第2章　整合發展環境初步與簡易 C#程式

2.1 整合發展環境初步與第一支 C#程式

整合發展環境(Integrated Development Environment，簡稱 IDE)將程式編寫(editing)、建置(building)、偵錯 (debugging)、執行 (execution)等各種工具合在一起，便利程式開發工作。本書所要介紹的 C#物件導向程式設計概念，可以使用各種C#語言整合發展環境。為說明方便，全書使用微軟公司的 Visual C# Express 2008 為例；但大部份的內容與所採用的整合發展環境關係不大。讀者如果使用其他 IDE，應該也可以找到對應的功能使用。

Visual C# Express 2008 可於http://www.microsoft.com/express/vcsharp/ 下載。同網址還提供一些教育參考資源，如Introduction to Visual C# 2008 Video（C#簡介影片）、Beginner Developer Learning Center (初學者學習中心)、Visual C# team's blogs (Visual C#開發團隊部落格)、Visual C# Express Forum (論壇)、MSDN (Microsoft Development Network Library，由各種微軟公司程式語言的詳細說明與整合發展環境使用方法介紹構成的知識庫)等，可以依需求瀏覽或下載。

下載 Visual C# Express 2008 後，執行檔案，在各個對話盒中選擇預先設定(default)，再依指示重新開機完成安裝。

自 Windows 系統「開始」選單之「程式集」中啟動 Visual C# Express2008，或者在 C:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE 中找到 VCSExpress 檔，設定捷徑到桌面，由桌面捷徑啟動。啟動後之畫面應與圖 2-1 類似。

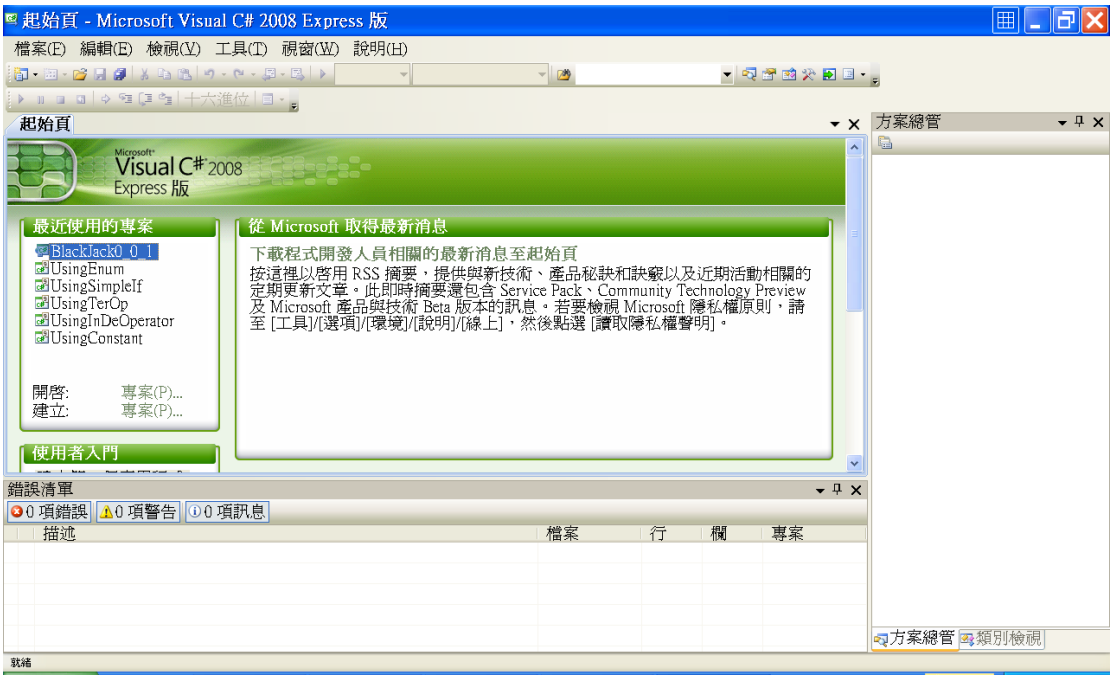

圖 2-1. Visual C# Express2008 起始畫面

設計一套程式以解決某一特定問題，稱為建立一個「專案」 (project)。集合幾個專案可以完成多個功能，稱為「方案」(solution)。從起始畫面的功能表「檔案」選項，選擇「新增」→「專案...」，系統隨即出現圖 2-2 的對話盒，供我們選擇所要開發的應用程式種類。對話盒中雖列有多種應用程式範本，本書將只介紹較基礎的「主控台應用程式」(console application programs)、「類別庫」(class library)及「Windows Form 應用程式」(Windows Form application programs)三種。物件導向的概念以「主控台應用程式」說明較為單純，而且「Windows Form 應用程式」需要相當清楚的物件導向觀念，所以要到第 10 章，已介紹完物件導向觀念後，才說明其設計方式。「類別庫」的用法安排於第 5 章。

