印

```
public partial class Form1 : Form
{
    Bitmap bitmap1;
    private void button1_Click(object sender, EventArgs e)
    {
        String filename = @"C:\Csbook\gwosheng.bmp" ;
        bitmap1 = new Bitmap(filename);
        Graphics g1 = this.pictureBox1.CreateGraphics() ;
        g1.DrawImage(bitmap1, 0, 0); // 使用 drawImage 將畫
        布影像載入控制項
    }

        private void pictureBox1_MouseDown(object sender,
        MouseEventArgs e)
    {
        Color pixelColor = bitmap1.GetPixel(e.X, e.Y);
        Pen pen1 = new Pen(pixelColor);
        Graphics g2 = this.pictureBox2.CreateGraphics();
        g2.DrawEllipse(pen1, e.X, e.Y, 10, 10);
        label1.Text = pixelColor.ToString();
        label2.Text = pixelColor.ToArgb().ToString();
    }
}
```

範例 16-5d

請設計一程式, 完成以下功能。當滑鼠按者 Picture1 左鍵並移動時, 可將 PictureBox1 的像素複製到 pictureBox2 的相對位置。

執行結果



程式列印

```
public partial class Form1 : Form
{
   public Form1()
   {
       InitializeComponent();
   }
   Bitmap bitmap1;
   Bitmap bitmap2;
       private void Form1_Load(object sender, EventArgs e)
       {
       //Picture1 於設計階段載入 gwosheng.bmp
       bitmap1 = (Bitmap)PictureBox1.Image;
       bitmap2 = new Bitmap(PictureBox2.Width, PictureBox2.Height);
  }

       private void PictureBox1_MouseMove(object sender,
       MouseEventArgs e)
  {
            Color pixelColor ;
            int x, y;
            x = e.X ; y = e.Y ;
            if (e.Button == MouseButtons.Left) {
                pixelColor = bitmap1.GetPixel(x, y);
                bitmap2.SetPixel(x, y, pixelColor);
                PictureBox2.Image = bitmap2;
            }
  }
}
```

自我練習

1. 同上範例, 但是每次複製滑鼠所指向的 6*6=36 像素。

Bitmap vs Graphics

Bitmap 的物件實體要如何使用繪圖方法呢？答案是使用 Graphics 介面的公用方法 FromImage, 如以下敘述：

```
Bitmap bitmap1 ;
// 指派畫布尺寸
bitmap1 = new Bitmap(PictureBox1.Width, PictureBox1.Height);
// 結合繪圖方法
Graphics g = Graphics.FromImage(bitmap1);
// 開始畫圖
Pen p1 = new Pen(Color.Red, 3);
g.DrawLine(p1, 30, 30, 100, 100);
```

drawImage 方法

drawImage 是 Graphics 介面的方法, 其用途是將 Bitmap 的物件實體, 顯示在指定的控制項。例如, 以下敘述, 可將 bitmap1 的內容輸出在表單位置 (0, 0) 的地方。

```
Graphics g1 = this.CreateGraphics();
g1.DrawImage(bitmap1, 0, 0);
```

以下敘述可將 bitmap1 的內容輸出在 PictureBox1 物件。

```
Graphics g2 = PictureBox1.CreateGraphics();
g2.DrawImage(bitmap1, 0, 0);
```

其次, 將 Bitmap 的物件實體, 指派給控制項的 Image 屬性, 亦可將 Bitmap 的物件實體輸出到控制項, 如以下敘述。

```
PictureBox1.Image = bitmap1 ;
```

程式的可移植性

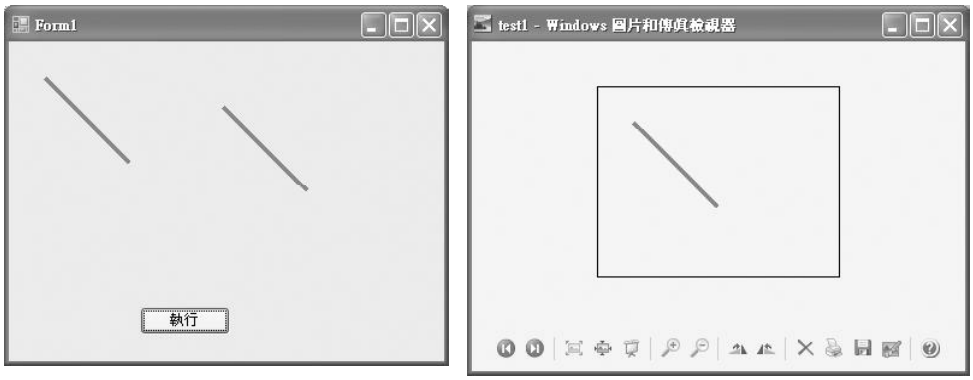
爲了提高程式的可移植性, 通常不要一開始就設定您的畫布是某一控制項, 因爲影像輸出的螢幕大小不一, 有可能是大的電腦螢幕、小筆電、網頁、手機、或 PDA 等。比較有遠見的方法是先將畫布指定在記憶體的某一部份, 如本例的的 Bitmap 執行個體, 待程式最後輸出階段, 再依所要輸出的目的, 指派到適當的控制項, 請看一下範例說明。

