第四章陣列

- 4.1 陣列
- 4-2 規則與不規則陣列
- 4.3 結構與結構陣列
- 4.4 方法
- 4.5 方法的使用
- 4.6 引數的傳遞方式
- 4.7 如何在方法間傳遞陣列
- 4.8 方法多載
- 4.9 區塊變數、區域變數、靜態變數與類別欄位

4.1 陣列 (Array)

4.1.1 陣列的宣告與建立

- 陣列是屬資料結構的串列。由多個相同資料型別的變數在記憶體中前後連續串接組成。
- 陣列使用時機
  是在需處理多個同性質資料時，以陣列中的陣列元素來取代同性質的變數，即將每個陣列元素視為一個變數來處理。
- 陣列元素是由陣列名稱其後跟著以中括號括住陣列註標值或稱索引值組成，來表示該陣列元素在陣列中是第幾個資料，陣列註標值由0開始算起。

4.1 陣列 (Array) Continue …

- score[0] 為 score 陣列的第一個陣列元素，
  score[2] 為第三個陣列元素，其他以此類推...。
- 在程式中可將每個陣列元素視為一個變數來處理。
- 陣列屬參考型別，宣告一個陣列
  ⇨ 編譯器會在記憶體中配置記憶體給此陣列名稱使用
  ⇨ 建立陣列時編譯器會依宣告的資料型別和陣列大小
  由陣列起始位址配置一塊連續記憶體空間給此陣列
  所有陣列元素使用，同時將陣列的起始位址存到陣列
  名稱的記憶體裡面
- 陣列名稱裡面存放不是資料本身，而是陣列的起始位址。

4.1 陣列 (Array)

Continue …

語法

宣告陣列語法：

```
資料型別[] 陣列名稱 ;
```

建立陣列語法：

```
陣列名稱 = new 資料型別[陣列大小];
```

語法

```
資料型別[] 陣列名稱 = new 資料型別 [陣列大小] ;
```

例 宣告並建立 score 為一個含有五個陣列元素的整數陣列，寫法：

```
int [] score;
score = new int [5];
```

合併成一行

```
int [] score = new int [5];
```

下面是各資料型別陣列的宣告和建立方式，其中 score 為陣列名稱，陣列元素有 score[0]、score[1]、score[2]、score[3]、score[4] 共 5 個陣列元素：

```
int[] score = new int[5];
long[] score = new long[5];
double[] score = new double[5];
string[] score = new string[5];
object[] score = new object[5];
```

4.1 陣列 (Array)

Continue …

陣列在使用前必須先宣告，賦予該陣列名稱、資料型別和陣列大小，接著使用 new 關鍵字建立陣列的物件實體，程式編譯時才能決定在記憶體中應配置多少個連續記憶空間給此陣列使用。C# 建立陣列的語法如下：

語法

1. 一維陣列：資料型別[] 陣列名稱 = new 資料型別[陣列大小];
2. 二維陣列：資料型別[,] 陣列名稱 = new 資料型別[第一維陣列大小,第二維陣列大小];

例 建立陣列名稱為 score，含有 score[0,0] ~ score[3,4] 共有 20 個陣列元素的整數陣列：

```
int [,] score;

score = new int [4,5];
```

合併成一行

```
int [,] score = new int [4,5];
```

4.1.2 陣列的初值設定

- 陣列經宣告和建立物件實體後
  ⇨ 必須賦予陣列元素初值才能存取陣列。
- 陣列元素相當於一個變數
  ⇨ 可用指定運算子(=)直接指定陣列元素初值。
- 宣告並建立 score整數陣列實體後
  再逐一指定 score[0]~ score[4] 五個陣列元素初值

```
int[] score = new int[5] ;
score[0]=50; score[1]=70 ; score[2]=65 ; score[3]=99 ;score[4]=78 ;
```

4.1.2 陣列的初值設定 Continue...

- C# 允許將宣告並建立陣列和設定陣列初值兩行敘述合併成一行。語法：

語法

```
資料型別[] 陣列名稱 = new 資料型別[] {陣列初值串列};
```

1. 建立 a 是一個含有 5 個整數的陣列，其中初值依序設為：
   a[0]=1 ； a[1]=2 ； a[2]=3 ； a[3]=4 ； a[4]=5 ；

4.1.2 陣列的初值設定 Continue…

2. 建立一個 a[3,4] 的二維整數陣列，其初值分別為：

a[0,0]=10； a[0,1]=20 ； a[0,2]=30 ； a[0,3]=40 ；

a[1,0]= 5 ； a[1,1]=15 ； a[1,2]=25 ； a[1,3]=35 ；

a[2,0]=12； a[2,1]=24 ； a[2,2]=36 ； a[2,3]=48 ；

寫法：

```
int [,] a = new int[]  {{10,20,30,40}, {5,15,25,35}, {12,24,36,48}};
```

4.1.3 陣列常用的屬性與方法

- 在 System.Array 類別提供建立、管理、搜尋和排序陣列的屬性和方法

```
int[] a1 = new int[3] ;
int[,] a2 = new int[3,4];
```

陣列物件的成員	功能
Length 屬性	取得指定陣列所有陣列元素的總數，傳回值為整數。例如： ① a1.Length ⇨ 傳回 3 ② a2.Length ⇨ 傳回 12
Rank 屬性	取得指定陣列維度的數目，傳回值為整數。一維陣列傳回 1，二維陣列傳回 2，依此類推。例如： ① a1.Rank ⇨ 傳回 1 ② a2.Rank ⇨ 傳回 2
IsReadOnly 屬性	傳回指定陣列是否唯讀，傳回值為布林值。例如： a1. IsReadOnly ⇨ 傳回 false

4.1.3 陣列常用的屬性與方法 Continue...

陣列物件的成員	功能
GetUpperBound 方法	取得指定陣列中某一維度上限，傳回值為整數。例如： ① a1.GetUpperBound(0) ⇦ 取 a1 第 1 維上限, 傳回 2 ② a2.GetUpperBound(1) ⇦ 取 a2 第 2 維上限，傳回 3
GetLowerBound 方法	取得指定陣列中某一維度的下限，傳回值為整數。陣列維度的下限是由 0 開始。
GetLength 方法	取得指定陣列中某一維度的陣列元素總數，傳回值為整數。例如： ① a1.GetLength(0) ⇦ 取 a1 第 1 維陣列元素總數，傳回 3 ② a2.GetLength(1) ⇦ 取 a2 第 2 維陣列元素總數，傳回 4
CreateInstance 方法	建立指定 陣列資料型別和大小的一維陣列(具有以零起始的索引)。範例於第 8 章介紹。 ① 產生一個含有五個陣列元素的 a1 整數陣列 Array a1 = Array.CreateInstance(typeof(Int32), 5); ② 產生一個 名稱為 a2 的 2x3 字串二維陣列 Array a2 = Array.CreateInstance(typeof(String), 2, 3);

4.1.3 陣列常用的屬性與方法 Continue...

