

# 第五章 方法



5.1 結構化程式設計

5.5 參數的傳遞方式

5.2 方法的簡介

5.6 陣列間參數的傳遞方式

5.3 亂數類別的使用

5.7 遞迴

5.4 方法的使用

5.8 多載

備註：可依進度點選小節



## 5.1 結構化程式設計

開發較大應用程式的過程中

- 程式設計人員和使用者彼此溝通不良
  - ⇒ 使得開發出的應用程式滿意度降低。
- 因在開發過程發生未察覺錯誤
  - ⇒ 降低使用者對該應用程式的品質信賴度。
- 對開發完成的應用程式測試不正確或文件說明不完整和明確
  - ⇒ 影響該應用程式日後維護。

設計大型的應用程式若能朝「結構化」去設計，就能避免上述事情的發生。



## ■ 結構化程式設計 (Structure Programming Design)

是指程式具有「由上而下程式設計」的精神及具模組化設計概念。

## ■ 由上而下程式設計 (Top-Down Programming Design)

是一種逐步細緻化的設計觀念，它具有層次性，將程式按照性能細分成多個單元，再將每個單元細分成各個獨立的模組，模組間儘量避免相依性，使得整個程式的製作簡單化。

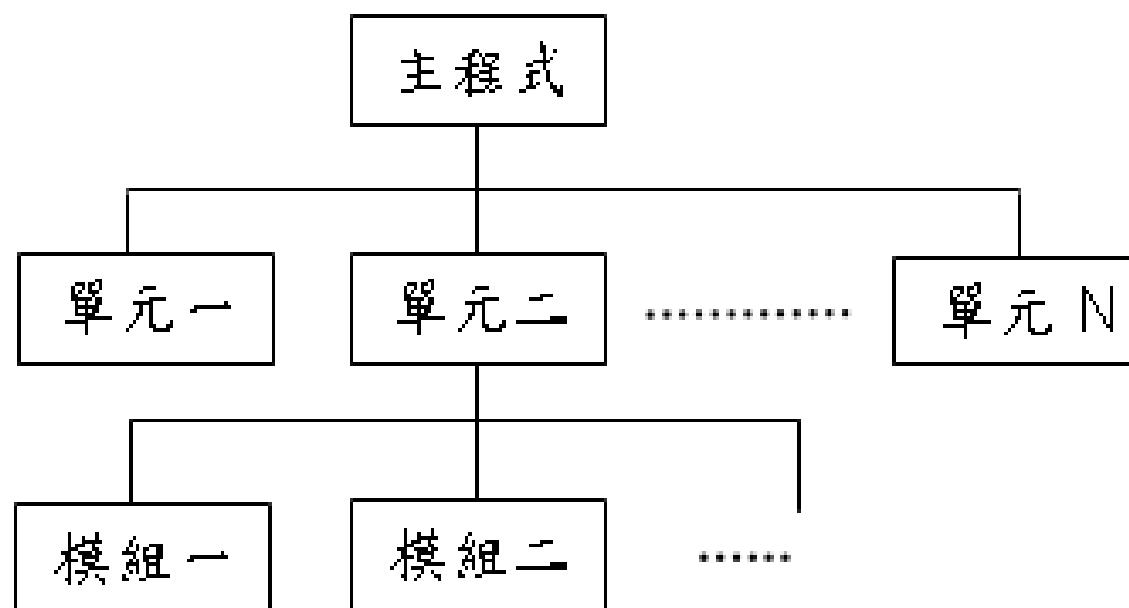
■ 設計一個完整的程式，就好像蓋房子一樣。

■ 設計程式亦如此。



■ 下圖是由頂端的主程式開始規劃，逐步往下層設計，層次分明有條理且易於了解，減少程式發生邏輯錯誤。

■ 此種設計觀念就是「由上而下程式設計」





- **模組化程式設計 (Modular Programming Design)**  
分析問題時，由大而小，由上而下，將應用程式切割成若干模組。
- 若模組太大可再細分小模組，使每個模組成為具獨立功能的程序或函式
- **C# 是物件導向語言**，一般都將函式稱為「**方法**」(Method)。
- 每個模組允許各自獨立分工撰寫和測試，可減輕程式設計者負擔且易維護和降低開發成本。