範例 16-5e

示範繪圖步驟

執行結果

1. 下圖左的左邊是顯示在表單的直線。
2. 下圖左的右邊是顯示在 PictureBox1 的直線。
3. 下圖右是存檔的結果。



程式列印

```
private void Button1_Click(object sender, EventArgs e)
{
    Bitmap bitmap1 ;
    // 指派畫布尺寸
    bitmap1 = new Bitmap(PictureBox1.Width, PictureBox1.Height);
    // 結合繪圖方法
    Graphics g = Graphics.FromImage(bitmap1);
    // 開始畫圖
```

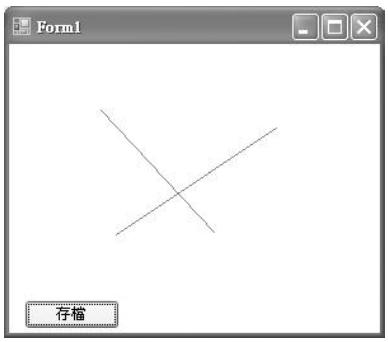
```
    Pen p1 = new Pen(Color.Red, 3);
    g.DrawLine(p1, 30, 30, 100, 100);
    // 顯示於表單
    Graphics g1 = this.CreateGraphics();
    g1.DrawImage(bitmap1, 0, 0);
    // 顯示於控制項
    Graphics g2 = PictureBox1.CreateGraphics();
    g2.DrawImage(bitmap1, 0, 0);
    // 顯示於控制項
    //PictureBox1.Image = bitmap1 // 使用 Image 屬性
    // 存檔
    bitmap1.Save(@"d:\vbbook\ch08\test1.bmp");
}
```

16-6 實例探討(二)

範例 16-6a

請以記憶體畫布的方式, 重做範例 16-2a 的小畫家。

執行結果



演算說明

1. 本例同時開啓兩個同樣大小的畫布, 一個是記憶體的正式畫布 bitmap1 (繪圖方法是 g1), 另一個畫布是表單, 其繪圖方法是 g2 。
2. 當使用者按者滑鼠左鍵移動時, 於表單畫布不斷的繪製直線、重置表單 (清除剛完成的直線), 所以此直線只是過渡性的直線。

3. 當放開滑鼠時, 才於 bitmap1 畫布畫出最終的直線, 且將 bitmap1 畫布的內容複製到表單。

4. 爲了減少螢幕的閃爍, 請將表單的 DoubleBuffered 屬性設爲 True 。

程式列印

```
public partial class Form1 : Form
{
    Bitmap bitmap1;
    Pen pen1 ;
    Graphics g1;
    int x1, y1, x2, y2;
    private void Form1_Load(object sender, EventArgs e)
    {
        // 請於設計階段設定表單的 DoubleBuffered=true , 可降低閃爍程度
        pen1 = new Pen(Color.Red, 1);
        // 於記憶體建立一個畫布一
        bitmap1 = new Bitmap(this.Width, this.Height);
        // 於畫布一取得 Graphics 物件的繪圖方法 g1
        g1 = Graphics.FromImage(bitmap1);
        g1.Clear(Color.White);
    }
    private void Form1_MouseDown(object sender, MouseEventArgs e)
    {
        x1 = e.X;
        y1 = e.Y;
    }
    private void Form1_MouseMove(object sender, MouseEventArgs e)
    {
        // 於表單取得 Graphics 物件的繪圖方法 g2
        Graphics g2 = this.CreateGraphics();
        // 只要是按著滑鼠左鍵移動
        if (e.Button == MouseButtons.Left){
            // 執行 Paint 方法, 將畫布內容複製到表單
            this.Refresh();
            // 於表單重畫直線
            g2.DrawLine(pen1, x1, y1, e.X, e.Y);
        }
    }
    private void Form1_MouseUp(object sender, MouseEventArgs e)
    {
```

```
        g1.DrawLine(pen1,  x1,  y1,  e.X,  e.Y);
        // 將畫布的結果複製到表單
        this.Invalidate();              // 或      this.Refresh();
    }
    private  void  Form1_Paint(object  sender,  PaintEventArgs  e)
    {
        e.Graphics.DrawImage(bitmap1,  0,  0);
    }
    private  void  button1_Click(object  sender,  EventArgs  e)
    {
        bitmap1.Save(@"c:\Csbook\ch08\test1.jpg");
    }
}
```

