

第二章 資料型別與主控台 應用程式輸出入



2.1 程式構成要素

2.5 主控台應用程式輸出入方法

2.2 常值與變數

2.6 Escape Sequence控制字元

2.3 如何宣告變數的資料型別

2.4 運算子與運算式

備註：可依進度點選小節



2.1 程式的構成要素

2.1.1 識別項

- 每個人一出生都需要取個名字來加以識別
- 在程式中所使用的變數、陣列、結構、函式、類別、介面和列舉型別等，也都需賦予名稱，以方便在程式中識別
- 名稱命名都需遵行識別項的命名規則。



■ 識別項的命名規則：


- ① 第一個字元需 Unicode 的大小寫字母、底線 _ 字元或中文字開頭，接在後面的字元可以是字母、數字或底線 _ 或中文字。中間不允許有空白出現。
- ② 識別項最大長度限16,383 個字元，不要太長以免難記且易造成輸入上的錯誤。
- ③ 一般關鍵字是不允許當作識別項，關鍵字前加前置字元@可當識別項。如 if 為關鍵字，@if 當識別項。
- ④ 識別項大小寫視為不同，如 tobe 和 ToBe 視為不同。
- ⑤ _pagecount、Part9、Number_Items 都是合法識別項。
- ⑥ 無效識別項：101Metro(數字開頭)，M&W(&字元)



2.1.2 陳述式 (Statement)

- 陳述式或稱敘述是高階語言所撰寫程式中最小可執行單位。
- 由一行一行的陳述式所成的集合構成一個程式(program)。
- 陳述式是由關鍵字、運算子、變數、常數及運算式等組合而成的。
- 撰寫程式時都是一行接一行由上而下撰寫。
- VC# 2008中每行敘述結尾是以「;」分號做區分，允許將多行敘述寫成一行，中間使用分號隔開即可，但為使程式方便閱讀建議不宜使用。



 + 為合併運算子，將前後雙引號括住的字串做字串合併，連接成字串後，顯示在訊息方塊中，寫法：
`MessageBox.Show("VC# 2008 is an" + "Object Oriented Language.");`

 將上面陳述式分成兩行書寫，在 + 運算子後面按  鍵跳至下一行繼續書寫 “Object Oriented Language.” 即可：
`MessageBox.Show("VC# 2008 is an" +
"Object Oriented Language.");`



2.1.3 關鍵字 (KeyWord)

- 所謂「關鍵字」或稱保留字(Reserve Word)，是對編譯器有特殊意義而預先定義的保留識別項。
- VC# 2008 系統所保留關鍵字，設計程式時不得拿來當作變數名稱。
- 若要使用關鍵字當變數時，必須在關鍵字最前面加上前置字元 @，才能當做程式中的識別項。
- 在撰寫 VC# 2008 程式時，若陳述式中有些字串以藍色字顯現，表示這些識別項就是 VC# 2008 的關鍵字。



abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	finally	do	double	else
enum	event	explicit	extern	false
finally	fixed	float	foreach	get
goto	if	in	int	interface
internal	is	long	namespace	new
null	object	operator	out	override
params	partial	private	protected	public
ref	return	sbyte	sealed	set
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	volatile
void	var	while	where	



2.2 常值與變數

- 電腦主要是用來處理資料。
- 依程式執行時該資料是否允許做 **四則運算**。分成：
 - ① 數值資料
 - ② 字串資料
- 依程式執行時資料是否具有 **變動性**，將資料分成
 - ① 常數(Constant)
 - ② 變數(Variable)



2.2.1 常數 (Constant)

■ 常數是程式執行過程中其值是**固定**無法改變。

■ 將常數細分成：

- ① 常值常數 (Literal Constant)
- ② 符號常數 (Symbolic Constant)

一. 常值常數

- 程式中直接以特定值的文數字型態存在於程式碼中稱為「常值常數」。
- 如：15、"Price" 等都屬於常值常數。
- 常值常數可為運算式一部份，也可指向一個符號常數或變數。