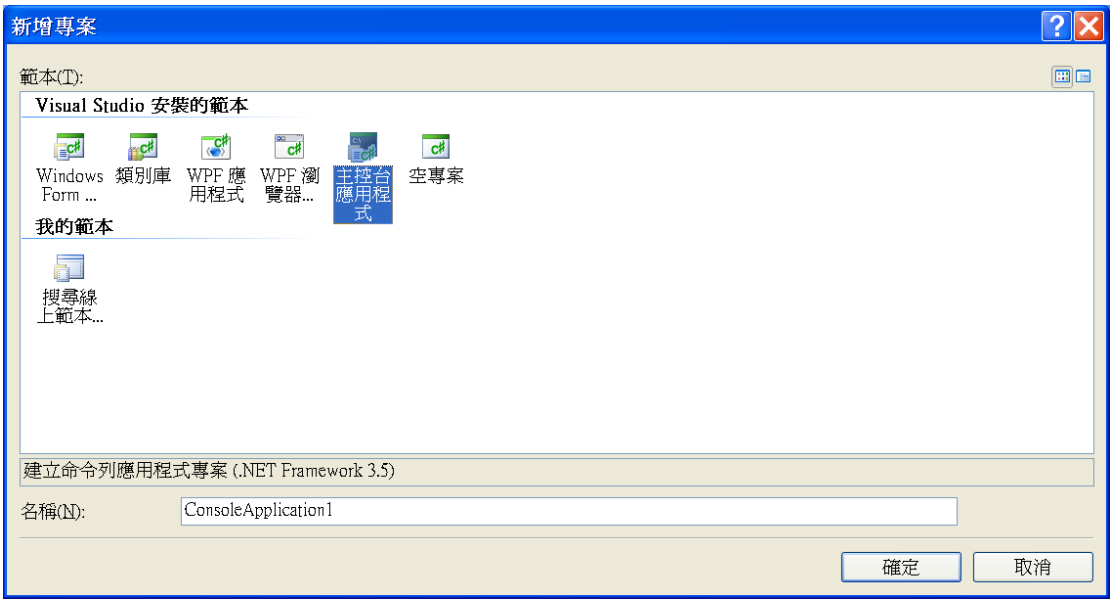

圖 2-2. 應用程式種類選擇畫面

我們的第一個 C#程式是一個主控台應用程式，所以選擇「主控台應用程式」，隨後在「名稱」右方的文字盒中，將預設的「ConsoleApplication1」改為「Hello」。按下「確定」按鈕後，進入圖 2-3 的工作畫面。由標題列可看到專案的名稱：Hello。畫面中標籤 Program.cs 下方的區域，便是程式碼的編寫區域，編寫方式與一般文書處理軟體如 Word 相同。系統在程式碼編寫區域已經自動產生了主控台應用程式的骨幹程式原始碼(source code)，供修改應用。左下方是訊息顯示區域，目前大部份空白。右方為方案總管，顯示方案中各專案的成員。目前方案總管顯示方案名稱亦為 Hello，其下只有一個專案：Hello。專案 Hello 之下列有三個項目：Properties、參考、Program.cs。Properties 和參考內含專案組成所需的資訊，於 Properties 和參考左方的「+」號按下滑鼠左鍵，即可看到其成員。這些資訊於本書中大多不需修改，所以暫時可以忽略。Program.cs 目前為 Hello 專案的唯一程式碼成員。相對於 Properties 和參考左方的「+」號，於程式碼區域中也有四個「−」號。倘若於「−」號按下滑鼠左鍵，若干程式碼即隱藏起來，或者說被摺疊起來，只剩原先在「−」號旁的程式敘述，且對應的「−」號也變成了「+」號。這個功能可以適時隱藏或展開程式細節，方便閱讀或搜尋特定程式單元。方案名稱如欲修改，可於方案名稱上按滑鼠右鍵，在跳出之選單中選擇「重新命名」，再直接修改方案名稱。