陣列物件的成員	功能

SetValue 方法	設定一維、二維 … 陣列中陣列元素的值。
	① 設定一維陣列第 3 個陣列元素的值為 10。
	a1.SetValue(10, 2);
	② 設定二維陣列第 1 列第 2 欄陣列元素的值 10。
	a2.SetValue(10, 0, 1);

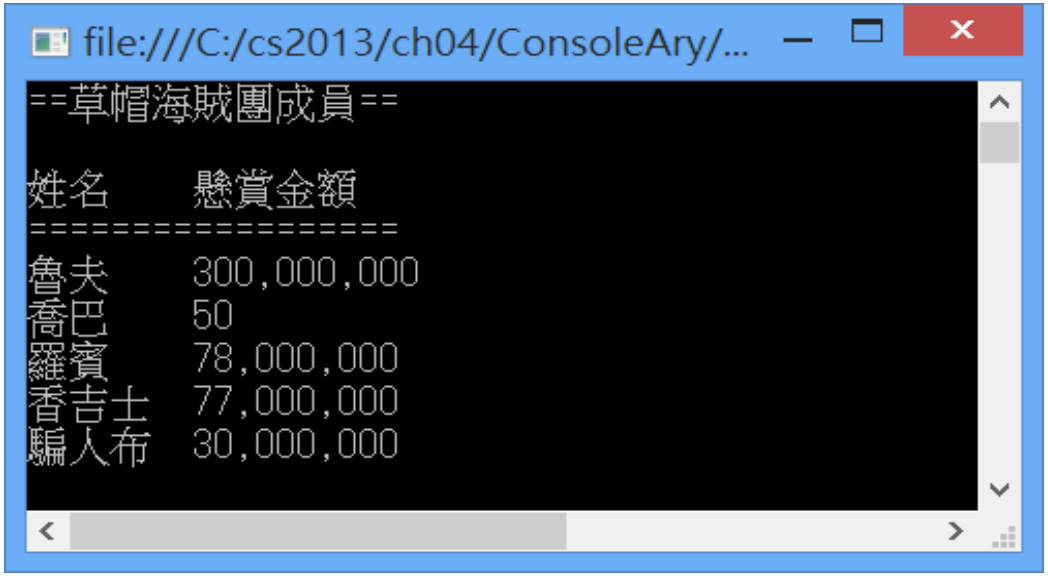
GetValue 方法	取得一維、二維 …陣列中位於指定位置的值。
	① 取得一維陣列第 3 個陣列元素的值。
	a1.GetValue(2);
	② 取得二維陣列第 1 列第 2 欄陣列元素的值。
	a2.GetValue(0, 1);

範例：ConsoleAry.sln

建立 RoleName 字串陣列存放卡通航海王的角色姓名，建立 Money 整數陣列存放各角色的懸賞金額。兩個陣列建立如下：

```
string[] RoleName = new string[] {"魯夫", "喬巴", "羅賓", "香吉士", "騙人布"};
int[] Money = new int[] {300000000, 50, 78000000, 77000000, 30000000};
```

將 RoleName 角色姓名及 Money 懸賞金額的各陣列元素依下圖顯示出來。



4.1.4 foreach … 敘述

- foreach 不用給予迴圈正確的初值、條件和終值。
- 自動將指定集合物件(如陣列)中的元素逐一指定給變數，代入迴圈內處理，一直到所有元素都處理完畢才離開迴圈，繼續執行接在 foreach… 敘述後面的敘述。

語法

```
foreach (資料型別 變數 in 集合物件 )
{
    [敘述區段]
    [break;]
    [敘述區段]
}
```

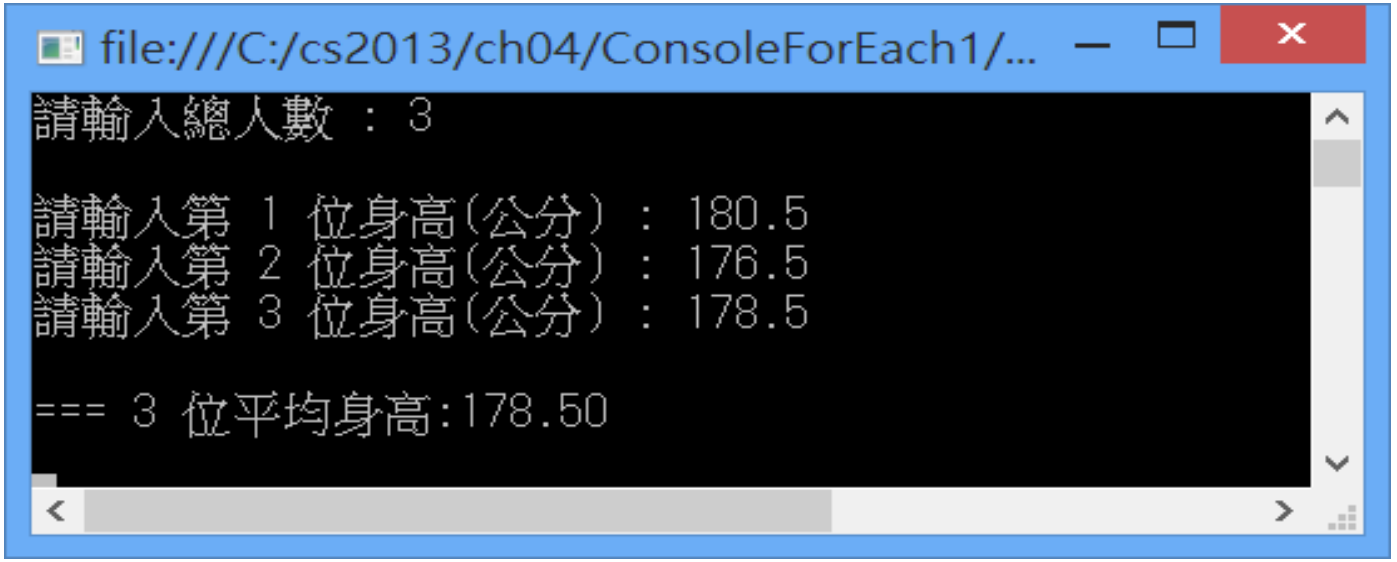
4.1.4 foreach … 敘述 Continue…

譬如：將 tall 陣列中所有的身高相加後的總和置入 sum 變數中，使用 foreach 寫法如下：

```
int[] tall = new int[] {10, 20, 30, 40, 50};
int sum = 0;
foreach (int height in tall)
{
    sum += height ;
}
```

