第 7 章　簡易小畫家

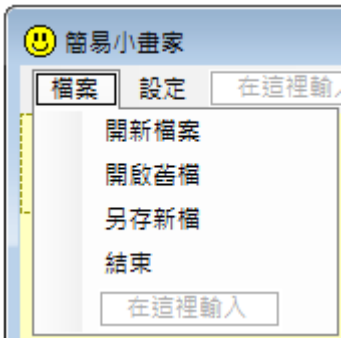
簡介：

上一章我們學會了一些處理影像物件的基本概念，這一章我們將繼續學習用 C#程式繪圖的技巧。比較複雜的是在 C#中，繪圖的操作層次分得很精細，且多半須以程式碼操作，包括影像與繪圖物件的建置、畫筆及筆刷顏色與粗細的設定等等，對初學者而言確實有點複雜，就請耐心學習了。

7-1 建立開檔存檔功能

[建立主功能表]

這個程式與上一單元一樣，主要還是由功能表(MenuStrip)操作，請先到工具箱叫出一個 menuStrip1 物件。這次我們是要畫圖的，所以除了開啟舊檔之外，還必須有新增空白圖檔，並回存作品(影像)的功能。如果你還記得之前的單元就會知道，另存新檔是比儲存檔案較簡單的動作，本單元就省略完整儲存功能的介紹，只作「另存新檔」就好了。請設計如下：



開啟影像檔案如前一單元一樣，需要用到 PictureBox 與 OpenFileDialog 物件，本單元也要作存檔動作，需要 SaveFileDialog，請到工具箱找出它們放到表單上。pictureBox1 的 SizeMode 屬性請設為 AutoSize，本單元以畫圖為主，不作縮放處理，進出的檔案都照原始大小顯示就好了！其次 openFileDialog1 與 saveFileDialog1 物件的 Filter 也都設為「*.jpg|*.jpg」，表示我們只處理 jpg 檔案的意思。當然你是可以設定處理更多格式的，如有需要請參考秀圖軟體單元。最後請將表單的 WindowState 設定為 Maximized(放到最大)，一般影像軟體總是希望畫面越大越好的！

[加入影像顯示盒]

仿照正版小畫家，請將 pictureBox1 放到視窗的左上角，如果你無法順利將 pictureBox1 貼齊到主功能表的下方，請查看一下 MenuStrip1 的 Size->Height(高度)值，將 pictureBox1 的 Location->Y 值(頂部座標，相當於 Top 屬性)設為與 MenuStrip1.Height(應該是 24 個像素點)一樣就貼齊了！

[開啟舊影像]

首先開啟舊檔的程式與前一單元相同如下：

```
private void 開啟舊檔ToolStripMenuItem_Click(object  sender, EventArgs e)
{
    if(openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        pictureBox1.Load(openFileDialog1.FileName);
    }
}
```

[儲存目前影像]

接下來另存新檔是將目前 pictureBox1 裡面的影像(Image)依使用者在 saveFileDialog1 中指定的檔名存到磁碟之中。因為我們之前已經將 Filter 設定為只處理 jpg 檔案，所以鍵入檔名時不需要再寫副檔名了！譬如你寫存入檔案「X」，就會變成 X.jpg 圖檔。請注意到本單元是要畫圖的，所以儲存的檔案可能是某圖檔被你叫出來塗鴉之後的「作品」，最好還是另存新檔，不要覆蓋原圖比較好。

```
private void 另存新檔ToolStripMenuItem_Click(object  sender, EventArgs e)
{
    if(saveFileDialog1.ShowDialog() == DialogResult.OK)
    {
        Bitmap bmp = new Bitmap(pictureBox1.Image);
        bmp.Save(saveFileDialog1.FileName);
    }
}
```

[Bitmap 點陣圖物件]

上述程式中的 Bitmap bmp = new Bitmap(pictureBox1.Image)是宣告一個稱為 bmp 的 Bitmap(點陣圖)物件，內容與 pictureBox1.Image 相同；接下來將此物件儲存(Save)為選定檔名 saveFileDialog1.FileName。此動作似乎有些多餘，但是直接用 pictureBox1.Image.Save 這樣的語法卻會產生「GDI+泛型錯誤」，同樣.NET Framework 版本的 VB 這樣寫卻是可以的！只能說是 C#語法比較嚴謹吧？