- 模組與模組間**資料**是透過**參數**來傳遞。
- 由於模組分開獨立撰寫，不同的應用程式可共用提昇生產力。
- 本章介紹的方法(或稱函式)都可視為**模組**。
- 一個製作好的模組，使用者只要給予輸入值，不必瞭解模組內的如何運作，便可輸出結果，有如各式各樣的積木，每塊積木相當於一個模組，使用者可依不同需求組出不同的東西出來。



- 任何一個程式的流程不外乎由循序結構、選擇結構(if...else...、switch...)，重複結構(for...、do...while)等程式區塊組合而成。
- 程式流程保持一進一出的基本架構，因而增加程式的可讀性。
- 在結構化程式設計，無論在規劃程式階段或編寫程式階段都注重由上而下設計和模組化的精神，使得程式層次分明、可讀性高、易分工編寫、易除錯與維護。



## 5.2 方法的簡介

- 在撰寫較大程式中，有些具有某種特定功能的程式區塊會在程式中多次重複出現，使得程式看起來冗長和不具結構化，C# 允許將這些程式區塊單獨編寫成一個方法(或稱函式)並賦予方法一個名稱，程式中需要時才進行呼叫。
- 方法符合模組化程式設計的精神。
- 將這些小模組交給不同程式設計師，同時進行編寫程式碼，最後只要透過最上層的主程式將各個方法連結起來就成為一個完整的大系統。





- 將程式開始執行的起點稱為「主程式」
- 在主控台應用程式下 C# 程式起點是 Main()方法
- Main()方法算是主程式。
- 一般的方法是不會自動執行，只有在被主程式或另一個方法呼叫(Call)時，一般的方法才會被執行。



## 程式中使用方法來建構程式碼的優點：

1. 方法可將大的應用程式分成若干不連續的邏輯單元，由多人同時進行編碼，可提高程式設計效率。
2. 相同功能的程式片段編寫成方法，只需寫一次，可在同個程式中多處使用；也可不需做太多修改或不用修改，套用到其它的程式共用，開發省時省力。
3. 可縮短程式長度，簡化程式邏輯，有助提高程式的可讀性。
4. 採用物件導向程式設計，不同功能的程式單元獨立成為某個類別的方法，使得較易除錯和維護。



■ 程式設計者應程式需求而自行定義的**使用者自訂方法**，在 C# 常用可分成下列兩種：

## 1. 事件處理函式 (Event-Handling Function)

依據使用者動作或程式項目所觸發 (Trigger) 的事件，此時會去執行該事件的事件處理函式，例如在 btnOk 鈕的上按一下，會觸發 btnOk 鈕的 Click 事件，此時會執行 btnOk\_Click() 事件處理函式。

## 2. 方法 (Method)

程式設計者自己定義的程式模組，可傳回一個結果值或不傳回值，一般都稱為函式，因 C# 為物件導向語言，在此我們稱為**方法**。



- 方法適用於進行重複或共用的工作。
- 例如常用的計算、文字和控制項的操作及資料庫作業。
- 可從程式碼中許多不同的位置呼叫方法，可將方法當作應用程式的建置區塊。



## 5.3 亂數類別的使用

「**內建類別**」是廠商將一些經常用到的數學公式、字串處理、日期運算以及方法的資料型別直接建構在該程式語言的系統內

### 系統內建類別

以 C# 來說就可用 .Net Framework 內建類別，程式設計者不必了解這些類別以及物件方法內部的寫法，只要給予參數，直接呼叫物件的方法名稱，便可得到輸出結果。

在 .Net Framework 提供的常用內建類別包括：  
亂數類別、數學類別、字串類別以及日期類別。



## ■ 命名空間 (Namespace)

由於 .Net Framework 所提供的類別很多，程式在編譯時，若將全部類別全部載入，對程式的執行效率影響很大，因此將所有類別分類，每個分類給予名稱稱為「命名空間」。

- 命名空間是微軟 .Net 用來將相關型別集合在一起的一種命名機制，以避免不同組件間發生名稱衝突。
- 譬如：常用的 Console 類別、String 類別、Random 類別...等都放在 System 這個命名空間中，程式中若有使用到這些類別，必須用 using 敘述引用 System 命名空間。
- 寫法： **using System;**