範例 16-6b

由於安全的考量, 現今監視系統的應用非常普遍。一般的監視系統是全程監視錄影的方式, 例如在一般的巷道及地下道所裝設的監視系統, 所有影像資料透過影像壓縮技術處理後, 儲存在一般媒體之中, 例如影帶或硬碟中, 以備將來查閱使用。

但是在一些特定的應用是監視系統爲了不要儲存大量靜止的影像, 因此發展由移動影像做爲觸發監控系統之錄影模式, 此種模式稱之爲動態偵測系統, 而動態偵測方法很多, 最常見的是影像相減法。

影像相減法是以攝影機擷取到的連續影像直接對相鄰影像進行相對像素之差值運作：

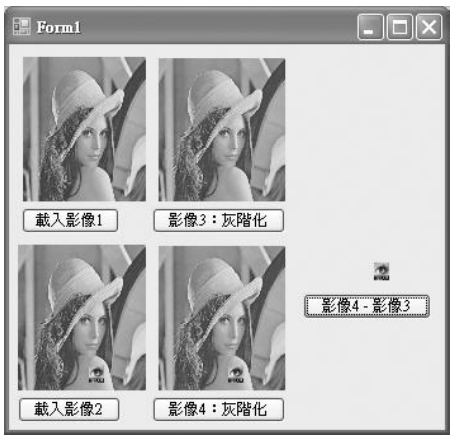
$$|P1(x, y)-P2(x, y)| > \tau$$

其中 P1 爲影像一之像素, P2 爲影像二之像素, τ 值爲容許誤差值。若符合上式方程式則該位置之像素保留下來, 反之則移除。做爲是否有物體進入監視範圍之判斷依據。

我們將寫一程式將輸入兩影像作影像相減去除背景, 取出主體影像, 實作步驟如下：

1. 載入影像。
2. 將載入之影像做灰階處理。
3. 將灰階處理後之兩影像作影像相減運算。

執行結果



程式列印

```
public partial class Form1 : Form
{
    Bitmap b1, b2, b3, b4, b5;
    // 載入影像 1
    private void button1_Click(object sender, EventArgs e)
    {
        openFileDialog1.Filter = "tif files|*.tif|png files|*.
        png|jpg files|*.jpg|bmp files|*.bmp" ;
        if (openFileDialog1.ShowDialog() == System.Windows.Forms.
        DialogResult.OK) {
            pictureBox1.ImageLocation = openFileDialog1.
            FileName;
            b1 = new Bitmap(openFileDialog1.FileName);
        }
    }
    // 載入影像 2
    private void button2_Click(object sender, EventArgs e)
    {
        openFileDialog1.Filter = "tif files|*.tif|png files|*.
        png|jpg files|*.jpg|bmp files|*.bmp";
        if (openFileDialog1.ShowDialog() == System.Windows.
        Forms.DialogResult.OK)
        {
            pictureBox2.ImageLocation = openFileDialog1.
            FileName;
            b2 = new Bitmap(openFileDialog1.FileName);
     }
    }
    // 灰階化
```

```
private void button3_Click(object sender, EventArgs e)
{
    b3 = new Bitmap(b1.Width, b1.Height);
    Color pixelColor;
    int ColorAvg ;
    for(int i=1;i<b1.Width;i++){
        for(int j=1;j<b1.Height;j++){
            pixelColor = b1.GetPixel(i, j);
            ColorAvg = (pixelColor.R / 3 + pixelColor.G / 3
            + pixelColor.B / 3);pixelColor = Color.FromArg
            (ColorAvg, ColorAvg, ColorAvg);
            b3.SetPixel(i, j, pixelColor);
 }
    }
    pictureBox3.Image = b3;
}

private void button4_Click(object sender, EventArgs e)
{
    b4 = new Bitmap(b2.Width, b2.Height);
    Color pixelColor;
    int ColorAvg;
    for (int i = 1; i < b2.Width; i++)
    {
        for (int j = 1; j < b2.Height; j++)
        {
            pixelColor = b2.GetPixel(i, j);
            ColorAvg = (pixelColor.R / 3 + pixelColor.G
            / 3+ pixelColor.B / 3);pixelColor = Color.
            FromArgb (ColorAvg, ColorAvg, ColorAvg);
            b4.SetPixel(i, j, pixelColor);
        }
    }
    pictureBox4.Image = b4;
}
// 影像 4 - 影像 3
private void button5_Click(object sender, EventArgs e)
{
    b5 = new Bitmap(b3.Width, b3.Height);
    Color pixelColor3, pixelColor4 ;
    for (int i = 1; i < b3.Width; i++)
    {
        for (int j = 1; j < b3.Height; j++)
        {
        pixelColor3 = b3.GetPixel(i, j);
```

```
                pixelColor4      =      b4.GetPixel(i,      j);
                if      (pixelColor3      !=      pixelColor4){
                    b5.SetPixel(i,          j,          pixelColor4);
                }
            }
        }
        pictureBox5.Image    =    b5    ;
    }
}
```