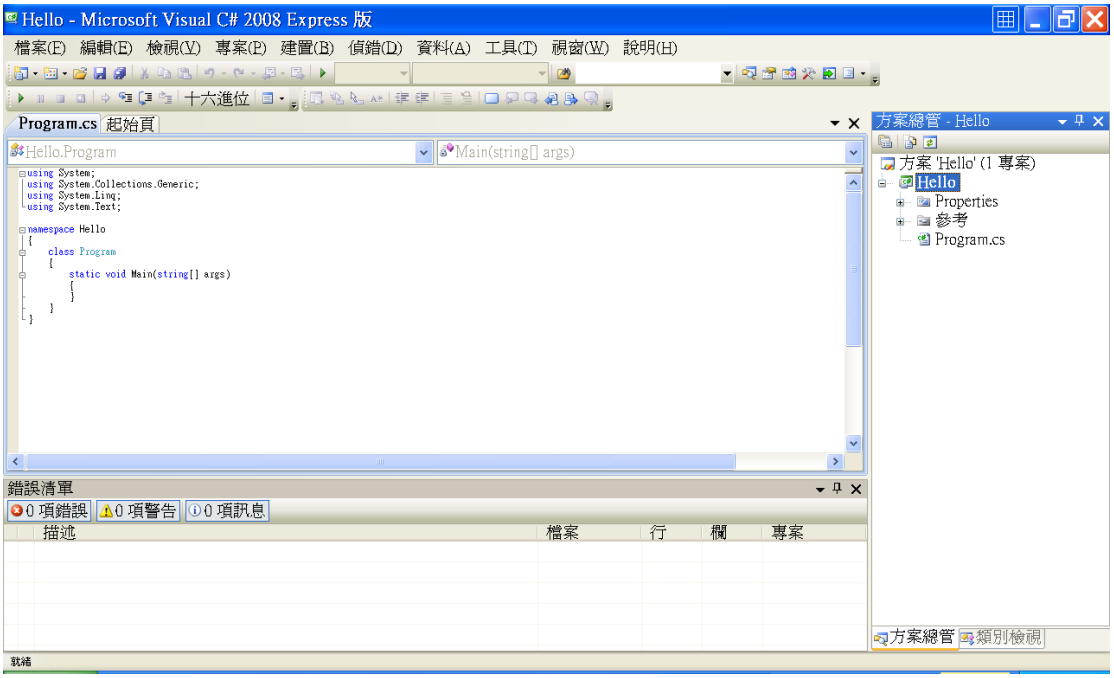

圖 2-3. 整合發展環境工作畫面

接著在程式碼工作區的骨幹程式碼內鍵入一行敘述(statement)：

```
Console.WriteLine("Hello");
```

使全部程式如圖 2-4 所示，便完成我們的第一支程式。程式內容於 2.3 節解說後當可了解，目前只需知道它執行後，將在螢幕顯示 Hello 字樣即可。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Hello
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello");
        }
    }
}
```

圖 2-4. 第一支程式

接下來檢查輸入的程式，注意字母大小寫不可弄錯，也不要拼錯字或漏掉大、小括弧與雙引號。確定無誤後，選擇功能表中的「建置」→「建置方案」。一切順利的話，訊息顯示區域應維持原狀，錯誤清單應顯示「0 項錯誤」、「0 項警告」、「0 項訊息」，而「描述」標題下應仍舊空白，僅有狀態列顯示由「就緒」變成「建置成功」。如果並非如此，請重新檢查程式，改正錯誤後，重新建置。

建置完成後，系統已將 C#原始碼(source code)翻譯為共同中介語言(Common Intermediate Language，CIL)，產生 CIL 的中間檔案以及完成連結其他程式庫的執行檔。選擇功能表中的「偵錯」→「啟動但不偵錯」，電腦即可直接解譯執行檔並執行，得到如圖 2-5 的主控台視窗輸出結果，驗證程式正確。按電腦鍵盤上的任一鍵，或滑鼠移至主控台視窗右上方「x」號處按左鍵，均可結束程式執行，主控台視窗消失。

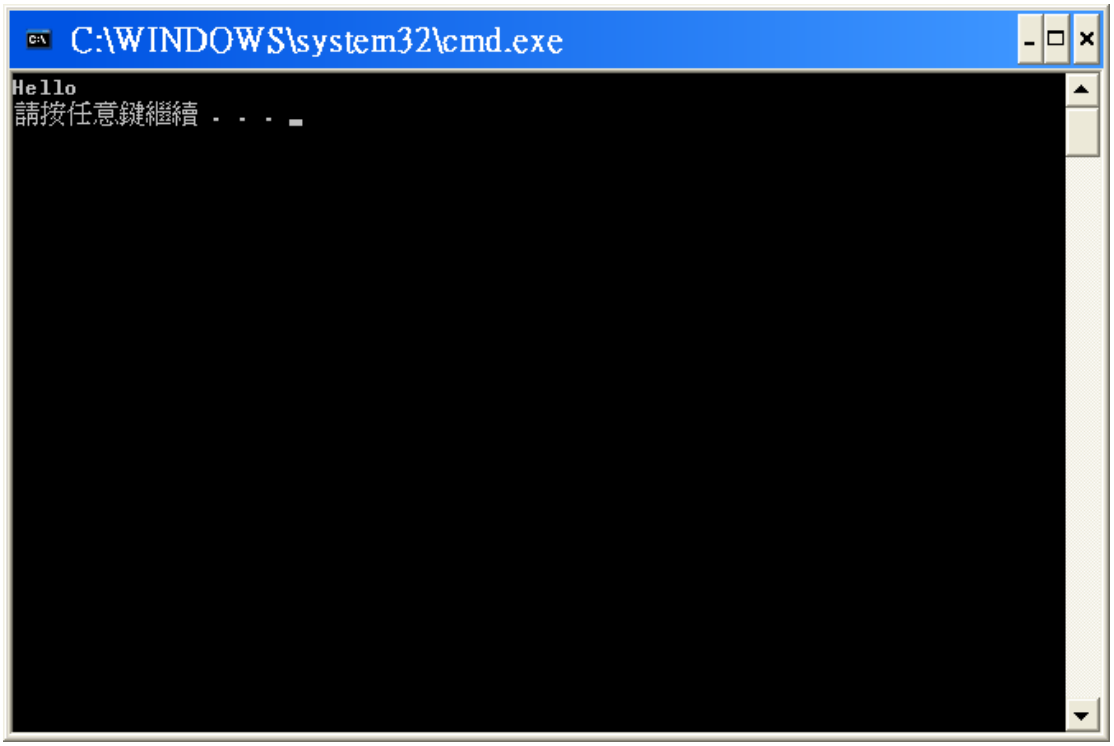

圖 2-5. 第一支程式的執行結果

完成並驗證程式後，選擇主功能表「檔案」→「全部儲存」，出現圖 2-6 的對話盒，接受預設，或利用「瀏覽...」按鈕選擇專案儲存位置，再按「儲存」按鈕，完成專案的儲存。如果要更改預設的專案儲存位置，可以選擇主功能表「工具」→「選項...」，得到類似圖 2-7 的選項對話盒。在對話盒左方的樹狀結構展開「專案和方案」，選取「一般」。對話盒右方如圖 2-7 所示，於「Visual Studio 專案位置」項下，按下對應的「...」按鈕，即可改變預設的專案儲存位置。假使專案要存在一個新的目錄檔案夾下，便得要先在 Windows 作業系統下產生新目錄檔案夾後，再回頭更改預設的專案儲存位置。