C# 允許使用的常值常數型別有：

常值常數

布林常值：true、false

整數常值：25, -30

浮點常值：24.5, 7.1E+10

字串常值："Hello", "24.5", "VC# 2008 從零開始"

字元常值："a", "8"

日期常值：#95/12/10#, 10:36PM#



二. 符號常數

- 符號常數是以有意義名稱來取代程式中不會改變的數字或字串。
- 是用來儲存應用程式執行過程中維持不變的值，不能像變數在程式執行過程中可變更其值或指派新值。
- 符號常數在程式中經宣告後，便無法修改或指派新的值。
- 符號常數是用 **const** 關鍵字及運算式來宣告並設定初值。



■ 符號常數宣告方式：

[存取修飾詞] const 資料型別 符號名稱 = [數值|字串|運算式];

可使用的關鍵字：

- ① public
- ② private (預設)
- ③ protected
- ④ internal

包括：

int/long/short
decimal
float/double
char/string
DateTime
bool/object

遵循識別項
的命名規則

中括號內擇一
不可省略



舉例說明：

```
const int DaysInYear=365;  
const int WeeksInYear=52;  
const int may=5;  
const double PI=3.1416;  
const bool pass=true;  
const string name="Viusal C# 2008" ;
```

多個符號常數一起宣告

```
const double x = 1.0, y = 2.0, z = 3.0;  
const int four = 4, five=5;
```

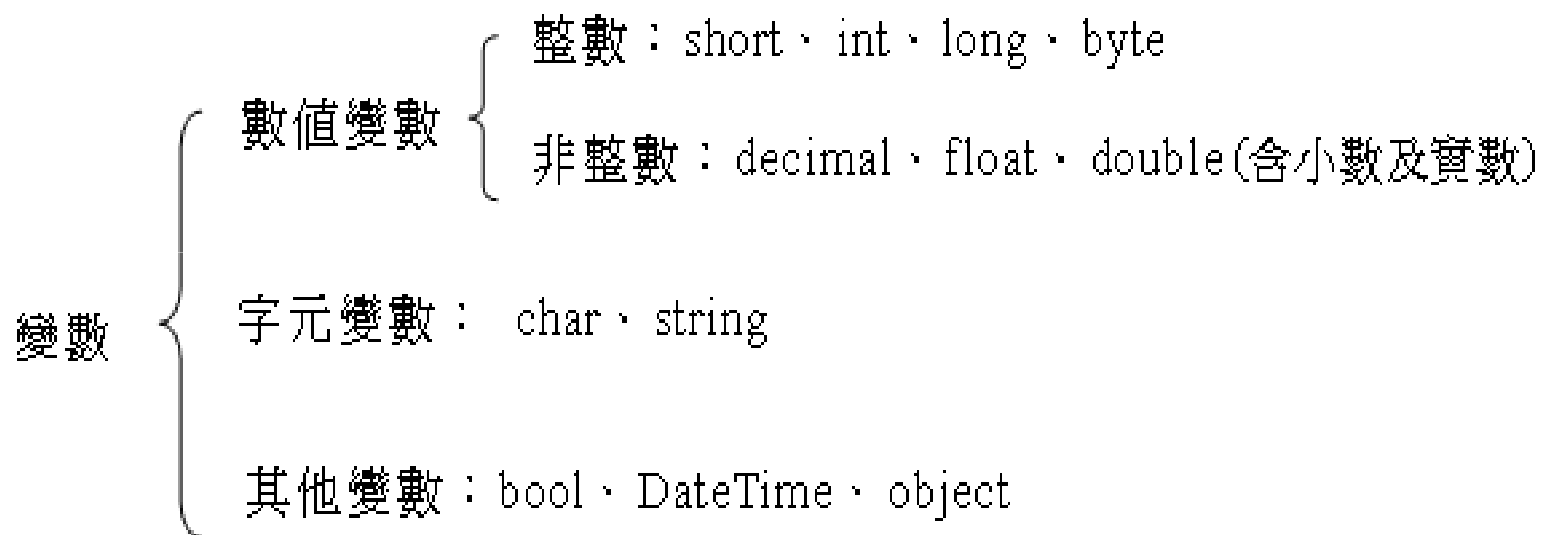


2.2.2 變數 (Variable)