範例 16-6c

在很多分群概要中, 一個未知向量 x, 會依照 x 和集合 C 間之鄰近函數 D(x, C) 的關係, 來決定 x 是否屬於集合 C 。以下有三種方式來定義鄰近函數 D(x, C)：

(1) 最大鄰近函數 (The max proximity function)：

$$D_{\max}^{ps}(\underline{x},C)=\max_{\underline{y}\in C} D(\underline{x},\underline{y})$$

(2) 最小鄰近函數 (The min proximity function)：

$$D_{\min}^{ps}(\underline{x},C)=\min_{\underline{y}\subset C} D(\underline{x},\underline{y})$$

(3) 平均鄰近函數 (The average proximity function)：

$$D_{avg}^{ps}(\underline{x},C)=\frac{1}{n_c}\sum_{\underline{y}\in C} D(\underline{x},\underline{y})$$

其中 nc 是集合 C 中的基數 (cardinality), 也就是集合 C 中元素的個數。

假設集合 $C=\{\underline{x}_1,\underline{x}_2,\underline{x}_3,\underline{x}_4,\underline{x}_5,\underline{x}_6,\underline{x}_7,\underline{x}_8\}$, 其中

$$\underline{x}_1=[1.5,\ 1.5]^T$$
$$\underline{x}_2=[2.0,\ 1.0]^T$$
$$\underline{x}_3=[2.5,\ 1.75]^T$$
$$\underline{x}_4=[1.5,\ 2.0]^T$$

$$\underline{x}_5 = [3.0,\ 2.0]^T$$
$$\underline{x}_6 = [1.0,\ 3.5]^T$$
$$\underline{x}_7 = [2.0,\ 3.0]^T$$
$$\underline{x}_8 = [3.5,\ 3.0]^T$$
未知向量 $\underline{x} = [6.0,\ 4.0]^T$

假設利用歐幾里德距離 (Euclidean distance) 來計算任兩點之間的距離, 則上述三種鄰近函數所得到的值分別如下：

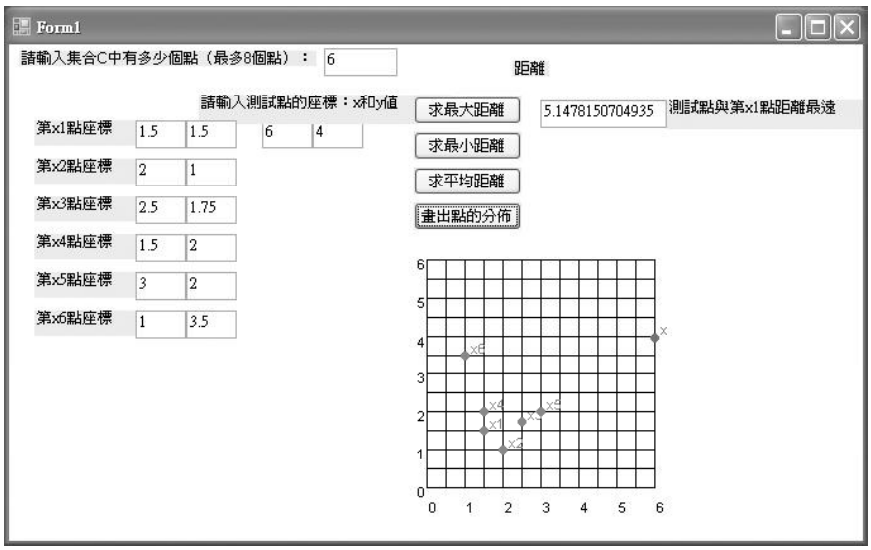
$$d^{ps}_{\max}(\underline{x},C) = \max_{\underline{y}\in C} d(\underline{x},\underline{y}) = d(\underline{x},\underline{x}_1) = 5.1478150704935$$
$$d^{ps}_{\min}(\underline{x},C) = \min_{\underline{y}\in C} d(\underline{x},\underline{y}) = d(\underline{x},\underline{x}_8) = 2.69258240356725$$
$$d^{ps}_{avg}(\underline{x},C) = \frac{1}{n_c}\sum_{\underline{y}\in C} d(\underline{x},\underline{y}) = \frac{1}{8}\sum_{t=1}^{8} d(\underline{x},\underline{x}_1) = 4.33490629135928$$