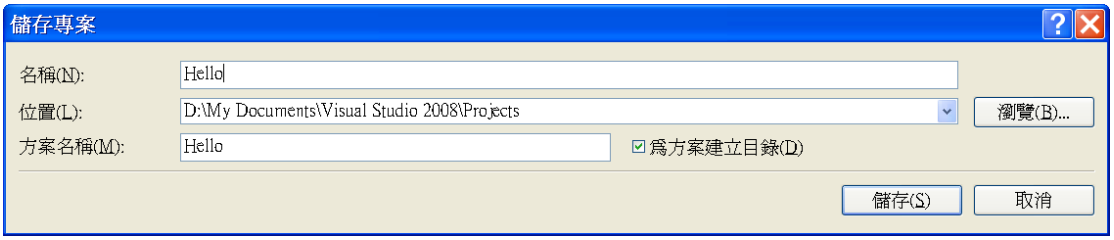

圖 2-6. 儲存專案對話盒

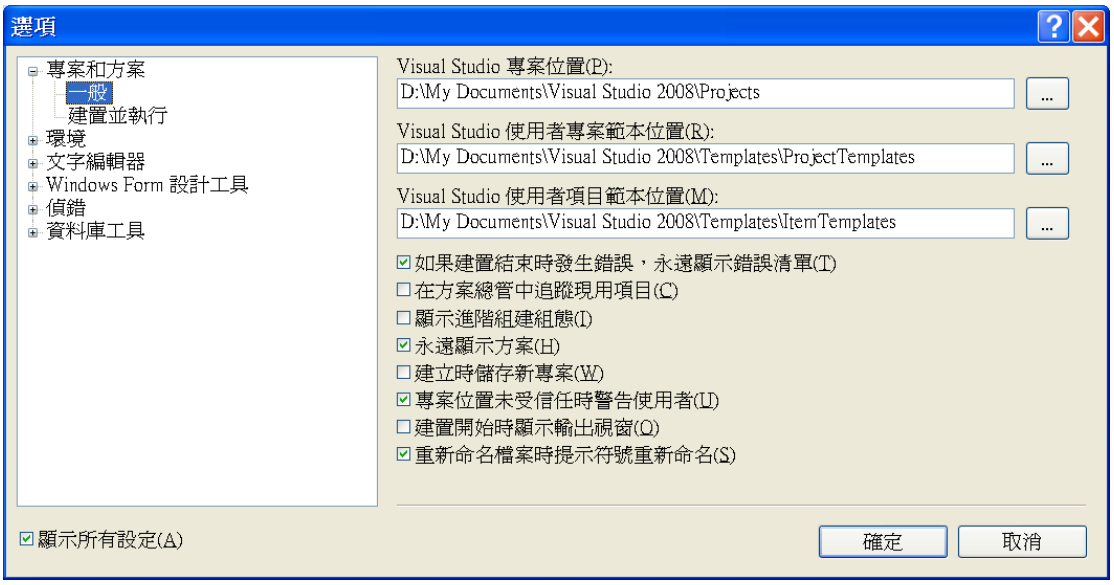

圖 2-7. 選項對話盒

存好專案與方案後，可以到方案的所在，打開與方案同名的檔案夾。方案檔案夾中會有一個與專案同名的檔案夾、一個與方案同名的方案檔(Microsoft Visual Studio Solution)、一個與方案同名的使用者設定檔(Visual Studio Solution User Options，可能是隱藏檔，我們一般用不到)。打開與專案同名的檔案夾，裏面有與專案同名的專案檔(Visual C# Project File)、與專案同名，副檔名為 csproj 的專案使用者設定檔(Visual Studio Project User Options File)、專案所包含的若干程式原始碼檔案(副檔名為 cs，例如 Program.cs)，以及三個檔案夾。三個檔案夾 bin、obj、Properties 分別主要存放執行檔、中間檔、以及專案的 Properties。打開 bin 檔案夾，可看到 Debug 與 Release 兩個檔案夾，分別存放偵錯器使用版本與正式發行版本的執行檔。正式發行的版本，是建置時進行最佳化(optimization)的結果。有可能在不影響程式執行結果的前提下，已經更動了程式敘述的順序，以加快執行的速度。偵錯器的版本則完全依照原始碼建置，不做任何更動，以方便逐步執行程式原始碼除錯。本書將在 2.4 節介紹偵錯器的使用，

2.2 第二支 C#程式與錯誤訊息

完成第一支 C#程式後，嘗試建立另一個主控台應用程式新方案與專案，命名為 SayHello。將系統產生的骨幹程式修改為如圖 2-8 所示之程式原始碼。建置之後執行，所得主控台視窗應與圖 2-5 相同。

```
/*
 * 第2支C#程式
 * 2/6/2009
 */

using System;