- 一個變數代表一個特定資料項目或值，並在記憶體預留一個位置來儲存該資料內容。
- 程式執行時碰到變數，會到該變數位址，取出該值加以運算。
- 變數和常數不一樣，程式執行過程中常數的值是固定不變，而變數允許重複指定不同的值。
- 使用變數前必須先宣告
- 宣告變數同時必須給予變數名稱，命名方式則遵循識別項命名規則，同時要設定該變數的資料型別以方便在程式進行編譯時配置適當記憶空間存放變數的內容。



變數的種類





$X = Y + 10 ;$

① 10 是常數

記憶體儲存10的位址，其內容固定無法改變。

② X、Y 是變數

儲存 X 和 Y 的位址內容是可改變。



2.3 如何宣告變數的資料型別

- 程式中使用的變數必須事先經過宣告
- 經宣告過變數可知道該資料的資料型別
- 程式在編譯時便可保留適當的記憶體空間給該資料使用。
- 宣告變數的資料型別，語法：

資料型別 變數名稱 ；



VC# 2008允許使用下表的資料型別來宣告變數：

資料型別	大小	該資料型別有效範圍
byte 位元組	1 Byte	宣告：byte a; 大小：0至255 (無正負號8位元整數)
sbyte 位元組	1 Byte	宣告：sbyte a; 大小：-128~127 (帶正負號8位元整數)
short 短整數	2 Bytes	宣告：short a; 大小：-32,768~+32,767
ushort 短整數	2 Bytes	宣告：ushort a; 大小：0~65,535
int 整數	4 Bytes	宣告：int a; 大小：-2,147,483,648至+2,147,483,647
uint 整數	4 Bytes	宣告：uint a; 大小：0~4,294,967,295
long 長整數	8 Bytes	宣告：long a; 大小：-9,223,372,036,854,775,808至 +9,223,372,036,854,775,807
ulong 長整數	8 Bytes	宣告：ulong a; 大小：0~18,446,744,073,709,551,615



float 單精確度	4 Bytes	宣告： <code>float a;</code> 大小： $\pm 1.5 \times 10^{-45}$ 至 $\pm 3.4 \times 10^{38}$ 有效位數7位數。 若宣告float 變數同時初始化，可加上後置字元 <code>f</code> 或 <code>F</code> ，例如： <code>float x = 3.5F;</code>
double 倍精確度	8 Bytes	宣告： <code>double a;</code> 大小： $\pm 5.0 \times 10^{-324}$ 至 $\pm 1.7 \times 10^{308}$ 有效位數15~16。 指定運算子(=)右邊的實數常值預設為double，如果欲將整數當成 double，在整數常值加後置字元 <code>d</code> 或 <code>D</code> ，例如： <code>double x = 10D;</code>
decimal 貨幣	16 Bytes	宣告： <code>decimal a;</code> 整數： $\pm 1.0 \times 10e^{-28}$ 至 $\pm 7.9 \times 10e^{28}$ 有效位數28~29。 如果要將數字實數常值當成 decimal 處理，必須加上後置字元 <code>m</code> 或 <code>M</code> ，如： <code>decimal myCash=1000m;</code> 或 <code>decimal myCash=1000M;</code>



char 字元	2 Bytes	<p>宣告：<code>char a;</code></p> <p>大小：U+0000至U+ffff (Unicode 16 位元字元) 即0~65,535(不帶無正負號)</p> <p>可寫成字元常值、16進位逸出序列或Unicode表示。也可寫成轉換整數字元碼。下列為所有char 宣告變數方式，並且以字元 A 將其初始化：</p> <pre>char char1 = 'A'; // 字元常值表示 char char2 = '\x0041'; // 十六進制表示 char char3 = (char)65; // 由整數常值轉成字元常值 char char4 = '\u0041'; // Unicode</pre> <pre>Console.WriteLine("{0} {1} {2} {3}", char1, char2, char3, char4);</pre> <p>輸出結果為： A A A A</p>
string 字串	依實際 需要	<p>宣告：<code>string a;</code></p> <p>大小：大約0至20億(2^{31})個字元。</p> <p>兩種格式，以雙引號頭尾括住。</p>
bool 布林	2 Bytes	<p>宣告：<code>bool a;</code></p> <p>大小：<code>true</code>(真), <code>false</code>(假)</p>