7-2 開新檔案

[建立新影像]

小畫家程式最常用的應該不是開啟舊檔來玩，而是開新檔案繪圖，甚至程式一開始就會開個空白的新檔案給我們使用！在此先寫一個開新檔案的程式：

```
private void 開新檔案ToolStripMenuItem_Click(object  sender, EventArgs e)
{
    pictureBox1.Image = new Bitmap(800, 600);
    Graphics g = Graphics.FromImage(pictureBox1.Image);
    g.Clear(Color.White);
}
```

我們先建置一個 800 x 600 點的點陣圖物件作為 pictureBox1 的 Image。接著宣告一個稱

為 g 的繪圖物件(Graphics)，此物件來自於(也就是會執行繪圖動作於)pictureBox1.Image，最後使用 g 將 Image 清除為白底(背景色)的狀況。

[Image 物件沒有 BackColor 屬性]

如果你不製作底色，預設的 Bitmap 是透明的，你可能會看不到圖在哪裡！但是程式碼蠻囉嗦的，不是嗎？或許你會說為何不能用類似 Image.BackColor=Color.White 這種語法呢？就是將底色當作屬性處理，希望以後的 C#版本可以這樣寫。但目前的 Graphics 物件總攬所有的繪圖動作，包括在 Image 上面畫複雜的圖案，譬如直線、方塊、圓圈或文字等等，連簡單的背景色設定也都必須經過它，程式看起來就複雜了，忍耐一下吧！試試看點選開新檔案會有一張白紙出現了！

[Form_Load 時開新檔案]

接下來我們希望程式啟動(Form_Load)時也會自動給一張新的空白圖，當然我們可以寫完全一樣的程式，也就是複製「開新檔案」的內容到 Form_Load 中，不過也可以這樣寫：

```
private void Form1_Load(object sender, EventArgs e)
{
    開新檔案ToolStripMenuItem_Click(sender, e);
}
```

是不是與前一單元的 SizeImage 等自訂副程式用法很像？就是執行到此時去呼叫『開新檔案』的程式，請記得小括號內的參數必須照寫，但物件型態名稱(object 與 EventArgs)則必須刪掉。這也告訴我們其實事件副程式與一般自訂副程式沒有甚麼不同，也可以直接呼叫的！至於為何按下「開新檔案」這個功能項目時會執行這個副程式？程式碼是在 Form1.Designer.cs 這個檔案有關「開新檔案」這個功能項目的設定程式內：

[事件副程式與事件的連結]

```
//
// 開新檔案ToolStripMenuItem
//
this.開新檔案ToolStripMenuItem.Name = "開新檔案ToolStripMenuItem";
this.開新檔案ToolStripMenuItem.Size = new System.Drawing.Size(152, 22);
this.開新檔案ToolStripMenuItem.Text = "開新檔案";
this.開新檔案ToolStripMenuItem.Click += new System.EventHandler(this.開新檔案
    ToolStripMenuItem_Click);
```

看到了嗎？最後一行的前面是事件名稱(….Click)，後面則是說新增(new)一個事件處理程式(EventHandler)，位置就在名為「開新檔案 ToolStripMenuItem_Click」的副程式。就是這行程式讓事件與副程式連線的！所以如果刪掉它，即使事件副程式的程式碼還在，也是沒有反應的！

當然，如果倒過來我們刪除了事件副程式的程式碼，卻留下上面的粗體字程式，就會出現找不到「開新檔案 ToolStripMenuItem_Click」副程式的錯誤。知道這層關係是蠻重要的！因為我們常常會刪除無用的事件副程式，此時這一行程式也要同時刪掉！另一方面，如果發

現某事件副程式寫好但沒有反應時，應該知道到這邊檢查看看連結事件與副程式的程式碼。

7-3 塗鴉

[用滑鼠拖曳動作繪圖]