namespace SayHello
{
    class Program
    {
        static void Main(string[] args)
        {
            string message; // 宣告變數
            message = "Hello";
            Console.WriteLine(message);
        }
    }
}
```

圖 2-8. 第二支程式原始碼

我們將利用第二支程式，說明如何處理錯誤訊息。首先把 **Console** 字首的大寫 **C** 改為小寫，使成為 **console**。重新建置後，會發現錯誤訊息如圖 2-9，顯示 SayHello 專案中，程式第 16 行，第 13 個字元欄位處，發生「名稱'console'不存在於目前內容中」的錯誤(error，俗稱 bug)。這表示系統不認識 **console** 這個名稱，應該檢查是否拼錯字、漏了宣告或 **using** 敘述(參看 2.3 節)。注意看程式碼編寫區，可以發現 **console** 字下產生波狀底線，指出此處發生錯誤。改正錯誤後，選擇主功能表的「建置」→「重建方案」，重新建置，執行後應該可以得到正確的結果。

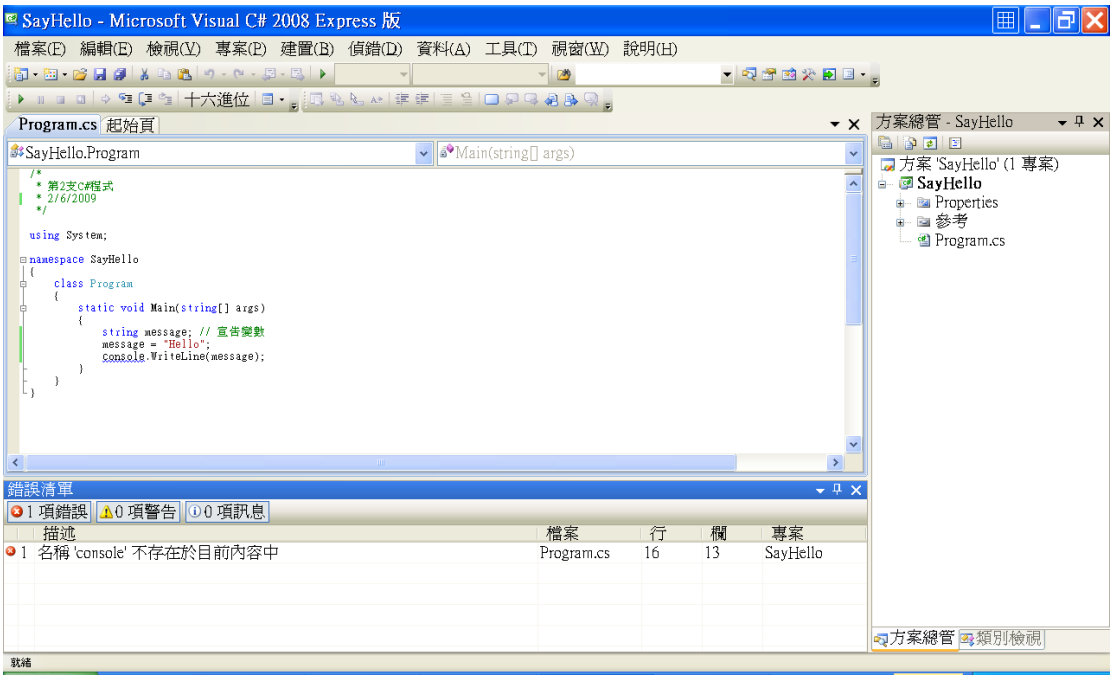

圖 2-9. 有錯誤訊息的畫面

若是程式較長，或者錯誤訊息有多起，可能沒那麼容易找到錯誤所在；如果能有程式碼行號顯示，就會方便許多。接下來就介紹顯示程式碼行號的方法：選擇主功能表「工具」→「選項...」，勾選下方「顯示所有設定」。在對話盒左方的樹狀結構展開「文字編輯器」，選取「C#」，勾選對話盒右方「顯示」字樣下的「行號」選項，再按「確定」按鈕離開，便可發現程式碼編寫區左方已出現行號。加上行號的程式原始碼如圖 2-10。

```
1  /*
2   * 第2支C#程式
3   * 2/6/2009
4   */
5
6  using System;
7
8  namespace SayHello
9  {
10     class Program
11     {
12         static void Main(string[] args)
13         {
14             string message; // 宣告變數
15             message = "Hello";
16             Console.WriteLine(message);
17         }
18     }
19 }
```

圖 2-10. 加上行號的第二支程式原始碼

其他常犯的錯誤如同一字大小寫前後不統一，漏掉分號「;」，大括弧「{」與「}」、雙引號「"」、「/*」與「/*」沒有配對等，在編寫程式時便會出現訊息，同時程式碼錯誤的地方變成紅字，提醒使用者注意改正。

2.3 程式大略解說

本節大略解說第一、二支程式內容，以使讀者了解 C#程式的基本語法。

從第二支程式開始：觀察 2.2 節專案 SayHello 的程式碼編寫區，可發現以符號「/*」與「/*」夾起來的文字，以及符號「//」之後的同一行文字都是綠色。它們代表程式設計者所添加的註解，以方便程式閱讀者了解程式的內容。在建置程式時，系統會自動跳過註解，因此不影響程式的執行；但是，經過一兩週後，程式設計者可能已忘記當初的想法，難以使用或修改程式。因此為自己及夥伴方便著想，應該儘量添加註解。通常程式檔案開始的地方，應該利用「/*」與「/*」，在其間記載程式目的、修改日期、作者、參考資料名稱等資訊；而在單行較複雜難懂的程式敘述後，則使用「//」加上程式敘述的說明。一般而言，程式註解著重在表明「為什麼」程式要這麼寫，而不在於重複程式敘述已經表現的內容。例如，第 14 行程式敘述

```
string message; // 宣告變數
```

的註解「宣告變數」，不是一個好的註解：因為在「//」前的同一行程式敘述便是在宣告變數(稍後說明)，所以此一註解並沒有增進我們對程式敘述的了解。「//」的效力只有一行，所以一行寫不完某一程式敘述的說明時，可以連續幾行都以「//」開始添加註解，並讓這幾行的「//」對齊。例如：

```
string message; // 宣告變數
                // 上一行註解不是好的註解
```

此外，有時也會在一行程式敘述的開頭加上「//」符號，或者在一段程式敘述前後加上「/*」與「/*」符號，把這一行或這一段程式暫時變成註解，在執行時跳過，以方便偵錯，算是註解符號的另一種用途。

其餘部份的文字呈現深藍色、淺藍色、咖啡色、與黑色，便是電腦可執行的程式碼。深藍色的字在 C#程式語言中有特別含意，稱為關鍵字(key word)或保留字(reserved word)，淺藍色的字是類別(class)或變數型別(type)，咖啡色的字是資

料字串，黑色的字是物件(object)、自定變數(variable)、常數(constant)、函式(function)、命名空間(namespace)的名稱或數值。以上提到的術語，稍後便會解釋。

由於 C#是純粹的物件導向(object-oriented)程式語言，我們必須先介紹物件(object)和類別(class)的觀念，才比較容易理解程式碼的涵義。所謂物件，在計算機科學中所指的是具有狀態資料與操作功能的抽象單元。例如，讀者家中的電視可以抽象地表示成一個具備頻道、音量等狀態資料與選台、調整音量等功能的單元。此一抽象單元可以一定程度地模擬讀者家中真正的電視，因此稱為一個電視物件。在這同時，世界上還有許多電視；每一部電視都可以對應到一個電視物件，各自保有其頻道、音量等狀態資料與選台、調整音量等功能。然而，所有電視物件整體來看，必有頻道、音量等狀態資料與選台、調整音量等功能。所以我們可以設定一個電視物件的藍圖，規定電視物件必須具備頻道、音量等狀態資料與選台、調整音量等操作功能。這個藍圖便稱為類別。根據電視類別的描述，設定不同形式之狀態資料與操作功能，便可產生不同電視物件。例如，電視物件甲只有三個無線電視頻道，音量最大只能到 40 分貝；電視物件乙有 256 個有線頻道，音量最大可調整到 50 分貝。電視物件甲與電視物件乙為不同的物件，但都是根據相同的電視類別描述(具備頻道、音量等狀態資料與選台、調整音量等操作功能)所產生。物件導向程式語言便是一種語法，讓我們可以方便地描述類別，而寫成的類別描述稱為物件導向程式。電腦執行物件導向程式時，可據以產生物件，操作其功能，變化物件狀態，進而解決我們的問題。由於物件概念很容易和真實與虛擬世界的事物對應，因此物件導向程式特別適合用以模擬真實與虛擬世界中的事物，例如圖書管理系統、線上購物網站、或角色扮演遊戲。

電腦程式執行時，需要儲存一些資料，也需要有一些方法來改變這些資料的內容，因此每一個要執行的電腦程式都可以看成一個物件。也因此我們需要撰寫一個類別來描述要電腦執行的程式物件的內容。這一類別此處叫作 **Program**，以保留字 **class** 加上類別名稱 **Program**，再加上一對大括弧「**{**」與「**}**」進行宣告。圖 2-10 程式碼的第 10 至第 18 行便是類別 **Program** 的宣告，其狀態資料與操作功能需於對應的大括弧之間描述。類別 **Program** 必備的操作功能便是告訴電腦如何執行工作，這一功能在圖 2-10 的第 12 至第 17 行描述，稱為主函式(main function)，寫法固定為