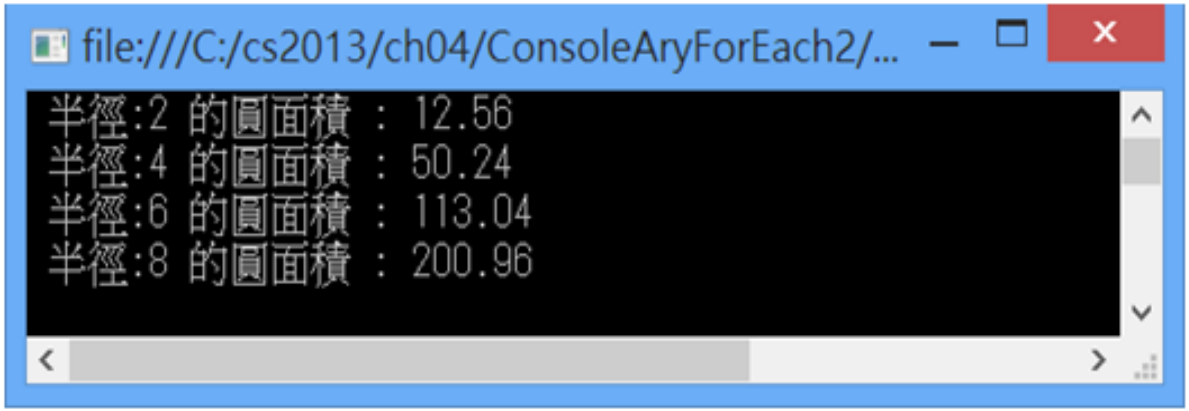
範例：ConsoleForEach1.sln

由鍵盤輸入總人數，再使用 for…迴圈由鍵盤依序輸入每個人的身高分別放入陣列，身高的單位為公分，身高允許有小數。再透過 foreach…敘述將陣列中每個元素的內容放入 height 變數，再累加到 sum 變數，最後依下圖格式顯示每位的身高和平均身高。



範例 ：ConsoleAryForEach2.sln

設計一個整數陣列，陣列元素依序為 2, 4, 6, 8 當圓之半徑。試以 Array.Each 方法，依序對取出的陣列元素執行 ShowArea()自定方法，將計算出的面積依下圖輸出顯示：



4.1.5 陣列的排序與搜尋

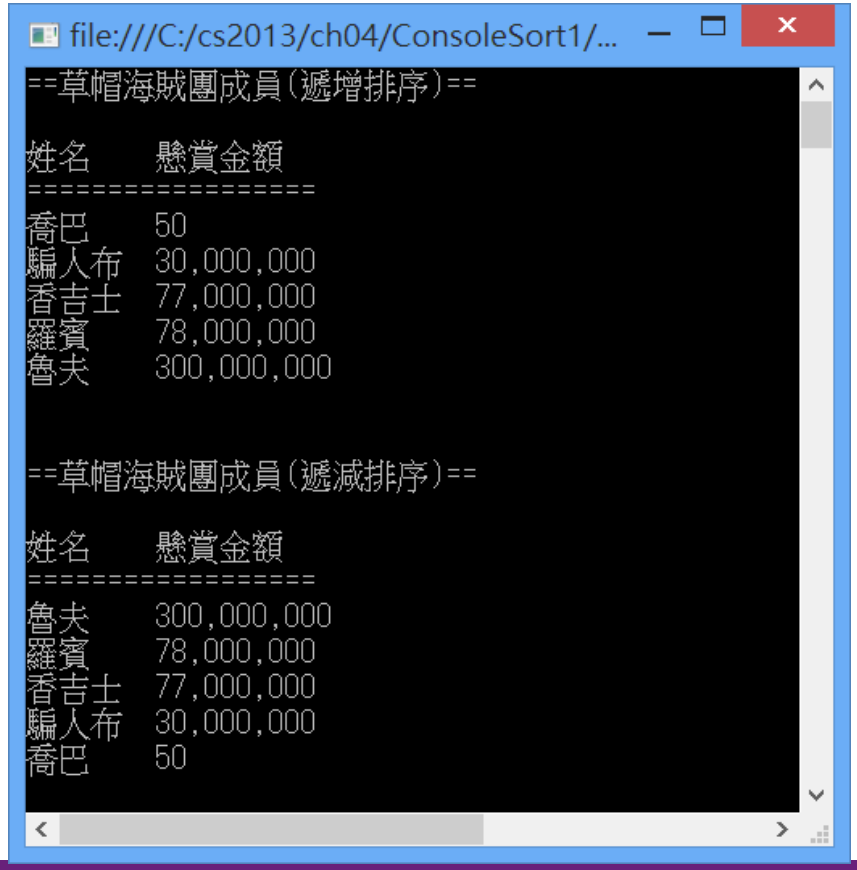
Array 類別靜態方法	功能
BinarySearch	[語法] int n=Array.BinarySearch(陣列名稱, 欲搜尋資料); [說明] 透過二分搜尋法搜尋資料是否存放於陣列中，使用此方法前必須事先使用 Array.Sort() 方法將陣列做遞增(由小到大)排序才可使用，適用於搜尋資料量較大的陣列，若找到資料傳回該陣列元素索引值，若沒找到傳回 -1。
Clear	[語法] Array.Clear(陣列名稱, 起始註標, 刪除個數); [說明] 將陣列中指定範圍內陣列元素內的內容清除。
IndexOf	[語法] int n=Array.IndexOf(陣列名稱, 欲搜尋資料); [說明] 搜尋資料是否存放於陣列中。若有找到資料會傳回該陣列元素索引值；若沒有找到資料則傳回-1。

4.1.5 陣列的排序與搜尋

Array 類別靜態方法	功能
Sort 方法	[語法 1] Array.Sort(陣列名稱); [說明 1] 對指定的一維陣列物件遞增(由小到大)排序。 [語法 2] Array.Sort(陣列名稱 1, 陣列名稱 2); [說明 2] 依據第一個引數陣列的索引值由小到大來排序，第二引數陣列的對應索引值亦跟著移動。 [語法 3] Array.Sort(陣列名稱 1, 起始註標，排序長度); [說明 3] 將指定一維陣列由指定起始陣列註標開始取指定長度的陣列元素進行由小而大遞增排序。 [譬如] 排序 score 陣列時，stu_name 陣列也會跟著更動。 Array.Sort(score, stu_name);
Reverse 方法	[語法 1] Array.Reverse(陣列名稱); [語法 2] Array.Reverse(陣列名稱,起始註標,排序長度); [說明] 將指定的一維陣列整個或由指定註標後面多少個陣列元素進行反轉。若陣列想遞減(由大到小)排序，可先執行 Array.Sort()方法進行遞增排序後，再執行 Array.Reverse()方法反轉陣列，則該陣列變成遞減排序。
GetEnumerator 方法	傳回 Array 的 IEnumerator。此方法於第八章介紹。

範例：ConsoleSort1.sln

修改 ConsoleAry.sln 專案，依 Money 陣列所存放的各角色懸賞金額做遞增和遞減排序，排序同時 RoleName 陣列所存放的角色姓名亦同時更動，結果如下圖所示。



4.2 規則與不規則陣列

4.2.1 規則陣列

- 規則陣列或稱矩陣陣列屬於二維陣列
- 規則陣列是指每列的陣列元素都相同。
- 語法：

語法

```
二維陣列：資料型別[,] 陣列名稱=new 資料型別[第一維陣列大小,第二維陣列大小];
```

4.2 規則與不規則陣列 Continue…

1. 利用下列任一方法即可在單一敘述中同時建立、設定和初始化多維陣列：

```
int[,] ary = new int [2,3] { {1,2,3}, {4,5,6} };
int[,] ary = new int [,]  { {1,2,3}, {4,5,6} };
int[,] ary = { {1,2,3}, {4,5,6} };
```

2. 存取單一陣列元素寫法 ： ary[1,1] = 5;