DateTime 日期	8 Bytes	<p>宣告：<code>DateTime a;</code></p> <p>大小： 1年1月1日0:00:00 ~ 9999年12月31日11:59:59 PM。</p> <p>[例] 顯示 "<u>2009/9/11 下午 03:30:00</u>"</p> <pre>string MyString = "Sep 11, 2009 15:30 pm"; DateTime MyDateTime = DateTime.Parse(MyString); Console.WriteLine(MyDateTime); DateTime d1 = DateTime.Now; Console.WriteLine("{0}", d1);</pre>
object物件 (預設值)	4 Bytes	<p>宣告：<code>object a;</code></p> <p>大小：可儲存任何資料型別</p>



■ 宣告變數時能在同一行敘述中同時宣告多個變數，各變數間使用逗號隔開。

■ 如：同時宣告 a 和 b 是整數變數。寫法：

```
int a , b ;
```

■ 宣告變數時亦允許同時對變數初始化(設定初值)，寫法：

```
資料型別 變數名稱1=初值1, 變數名稱2=初值2, ... ;
```



[例1] 宣告 a 是一個整數變數且初值為10。寫法：

```
int a=10;
```

[例2] 宣告 a 是一個整數變數且初值為10；

宣告 b 為布林變數且初值為 false。寫法：

```
int a =10 ; bool b =false;
```

[例3] 宣告 a, b, c 是倍精確度變數，初值依序1.1, 2.2, 3.3

寫法：

```
double a=1.1, b=2.2, c=3.3;
```



2.4 運算子與運算式

2.4.1 運算子與運算元

- 運算子是指運算的符號
- 如四則運算的 $+$ 、 $-$ 、 \times 、 \div 、 \dots 。
- 程式中利用運算子可將變數、常數及函式形成一個運算式(Expression)
- 運算式就是由運算子和運算元組合而成的。
- 譬如：`price*0.05`



① 一元運算子(Unary Operator)

運算時，在運算子前面只需要一個運算元。

採前置標記法 (Prefix Notation)。如：- 5。

② 二元運算子(Binary Operator)

運算時在運算子前後各需一個運算元，採中置標記法 (Infix Notation)。如：5+8。

③ 三元運算子(Tenary Operator)

運算時需有三個運算子才能做運算，? ... : ...。語法：

變數=條件式 ? 值1 : 值2;

若條件式結果為 true，則將 值1 傳給變數，
否則將 值2 傳給變數。



[例] 檢查 score 成績變數是否及格？

若及格，將 “PASS” 傳給 str 字串變數，

若不及格，傳回 “DOWN”。

```
int score = 59;
```

```
string str = (score >= 60 ? "PASS" : "DOWN");
```