我們將寫一個程式, 具有下列功能：

1. 當程式剛執行時會出現一視窗, 提示使用者先輸入集合中有多少個向量, 爲了簡化問題, 最少 1 個向量, 最多 8 個向量, 而且向量僅使用二維 x, y 空間座標, 即用點數來表示, 而且其 x 和 y 座標都落在 (0, 0)~(6, 6) 之間。
2. 當使用者輸入上述點數後, 程式會自動出現集合 C 中點數的 x 和 y 座標, 讓使用者輸入所有點的座標。而且程式會自動出現未知向量 x (測試點), 讓使用者輸入 x 和 y 座標。
3. 按 " 求最大距離 " 鈕, 會利用最大鄰近函數, 來求測試點與集中 C 中, 哪一點的距離最大？並且顯示距離與點數編號。
4. 按 " 求最小距離 " 鈕, 會利用最小鄰近函數, 來求測試點與集中 C 中, 哪一點的距離最小？並且顯示距離與點數編號。
5. 按 " 求平均距離 " 鈕, 會利用平均鄰近函數, 來求測試點與集中 C 中, 所有點的平均距離, 並且顯示距離與點數編號。

6. 按 " 畫出點的分佈 " 鈕, 會畫出集合 C 中的點數, 測試點 x 的點數, 並用不同顏色來區分集中 C 和 x 的點數, 同時 x, y 二維空間座標也要畫出來。

執行結果



演算說明

1. 本例的集合至多 8 個點, 如下表所示：本例以 C 陣列儲存：
2. 輸入集合個數。

集合	x1	x2	x3	x4	x5	x6	x7	x8
x	1.5	2.0	2.5	1.5	3.0	1.0	2.0	3.5
y	1.5	1.0	1.75	2.0	2.0	3.5	3.0	3.0

3. 輸入任一點座標。本例假設是 x1, y1
4. 計算最小鄰近距離。將(x1, y1)一一代入集合 C, 求其距離, 並求其極小值。
5. 計算最大鄰近距離。將(x1, y1)一一代入集合 C, 求其距離, 並求其極大值。