```
static void Main(string[] args)
{
    /*
        . . .
    */
}
```

每一專案只能有一個主函式。專案程式開始執行時，電腦會找到主函式，依序執行主函式的大括弧之間，以分號「**;**」分開的程式敘述。所以主函式也可稱為專案程式的進入點(entry point)。關鍵字 **static** 和 **void** 的意義留到第 5 章解說；主函式名稱 **Main** 之後，小括弧內的引數(argument)宣告「**string[] args**」通常用不到，說明從略。注意主函式的大括弧包含在類別 **Program** 的大括弧中，而類別 **Program** 的大括弧又包含在 **namespace SayHello** 的大括弧中，形成「大圈圈裏頭有個小圈圈，小圈圈裏頭還有個黃圈圈」的套疊式巢狀結構(nested structure)，要注意之間的對應，不能錯亂。為幫助我們分清每一個左右大括弧對應的另一半，程式碼編輯器自動將每一對大括弧獨立成行，而且前面加入相同數量的空白，稱為縮排(indentation)。同一對大括弧內的程式敘述，也維持相同的縮排，如圖 2-10 程式中的第 14 至 16 行。縮排是很好的程式設計習慣，幫助我們快速看清程式架構，避免錯誤。附帶一提，有的書喜歡使用另一種縮排方式，以圖 2-10 的主函式為例，寫為

```
static void Main(string[] args) {
   string message; // 宣告變數
   message = "Hello";
   Console.WriteLine(message);
}
```

把左大括弧移到上一行，可以節省一行的空間。

接著說明程式敘述：圖 2-10 第 14、15 行

```
string message; // 宣告變數
message = "Hello";
```

宣告一個字串變數 **message**，並且設定其值為 **Hello**。此處所說的變數，意義與數學上的變數相似，代表一個可以改變其值的符號。所以同一程式中，隨後可以改變變數 **message** 的值，比如，在宣告設值後，以

```
message = "Fine, thank you";
```

使變數 **message** 改代表字串 **Fine, thank you**。注意符號「**=**」並不是代表數學中的「相等」意義，而是代表「設值」(assignment)。宣告與設值的涵義進一步說明如下：

一般電腦以 word 為記憶與計算處理的單位，每個 word 又由 16、32、64 或更多位元構成。在電腦記憶體中，這些 word 可以想成排為一列，每一個 word 給定一個位址(address)。每一個 C#語言中的變數代表一個或數個 word 形成的記憶體範圍。記憶體範圍內儲存的 0 與 1 樣式，便代表此一變數的值。不同的 0 與 1 樣式可以代表各式各樣的資料，其解讀由程式設計者決定。所以，在程式使用一個變數前，要先宣告，以便 IDE 建置程式時，畫定某個記憶體範圍給此一變數，並自動把變數名稱轉譯為記憶體區域的位址，且規定好記憶體範圍內的資料解讀方式。圖 2-10 第 14 行宣告因此可解釋為在記憶體設定一塊區域，命名為 **message**，並將以字串解讀其內容；而第 15 行的設值敘述則代表在此一區域填入 H、e、l、l、o 六個字元，如圖 2-11 所示。

假若將圖 2-10 第 14 行變數宣告敘述之前加上「**//**」符號，使其變為註解。重新建置後，將發現兩個「名稱'message'不存在於目前內容中」的錯誤訊息，分別發生在剩下的兩行程式敘述，並可看到程式碼內兩處 **message** 名稱下都出現波狀底線。這是因為變數 **message** 未經宣告，表示計算機未分配記憶區域給它，也不知如何解讀其中的 0 與 1 樣式，使程式無法建置。去掉第 14 行程式前的「**//**」符號，使之恢復為程式敘述，但改將第 15 行的設值敘述變成註解，則第 16 行的 **message** 名稱下出現波狀底線，訊息區並有「使用未指定的區域變數'message'」的錯誤訊息。此時系統雖有分配記憶區域給變數 **message**，也知將以字串解讀其內容，但是沒有設定其初值(initial value)，記憶區域內可能是系統前次留下的 0 與 1 樣式，其值並不一定，因此第 16 行敘述顯示變數 **message** 內容時，將會在主控台視窗出現亂碼。有的 IDE，如 Visual C# Express 2008，則會在分配記憶區域時，將變數內容設為空字串符號「**null**」(通常就是把所有位元設為 0)，主控台視窗不會顯示任何字樣。為避免此種初值錯誤，IDE 在建置時即將其挑出並警示。以上變數未宣告及未設初值兩種錯誤於程式設計中最為常見，因此如果可能，儘量將第 14、15 行敘述合併，寫為

```
string message = "Hello"; // 示範宣告變數並設初值
```

同時完成宣告與設定初值。

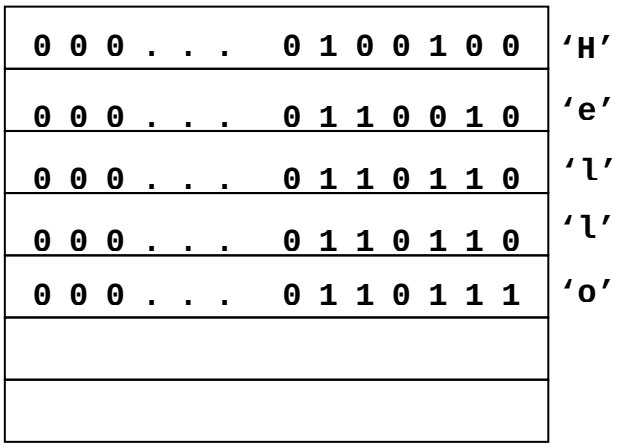

圖 2-11. 計算機記憶體示意圖

其次說明圖 2-10 的第 16 行敘述

```
Console.WriteLine(message);
```

此處的 **Console** 為系統提供的類別，代表主控台，即類似圖 2-5 的螢幕視窗(輸出)或鍵盤(輸入)。**WriteLine** 為類別 **Console** 內建的靜態(static)函式(function)。C#中的函式觀念與數學中的函數符號相似，例如數學中的 $\sin(x)$ 和 $\sin(\pi/2)$ 分別代表進行變數 x 和常數 $\pi/2$ 的正弦函數值計算，而 C#中的

```
Console.WriteLine(message);
```

」

與

```
Console.WriteLine("Hello");
```

則分別代表進行於主控台視窗顯示字串變數 **message** 內容與 Hello 字樣的操作。進一步的函式與靜態觀念，將在第 5 章說明。

往後的程式可能會包含許多類別，為方便整理，也為了可以重複使用之前別處用過的類別名稱，我們可以將一些類別包裝，放入某一名稱的命名空間(namespace)。在 Visual C# Express 2008 中，每一個專案都會自動產生一個同名的命名空間，在同一個專案中編寫的類別，通常就屬於這個與專案同名的命名空間。例如，圖 2-10 的類別 **Program** 位於第 8 行宣示的命名空間 **SayHello** 內，而 **SayHello** 正是圖 2-10 程式所屬的專案名稱。系統提供的一些常用類別，包括 **Console**，都包裝在命名空間 **System** 內。使用某個命名空間內的類別時，需要在程式碼之前加上 **using** 敘述。例如，圖 2-10 程式用到命名空間 **System**

的類別 **Console**，便加上