■ 若運算子按照特性分類分成六大類：

1. 算術運算子
2. 指定(複合)運算子
3. 關係運算子
4. 邏輯運算子
5. 合併運算子
6. 移位運算子

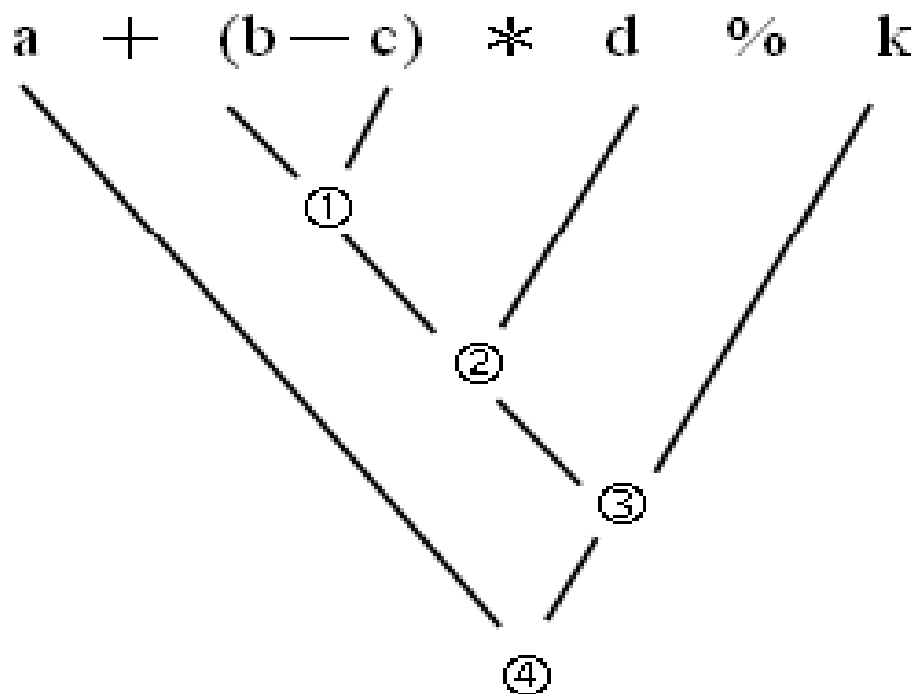


2.4.2 算術運算子

運算子	說明	範例
()	小括號	$10*(20+5)=250$
^	次方	$\text{Math.Pow}(5,3) \Rightarrow 125$
-	負號	-5 , $(-5)^3=-125$
*、/	乘、除	$5*6/2=15$ double num1; $\text{num1}=10/4 \Rightarrow 2$ $\text{num1}=10/4.0 \Rightarrow 2.5$ $\text{num1}=10.0/4 \Rightarrow 2.5$ $\text{num1}=10/3.0 \Rightarrow 3.33333333333333$
%	相除取餘數	$8 \% 5=3$, $12 \% 4.3=3.4$
+、-	加、減	$20-6+5=19$



■ $a + (b - c) * d \% k$ 運算式執行順序：





2.4.3 關係運算子

- 「關係運算子」亦稱「比較運算子」。
- 程式中遇到兩個數值或字串需做比較時，就要用到「關係運算子」。
- 關係運算子執行運算時需用到兩個運算元，且這兩個運算元必須同時是數值或字串方可比較，經過比較後會得到 **true**(真) 或 **false**(假)。
- 程式中可透過此種運算配合選擇結構陳述式來改變程式執行流程。



關係運算子：

運算子	運算式	範例
< 小於	$x < y$	$5 < 2 \Rightarrow \text{false}$
<= 小於等於	$x \leq y$	$5 \leq 2 \Rightarrow \text{false}$
> 大於	$x > y$	$5 > 2 \Rightarrow \text{true}$
>= 大於等於	$x \geq y$	$5 \geq 2 \Rightarrow \text{true}$
== 等於	$x = y$	$5 == 2 \Rightarrow \text{true}$
!= 不等於	$x <> y$	$5 != 2 \Rightarrow \text{false}$



2.4.4 邏輯運算子

- 一個關係運算式就是一個條件。
- 當有多個關係式要一起判斷時需用到邏輯運算子來連結。
- VC# 2008所提供的邏輯運算子如下：

1. and 邏輯運算子

A	B	A && B
true	true	true
true	false	false
false	true	false
false	false	false

- 例 $70 < \text{score} \leq 79$ 條件式：`(score) > 70 && (score <= 79);`



2. or 邏輯運算子

A	B	A B
true	true	true
true	false	true
false	true	true
false	false	false

[例] `score < 0` 或 `score ≥ 100` 條件式寫法：
`(score < 0) || (score ≥ 100)`



3. xor邏輯運算子

A	B	$A \wedge B$
true	true	false
true	false	true
false	true	true
false	false	false