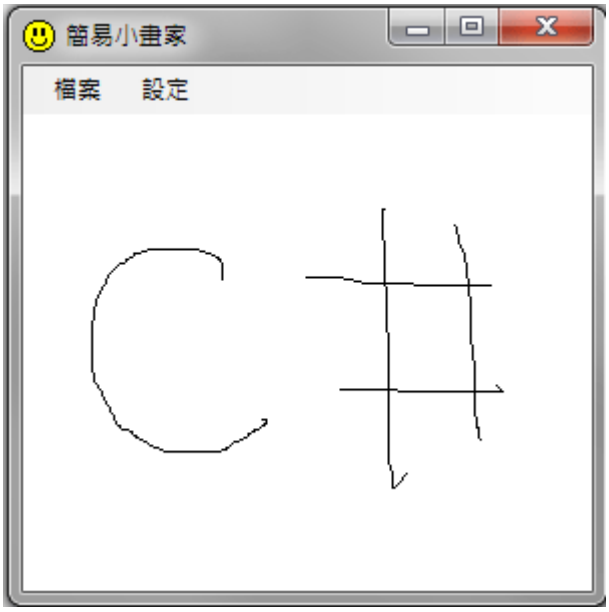
有了空白的畫圖紙，第一件事當然就是想給它亂塗鴉一下的啦！要寫塗鴉的程式必須先分解細部動作：先是滑鼠對著 pictureBox1 上面的某處(起點)壓下左鍵，這就是 MouseDown 事件啦！接著使用者繼續壓住滑鼠左鍵移動滑鼠到新的一點(終點)，就是 MouseMove 事件了！每次的移動我們都要立即畫一個小線段，從起點到終點，畫完之後將終點變成新的起點，這樣滑鼠繼續移動時就會繼續畫隨後的小線段，這些小線段連起來就是你的塗鴉軌跡了！所以我們要將程式寫在 picturBox1 的滑鼠事件中，請先到設計頁面點選 pictureBox1，再到屬性視窗找到 MouseDown 與 MouseMove 建立事件副程式框架，程式碼如下：

```
int x0, y0;
private void pictureBox1_MouseDown(object  sender, MouseEventArgs e)
{
    x0 = e.X;
    y0 = e.Y;
}

private void pictureBox1_MouseMove(object  sender, MouseEventArgs e)
{
    if(e.Button == MouseButtons.Left)
    {
        Graphics g = Graphics.FromImage(pictureBox1.Image);
        g.DrawLine(Pens.Black, x0, y0, e.X, e.Y);
        x0 = e.X;
        y0 = e.Y;
        pictureBox1.Refresh();
    }
}
```

[Graphics 與 Pens 的設定]

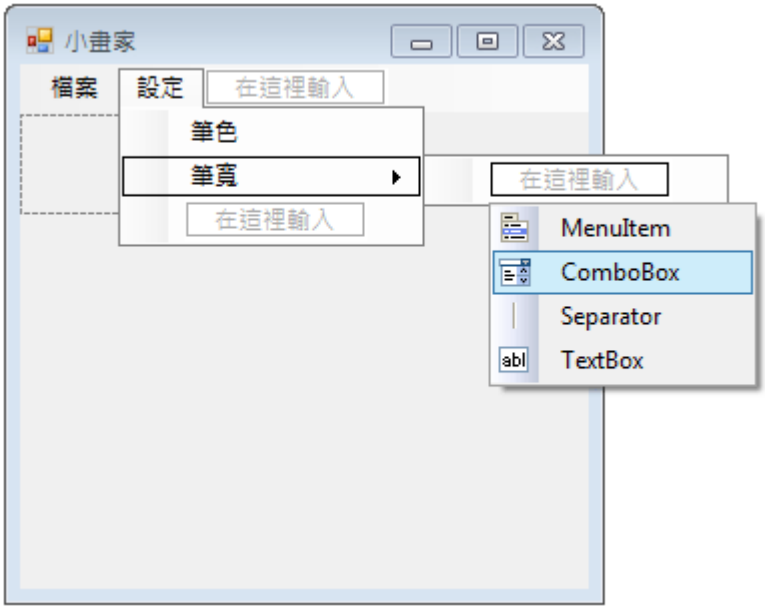
上面程式碼中的 x0 與 y0 就是繪圖起點的座標，當 MouseDown 事件發生時先記住這個位置，所謂 e.X 與 e.Y 就是滑鼠目前位置的座標。接下來滑鼠移動了！我們首先宣告繪圖物件 g，當然是要它指向目前的 pictureBox1.Image 這個影像了！請繪圖物件 g 畫一個線段(DrawLine)，使用黑色的筆(Pens.Black)，起終點座標為 (x0, y0) - (e.X, e.Y)，畫完之後將終點變成新的起點 x0 = e.X，y0 = e.Y。試試看！塗鴉動作應該可以執行了！這段程式碼其實與前面記事本單元的拖曳程式很類似，只是多了繪圖動作而已。別忘了你也可以使用剛剛寫的存檔程式將作品典藏哦！