```
using System;
```

的敘述，如圖 2-10 程式中的第 6 行所示。如果將這一行敘述前面加上「**//**」符號，使其變為註解，重新建置後，將發現「名稱'Console'不存在於目前內容中」的錯誤訊息。這是因為系統只在目前的命名空間找尋 **Console** 名稱，所以發生這樣的錯誤。由是可知「**using System;**」敘述主要在提示系統，把命名空間 **System** 也納入搜尋範圍，這就可以解決問題。如果還是不寫「**using System;**」敘述，也可以把圖 2-10 的第 16 行改成

```
System.Console.WriteLine(message);
```

明確告訴系統使用命名空間 **System** 內宣告的類別 **Console**。

命名空間中可以再定義命名空間，形成巢狀結構。例如，圖 2-4 的第一支程式中，系統產生的骨幹程式碼有四個 **using** 敘述：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

其中的**Linq**和**Text**是命名空間**System**中的子命名空間，而**Generic**是命名空間**System**的子命名空間**Collections**內的子命名空間。顯然，這些命名空間的表示方法有如Windows作業系統下的階層式檔案夾系統，只是改用符號「**.**」來分別上下層的關係。注意**using**敘述效力只及同一層命名空間內的類別，不能觸及其子命名空間與上一層命名空間中的類別。所以命名空間**System**中看不到子命名空間**Linq**和**Text**內的類別，而命名空間**System.Collections**內也看不到其子命名空間**Generic**內宣告的類別。因此，骨幹程式碼的這四個**using**敘述必須完整寫出命名空間的完整路徑名稱。在圖 2-10的第二支程式碼中，用不到**Linq**、**Text**、**Generic**內的類別，所以將對應的**using**敘述刪去。其實第一支程式也用不到，但為減少讀者初次嘗試程式設計的困難，所以仍予保留。

假設某方案有甲、乙兩專案。如前所述，兩專案編寫的類別通常放在各自專案名稱的命名空間中。如果乙專案要使用甲專案內的類別，便需依照上述的原則，在程式碼中增添適當的**using**敘述。除此而外，IDE內的方案總管也要做一些設定，留待第5章說明。

2.4 偵錯器使用初步

以上各節所討論的拼錯字、大小寫顛倒、大小括弧未配對、變數未宣告、變數未設初值等錯誤，稱為語法錯誤(syntax error)，相當於自然語言中的文法錯誤，一般均可於建置時發現，並顯現錯誤訊息，通常不算是太大的困擾。但有另一種情形，其句法並無錯誤，也能順利建置成功，但程式執行之後，檢查其結果卻與預期不同。此種狀況，稱為語義錯誤(semantics error)或邏輯錯誤(logic error)，通常是由於程式設計者對程式的構想有誤，或對程式語法涵義的誤解，或輸入設定的資料不正確。一般而言，這一類的錯誤很難找出更正，最讓人頭痛。經驗顯示，大多數程式設計師可能只花大約 20%的時間寫新程式，卻以 80%的時間尋找更正程式中的語法與語義的錯誤，而且程式越大越複雜，偵錯(debug)時間比例愈高。

要找到語義錯誤，通常需要反覆閱讀程式碼，搭配以紙筆與計算器(calculator)模擬計算機執行程式過程(稱為程式追蹤，program tracing)，加上對問題與程式目的的了解，與一點運氣。另外，經驗顯示，向夥伴逐行解說程式碼(稱為程式複核，program walk-through)時，常常可以發現自己想法邏輯錯誤的所在，而夥伴的不同看法有時也有幫助。除此之外，IDE 內的偵錯器是一項相當有力的工具，它可以縮小錯誤所在的程式原始碼範圍，使錯誤突顯出來。首先，利用偵錯器確定相關變數或物件的內容在某一敘述執行前均為正確；其次，以偵錯器讓程式由該敘述開始，執行若干步後暫停；用偵錯器檢視變數或物件內容的變化，與預期內容(通常由紙筆與計算器的程式追蹤獲得)比較：如果有不同，就表示這幾步牽涉到的程式敘述中有語義錯誤產生，要仔細閱讀了解；或者回到前一正確敘述處，減少偵錯器執行的步數，使範圍更小，更容易發覺錯誤所在。以下便就圖 2-10 的第二支程式略加改動，說明 Visual C# Express 2008 偵錯器的使用。

由於第二支程式的邏輯想法相當簡單，不容易出錯，所以我們把第 15 行的資料字串改為**"Helo"**。重新建置專案並執行，可以發現主控台視窗顯示 Helo 字樣，而不是假設的 Hello，這構成了一個語義錯誤。雖然這個錯誤很容易找到，我們還是嘗試用偵錯器來幫忙尋找錯誤所在。

首先找到程式進入點(主函式對應的左大括弧「**{**」)後的第一個可執行敘述(變數宣告不算)，即第 15 行的設值敘述，在最左端的淺灰色直長條處按下左鍵，就會看到在第 15 行最左端淺灰色直長條處出現一個紅色圓形記號，對應敘述顯示在紅褐底色之方塊上，表示本行程式敘述執行前須暫停。重新建置後，選擇主功能表的選項「偵錯」→「開始偵錯」，程式便會執行到第 15 行之前，而紅色圓形記號中出現一個黃色箭頭，對應敘述也顯示在黃底色之方塊上，表示即將執行本行敘述。此時的畫面如圖 2-12，左下方的訊息區現在成了檢視變數內容的地

方。在「區域變數」標題下的區域顯示兩個變數：**args** 和 **message**，**args** 是主函式 **Main** 的引數，略去不討論；**message** 的值顯示為空字串 **null**，表示尚未設值。

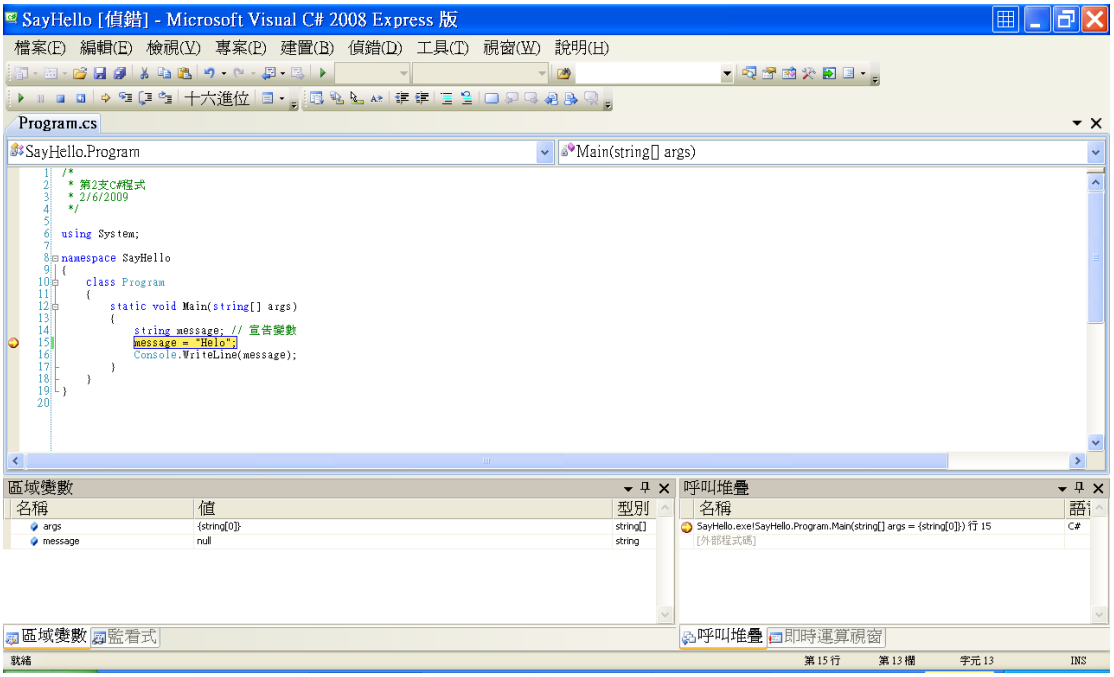

圖 2-12. 第二支程式偵錯畫面

接著在主功能表選擇「偵錯」→「逐步執行」，可看到黃色箭頭向下移動一行，來到第 16 行，顯示已執行第 15 行，將要執行第 16 行。在這同時，左下方的 **message** 變數內容變成"Helo"，而且字體為紅色，表示是由剛才執行的程式敘述所改變。假定我們因觀察變數內容發現了錯誤，便可於主功能表選擇「偵錯」→「停止偵錯」，離開偵錯模式。接著將程式碼中的資料字串改正，重新建置，逐步執行偵錯。這次應當發現 **message** 變數的內容正確。繼續逐步執行，直到黃色箭頭抵達主函式對應的右大括弧「**}**」，這時觀察主控台視窗，可看到 Hello 字樣，是剛才執行第 16 行敘述的正確結果。繼續逐步執行，可看到程式順利結束，主控台畫面消失。程式既已改正，中斷點便沒有必要存在。將滑鼠移到中斷點的紅色圓形記號，按下左鍵，紅色圓形記號消失，中斷點刪除。

有時程式設有多個中斷點，於某個中斷點執行一步，沒有發現問題，可以在偵錯器主功能表選擇「偵錯」→「繼續」，直接執行到下一個中斷點，不必逐步執行好幾次。偵錯時發現小錯誤，也可以當場修改，於主功能表選擇「偵錯」→「重新啟動」，重新偵錯。

偵錯器除了協助找出語義錯誤外，也可以幫忙我們了解程式語言。假設我們新學某種程式敘述，或者忘了某種程式敘述的功能時，可以寫一個包含此一程式

敘述的簡單程式，以偵錯器執行，並觀察有關變數的內容變化，當可深入明瞭此一程式敘述的效用。有一本書的作者提到：他教程式設計時先教偵錯器的使用，結果他班上學生的期中、期末考成績總比其他班高出一截，甚至被懷疑洩題。這就是因為他的學生會自行利用偵錯器增進對教學內容的了解，所以學習成效提高。本書之後的討論，也會儘量利用偵錯器說明較困難的程式敘述。

2.5 基本輸入與輸出

由第二支程式已知可用 **Console.WriteLine**，使電腦顯示一行訊息。其次必須能讓電腦讀取我們由鍵盤輸入的資料，以便依照鍵入的資料，產生不同的回應。圖 2-13 的程式示範混合使用輸出與輸入敘述，使電腦可以取得使用者的姓名，並稱呼使用者姓名來問候。