6. 計算平均鄰近距離。將(x1, y1)一一代入集合 C, 求其距離, 並求其平均值。
7. 輸出以上資料
8. 在座標平面繪出以上點。

程式列印

```
public partial class Form1 : Form
{
  int n =0;
  double[, ] a = { { 0, 0 }, { 1.5, 1.5 }, { 2.0, 1.0 }, { 2.
  5 1.75 } { 1.5 2.0 } { 3.0 2.0 } { 1.0 3.5 } { 2.0
  3 . 0 }, { 3.5, 3.0 } };
  double x = 6.0 ;
  double y = 4.0;
  Label[] lbl =new Label[8];
  TextBox [, ] txta = new TextBox[8, 2];
  TextBox txtx, txty;
  Label lblxy ;

  private void TextBox1_TextChanged(object sender, EventArgs e)
  {
      n = int.Parse(TextBox1.Text);
      for(int i=1;i<=n;i++){
          lbl[i] = new Label();
          lbl[i].Left = 20 ; // 設定位置
          lbl[i].Top = 30 + 30 * i;
          lbl[i].Height = 20; // 大小
          lbl[i].Width = 80;
          lbl[i].Text = "第 x" + i.ToString() + "點座標 ";
          lbl[i].Visible = true;
          this.Controls.Add(lbl[i]);
      }
      for(int i=1;i<=n;i++){
          for(int j=0;j<=1;j++){
          txta[i, j] = new TextBox();
          txta[i, j].Left = 100 + 40 * j ; // 設定位置
          txta[i, j].Top = 30 + 30 * i;
          txta[i, j].Height = 20 ; // 大小
          txta[i, j].Width = 40;
          txta[i, j].Text = a[i, j].ToString();
          txta[i, j].Visible = true;
          this.Controls.Add(txta[i, j]);
```

```
        }
    }
        lblxy = new Label();                          //產生一個物件
        lblxy.Location = new Point(150, 40);  // 位置
        lblxy.Width = 200;
        lblxy.Text = "請輸入測試點的座標：x 和 y 值";
        lblxy.Visible = true;
        this.Controls.Add(lblxy);                     //將控制項加入方案

        txtx = new TextBox();                         //產生一個物件
        txtx.Location = new Point(200, 60) ;  // 位置
        txtx.Width = 40;
        txtx.Text = (6.0).ToString();
        txtx.Visible = true;
        this.Controls.Add(txtx);                      //將控制項加入方案

        txty = new TextBox();                         //產生一個物件
        txty.Location = new Point(240, 60);   // 位置
        txty.Width = 40;
        txty.Text = (4.0).ToString();
        txty.Visible = true;
        this.Controls.Add(txty);                      //將控制項加入方案
    }

    private void Button1_Click(object sender, EventArgs e)
    {
        double max = 0 ;
        int maxno = 0 ;
        int i, j ;

        for( i=1;i<=n;i++){
            for( j=0;j<=1;j++){
                a[i, j] = double.Parse(txta[i, j].Text);
            }
        }
    x = double.Parse(txtx.Text);
    y = double.Parse(txty.Text);
    double len ;
    for(i=1;i<=n;i++){
        len = Math.Sqrt( Math.Pow(x - a[i, 0], 2) + Math.Pow
        (y - a[i, 1], 2) ) ;
        // 請留意括號位置
        if (len > max){
            max = len ;
            maxno = i ;
        }
```

```
}

TextBox txtmax ;
txtmax = new TextBox() ; // 產生一個物件
txtmax.Location = new Point(420, 44); // 位置
txtmax.Width = 100;
txtmax.Text = string.Format("{0}", max);
txtmax.Visible = true;
this.Controls.Add(txtmax); // 將控制項加入方案

Label lblmax ;
lblmax = new Label(); // 產生一個物件
lblmax.Location = new Point(520, 44); // 位置
lblmax.Width = 200;
lblmax.Text = "測試點與第 x" + String.Format("{0}", maxno) +
"點距離最遠";
lblmax.Visible = true;
this.Controls.Add(lblmax); // 將控制項加入方案
}

private void Button2_Click(object sender, EventArgs e)
{
    // 自行發揮
}

private void Button4_Click(object sender, EventArgs e)
{
    String ns = TextBox1.Text ;
    if (ns==""){
        MessageBox.Show("集合 C 中輸入點數錯誤, 先用預設的 8 個
        點, 來畫出點的分佈。請在輸入一次。");
    }else{
        Graphics g = this.CreateGraphics();
        Pen p1 = new Pen(Color.Black, 1);

    int x0 = 330;
    int y0 = 350;

    // 水平放大倍率
    int sx = 30 ;
    // 垂直放大倍率
    int sy = 30;
    int dy = -1;

    // 清除螢幕
```