4. not 邏輯運算子

A	$\neg A$
true	false
false	true



2.4.5 指定運算子

運算子	運算式	假設x=5, y=2 運算後的x值
=	$x=y$	$x=5$
+=	$x+=y$ 相當於 $x=x+y$	$x=5+2=7$
-=	$x-=y$ 相當於 $x=x-y$	$x=5-2=3$
=	$x=y$ 相當於 $x=x*y$	$x=5 \times 2=10$
/=	$x/=y$ 相當於 $x=x/y$	$x=5/2=2.5$
%=	$x\%=y$ 相當於 $x=x\%y$	$x=x\%2=1$
^=	XOR邏輯運算 $x^=y$	$x=x^y \rightarrow 7$ 運算過程如下： $\begin{array}{r} 0101_2=5_{10} \\ \text{Xor } 0010_2=2_{10} \\ \hline 0111_2 \rightarrow 7_{10} \end{array}$



運算子	運算式	假設x=5, y=2 運算後的x值
&=	AND邏輯運算 $x \&= y$	$x = x \& y \rightarrow 0$ 運算過程如下： $\begin{array}{r} 0101_2 = 5_{10} \\ \text{and } 0010_2 = 2_{10} \\ \hline 0000_2 \rightarrow 0_{10} \end{array}$
=	OR邏輯運算 $x = y$	$x = x y \rightarrow 7$ 運算過程如下： $\begin{array}{r} 0101_2 = 5_{10} \\ \text{Xor } 0010_2 = 2_{10} \\ \hline 0111_2 \rightarrow 7_{10} \end{array}$
<<=	$x <<= 1$ 相當於 $x = x << 1$ (左移一位即x乘以2)	$x = 5 << 2$ 左移兩位相當乘以4 $= 20$
>>=	$x >>= 1$ 相當於 $x = x >> 1$ (右移一位即x除以2)	$x = 5 >> 2$ 右移兩位相當除以4 $= 1$



2.4.6 合併運算子

■ + 符號除可當加法運算子外，也可用來合併字串。

■ 若 + 運算子前後的運算元都是

- ① 數值資料 (byte、short、int、long、float、double 或 decimal)，會視為加法運算處理，其結果為數值。
- ② 字串資料，視為合併運算子，將兩個運算元前後合併成一個字串。

■ 譬如：

```
string myStr ;  
int a ;  
myStr= "To be " + "Or Not to be";  
a = 20 + 30 ;
```



2.4.7 移位運算子

■ 移位運算子用在數值資料。

■ 對二進制正整數或帶小數整數。

① 該數值往左移一個位元，即將該數值乘2。

② 若往右移一個位元，即該數值除以2。

■ 可用移位運算子：

① <<：左移運算子

② >>：右移運算子

■ 譬如：

```
int a=10;
```

```
Console.WriteLine(a>>1); //  $10_{10}=1010_2 \rightarrow$  右移一位  $\rightarrow 0101_2=5_{10}$ 
```

```
Console.WriteLine(a<<2); //  $10_{10}=1010_2 \rightarrow$  左移兩位  $\rightarrow 101000_2=40_{10}$ 
```



2.4.8 運算子優先順序和順序關聯性

- 運算式的運算子優先執行先後順序是由運算子的優先順序和順序關聯性 (Associativity) 來決定的。
- 當運算式包含多個運算子時，運算子優先順序會控制評估運算式的順序。
- 如運算式 $x + y * z$ 的評估方式是 $x + (y * z)$ ，因為 $*$ 運算子的運算次序比 $+$ 運算子高。
- 下表由高至低列出各運算子優先執行順序；同列內運算子具有相同優先順序，且是依下表中第三欄指定方向進行運算：



優先次序	運算子	同一列運算子運算方向
1	◻、[]、註標	由內至外
2	+、-	由內至外
3	*、/	由左至右
4	%	由左至右
5	+、-	由左至右
6	&	由左至右
7	<<、>>	由左至右
8	=、 =、<、>、<=、>=	由左至右
9	! (Not 運算子)	由左至右
10	&& (And 運算子)	由左至右
11	(Or 運算子)	由左至右
12	^ (Xor 運算子)	由右至左
13	=、+=、-=、*=...	由右至左