```
1  /*
2   * 示範基本輸入, 輸出敘述
3   * 2/12/2009
4   */
5
6  using System;
7
8  namespace SayHelloIO
9  {
10     class Program
11     {
12         static void Main(string[] args)
13         {
14             Console.WriteLine("Enter your name");
15             string name = Console.ReadLine();
16             Console.WriteLine("Hello," + name);
17         }
18     }
19 }
```

圖 2-13. 基本輸入與輸出之示範程式

以下為圖 2-13 程式的解說：第 14 行

```
Console.WriteLine("Enter your name");
```

在主控台視窗顯示 Enter your name 字樣，提醒使用者鍵入姓名，通常稱為「提示」(prompt)。完成這一行顯示後，主控台視窗的游標便移到下一行。由此可知 **WriteLine** 其實是「寫一行」(write a line)的意思，寫完就換行。接下來的第 15

行敘述

```
string name = Console.ReadLine();
```

表示由鍵盤讀取字元，並將讀得的字元依次組成一個資料字串，存入變數 **name** 之中。使用者每按一個鍵，主控台視窗便顯示對應的字元，直到使用者按下 Enter 鍵換行為止，所以 **ReadLine** 代表「讀一行」(read a line)。最後，第 16 行

```
Console.WriteLine("Hello," + name);
```

執行後，主控台視窗便在 Hello,字樣後，出現使用者剛才輸入的姓名字串。注意這裏的「**+**」號，代表將前後字串連結成新字串(concatenation)，並不是相加的意思。

到此，讀者可以建立專案 SayHelloIO，將圖 2-13 程式鍵入，並予建置、執行。確認程式執行無誤後，將方案儲存。讀者接著可以修改這個程式，讓電腦提示問題，於使用者鍵入單行答案後，讓電腦複誦(稱為 echo)或作出適當的回應。讀者可重複若干次提問與回答，讓電腦與使用者對話。

完整經過偵錯的程式可以作為往後的參考，也可用來衍生新程式，並作為比對與偵錯新程式的依據，是極為可貴的資產。因此修改程式以解決新問題時，最好不要動到已經能解決某一問題的專案。此時可以將舊程式碼複製到新專案再修改，方法如下：新增一個專案，利用主功能表「檔案」→「開啟」→「檔案...」，開啟專案 SayHelloIO 中的 Program.cs，此時程式編寫區會出現兩個 Program.cs 的標籤，並且顯現舊專案 SayHelloIO 的 Program.cs 程式碼，如圖 2-14 所示。以主功能表「編輯」→「全選」，選取所有程式碼；再以「編輯」→「複製」複製到剪貼簿；關閉舊 Program.cs 檔；以滑鼠選取所有新 Program.cs 的骨幹程式碼；選擇「編輯」→「貼上」，可看到新專案的 Program.cs 中，系統產生的程式碼已由圖 2-13 的程式碼取代。讀者接下來就可以依新需求改寫程式，而仍能保持舊專案 SayHelloIO 的完整。

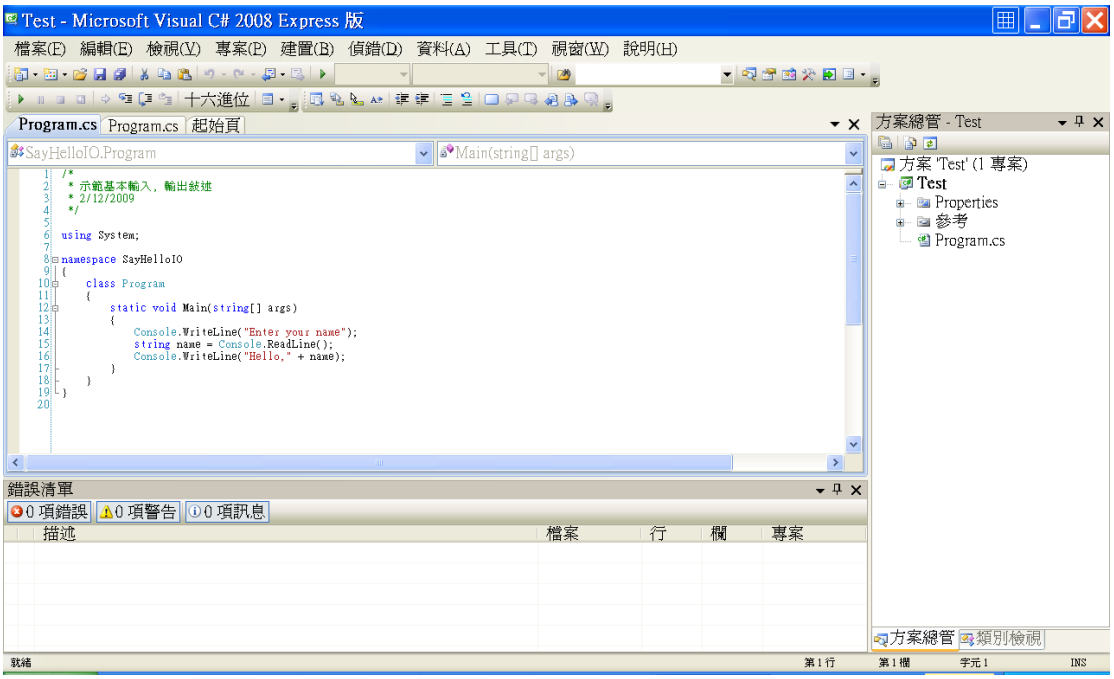

圖 2-14. 於新專案開啟舊專案 SayHelloIO 的 Program.cs 檔，出現兩個 Program.cs 標籤

2.6 剪貼視窗內容與列印程式碼

完成某階段的程式開發後，常須撰寫報告或留下書面記錄。此時可能需要列印程式碼，或擷取執行時出現的某一視窗(例如主控台視窗)圖像。

先說明列印程式碼的方法：選取主功能表「檔案」→列印，選好印表機、列印範圍、份數、列印內容(勾選是否隱藏摺疊區域、是否包含行號)，再按「確定」按鈕即可印出目前程式編寫區的程式碼。

其次解說擷取視窗圖像的辦法：首先將游標移進目標視窗內，同時按下鍵盤上的 Ctrl、Alt、Prt Sc 三個按鍵，視窗圖像便已複製到剪貼簿中。再打開文書處理軟體或 Power Point，將剪貼簿內容貼上，隨後再以圖片工具裁剪出所需部份，調整其大小即可。

主控台視窗預設為黑底白字，列印時會耗費大量碳粉或黑色墨水，不經濟也不環保。此時可在擷取視窗前，於主控台視窗範圍內，按滑鼠右鍵，選擇彈出功能表的「全選」選項，即可看到主控台視窗內的輸出輸入部份已經反白，其他部份仍為黑色。接著於視窗左上角第一個字元處按下滑鼠左鍵，主控台視窗又變回黑底白字，僅有左上角第一個字元處為白底黑字之小方塊。由左上角第一個字元開始，按住滑鼠左鍵，向右下方拖曳，直到整個主控台視窗成為白底黑字，如圖 2-15，就得到所需的主控台視窗圖像了。另一種比較一勞永逸的辦法如下：在主控台視窗的藍色標題框架內，按滑鼠右鍵，選擇「預設值」，即出現如圖 2-16

的對話盒。將選定色彩值的紅、綠、藍均調為 255，再選擇左方「螢幕文字」，將選定色彩值的紅、綠、藍均調為 0。此時的主控台視窗仍為黑底白字，要把它關掉，重新執行一次程式，就得到白底黑字的結果了。以後執行程式得到的主控台視窗便都是白底黑字，在整個電腦螢幕上較不明顯，有些人可能較不習慣。這時可能用前一種方法較好，僅在需要時產生反白的效果。讀者可依偏好選用適當方式。

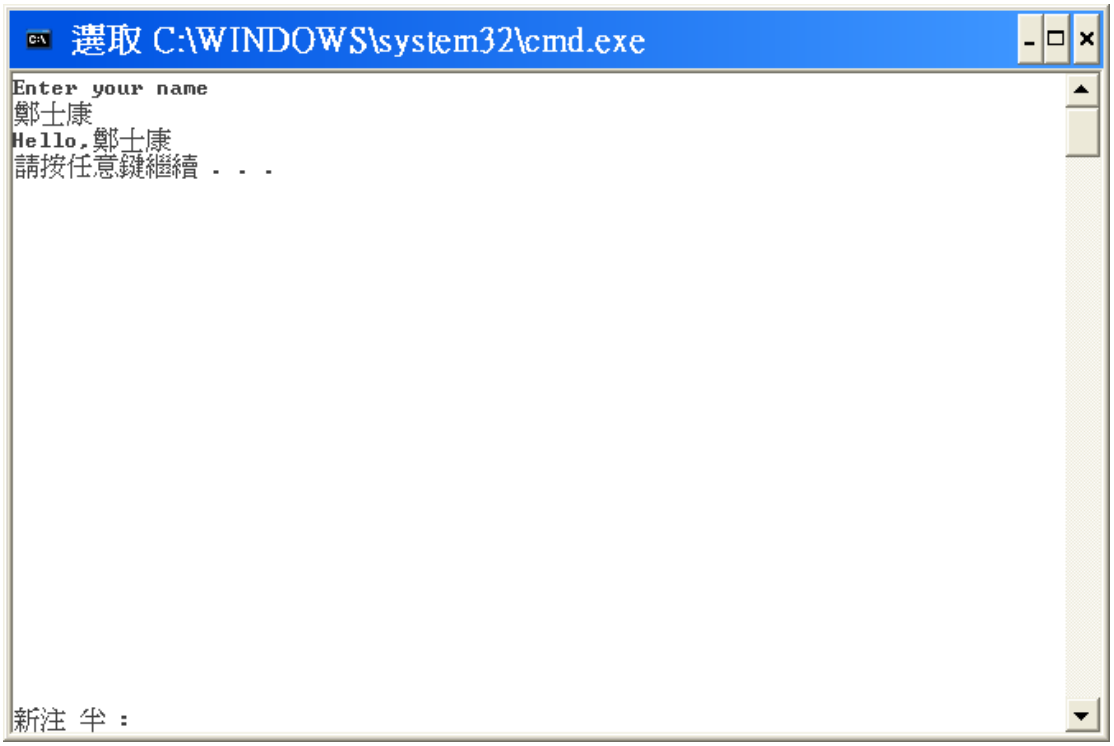

圖 2-15 白底黑字的主控台視窗

圖 2-16 主控台預設值對話盒

2.7 本章複習

1. 什麼是整合發展環境(IDE)?
2. 專案與方案有什麼分別?
3. 說明註解的用途與撰寫時機。
4. 舉例說明物件與類別的觀念。
5. 寫出主控台程式的主函式架構。
6. 說明變數宣告與設值時，電腦對應的處理。
7. 舉例說明命名空間的用途與宣告方式。
8. 語法錯誤與語義錯誤有何不同? 何者較棘手?
9. 有那些方法可以幫助程式設計師找出語義錯誤?
10. 大略說明偵錯器的用途與用法。
11. 已能解決某一問題的專案為什麼要儘量不去更動?
12. 如何讓主控台視窗內容變成白底黑字，並擷取其圖像?