7-4 調整畫筆的顏色與粗細

[設定 Pen 物件屬性]

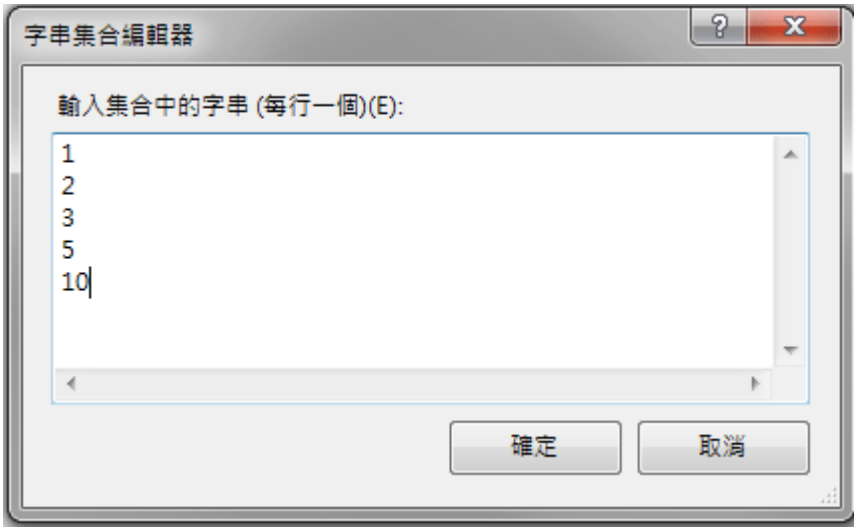
剛剛的程式是使用系統預設的黑筆(Pens.Black)，粗細也是預設的一個像素點寬，如果要用自己設定的筆就要宣告一個「筆物件」(Pen)來繪圖。像繪圖物件一樣，其實筆物件也可以隨時宣告設定顏色與粗細來使用！請先在功能表加入以下功能項目：



[使用 ComboBox]

請注意到上圖最右邊我們是將新選項的右端選單拉開加入一個 ComboBox，這是一個下拉式選單，建立好之後編輯其 Items(項目)集合屬性如下圖，當然項目就是畫筆粗細的選擇，你可以自由發揮。寫好這個集合之後也要將 ComboBox 的 Text 屬性改為 1，這是筆的預設粗

細值。請注意到：Combo 是複合的意思，表示它是文字輸入方塊(TextBox)與選單物件的複合體，使用者除了在有限的選項中選擇之外，直接輸入不同於選項的數值也是可以的！但是筆粗細必須為整數，如果輸入 2.5 程式實際繪圖時就會當掉！

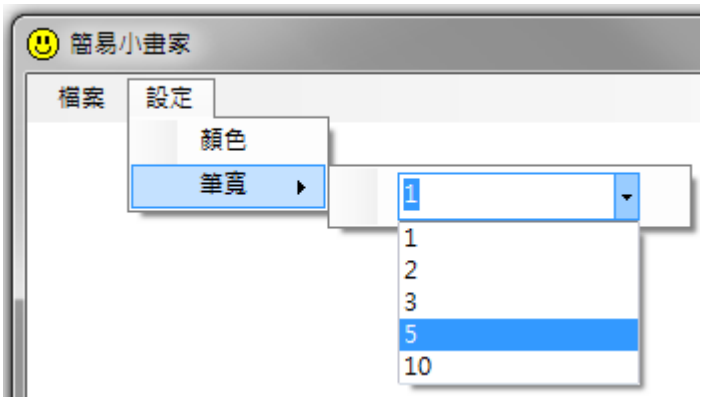


[使用 ColorDialog]