■ **Random** 類別可用來產生亂數。

■ 宣告和建立方式：

1. 先宣告一個名稱為 **ranObj** 是指向 **Random** 型別的物件參考。
2. 使用 **new** 來初始化建立 **ranObj** 物件，即建立 **ranObj** 為 **Random** 類別的物件實體。

■ 寫法：**Random ranObj = new Random();**



■ 使用 Random 類別提供的 Next() 方法產生某個範圍的亂數值。

■ 方式：

① 產生介於 0~2,147,483,647( $=2^{31}-1$ )間的亂數值  
並指定給 ranNum 整數變數：

```
int ranNum = ranObj.Next();
```

② 產生 0~4 間的亂數值並指定給 ranNum 整數變數，寫法：

```
int ranNum = ranObj.Next(5);
```

③ 產生 7~99 間的亂數值並指定給 ranNum 整數變數，寫法：

```
int ranNum = ranObj.Next(7, 100);
```





### 範例演練

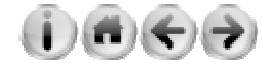
試使用 **for** 迴圈與 **Random** 類別來產生兩組1~10 之間的五個亂數，觀察兩次執行結果是否一樣？

```
file:///C:/CSharp/chap05/...  
== Pass1 : 9 7 2 8 5  
== Pass2 : 3 5 2 5 2_
```

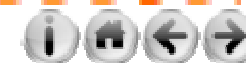
第一次執行結果

```
file:///C:/CSharp/chap05/...  
== Pass1 : 3 1 9 5 7  
== Pass2 : 1 4 6 9 6
```

第二次執行結果



```
// FileName : RndObj.sln
01 static void Main(string[] args)
02 {
03     Random rndObj = new Random();
04     // 一組 1-10 的五個亂數
05     Console.Write("== Pass1 : ");
06     for (int k = 1; k <= 5; k++)
07     {
08         Console.Write(" {0}", rndObj.Next (1, 11));
09     }
10     Console.WriteLine();
11     //第二組 1-10 的五個亂數
12     Console.Write("== Pass2 : ");
13     for (int k = 1; k <= 5; k++)
14     {
15         Console.Write(" {0}", rndObj.Next(1, 11));
16     }
17     Console.Read();
18 }
```



## 5.4 方法的使用

- C# 為物件導向的程式語言，方法必須放在類別內不可獨立在類別外，和傳統結構化程式設計不同。
- 使用靜態方法並不需建立類別的物件實體即可直接呼叫使用
- 迴圈和方法(或稱函式)的應用範圍不相同：
  - ① 迴圈每次執行時，可在程式相同的地方重複執行某一段程式碼
  - ② 方法可在程式中任何地方被重複呼叫使用。
- 方法需在程式中先定義，再透過呼叫方式來執行該方法。



## 5.4.1 方法的定義

■ 定義方法主要是先為該方法設定：

- ① 傳回值的資料型別
- ② 為方法命名
- ③ 指定呼叫該方法時到底要傳入多少個引數
- ④ 同時宣告這些引數的資料型別

■ 以上都在定義方法第一行敘述中書寫，接著在該方法的主體內撰寫相關程式碼。基本語法：

```
[static] 傳回值資料型別 方法名稱 ( 引數串列 )  
{  
    .....  
    // 方法主體  
    .....  
}
```



## 方法的命名：

■ 比照 識別項 命名規則：

- ① 引數串列超過一個以上，中間使用逗號隔開。
- ② 每個方法和變數一樣都有一個資料型別，該資料型別是放在方法名稱前面，由它來決定傳回值的資料型別。
- ③ 若在方法前加 **static**，即將方法定義為靜態方法，靜態方法不用建立類別的物件實體即可以呼叫使用。
- ④ 若方法之前省略**static** 即為類別的方法，類別的方法必須要建立物件實體才能呼叫。



- 方法是一組以 { 開頭而以 } 結束所組成的程式區塊。
- 每當方法被呼叫時，會跳到方法後面的第一個可執行的陳述式，開始往下執行，當碰到 } 右大括號或 **return** 陳述式才離開方法，接著再返回原呼叫陳述式處。
- 方法內引數串列寫法：

```
static int compute ( int r, ref double v )  
{  
    .....  
}
```

虛引數串列



## 實引數和虛引數：

■ 將呼叫陳述式的引數串列稱為「實引數」(Actual Argument)