2.5 主控台應用程式輸出入方法

- **Console(主控台)**是系統命名空間(NameSpace)內所定義的類別之一。
- **Console** 類別提供基本支援從主控台讀取和寫入字元的應用程式。
- 如：
Read 或 **ReadLine**方法提供由鍵盤輸入字元。
Write 或 **WriteLine**方法將資料顯示在螢幕上。



2.5.1 Write / WriteLine 方法

一. Console.Write() 方法

- 用來寫入標準輸出資料流，將接在Console.Write後面小括號內頭尾用雙引號括住的字串，顯示在螢幕目前插入點游標處，當字串顯示完畢插入點游標會停在字串的最後面。譬如將“光陰的故事”顯示在螢幕目前游標處，寫法如下：

```
Console.Write("光陰的故事");
```

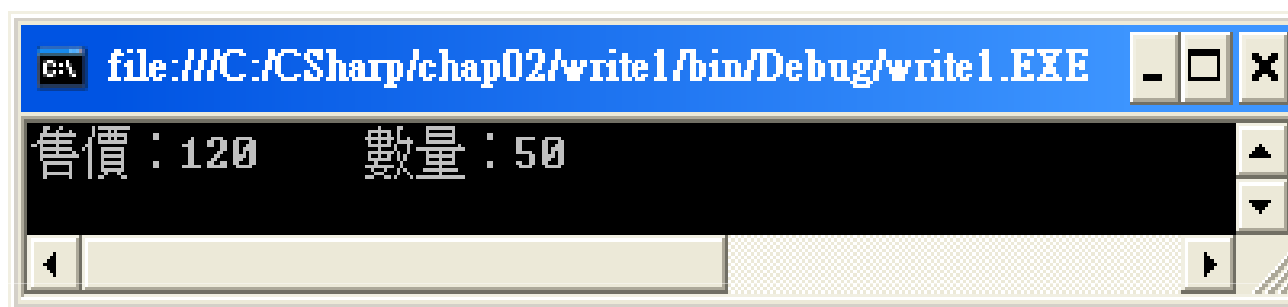
- 如果輸出的字串中間要插入指定的變數或運算式的內容。譬如：假設單價price整數變數值為120，數量為常數常值50，欲在目前游標處顯示“單價：120 數量：50”，寫法如下：

```
int price=120;  
Console.Write("售價：{0} 數量：{1}", price, 50);
```



範例演練

寫出輸出結果如下的程式碼。



```
10 static void Main(string[] args) //Main()方法為程式進入點
11 {
12     int price = 120;
13     Console.Write("售價：{0} 數量：{1}", price, 50);
14     Console.Read();
15 }
```



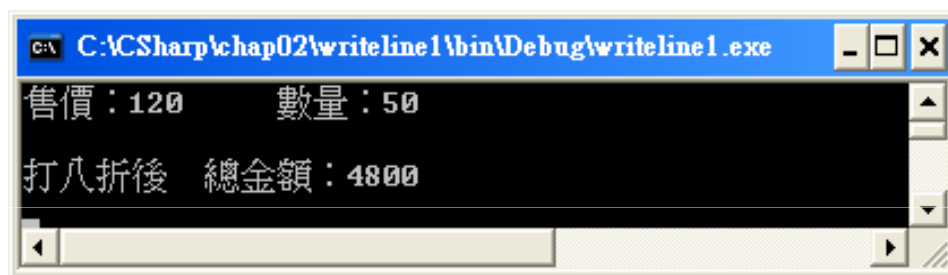
二. Console.WriteLine() 方法

- 功能和 **Console.Write()** 方法一樣，用來輸出資料流。
- 兩者的差異在**Console.WriteLine()** 方法會在輸出字串的最後面自動加入換行字元(Carriage Return)，使得游標自動移到下一行的最前面。
- 若接在 **Console.WriteLine()** 小括號內未加任何參數可用來空一行。