3. 以雙層迴圈來依序存取陣列元素先做初始化再讀取陣列元素

```
int[ , ] ary = new int[2,3];
for(int i =0; i < 2 ; i++)
    for(int  j = 0 ;  j < 3 ;  j++) {
        ary[ i , j] = 0;
        Console.Write("{0} ", ary[i, j]);
    }
```

4.2.2 不規則陣列 (Jagged Array)

- 或稱非矩形陣列。
- 指陣列中每列的長度不相同稱為 不規則陣列。
- 不規則陣列就是陣列中還可存放陣列，每列長度不相同，可用來建立每列不同長度的表格。
- 語法：

語法

```
資料型別[][] 陣列名稱 = new 資料型別[陣列大小][];
```

4.2.2 不規則陣列 (Jagged Array)

1. 建立不規則陣列的方式如下：

   ① 先宣告一個不規則陣列。譬如宣告一個名稱為 myjag 的不規則陣列，此不規則陣列含有 3 列，寫法：

```
int[][] myjag=new int[3][];
```

   ② 接著分別設定不規則陣列每列陣列的大小，例如將第 0、1、2 列分別配置 3、2、4 個陣列元素，寫法：

```
myjag[0] = new int[3];
myjag[1] = new int[2];
myjag[2] = new int[4];
```

經過上面的宣告所建立的不規則陣列如下：

myjag[0][0]	myjag[0][1]	myjag[0][2]	
myjag[1][0]	myjag[1][1]		
myjag[2][0]	myjag[2][1]	myjag[2][2]	myjag[2][3]

3. 如何設定和存取不規則陣列的元素，其寫法如下：

```
myjag[1][1] = 5；
```

要注意存取不規則陣列元素的寫法 myjag[1][1] 和一般二維(矩形)陣列的寫法 myjag[1,1] 是不相同的。

4. 如何取得不規則陣列第 i 列的長度，其寫法如下：

```
int num = myjag[i].Length;
```

範例：ConsoleAmount.sln

某公司台北、台中、高雄三個營業處各分處銷售金額(單位：仟元)如下表。各分公司的營業處個數不盡相同，使用不規則陣列求各分公司營業總額及營業比率。

分處 / 營業處	第一處	第二處	第三處	第四處
台北分公司	amt[0][0]=1100	amt[0][1]=2200	amt[0][2]=3300	—
台中分公司	amt[1)[0)=1500	amt[1][1]=2500	—	—
高雄分公司	amt[2][0]=1000	amt[2][1]=2000	amt[2][2]=3000	amt[2][3]=4000

執行結果


file:///C:/cs2013/ch04/ConsoleAmount/bin/Debug/...
第一處 第二處 第三處 第四處 （單位：千元）
1100 2200 3300
1500 2500
1000 2000 3000 4000
台北分公司營業額：NT$6,600,000.00元 營業率：32.04%
台中分公司營業額：NT$4,000,000.00元 營業率：19.42%
高雄分公司營業額：NT$10,000,000.00元 營業率：48.54%
總營業額：NT$20,600,000.00元

4.3 結構與結構陣列

●4.3.1 結構變數

1.定義 Student 結構

語法

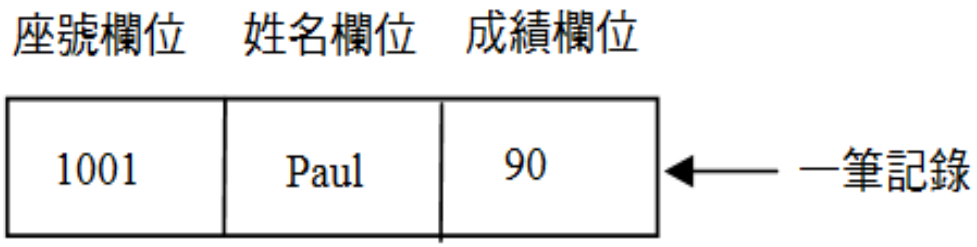
```
struct 結構名稱{
    資料型別 1   欄位名稱 1;
    資料型別 2   欄位名稱 2;
        ⋮
}
```

2. 宣告結構變數

語法

```
struct   結構名稱   結構變數;
```

譬如：定義結構名稱為 Student 的結構，每筆記錄是由座號(No)、姓名(Name)、成績(Score)三個不同資料型別的欄位構成一筆記錄，該記錄如下圖所示：



1. 定義 Student 結構

```
struct Student  {
        public int No;
        public string Name;
        public int Score;
}
```

2. 宣告 bcc 為具有 Student 結構的結構變數：

```
Student bcc ;
```

3. 設定結構變數的初值

結構變數允許將不同資料型別的資料合在一起，存取欄位資料時採取結構變數名稱和欄位名稱中間用小數點隔開來表示。譬如下面敘述即是設定 bcc 計概結構變數各欄位的初值：

```
bcc.No = 1001;
bcc.Name = "Paul" ;
bcc.Score = 90 ;
```

4.3.2 結構陣列

- 一個結構變數表一筆資料；多筆資料須用結構陣列存放。
- 所謂「結構陣列」是指陣列中的每個陣列元素都對應到一筆記錄。

例定義結構名稱為 Student 的結構，每筆記錄是由座號(No)、姓名(Name)、成績(Score)三個不同資料型別的欄位構成一筆記錄，共有五筆記錄其寫法如下：

1. 定義 Student 結構

```
struct Student   {
      public int No;
      public string Name;
      public int Score;
}
```

2. 宣告 bcc 為具有 Student 結構的結構陣列，大小為 4

```
Student[] bcc = new Student[4];
```

4.3.2 結構陣列 Continue…

3. 設定 bcc 結構陣列的初值

```
bcc[0].No = 1001;   bcc[0].Name = "Paul";   bcc[0].Score = 85;
bcc[1].No = 1002;   bcc[1].Name = "Jack";   bcc[1].Score = 80;
bcc[2].No = 1003;   bcc[2].Name = "Mary";   bcc[2].Score = 70;
bcc[3].No = 1004;   bcc[3].Name = "Jane";   bcc[3].Score = 90;
```

可以將上面(2)和(3)宣告和設定初值合併成下面敘述：

```
Student[] bcc = new Student[] {
    new Student() {No= 1001, Name="Paul", Score=85},
    new Student() {No= 1002, Name="Jack", Score=80},
    new Student() {No= 1003, Name="Mary", Score=70},
    new Student() {No= 1004, Name="Jane",  Score=90},
};
```

4.3.2 結構陣列 Continue…

- 使用結構陣列的好處
  當欲對某個欄位做排序時，須對兩筆資料做互換時，資料彼此互換所寫的敘述較精簡
- 譬如：temp為具有Student的結構變數，
  當 bcc[i] 和 bcc[j] 兩個結構陣列元素需做交換只需三行敘述。寫法 ：

```
temp = bcc[i];
bcc[i] = bcc[j];
bcc[j] = temp;
```

4.3.2 結構陣列 Continue…

若改用陣列名稱分別為 No、Name 和 Score 的三個一維陣列來存放座號、姓名、成績。若第 i 個和第 j 個陣列元素的資料需做互換時，必須寫 9 行敘述。寫法：

```
temp = No[i];      No[i] = No[j];      No[j] = temp;
temp = Name[i];    Name [i] = Name[j]; Name [j] = temp;
temp = Score[i];   Score [i] = Score[j]; Score [j] = temp;
```