■ 將被呼叫方法的引數串列稱為虛引數(Dummy Argument)。

### ■ 參考呼叫

虛引數串列的引數資料型別前面設定為 **ref**。此時實引數和虛引數會佔用相同記憶位址，表示除了允許接收由原呼叫陳述式對應的實引數外，離開時也可以將值傳回給原呼叫敘述對應的引數。

### ■ 傳值呼叫

若虛引數資料型別前面沒有加 **ref**。表該引數只能接收由原呼叫陳述式對應的引數，但離開方法時無法將值傳回。



## 傳入值和傳回值

### 傳入值

是指當呼叫方法時，可透過引數串列來取得由呼叫該方法的陳述式處所傳入的常數、變數或運算式等引數當作初值。

### 傳回值

是指方法將執行結果傳回給原呼叫程式的值，方法使用 **return** 敘述來傳回值。語法：

**return**(運算式)

 若方法的傳回值設為 **void**，表呼叫該方法沒傳回值。





## return 敘述

使用 **return** 陳述式來指定傳回值，當此陳述式執行完畢立即將控制權傳回給原呼叫程式，接在 **return** 後面的陳述式都不會執行。

```
static double compute(double r)
{
    :
    return (4.0/3.0*PI*r*r*r); //傳回計算後的結果值
    :                          //並將控制權由此離開方法返迴呼叫處
    :                          //此處不會繼續執行
}
```



## 方法使用上注意事項：

- 方法除可寫在 **Main()** 主程式前面，也可將方法寫在 **Main()** 主程式的後面
- 方法必須在類別內。
- 程式中使用方法有限制，方法內不允許再定義另一個方法。
- 方法前可加 **public**、**private** 等來設定方法有效的存取權限。
- 若宣告為 **public** 表示該方法的存取範圍沒有限制。
- 若宣告為 **private** 該方法只能在目前的類別中使用。



## 5.4.2 方法的呼叫

■ 當靜態方法定義完畢，呼叫的敘述與靜態方法在同類別內，便可用下面方式呼叫

■ 將上節定義的方法稱為 **被呼叫的程式**。

下列呼叫方法的敘述稱為 **呼叫陳述式**。

■ **語法1：**

**變數名稱 = 方法名稱([引數串列]);** // 傳回一個結果值

**volume=compute(r);**

**volume=20+compute(r+3);**

■ **語法2：**

**方法名稱([引數串列]);**// 不傳回值



- 在撰寫呼叫陳述式和被呼叫程式間做引數傳遞時注意：
  - ① 實引數和虛引數的個數要相同
  - ② 兩者間對應引數的資料型別要一致
- 實引數允許使用常值、變數、運算式、陣列、結構、物件當引數。
- 虛引數不允許使用常值、運算式當引數，允許使用變數、陣列、結構、物件當引數。



## 呼叫方法1 - 呼叫同類別之靜態方法

呼叫時可直接撰寫靜態方法名稱再傳入引數。

下面寫法呼叫 **add()** 靜態方法，將實引數 1 傳給 虛引數 **a**，實引數 6 傳入給 虛引數 **b**，最後透過 **return** 將 **a+b** 結果傳回原呼叫敘述 等號左邊的 **sum** 變數。

```
class Program    // Program 類別
{
    static void Main(string[] args)    // Main() 主程式
    {
        // 呼叫 add 方法，會將 add 方法的結果傳回給 sum 整數
        int sum = add (1, 6);
    }
    static int add (int a, int b)    // 被呼叫的 add 方法
    {
        return a+b;                // 傳回 a+b 兩數相加
    }
}
```

Diagram illustrating the call to the **add** method. In the **Main** method, the call **add(1, 6)** is shown. Arrows point from the arguments **1** and **6** to the parameters **a** and **b** in the **add** method signature, with the label "傳入" (pass) written below each arrow.