範例演練

製作下圖程式，欲顯示下圖範例輸出格式，必須使用三行 `Console.WriteLine()` 方法來書寫。




```
05 static void Main(string[] args)
06 {
07     //宣告整數變數price為120，整數變數qty為50
08     int price = 120, qty = 50;
09     //印出資料後游標往下移一行
10     Console.WriteLine("售價：{0} 數量：{1}", price, qty);
11     Console.WriteLine(); //游標往下移一行
12     Console.WriteLine("打八折後 總金額：{0}", price * qty * 0.8);
13     Console.Read(); //暫停畫面，等待使用者按下 [Enter] 鍵
14 }
```




2.5.2 Read / ReadLine 方法

一. Console.Read() 方法

- 從標準輸入裝置(鍵盤)讀取一個字元，不等待按  鍵便直接讀取。
- 使用時機是當需按任意鍵繼續時或只允許需要輸入一個字元時使用。
- 可將輸入字元放入 **char** 資料型別所宣告字元變數。



二. Console.ReadLine() 方法

- 從標準輸入裝置(鍵盤)讀取一整行字元，一直到按  鍵為止。
- 使用時機是當需要輸入字串時用，可將輸入字串放入 **string** 資料型別宣告的字串變數。
- 由於從 Console.ReadLine() 方法傳入的資料是屬**string**型別，若使用 Console.ReadLine() 方法將傳入的字串資料指定給整數變數時，必須使用 `int.Parse()` 或 `Convert.ToInt32()` 方法將字串轉成整數。



範例演練

試使用 `Console.ReadLine()` 方法取得由鍵盤輸入的書名、售價、數量，並使用 `int.Parse()` 或 `Convert.ToInt32()` 方法將售價和數量的資料轉成整數資料型別，最後再使用 `Console.WriteLine()` 方法將售價乘數量的金額顯示出來。

```
file:///C:/CSharp/chap02/readline1/bin/Debug/readline1.EXE

博碩電腦圖書廣場
=====
1. 書名: Visual C# 2008從零開始
2. 售價: 500
3. 數量: 6
=====
4. 金額: 3000
```



```
05  static void Main(string[] args)
06  {
07      string str1;
08      int price, qty;
09      Console.WriteLine();
10      Console.WriteLine(" 博碩電腦圖書廣場");
11      Console.WriteLine("=====");
12      Console.Write(" 1. 書名 : ");
13      str1 = Console.ReadLine(); // 輸入書名並指定給str1變數
14      Console.Write(" 2. 售價 : ");
15
16      price = int.Parse(Console.ReadLine());
17      Console.Write(" 3. 數量 : ");
18
19      qty = Convert.ToInt32(Console.ReadLine());
20      Console.WriteLine("=====");
21      Console.WriteLine(" 4. 金額 : {0}", price * qty);
22      Console.Read();
23  }
```




2.6 Escape sequence 控制字元

- 在 C# 中若欲印出「'」單引號、「"」雙引號或「\」倒斜線等符號，需使用「逸出序列」來達成。
- 當編譯器遇到這些逸出字元時，將使得接在倒斜線字元（\）後的字元，被當成某種特殊意義的符號來處理。

**[簡例]**

```
Console.WriteLine("\光陰的故事"); //印出 "光陰的故事"  
Console.WriteLine("Jack's Wang"); //印出 Jack's Wang  
Console.WriteLine("Why 1\\2");      // Why 1\2
```

逸出序列	說明
\'	插入一個單引號
\"	插入一個雙引號
\\	插入一個倒斜線，當程式定義檔案路徑時有用
\a	觸發一個系統的警告聲
\b (Backspace)	退一格
\f (Form Feed)	跳頁
\n (New line)	換新行
\r (Return)	游標移到目前該行的最前面。
\t (Tab)	插入水平跳格到字串中
\u d d d d	插入一個 Unicode 字元
\v	插入垂直跳格到字串中
\0 (Null space)	代表一個空字元