4.4 方法 (Method)

- C# 都將程式寫在預設名稱為 Program 類別內。
- 類別是由屬性和方法組成
  ⇨ 屬性和方法就成為類別的成員(Member)。
- C# 的方法相當於 VB 的函式或程序
  相當於 C 語言和 C++ 的函式。
- 方法即為函式或程序，只是在不同的程式語言
  而有不同命名方式而已。
- C# 屬物件導向程式語言，為符合物件導向封裝的特性，
  對於物件內的屬性內容的存取儘量透過物件本身所提供
  的方法來存取。

4.4 方法 (Method)

Continue…

- 方法具有下面特點：

1. 方法擁有自己的名稱，使用識別字來命名。
   名稱不允許和變數、常數或定義在類別內的屬性或
   其他方法名稱重複。
2. 方法內所宣告的變數(即屬性)屬區域變數，在不同方法
   內所宣告的變數互不相關，允許使用相同區域變數名稱
   有效範圍限在該方法內。
3. 方法具有特定功能，程式碼簡單明確，可讀性高
   且易除錯和維護。

4.4 方法 (Method)

Continue…

- C# 提供的方法依特性分成三大類：
  1. 系統提供的方法
  2. 使用者自定的方法
  3. 事件(Event)

4.5 方法的使用

語法

```
[成員存取修飾詞] static 傳回值型別 方法名稱 (引數串列)
{
    [方法主體]
    [return 運算式;]
}
```

存取修飾詞	功能
public	類別或類別成員存取範圍完全沒有限制。
protected	類別或類別成員存取範圍限於父類別、或繼承自父類別的子類別可使用。
internal	類別或類別成員存取只限於目前專案。
protected internal	類別或類別成員存取只限於目前專案或繼承自父類別。
private	只有本身的類別可以存取。(預設)

如何在程式中撰寫「使用者自定方法」

譬如：定義一個只有本身類別才可叫用的 add 靜態方法，功能為兩數相加。呼叫此方法時傳入兩個變數給 a 和 b 整數變數，最後透過 return 敘述將 a 和 b 兩數和的結果傳回，由於傳回值為整數，因此在 add 名稱前面的傳回值型別必須設為 int。且此方法允許讓本身類別使用，成員存取修飾詞必須設為 private(或省略 private)，且加上 static 宣告為靜態方法。寫法：

```
private static int add (int a , int b )  //靜態方法
{
   int c;   // 區域變數
   c=a+b;
   return c;
}
```

4.5.2 如何呼叫方法

語法

1. 同一類別：
   方法名稱([引數串列]) ;
2. 不同類別：
   類別名稱.方法名稱 ([引數串列]) ;

4.5.2 如何呼叫方法

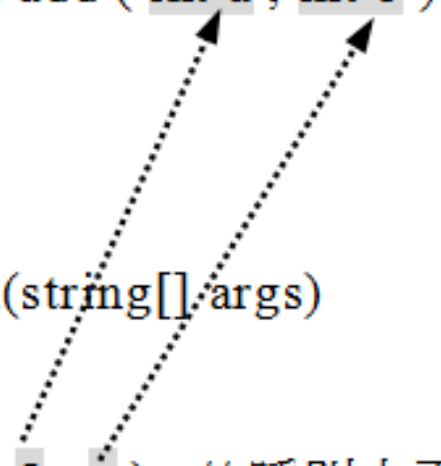
Continue…

同類別的方法呼叫

```
class Class1
{
    private static int add ( int a , int b )
    {

    }

    static void Main(string[] args)
    {
        int y = 6 ;
        int z = add ( 5 , y ) ; // 呼叫上面同類別的 add 方法
    }
}
```



4.5.2 如何呼叫方法

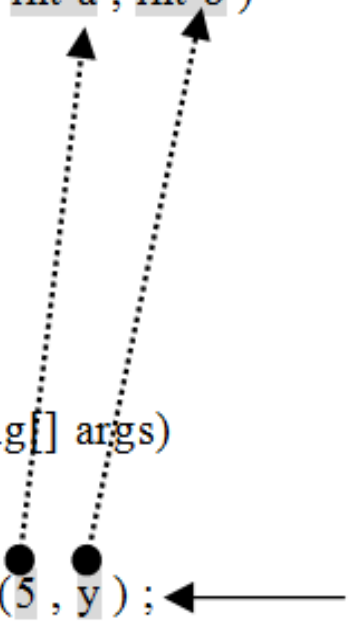
Continue…

不同類別的方法呼叫

```
class Class1
{
    public static int add ( int a , int b )
    {

    }
}

class Class2
{
    static void Main(string[] args)
    {
        int y = 6 ;
        int z = Class1.add (5 , y ) ;
    }
}
```

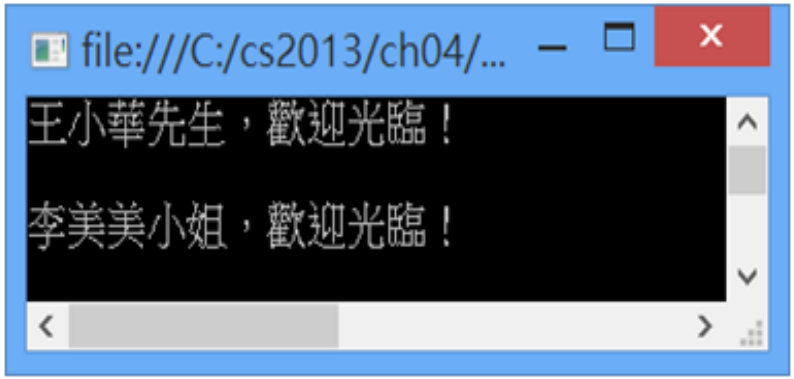


在 Class2 類別中，呼叫 Class1 類別定義的 add 方法

範例 ：ConsoleMethod1.sln

練習在 Program 類別中撰寫 void(不傳回值) 型別的使用者自定方法，其方法名稱為 Login，Main()方法呼叫 Login()方法時會傳遞兩個引數，第一個引數是姓名(字串資料型別)、第二個引數為 true/false(布林資料型別)，若為 true 表示男性，false 為女性。兩個引數均採傳值呼叫(Call by Value)方式傳遞。

執行結果



範例：ConsoleMethod2.sln

延續上例練習在 Program 類別中撰寫具有傳回字串資料型別的使用者自定方法，其方法名稱為 GetWelcome()，Main()主程式呼叫 GetWelcome()方法時，同上例，兩個引數均採傳值呼叫(Call by Value)方式傳遞，第一個引數是姓名(字串資料型別)、第二個引數為 true/false(布林資料型別)，若為 true 表示男性，false 為女性。執行結果與上例 ConsoleMethod1.sln 相同。

範例：ConsoleMethod3.sln

延續上例練習在 Class1 類別中撰寫傳回字串資料型別的使用者自定靜態方法，該方法名稱為 GetWelcome。然後在 Program 類別中呼叫 Class1 類別的 GetWelcome 方法。執行結果同範例 ConsoleMethod1.sln。