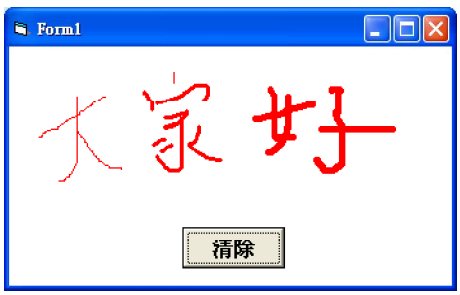
```
        g.Clear(Color.White);
        // 平移
        g.TranslateTransform(x0,          y0);
        int    x1,  y1,  x2,  y2  ;
        for(double          i=0;i<=6;i+=0.5){
              // 水平線
              x1  =  0;
              y1  =  (int)(  i  *  sy  *  dy);
              x2  =  6  *  sx;
              y2  =(int)(  i  *  sy  *  dy);
              g.DrawLine(p1,   x1,   y1,   x2,   y2);

              // 垂直線
              x1  =  (int)(i  *  sx);
              y1  =  0;
              x2  =  (int)(i  *  sx);
              y2  =  6  *  sy  *  dy;
              g.DrawLine(p1,   x1,   y1,   x2,   y2);

        }
        // 標示座標
        Font  font  =  new  Font("Arial",  8);
        SolidBrush     sBrush=     new     SolidBrush(Color.Black);

        for(int          i=0;i<=6;i++){
              // 水平座標
              x1  =  i  *  sx  ;
              y1  =(int)(  -0.3  *  sy  *  dy);
              g.DrawString(     i.ToString().Trim()     ,     font,     sBrush,
                 x1,  y1);
              // 垂直座標
              x1  =  (int)(-0.3  *  sx);
              y1  =  (int)((i  +  0.1)  *  sy  *  dy);
              g.DrawString(i.ToString().Trim(),     font,     sBrush,     x1,     y1);
        }
        // 標點
        Pen  p2  =  new  Pen(Color.Red,  1);
        SolidBrush    sBrush2    =    new    SolidBrush(Color.Red);
        // 讀取資料
        for(int          i=1;i<=n;i++){
              for(int          j=0;j<=1;j++){
                    a[i,     j]     =     double.Parse(txta[i,     j].Text);
              }
        }
        x     =     int.Parse(txtx.Text);
        y     =     int.Parse(txty.Text);
```

```
        int height, width ;
        for(int i=1;i<=n;i++){
            x1 = (int) ((a[ i, 0] - 0.1) * sx);
            y1 = (int)((a[i, 1] + 0.1) * sy * dy);
            width =(int)( 0.2 * sx);
            height =(int)( 0.2 * sy);
            g.DrawEllipse(p2, x1, y1, width, height);
            // 填滿顏色
            g.FillEllipse(sBrush2, x1, y1, width, height);
            // 標示文字
            x2 = (int) ((a[ i, 0] + 0.1) * sx);
            y2 =(int)( (a[i, 1] + 0.4) * sy * dy );
            g.DrawString("x" + i.ToString().Trim(), font,
            sBrush2, x2, y2) ;
        }
        / / x y
        Pen p3 = new Pen(Color.Green, 1);
        SolidBrush sBrush3 = new SolidBrush(Color.Green);
        x1 =(int)( (x - 0.1) * sx );
        y1 =(int)( (y + 0.1) * sy * dy);
        width =(int) (0.2 * sx);
        height = (int)(0.2 * sy);
        g.DrawEllipse(p3, x1, y1, width, height);
        g.FillEllipse(sBrush3, x1, y1, width, height);
        x2 = (int)((x + 0.1) * sx);
        y2 = (int)((y + 0.4) * sy * dy);
        g.DrawString("x", font, sBrush3, x2, y2);
    }
}
```

習題

1. 試寫一個程式, 使用滑鼠作爲畫筆, 滑鼠按鍵左、右分別代表紅色與綠色；鍵盤的 SHIFT 、 CTRL 及 ALT 鍵分別代表筆寬爲 1 、 2 、 4 。

   執行結果：



2. 在影像處理的技術中, 灰階化是讓 RGB 三個顏色都一樣, 在處理上, 我們純粹用一個數值來代表明亮度, 譬如說, 使用 255 來代表最亮的白色, 以及使用 0 來代表最暗的黑色, 0 與 255 中間就是所謂的灰階地帶, 0 ～ 255 剛好可以用一個 byte 來表示, 而要將圖檔轉爲灰階, 可採用如下公式來計算：

```
灰階亮度 = 0.299* 紅色 + 0.587* 綠色 + 0.114* 藍色
```

如此一來, 即可單純地將所有色彩以亮度來表示, 而透過這個式子轉換過後的圖檔, 即是灰階化後的圖檔。利用灰階影像可再進行處理與應用, 例如：

(1) 影像二值化 (binarize)：此功能可將影像中的物體與背景分離。

$$G(x,y) = \begin{matrix} 0 & g(x,y) \le T_0 \\ 255 & g(x,y) > T_0 \end{matrix}$$

其中 G(x, y) 爲二值化後之灰階值, g(x, y) 爲像素 (x, y) 之灰階值, to 爲門檻值。

(2) 影像反白 (inverse)

```
G(x, y) = 255 - g(x, y)
```

其中 G(x, y) 爲反白後之灰階值, g(x, y) 爲像素 (x, y) 之灰階值。

(3) 影像對比拉伸 (stretch)