還有選擇顏色時需要一個 ColorDialog 對話方塊，也請準備好，位置在工具箱的「對話方塊」分類。這樣就可以開始寫程式了！請雙擊「筆色」進入程式碼頁面，寫程式如下：

```
private void 筆色ToolStripMenuItem_Click(object  sender, EventArgs e)
{
    colorDialog1.ShowDialog();
}
```

上述程式只是讓選色方塊顯示出來供使用者選色而已，至於選畫筆粗細的介面可以直接操作，連開啟程式都不必寫了！操作畫面如下：

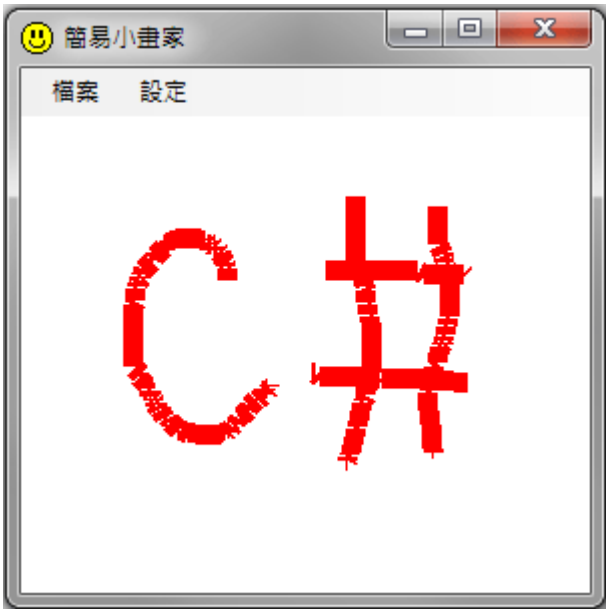


[建立 Pen 物件的程式]

但這樣還是不會產生改變畫筆的效果，程式必須修改，使用這兩個操作的結果去「製作」需要的畫筆來執行繪圖。請將塗鴉部分的 MouseMove 事件副程式修改如下：

```
private void pictureBox1_MouseMove(object  sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left)
    {
        Graphics g = Graphics.FromImage(pictureBox1.Image);
        Pen p = new Pen(colorDialog1.Color, int.Parse(toolStripComboBox1.Text));
        g.DrawLine(p, x0, y0, e.X, e.Y);
        x0 = e.X;
        y0 = e.Y;
        pictureBox1.Refresh();
    }
}
```

與原來程式只有兩行的差異，一是宣告一支新畫筆(new Pen)，筆色來自 colorDialog1 的 Color 屬性，筆的粗細則來自下拉式選單(toolStripComboBox1)的文字(Text)所代表的整數，在此必須用 int.Parse 將文字資料轉為整數數值；接著就用這支筆(p)置換原來系統預設的黑筆(Pens.Black)。如果 toolStripComboBox1 的超長名稱對你來說有些困擾，建議你到表單點選該物件，再到屬性欄中拷貝它的 Name 屬性來寫程式，就保證不會打錯字了！你要直接將它改名(Name 屬性)也是可以的。試試看！現在你可以任意改變筆色與粗細了！



7-5 畫直線

正版的小畫家可以畫直線、矩形或圓形等形狀，與塗鴉不同的是他們在拖曳過程中只會顯示一條線或一個形狀，不會一移動就一直畫，到最終放開滑鼠時就只有最後的一個圖案會留下！這幾個功能的核心技巧很相似，本單元只示範拉直線的作法，為了簡化操作介面就使用滑鼠「右鍵」表示拖曳直線，左鍵保持為之前的塗鴉之用。

[動態復原底圖影像]