範例：ConsoleMethod4.sln

延續上例練習在 Class1 類別中撰寫傳回字串資料型別的使用者自定方法，該方法名稱為 GetWelcome。在 Program 類別中呼叫 Class1 類別的 GetWelcome 方法，必須使用 new 保留字建立 Class1 類別的實體，然後才可以使用 Class1 類別的 GetWelcome 方法。執行結果同範例 ConsoleMethod1.sln。

4.6 引數的傳遞方式

- 使用 return 敘述一次只能傳回一個值或不傳回值返回到原呼叫處。
- 若方法A 某個敘述呼叫另一個 方法B 時，需一次傳回兩個以上值時， return 無法做到
- 必須用參考呼叫或傳出參數達成，
- 方法引數傳遞方式有三種：
  1. 傳值呼叫(Call By Value)
  2. 參考呼叫(Call By Reference)
  3. 傳出參數(Output parameter)

4.6.1 傳值呼叫

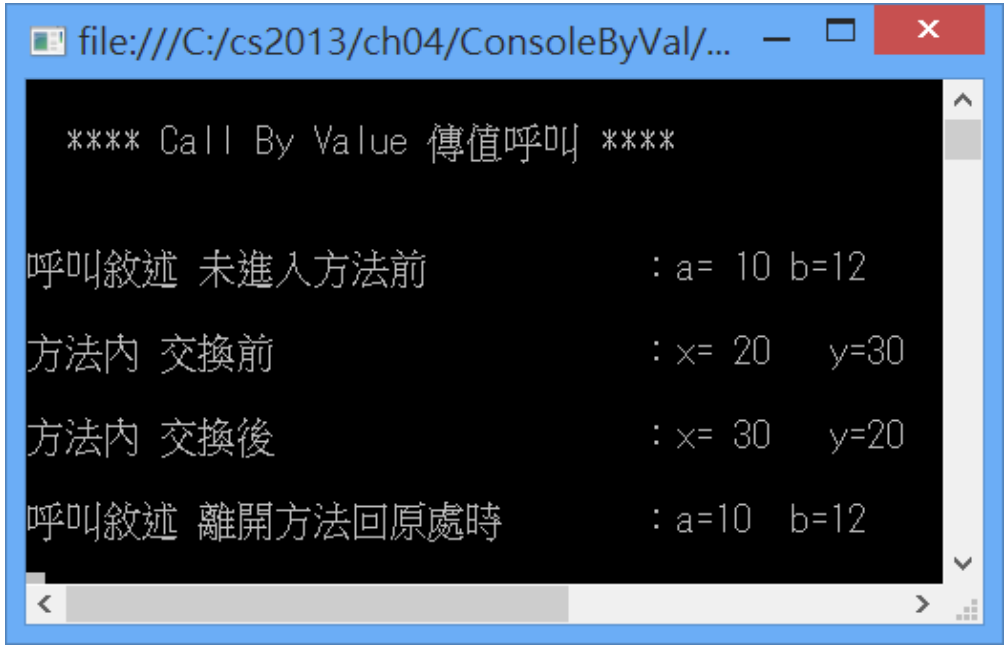
- 傳值呼叫是指呼叫方法時，呼叫方法的實引數複製一份給被呼叫方法的虛引數
- 實引數與虛引數分佔不同記憶體位址。
- 當被呼叫方法內虛引數修改時，結果不影響對應實引數。
- C# 預設為傳值呼叫。適用於方法內結果不影響該方法外的變數時使用，具有保護變數不被修改的特性。
- 傳值呼叫中，實引數可為變數、常數或運算式。

```
CallValue( a,  7,  a + 10) ;
```

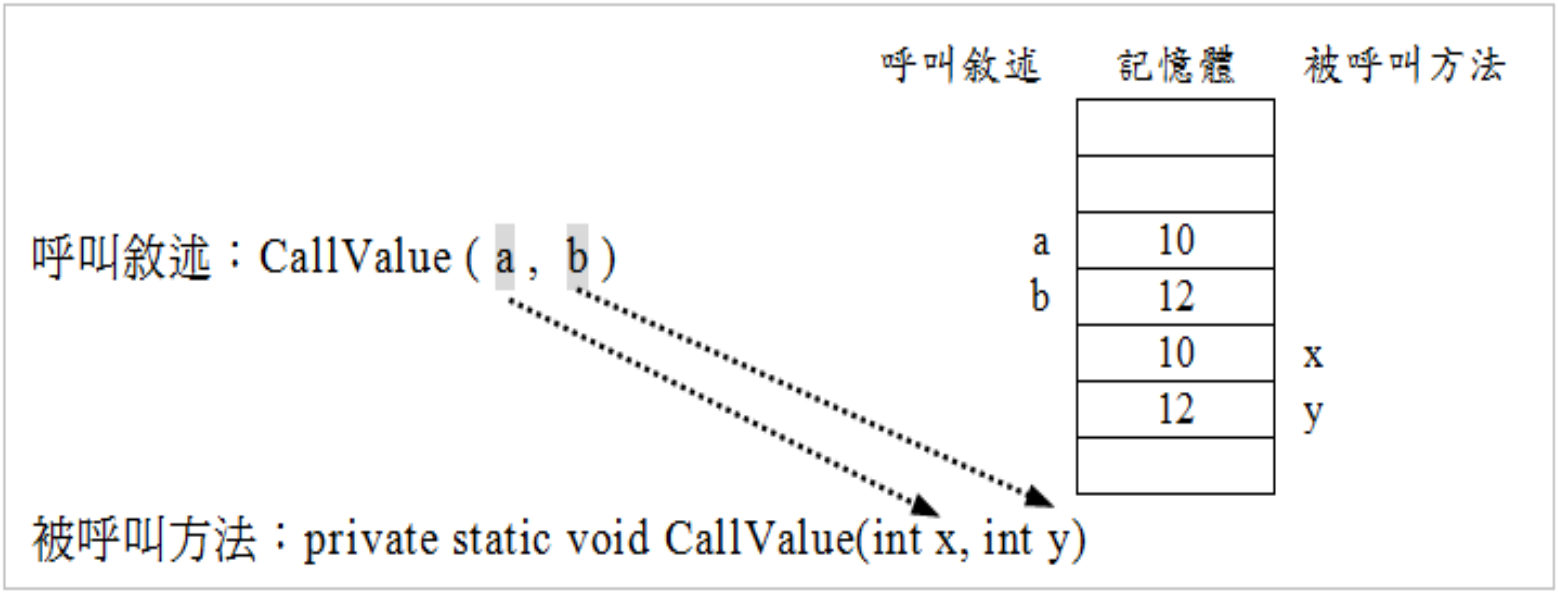
變數　常數　運算式

範例 ：ConsoleByVal.sln

練習撰寫同類別方法間的參數傳值呼叫。本例將 a 和 b 的值以傳值呼叫方式，將 a 和 b 傳給 CallValue 方法()中的 x 和 y 值，在此方法內將 x 和 y 的值互換。請觀察第 23 行呼叫敘述叫用第 5 行 CallValue()方法之前的 a、b 值，進入 CallValue()方法內 x、y 互換前後的值，以及離開 CallValue()方法回到原呼叫敘述時 a、b 的值。