## 呼叫方法 2 - 呼叫不同類別之靜態方法，

呼叫時需撰寫完整的類別名稱及 **public** 公用型靜態方法名稱，接著再傳入方法的引數即可。

下面寫法呼叫 **Class1** 類別的 **add()** 靜態方法並將實引數 1 傳給虛引數 **a**，實引數 6 傳入給虛引數 **b**，最後透過 **return** 敘述將 **a+b** 結果傳回給原呼叫敘述等號左邊的 **sum** 變數。

```
class Program    // Program 類別
{
    static void Main(string[] args) // Main 主程式
    {
        // 直接呼叫 Class1 類別的 public 公用型 add 靜態方法
        int sum = Class1.add(1, 6);
    }
}

class Class1 // Class1 類別
{
    public static int add(int a, int b) //被呼叫的 add 方法為 public
    {
        return a+b; // 傳回 a+b 兩數相加
    }
}
```

Diagram illustrating the call: Arrows point from the arguments '1' and '6' in the `Class1.add(1, 6);` call to the parameters 'a' and 'b' in the `add(int a, int b)` method definition, labeled '傳入' (pass).



## 呼叫方法 3 - 呼叫不同類別之方法

首先必須用 **new** 關鍵字建立該類別的物件實體，接著透過「**物件.方法名稱()**」來呼叫即可。

下面寫法先建立 **c** 屬於 **Class1** 類別的物件，接著呼叫 **c** 物件的 **add** 方法並傳入虛引數 **1** 和 **6** 給實引數 **a** 和 **b**，最後透過 **return** 將 **a+b** 的結果傳回給原呼叫敘述等號左邊的 **sum** 變數。

```
class Program //Program 類別
{
    static void Main(string[] args)    // Main 主程式
    {
        Class1 c=new Class1();    // 建立 Class1 類別的 c 物件
        int sum = c.add(1, 6);    // 呼叫 c 物件的 add 方法
    }
}

class Class1    // Class1 類別
{
    public int add (int a, int b) // 被呼叫的 add 方法，非靜態方法
    {
        return a+b;    // 傳回 a+b 兩數相加
    }
}
```

Diagram illustrating the method call: Arrows labeled "傳入" (Pass) point from the arguments `1` and `6` in the `c.add(1, 6)` call to the parameters `a` and `b` in the `add` method definition.



## 範例演練

試寫一個方法，名稱為 **Compute**。先由鍵盤輸入半徑 (radius)，再將 radius 實引數傳給 **Compute()** 方法的 **r** 虛引數，計算出體積後，再將結果由方法本身傳回給 **volume**，最後再顯示輸入的半徑和計算出的體積。

```
file:///C:/CSharp/chap05/Medhod1/bin/Debug/Medhod1.EXE
請輸入半徑<公分> : 5
半徑 = 5公分 體積 = 523.6 立方公分
```





### Step1 宣告 PI 靜態變數

靜態方法內只能存取靜態變數(即靜態欄位)，  
將圓周率 PI 宣告成 **private** 私有靜態變數，  
因此 PI 私有靜態變數需在 **Main()** 方法前面或後面宣告。  
寫法：

```
private static double PI= 3.1416;
```

### Step2 定義 Compute() 方法

在 **Main()** 方法前面或後面定義下列 **Compute()** 方法用來計算  
並傳回圓的體積，寫法：

```
static double Compute(double r)
{
    return (4.0 / 3.0 * PI * r * r * r);
}
```

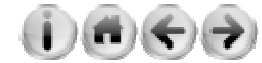


### Step3 撰寫 Main() 方法主程式

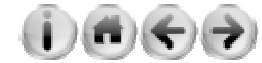
在Main()方法內宣告相關變數，volume 存放體積，radius 存放半徑，為求體積正確，變數資料型別設為double。

### Main()主程式寫法：

```
static void Main(string[] args)
{
    double volume, radius;
    Console.Write(" 請輸入半徑(公分) : ");
    radius = double.Parse(Console.ReadLine());
    //呼叫Compute靜態方法並傳入半徑，最後再將結果傳回給volume
    volume = Compute(radius);
    Console.WriteLine();
    Console.WriteLine(" 半徑 = {0}公分 體積 = {1} 立方公分",
        radius, volume);
    Console.Read();
}
```



```
// FileName : Medhod1.sln
01 //宣告靜態成員變數PI(即靜態欄位)
02 private static double PI = 3.1416;
03
04 //Compute靜態方法可算出圓的體積
05 static double Compute(double r)
06 {
07     // 靜態方法才能使用靜態變數
08     return (4.0 / 3.0 * PI * r * r * r);
09 }
10
```



```
11 static void Main(string[] args)
12 {
13     double volume, radius;
14     Console.Write(" 請輸入半徑(公分) : ");
15     radius = double.Parse(Console.ReadLine());
16     // 呼叫Compute靜態方法並傳入半徑，最後再將結果傳回給volume
17     volume = Compute(radius);
18     Console.WriteLine();
19     Console.WriteLine(" 半徑 = {0}公分 體積 = {1} 立方公分", radius, volume);
21     Console.Read();
22 }
```