這個程式技巧的關鍵在於當使用者按下滑鼠時要記住當時的畫面(存為一個影像物件)，每次滑鼠移動時都會再次拷貝這個影像(沒有直線的畫面)為繪圖目標，並從滑鼠壓下的那一

點畫一條直線到目前滑鼠位置，這樣每一個 MouseMove 動作都只會畫出一條線的影像，因為下次移動時又是從沒有這條線的原來影像(沒有直線的畫面)重新複製來繪製。這樣一來，每一次移動滑鼠的小移動就要進行一次圖像轉移，好像讓電腦很操勞，會不會因此產生停格呢？早期的電腦程式是有可能，現在電腦影像管理效能好多了，何況螢幕畫面本來就每秒更新數十次，放心吧！絕對沒事的！

包括塗鴉在內的繪圖動作程式現階段請修改如下：

```
int x0, y0;
Bitmap bg;
private void pictureBox1_MouseDown(object  sender, MouseEventArgs e)
{
    x0 = e.X;
    y0 = e.Y;
    bg = new Bitmap(pictureBox1.Image); //記憶未畫線影像
}

private void pictureBox1_MouseMove(object  sender, MouseEventArgs e)
{
    switch (e.Button)
    {
        case MouseButtons.Left:
            Graphics g = Graphics.FromImage(pictureBox1.Image);
            Pen p = new Pen(colorDialog1.Color,  int.Parse(toolStripComboBox1.Text));
            g.DrawLine(p, x0, y0, e.X, e.Y);
            x0 = e.X;
            y0 = e.Y;
            pictureBox1.Refresh();
            break;
        case MouseButtons.Right:
            Bitmap bf = new Bitmap(bg);
            Graphics gg = Graphics.FromImage(bf);
            Pen pp = new Pen(colorDialog1.Color,  int.Parse(toolStripComboBox1.Text));
            gg.DrawLine(pp,  x0, y0, e.X, e.Y);
            pictureBox1.Image = bf;
            break;
    }
}
```

上述畫直線程式先宣告一個 bg 的影像物件記錄滑鼠剛剛壓下時的畫面，接著進到滑鼠移動的程式，如果按的是左鍵就執行之前的塗鴉程式；如果是右鍵就先建立一個 bg 的副本 bf，在 bf 上面畫一條直線，從滑鼠壓下時的位置(x0, y0)到目前滑鼠位置(e.X, e.Y)，然後貼到 pictureBox1.Image 上面。如果滑鼠鍵被放開時此線就固定下來成為影像的永久成員了，如果按住繼續移動又會從原先 bg 的狀態重畫直線，這就是用拖曳方式畫直線的密技了！

[Graphics 物件的其他功能]

如果你想如正版小畫家一樣，畫出方塊或橢圓需要改變的只是將 DrawLine 的那一部分指令，改成 DrawRectangle 就是矩形；DrawEllipse 就是橢圓。如果要畫成實心的方或圓形就改成 FillRectangle 或 FillEllipse 即可，甚至有個 FillPie 的指令可以畫出扇形的形狀。這些就留給同學作為課後的進階挑戰囉！

7-6 使用工具提示顯示顏色值

[ToolTop 物件的使用]

在此介紹一個工具箱「通用控制項」分類中的 ToolTip 物件，它的直接翻譯就是「工具提示」，請先將一個 toolTip1 加入設計表單之中。相信所有用過電腦的人都會知道滑鼠移到某個物件上面時，常常會有個小小的黃色文字方塊「溫柔」的浮現出來，告訴你這個東西是幹嘛用的等等資訊，那就是工具提示了！請先到工具箱的「一般控制項」中找出它加入表單。請注意！工具提示不是猛跳出來，必須用滑鼠點選關閉的那種視窗哦！那個東西叫做訊息框(MessageBox)。

