第2章 資料型態、變數與運算子

- 資料，是任何事件的核心。
  - 一個事件隨著狀況不同，會產生不同資料及因應之道
  - 例一：隨著交通事故通報資料的嚴重與否，交通警察大隊派遣處理事故的人員會有所增減
  - 例二：隨著年節的到來與否，鐵路局對運送旅客的火車班次會有所調整
- 對不同事件所處理的資料之型態也就也不盡相同
  - 例一：對乘法「*」事件　　處理的資料一定為數字
  - 例二：對「輸入姓名」事件　處理的資料一定為文字
- 了解資料的基本型態是學習程式設計的基本課題

2-1 資料型態

- 資料處理包括資料輸入、資料運算及資料輸出
- 程式中使用的資料，都儲存在記憶體位址中
  - 設計者是透過變數名稱來存取記憶體中的對應資料，而這個變數名稱就相當於某個記憶體位址的代名詞
- 變數型態：
  - 實值型態（Value Type）
  - 參考型態(Reference Type)

■ 實值型態：

1. 整數型態
2. 浮點數型態
3. char(字元)型態
4. bool(布林)型態
5. enum(列舉)型態
6. struct(結構)型態

■ 整數型態：

1. sbyte(帶正負號的位元組整數)
2. byte(不帶正負號的位元組整數)
3. short(帶正負號的短整數)
4. ushort(不帶正負號的短整數)
5. int(帶正負號的整數)
6. uint(不帶正負號的整數)
7. long(帶正負號的長整數)
8. ulong(不帶正負號的長整數)

- 浮點數型態
  1. float(單精度浮點數)
  2. double(倍精度浮點數)
- char型態的資料，除了可依據C/C++的規則呈現外，還可直接以16bits Unicode編碼方式來表示
- bool型態的資料，其內容只能「true」或「false」
- 每一個基本資料型態變數，一次只能儲存一項資料

- 參考型態變數儲存的不是它所指向的資料，而是此資料所在的記憶體位址之起始位置。透過參考資料型態變數中的記憶體位址，才能存取該資料

- 參考型態變數
  - String(字串)變數
    - 請參考「第六章 內建類別」
  - 陣列變數
    - 請參考「第七章 陣列」
  - class(類別)變數
    - 請參考「第六章 內建類別」和「第九章 自訂類別」
  - interface(介面)變數
    - 請參考「第十一章 抽象類別和介面」

2-1-1 整數型態

■ 整數:不帶有小數點的數字

表2-1 整數型態所佔用的記憶體空間及範圍

資料型態	佔用記憶體空間	資料範圍
sbyte	1 byte	-128至127
byte	1 byte	0至255
short	2 byte	-32768至32767
ushort	2 byte	0至65535
int	4 byte	-2147483648至2147483647
uint	4 byte	0至4294967295
long	8 byte	-9,223372036854775808至 9,223372036854775807
ulong	8 byte	0至 18446744073709551615

- 整數運算時，通常是以十進位方式來表示整數
  - 在有些特殊的狀況下，被要求以二進位方式或十六進位方式來表示整數
  - 二進位表示整數的方式，是直接在數字前加上「0b」或「0B」
  - 十六進位表示整數的方式，是直接在數字前加上「0x」或「0X」
- 例：宣告兩個整數變數b及h，且b的初值為6_{10}，h的初值為58_{10}。以二進位方式表示b的值，十六進位方式表示h的值。

解：
```
int b=0b110; // 或int b=0B110;  6₁₀ 等於 110₂
int h=0x3a;  // 或int h=0X3a;  58₁₀ 等於 3a₁₆
```

- 若一整數常數無特別註明，則預設為int型態。
  - 例：1234是int型態
- 若一整數常數想要代表long(長整數)，則必須在數字後面加上L或l
  - 例：56L或56l是long型態
- 若一short整數常數超過short整數資料型態的範圍，則編譯時會產生以下錯誤訊息：

  常數值 'xxx' 不可轉換成 'short
  (常數識別字xxx 的值超過short 型態的範圍)
  - 例：若將32768 存入short型態的變數中，則會出現錯誤訊息「常數值 '32768' 不可轉換成 'short'」
  - 其他的整數資料型態，也有類似狀況出現