## 範例演練

使用名稱為CheckYear()的方法，此方法傳回型別為 void，由鍵盤輸入西元多少年(year)，採傳值呼叫方式將 year 變數傳給 CheckYear()方法中的 y 整數變數，將 y 經判斷是否為閏年，若是印出”閏年”訊息，否則印出”平年”訊息。

閏年計算規則「能被4整除且不被100整除或能被400整除的年份」

```
if (y % 4 == 0 && y % 100 != 0 || y % 400 == 0)
{
    // 閏年
}
else
{
    // 平年
}
```





**FileName : Medhod2.sln**

**01 //呼叫CheckYear靜態方法，可判斷並顯示傳入的y年份是閏年還是平年**

**02 static void CheckYear (int y)**

**03 {**

**04     if (y % 4 == 0 && y % 100 != 0 || y % 400 == 0)**

**05     {**

**06         Console.WriteLine("\n=== {0} 年 為 閏年! ===", y);**

**07     }**

**08     else**

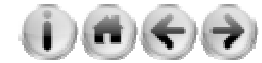
**09     {**

**10         Console.WriteLine("\n=== {0} 年 為 平年! ===", y);**

**11     }**

**12 }**

**13**



```
14 static void Main(string[] args)
15 {
16     int year;
17     Console.Write("請輸入年份:");
18
19     try
20     {
21
22         year = int.Parse(Console.ReadLine());
23
24         CheckYear(year);
25     }
26     catch (Exception ex) 27     {
28         Console.WriteLine(ex.Message );
29     }
30     Console.Read();
31 }
```



## 5.5 參數的傳遞方式

■ 方法都是使用者應程式需求而自訂的函式。

■ 方法必須透過呼叫才能執行。

■ 參數傳遞

是指自訂方法(或稱函式)被呼叫開始執行前，應指定哪些實引數要從呼叫陳述式傳入被呼叫方法的虛引數及哪些虛引數在離開自訂方法要傳回給呼叫陳述式的實引數。

■ C# 提供兩種方式做參數傳遞：

1. 傳值呼叫(Call By Value)
2. 參考呼叫(Call By Reference)





- C# 中，在被呼叫方法引數串列中的虛引數及呼叫陳述式的實引數前面可指定或省略 **ref** 關鍵字。
- 若省略 **ref** 關鍵字代表以傳值 (By Value) 方式來傳遞引數至方法。
- 若虛引數型別之前加上 **ref** 關鍵字則以參考 (By Reference) 方式來傳遞引數至方法。
- 以傳值方式來傳遞引數屬於數值型別 (Value Type) 。
- 以參考方式來傳遞引數屬於參考型別 (Reference Type) 。



## 5.5.1 傳值呼叫

- **傳值呼叫** 是指當呼叫陳述式將實引數傳給虛引數只做傳入的動作。
- 也就是說 C# 此時將虛引數視為**區域變數**，自動配置新的記憶位址給虛引數來存放實引數傳入的內容。
- 實引數和虛引數兩者在記憶體分別佔用**不同的記憶位址**，當虛引數在方法內資料有異動時，並不會影響實引數的值。
- 當離開自訂方法時，虛引數佔用的記憶體位址自動釋放，交還給系統，當程式執行權返回到原呼叫的陳述式時，實引數的內容仍維持不變。
- 用傳值方式來傳遞引數，表示該自訂方法無法變更實引數中原呼叫程式碼裡的變數內容。
- C# 對參數傳遞預設採傳值呼叫。



## 5.5.2 參考呼叫

- 所謂「**參考呼叫**」是指當呼叫陳述式將實引數傳給虛引數時，可做傳入和傳出的動作。
- 也就是說實引數和虛引數兩者參用相同的記憶體位址來存放引數，當虛引數在方法內資料有異動時，離開自訂方法時，虛引數解除參用，但實引數仍繼續參用，待當程式執行權返回到原呼叫的陳述式時，實引數的內容已是異動過的資料，這表示參考呼叫是可以修改變數本身。
- 引數做參考呼叫時，虛引數及實引數的資料型別之前要加上 **ref** 關鍵字來表示。