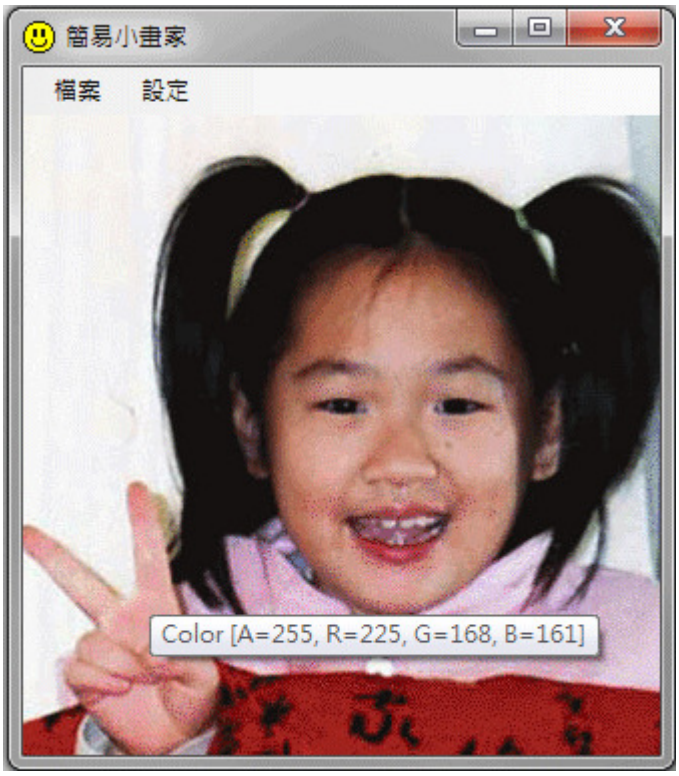
[數位影像的基本結構→像素點→RGB 三原色]

在此我們希望使用工具提示顯示滑鼠所在位置的顏色資訊。典型數位影像的結構是每個像素點都是由 RGB，也就是紅綠藍三原色以不同的亮度值組成，每種色光亮度的值介與 0~255 之間。要取得一個像素點的顏色資訊是使用 GetPixel 的指令將某一像素點從 Image 物件中擷取出來，當然必須指定是影像上 X 與 Y 座標分別為多少的那一點。

我們想將顯示程式寫在 MouseMove 事件中，之前此事件副程式已經用於畫直線及塗鴉，它們分別已佔用了滑鼠的右與左鍵，這裡就將顯示顏色資訊的程序寫在「未按鍵」(None)的狀況，程式碼如下：

```
case MouseButtons.None:
    bg = new Bitmap(pictureBox1.Image);
    Color C = bg.GetPixel(e.X, e.Y);
    toolTip1.SetToolTip(pictureBox1, C.ToString());
    break;
```

首先宣告 bg 與目前的 pictureBox1.Image 相同，接著在 bg 內滑鼠所在的位置取出一個像素點 C，所謂的像素點(Pixel)在數位影像中的實質意義就是一個點的顏色資訊，所以資料型態宣告為 Color。最後呼叫 toolTip1 設定工具提示(SetToolTip)於 pictureBox1 物件之內，提示內容為該處像素點(C)的內容(ToString)。試試看執行畫面應該如下：



[透明度也是顏色資訊]

在此我們有點偷懶的直接將顏色變數轉成字串(C.ToString())，你會發現轉出來的除了 RGB 值之外還有一個 A 值，這是與顏色「透明度」相關的值，255 表示完全不透明，0 就是完全透明了！事實上也可以使用 C.R 取得紅色，C.G 取得綠色，C.B 取得藍色等語法來操作這些資訊。

[使用 GetPixel 的意義]

GetPixel 的使用其實是非常重要的一個里程碑，這表示我們知道如何取得影像中任何一個像素點的詳細資訊，事實上你還可以用相對的 SetPixel 函數將顏色改變之後再「放」回影像裡面去！有了它們，任何進階的影像處理程式都可以依據這個邏輯來寫了！

7-7、進階挑戰

一、如何模仿畫直線的方式繪製出方塊或圓圈？
提示：使用繪圖物件 Graphics 的 DrawRectangle, DrawEllipse, Fill…等功能。

二、如何製作調整畫紙大小的功能？
提示：可用記事本單元的 GroupBox 小視窗方式輸入影像的寬高值。

三、如何製作復原功能？
提示：使用多個背景影像物件陣列記住每個步驟的影像畫面，需要復原時回貼即可。

課後閱讀

[程式繁簡與執行速度的兩難]