影像對比拉伸函數可將影像變得更爲清晰, 此功能是提高影像之對比。

$$G(x,y) = \frac{g(x,y) - \min}{\max - \min} \times 255$$

其中 G(x, y) 爲對比拉伸後之灰階值, g(x, y) 爲像素 (x, y) 之灰階值, min 爲影像中最小之灰階值, max 爲影像中最大之灰階值。

請依據以上敘述, 設計一個程式, 具備以下功能：

(1) 載入一個彩色電腦圖檔 (*.bmp 或 *.gif)。

(2) 按下 " 灰階化 " 命令鍵後, 出現灰階化之圖。

(3) 按下 " 反白 " 命令鍵後, 出現反白處理之圖。

(4) 輸入門檻值 to (範圍爲 0 ～ 255) 後按下 " 二值化 " 命令鍵後, 出現二值化處理之圖。

(5) 按下 " 對比拉伸 " 命令鍵後, 計算出該影像之最大與最小灰階值且出現對比拉伸處理之圖。

3. 線性回歸（Linear Regression）說明：在二維的平面上, 若有許多資料點, 線性回歸用來找出一條最接近這些資料點的直線方程式:

```
y = mx + b
```

以便得到最好的逼近結果, 上述式子中, m 表示斜率, b 表示 y 軸截距, m 和 b 就是所謂的回歸係數。而最小平方逼近法, 常被用來求線性回歸的係數 m 和 b, 最小平方逼近法是將觀察的 y 值和預測的 y 值做相減, 再求其平方的總和, 經由公式的推導, 得到最小平方線的斜率（m）公式如下：

$$m = \frac{(\sum xy) - (\sum x)\overline{y}}{(\sum x^2) - (\sum x)\overline{x}}$$

得到最小平方線的截距 (b) 公式如下：

$$b = \overline{y} - m\overline{x}$$

其中：

$\sum x$　表示 x 值的總和

$\sum x^2$　表示 x 值平方的總和

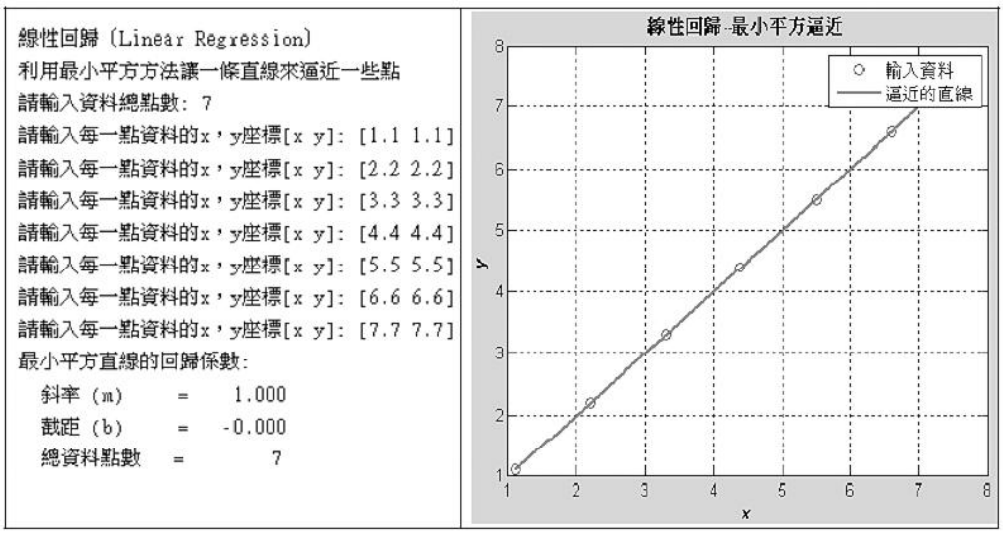
$\sum xy$　表示 x 值與 y 值乘積的總和

$\overline{x}$　表示 x 值的平均值

$\overline{y}$　表示 y 值的平均值

請寫一個程式, 當輸入一些資料點（x, y）, 計算其最小平方斜率 m 和 y 軸截距 b 。資料點採用鍵盤輸入, 另外, 畫出每一個別資料點和最小平方逼近的直線。資料點數目 N 爲 2 ≦ N ≦ 10 。

範例：此範例輸入 N=7 個資料點（1.1, 1.1）, （2.2, 2.2）, （3.3, 3.3）, （4.4, 4.4）, （5.5, 5.5）, （6.6, 6.6）, （7.7, 7, 7）, 利用最小平方法求的的斜率 m ＝ 1.0 和 y 軸截距 b ＝ 0.0 or -0.0 。執行過程和結果如下, 小圓圈表示輸入資料點。



M • E • M • O