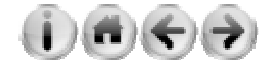
## 範例演練

修改 **Medhod2.sln** 範例。

試使用名稱為 **CheckYear()** 方法，由鍵盤輸入西元多少年 (**year**)，採傳值呼叫方式將 **year** 變數傳給 **CheckYear()** 方法中的 **y** 整數變數；

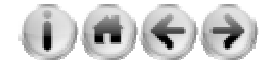
採參考呼叫使實引數 **str1** 和虛引數 **s1** 佔相同記憶位址。

經判斷若 **y** 為閏年，則指定**s1**(即**str1**)傳回 ” 閏年 ” 訊息，若是平年，則**s1**(即**str1**)傳回 ” 平年 ” 訊息。



FileName : Medhod3.sln

```
01  // y虛引數採傳值呼叫，s1虛引數採參考呼叫
02  //s 1虛引數與實引數會共用同一記憶空間
03  static void CheckYear(int y, ref string s1)
04  {
05      if (y % 4 == 0 && y % 100 != 0 || y % 400 == 0)
06      {
07          s1 = "閏年! ";
08      }
09      else
10      {
11          s1 = "平年! ";
12      }
13  }
14
```



```
15 static void Main(string[] args)
16 {
17     int year;
18     string str1= "";
18     Console.Write("請輸入年份：");
19     try
20     {
21         year = int.Parse(Console.ReadLine());
22
23         CheckYear(year, ref str1);
24         Console.WriteLine("\n=== {0}年爲{1} === ", year, str1);
25     }
26     catch (Exception ex)
27     {
28         Console.WriteLine(ex.Message);
29     }
30     Console.Read();
31 }
```



## 5.6 陣列間參數的傳遞方式

- 方法間引數的傳遞，除常值、變數外，還可用陣列、結構，或物件來傳遞。
- 若引數是陣列，單一個陣列元素可依需求採傳值或參考呼叫。
- 物件或整個陣列做參數傳遞必須使用參考呼叫。
- 假設 `myAry` 為一個整數陣列，透過 `myMethod` 方法作參數傳遞。其寫法如下：



## 1. 傳遞陣列元素

主程式：

⋮

myMethod(myAry[2], myAry[4]); //傳遞陣列元素

⋮

方法 myMethod()：

static void myMethod(int a, ref int b){...}

.....

傳值

傳參考





## 2. 傳遞整個陣列

被呼叫程式虛引數資料型別之後必須加上 `[]`，  
原呼叫陳述式可直接設定陣列名稱即可。

寫法：

主程式：

⋮

`myMethod(myAry);`    //傳整個陣列

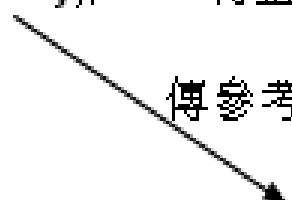
⋮

方法 `myMethod()`：

`static void myMethod( int[] a){...}`

.....

傳參考

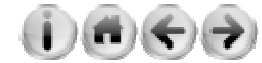




### 範例演練

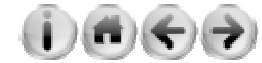
試寫一個將 **ary1** 整個整數陣列傳給 **GetMin()** 方法，  
接著即可傳回 **ary1** 陣列中的最小值。

```
file:///C:/CSharp/chap05/Send Array...  
ary1陣列為->10 88 6 34 77  
ary1陣列最小數為：6
```



FileName : SendArray.sln

```
01 // GetMin()方法找出傳入的陣列中的最小值
02 static int GetMin(int[] tempAry)
03 {
04     int min = tempAry[0];
05
06     for (int i = 1; i <= tempAry.GetUpperBound(0); i++)
07     {
08         if (tempAry[i] < min)
09         {
10             min = tempAry[i];
11         }
12     }
13     return min;
14 }
15
```



```
16 static void Main(string[] args)
17 {
18     //建立並初始化ary1陣列
19     int[] ary1 = new int[] { 10, 88, 6, 34, 77 };
20     Console.Write("ary1陣列為->");
21     //逐一印出陣列中的每一個陣列元素
22     for (int i = 0; i <= ary1.GetUpperBound(0); i++)
23     {
24         Console.Write("{0} ", ary1[i]);
25     }
26     Console.WriteLine();
27     Console.WriteLine("\nary1陣列最小數為：{0}", GetMin(ary1) );
28     Console.Read();
29 }
```