相信多數同學學到本單元一定會被 Bitmap，Graphics 甚至 Pens，Brush 等等東西弄得很煩。畫幾條線而已，怎麼弄得那麼複雜啊？確實如此，在「美好」的 VB6 時代，畫一條線真的只要一個指令：**Picture1.Line(X1,Y1)-(X2,Y2)** 就搞定了！連我都好懷念。但是簡單的 VB6 繪圖語法執行速度卻非常的慢，也因此當年要寫影像程式時，多數專家都不會用 VB，而是選擇 C++語言，當然現在的 C#執行速度一定比 VB6 甚至舊版的 C++更快的！只是語法並未隨之簡化，仍比 VB6 複雜許多，想寫影像程式這些基本功還是必須練好的！就讓我們對這幾個名詞深入了解一下吧？

一、Bitmap、Graphics 與影像容器

上一單元課後閱讀介紹過 Bitmap 物件，它是被繪圖的目標，如同一張圖畫紙，或者一張相片！Graphics 呢？它好像是個繪圖機、畫家或者說繪圖的「代理人」，宣告它時必須提到它是依附於(用來畫)哪一個 Bitmap 物件的。也就是說如果沒有指定的 Bitmap，Graphics 其實是不能單獨的存在！下行程式就是說明 g 物件依附於 picturBox1.Image 這個影像。

```
Graphics g = Graphics.FromImage(pictureBox1.Image);
```

接下來目標影像物件(Bitmap)被繪圖的動作都必須指定這個 Graphics(g)去執行，但是操作個別像素點的 GetPixel 與 SetPixel 指令例外，它們可以直接讀寫像素點(Pixel)，也就是 Bitmap 的內容，不必用到 Graphics。Graphics 物件必然的會將所有的繪圖動作畫到它依附的那個 Bitmap 物件之上。但是煩人的小陷阱是這個畫好的 Bitmap 物件還是只存在電腦的記憶體裡面，如果沒清楚地交代要顯示在哪一個影像「容器」(如 pictureBox1)裡面，或者忘了『更新』(Refresh)容器內的影像內容，使用者還是看不到結果的！下面的程式碼就是在作這些事情：

```
pictureBox1.Image = new Bitmap(800, 600);

pictureBox1.Refresh();
```

上面的 pictureBox1 就是影像的「容器」，最典型的容器當然是 PictureBox，其他還有表單的 BackgroundImage，Label 物件也有 Image 屬性，凡是可以載入影像(有 Image 或 BackgroundImage 屬性)的物件都可以當影像容器。總之，請記得影像物件畫好後一定要放到某個容器內，而且有時候影像用程式改變之後還得寫個指令(Refresh)更新一下才行！

二、Pen 與 Brush

Pen 是筆，Brush 是刷子！要畫出線條狀的東西(Draw)得用筆，要將一個區塊塗滿(Fill)一種顏色就要用刷子了！這就是基本的使用邏輯。因此如果繪圖物件要 Draw 一個方塊，表示是畫一個方塊形狀的邊框，就應該用 Pen，如同本單元的內容，Pen 可以自己宣告定義筆色與筆寬，但是 C#程式軟體預設就有許多現成的筆可以用，它們的名字是這樣的：

Pens.Black，Pens.Red…等等。因此畫個黑框矩形的程式就會是這樣：

g.**Draw**Rectangle(Pens.Black, x1, y1, width, height);

要畫個實心的(Fill)黑方塊呢？程式就是：

g.**Fill**Rectangle(Brushes.Black, x1, y1, width, height)

如果想自己定義筆刷的顏色呢？很奇怪的！必須宣告的物件不是 Brush 或 Brushes，而是「**SolidBrush**」！所以畫一個實心黃色矩形的程式如下：

SolidBrush B = new SolidBrush(Color.Yellow);

g.**Fill**Rectangle(B, x1, y1, width, height);

確實有點麻煩，但是懂得以上幾個東西的基本意義與用法之後，你以後要寫繪圖程式時應該就不會覺得摸不著頭緒了！