3. 實引數與虛引數的傳遞情形如下：



4.6.2 參考呼叫

- 方法A 呼叫方法B 時，若希望將 方法B 的執行結果回傳給 方法A 需使用參考呼叫。
- 若能將實引數和虛引數佔用相同記憶體位址，虛引數有改變，對應實引數亦跟改變，便可將方法B執行的結果回傳給方法A。
- 操作方式
  將虛引數及實引數宣告為 ref 即成參考呼叫。
- 參考呼叫實引數必須是變數、陣列或物件(即參考資料型別)，不可為常數或運算式
- 實引數必須給予初值才能使用

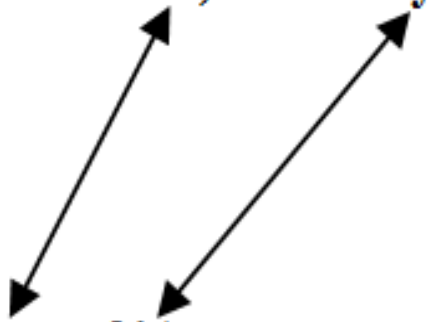
4.6.2 参考呼叫 Continue…

1. 定義方法

```
private static void CallRef(ref int x,  ref int y)
{

}
```

2. 呼叫方法： CallRef (ref a, ref b);

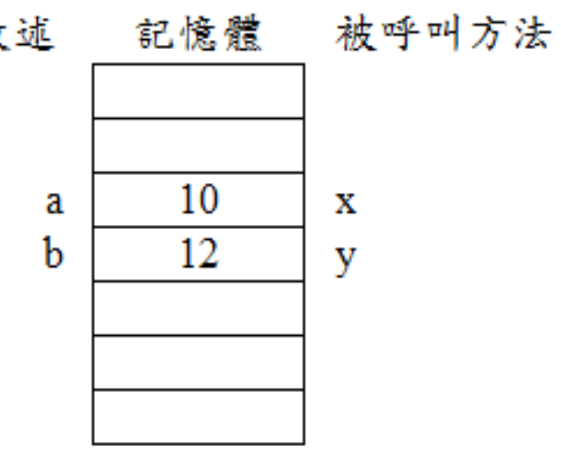


範例：ConsoleByRef.sln

練習撰寫參考呼叫。第 23 行呼叫敘述將 a=10、b=12 以參考呼叫方式將值傳給第 5 行 CallRef()方法的 x、y，在該方法內將 x、y 值設成 20、30，將 x、y 互換，觀察 a、b 是否跟隨 x、y 值改變。

執行結果

```
int a, b;
a=10;
b=12;
```



呼叫敘述：CallRef (ref a , ref b)

被呼叫方法：private static void CallRef(ref int x, ref int y)

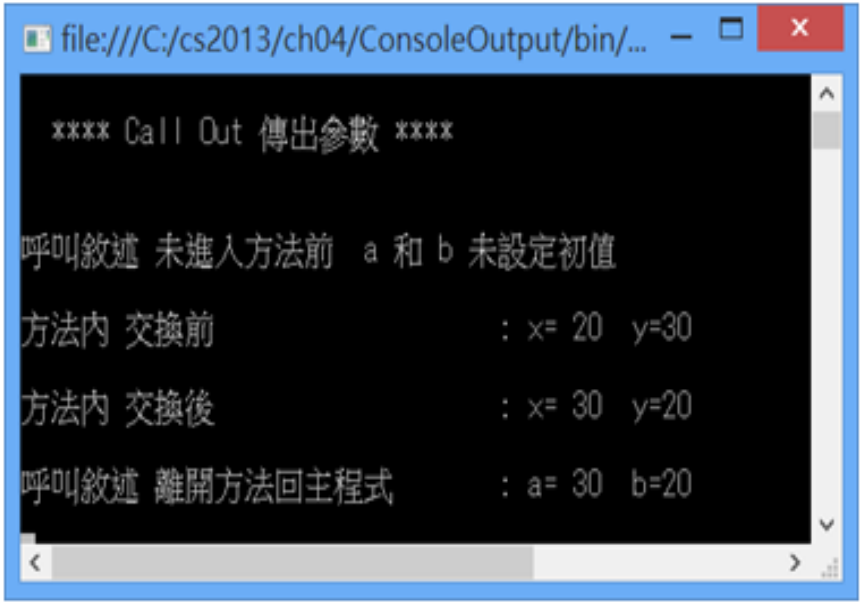
4.6.3 傳出參數

- 傳出參數與參考呼叫的實引數和虛引數都是共同佔用相同的記憶位址。
- 兩者主要差異在於
  - ⇨ 傳出參數的實引數變數不必設定初值即可作參數傳遞。
  - ⇨ 參考呼叫必須先設定初值才能傳遞參數。
- 若在「呼叫敘述」及「被呼叫方法」的引數串列參數前面加上out，即變為傳出參數。

範例：ConsoleOutput.sln

練習撰寫傳出參數。按照輸出結果撰寫進入方法前，和進入方法後各變數交換的變化情形。

執行結果



4.7 如何在方法間傳遞陣列

- 欲將整個陣列實引數傳給被呼叫方法虛引數必須使用參考呼叫。
- 在使用者自定方法中虛引數須在資料型別前加 ref ，及在該資料型別之後加上 [ ] 中括號和陣列名稱即可
- 中括號內不設定陣列大小
- 在呼叫敘述的實引數內，在陣列名稱前加 ref 即可
- 陣列名稱後不必接 [ ] 中括號。
- 由於陣列名稱即代表陣列的起始位址，因此，實引數及虛引數內陣列名稱前面的 ref 可同時省略並不會發生錯誤。但是實引數或虛引數若只有一方有寫ref，則程式編譯時會發生錯誤。

範例：ConsoleGetMax.sln

定義一個 GetMax()方法，該方法傳回陣列中的最大值。當主程式呼叫 GetMax()方法時，將整個陣列以參考呼叫(Call By Reference)方式傳遞給此方法，結果會傳回陣列中的最大值並顯示出來。

4.8 方法多載 (OverLoading)

- 是允許在類別中，建立多個相同名稱的方法
- 做法是以虛引數的引數個數或引數資料型別的不同來區別相同名稱的方法。
- 可讓使用者減少方法命名困擾。
- 譬如下面sum() 有四個方法多載：

1. 當 x 和 y 兩個引數都是整數時呼叫此方法：

```
static int sum(int x, int y )
{
        return (x + y);
}
```

2. 當 x、y 和 z 三個引數都是整數時呼叫此方法：

```
static int sum(int x, int y, int z ) {
  return (x + y + z);
}
```

3. 當 x 和 y 兩個引數都是字串時呼叫此方法：

```
static string sum(string x, string y) {
      return (x + y);
}
```

4. 當 x、y 和 z 三個引數都是字串時呼叫此方法：

```
static string sum(string x, string y, string z) {
      return (x + y + z);
}
```