## 5.7 遞迴 (Recursive)

- 遞迴方法(或稱遞迴函式)  
是指方法中再呼叫自己本身就構成遞迴。
- 像數學求階乘、排列、組合、數列等都可使用。
- 撰寫遞迴方法時必須在方法內有能離開方法的條件式，否則很容易造成無窮迴圈。



## 範例演練

- 西元1202年義大利數學家費波納西(Fibonacci)在他出版的「算盤全書」中介紹費波納西數列。
- 該數列最前面兩項係數都為1，其它項係數都是由位於該項係數前面兩項係數相加之和。該數列依序：1、1、2、3、5、8、13、21、...等以此類推下去。
- 當數字越大時，將前項數字除以緊接其後之數字，比值有逐漸向0.618 收斂。此比率就是「黃金比率」。
- 在大自然植物的花瓣、美學、建築、股票趨勢分析等都看到它的蹤影。現在就以此數列來撰寫一個遞迴方法，由鍵盤輸入一個正整數，若輸入8，下圖會顯示出費波納西數列的前八個係數：

```
file:///C:/CSharp/chap05/Recursive/bin/Debug/Recursive.EXE
=== 請輸入欲列印到第幾個費波納西係數：8
=== 費波納西數列的係數為：
1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 ,
```



FileName : Recursive.sln

01 //定義Fib方法可傳回第n個費波納西係數

02 **static int Fib(int n)**

03 {

04     **if (n == 1 || n == 2)**

05     {

06         **return 1;**

07     }

08     **else**

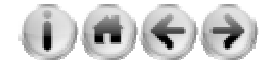
09     {

10         **return (Fib(n - 1) + Fib(n - 2));**

11     }

12 }

13



```
14 static void Main(string[] args)
15 {
16     int keyin;
17     Console.Write("=== 請輸入欲列印到第幾個費波納西係數 : ");
18     try
19     {
20         keyin = int.Parse(Console.ReadLine());
21         Console.WriteLine("\n=== 費波納西數列的係數為 : ");
22         Console.Write("  ");
23         for (int i = 1; i <= keyin; i++)
24         {
25
26             Console.Write("{0} , ", Fib(i));
27         }
28     }
29     catch (Exception ex)
30     {
31         Console.WriteLine(ex.Message);
32     }
33     Console.Read();
34 }
```






## 5.8 多載 (Overloading)

### 多載

是指多個方法(或函式)可以使用相同的名稱

 是透過不同的引數串列個數及引數的資料型別來加以區分。

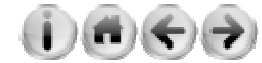


### 範例演練

定義兩個相同名稱的 **GetMin()** 方法：

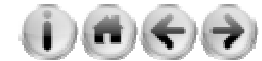
- ① 一個用來傳回兩個整數的最小數。
- ② 一個用來傳回整數陣列中的最小數。

```
file:///C:/CSharp/chap05/OverLoads/...  
5, 7最小數為5 :  
25, 6, 899, 3最小數為899 :
```



## FileName : OverLoads.sln

```
01 // 傳回 a, b 中的最小數
02 static int GetMin(int a, int b)
03 {
04     return (a < b ? a : b);
05 }
06
```



```
07 //傳回b陣列中的最小數
08 static int GetMin(int[] b)
09 {
10     int min = b[0];
11     for (int i = 0; i <= b.GetUpperBound(0); i++)
12     {
13         if (min < b[i])
14         {
15             min = b[i];
16         }
17     }
18     return min;
19 }
20 // -----
21 static void Main(string[] args)
22 {
23     Console.WriteLine("5, 7最小數為{0} : ", GetMin(5, 7));
24     int[] ary = new int[] { 25, 6, 899, 30 };
25     Console.WriteLine("25, 6, 899, 3最小數為{0} : ", GetMin(ary));
26     Console.Read();
27 }
```