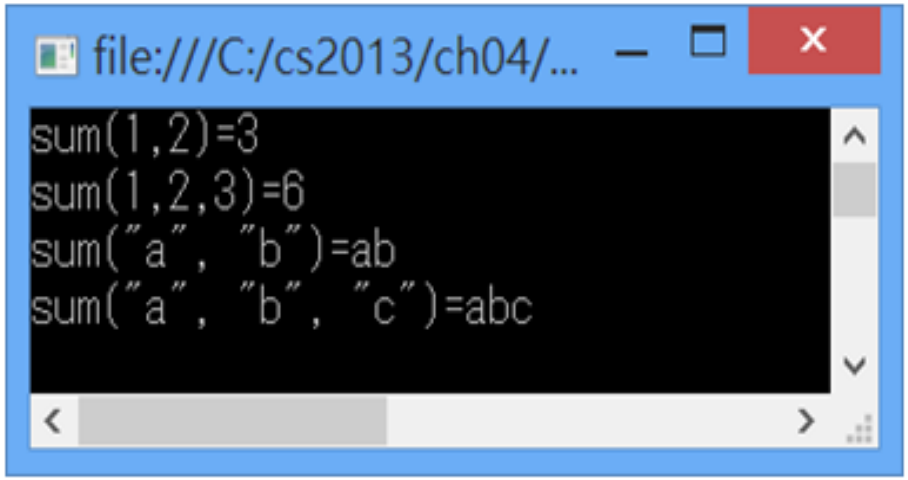
程式執行時：

1. 當執行 sum(2,3) 時，呼叫上面第 1 個方法。
2. 當執行 sum(2,3,4) 時， 呼叫上面第 2 個方法。
3. 當執行 sum("a", "b")時，呼叫上面第 3 個方法。
4. 當執行 sum("a", "b", "c")時，呼叫上面第 4 個方法。

範例：ConsoleOverLoads.sln

將上面簡例說明，以 sum() 方法的多載撰寫出完整程式碼。

執行結果



4.9 區塊變數、區域變數、靜態變數與類別欄位

4.9.1 區塊變數(block level variables)

- 是指程式中前後以大括號括住的多行敘述區塊，在此區塊內所宣告的變數
- 例如for、switch 敘述區塊內所宣告的變數。
- 在某個區塊中所宣告的變數，只能在這個區塊中使用，區塊以外的程式碼，即使在同一個方法中都無法存取，例如：

4.9.1 區塊變數 (block level variables)

```
class Class1
{
    static void Main(string[] args)
    {
        for(int i=1 ; i<=6 ; i++)
        {
            Console.WriteLine(i) ;
        }
        Console.WriteLine(i) ;
    }
}
```

4.9.2 區域變數 (local variables)

- 區域變數是在方法(函式)內宣告的變數，或是方法的引數
- 區域變數一離開方法，其生命週期即馬上結束。
- 例如
  下面程式片段 MyMethod() 方法中的
  name、k 及str 是區域變數。

4.9.2 區域變數

```
class Class1
{
    private void MyMethod(string str)
    {
        string name = "蔡小龍" ;          // 區域變數 name
        name += str ;                     // 區域變數 str
        int k ;                           // 區域變數 k
        for(int i=1 ; i<=6 ; i++)         // 區塊變數 i
        {
            Console.WriteLine(i) ;
        }
        // Console.WriteLine(i) ; i 為區塊變數, 故此處 i 無法使用
    }
}
```

4.9.3 靜態變數

- 在類別內宣告的變數(即屬性)前加 static
  此變數在類別內變成靜態變數或稱靜態欄位。
- 靜態變數不能在方法內宣告，離開方法後變數值不會釋放掉，下次再呼叫此方法時，保留的變數值仍可繼續使用
- 靜態變數的生命週期是到程式停止執行為止。
- 當程式執行時，static 變數會被放在全域變數區，此時不需建立物件實體就能存取此變數，且多個物件共用一份靜態變數。

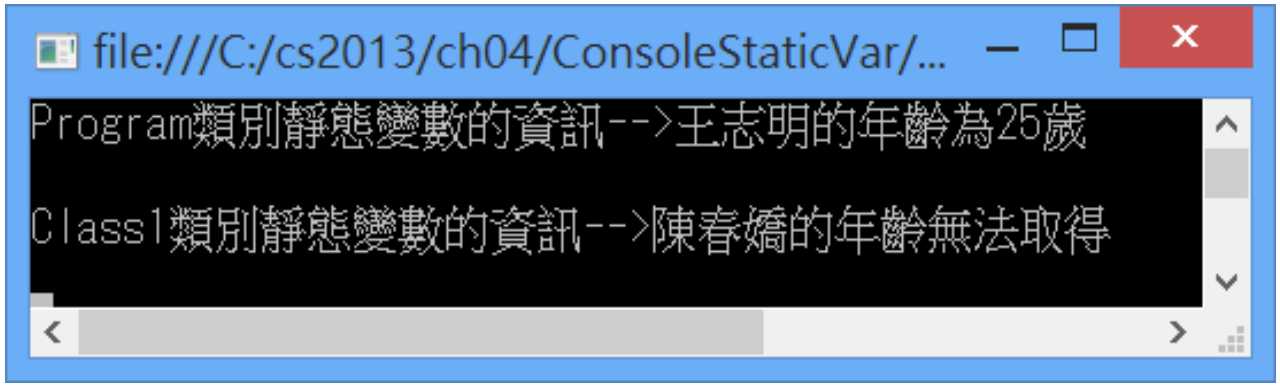
範例：ConsoleStaticVar.sln

練習使用靜態變數。本範例說明如下：

① 用 static 修飾詞所宣告的類別成員，不必建立物件實體即可使用。

② 本例第 11,12 行的 Program 類別內建立 age 及 name 靜態變數，且宣告為 private，因此只有 Program 類別內程式才可以使用 age 及 name 變數，且在 Program 類別中使用 age 及 name 變數不用撰寫完整名稱，如第 15 行程式使用 name 及 age 變數。

③ 本例第 5 行的 Class1 類別建立 private 的 age 變數，因此只有 Class1 類別可以使用 age 變數；第 6 行使用 static 建立 name 靜態變數，且宣告為 public，因此 name 靜態變數的存取範圍沒有限制，若在不同類別要使用 Class1 類別的 name 靜態變數，必須撰寫完整名稱才可以使用，如第 16 行敘述 Class1.name 即是在 Program 類別中使用 Class1 類別的 name 靜態變數。



定義為 static 靜態變數，在宣告變數之後若沒有指定變數的初值，C# 會依照變數的資料型別，直接設定其預設值，下表為各種資料型別的預設值。

資料型態	預設值
sbyte,byte,short,ushort,int,uint,long,ulong	0
char	'\x0000'
float	0.0f
double	0.0d
decimal	0.0m

4.9.4 類別欄位(非靜態成員)

- 物件屬性(或稱欄位)和靜態變數(加上static修飾詞或稱靜態成員)一樣。
- 不同地方是因物件屬性放在 Heap區，靜態變數是放在全域變數區，因此物件屬性必須建立該物件實體(執行個體)才能使用。
- 當物件生命週期結束後，非靜態物件屬性生命週期會跟著結束。

```
class MyClass {
    public static int vNo ;          // 靜態變數，或稱靜態成員
    public string vName ;            // 物件屬性，也可稱為物件欄位
    ..............................

}
```

本章結束 ...