2-1-2 浮點數型態

- float(單精度浮點數)：帶有小數點的數字。電腦會提供4個位元組(byte)的記憶體空間給float型態的資料存放
- double(倍精度浮點數)：帶有小數點的數字。電腦會提供8個位元組(byte)的記憶體空間給double型態的資料存放

2-1-2 浮點數型態

■ 浮點數資料型態所佔用的記憶體空間及約略範圍

資料型態	佔用記憶體空間	資料約略範圍
float	4 byte	$-3.4028235*10^{38}$至$-1.4*10^{-45}$ 與 $1.4*10^{-45}$至$3.4028235*10^{38}$
double	8 byte	$-1.79769313486231 57*10^{308}$至 $-4.9*10^{-324}$ 與 $4.9*10^{-324}$至$1.79769313486231 57*10^{308}$

- 若一float型態的常數(例：**4e+38F**)超過float型態的範圍，則編譯時會產生錯誤訊息：

  浮點常數的值超出類型 'float' 的範圍

  - float型態的資料儲存時，一般只能準確6~7位(整數位數+小數位數)
- 若一double型態的常數(例：4e+308)超過double型態的範圍，則編譯時會產生錯誤訊息：

  浮點常數的值超出類型 'double' 的範圍

  - double型態的資料儲存時，一般只能準確14~15位(整數位數+小數位數)
- 有關浮點數準確度，請參考「3-3 發現問題」之「範例4」

- 浮點數型態的資料，呈現方式有下列兩種：
  - 以一般常用的小數點方式來呈現
    - 例：9.8、-3.14、1.2f 、-5.6F
  - 以科學記號方式來呈現
    - 例：-2.38e+01、5.143E+21

2-1-3 字元型態(char)

- 若文字資料的內容，只有一個中文字或一個英文字母或一個全形字或一個符號，則稱此文字資料為字元(char)型態資料
  - char型態資料，必須在放在一組單引號「'」中
  - 有一些具有特殊意義的字元(例，雙引號「"」)或用來指定螢幕游標的動作(例，定位鍵「Tab」)，必須以一個反斜線「\」，後面加上該字元或該字元所對應的Unicode(萬國碼)，才能顯示在螢幕上或產生指定的動作這種組合方式，稱為「逸出序列」(Escape Sequence)
    - 逸出序列相關說明，參考「表2-3 常用的逸出序列」

表2-3 常用的逸出序列

逸出序列	作用	對應的十進位ASCII碼	對應的十六進位Unicode碼
\n	換列字元(New Line)：讓游標移到下一列的開頭	10	'\u000A'
\b	倒退字元(Backspace)：讓游標往左一格，相當於按「Backspace」鍵	8	'\u0008'

逸出序列	作用	對應的十進位ASCII碼	對應的十六進位Unicode碼
\t	水平跳格字元(Horizontal Tab)，讓游標移到下一個定位格，相當於按「Tab」鍵	9	'\u0009'
\r	歸位字元(Carriage Return)：讓游標移到該列的開頭，相當於按「Home」鍵	13	'\u000D'

逸出序列	作用	對應的十進位ASCII碼	對應的十六進位Unicode碼
\"	顯示雙引號「"」	34	'\u0022'
\'	顯示單引號「'」	39	'\u0027'
\\	顯示反斜線「\」	92	'\u005C'

[註]
C#語言預設定位格位置為水平的
1,9,17,25,33,41,49,57,65,73

- 在C#環境中，char型態資料是以16位元的無號數整數Unicode碼來表示，而不是以8位元的無號數整數ASCII碼來表示
- 無論是一個中文字或一個英文字母或一個全形字或一個符號，都是以一個Unicode碼來表示字元
- Unicode碼的範圍，介於'\u0000'到 '\uFFFF'間
- 「中文」字元所對應的Unicode碼區間為[4e00,9eae]
- 英文字元對應的Unicode碼區間為[0041,005A]及[0061,007A](以十進位表示，則在[65,90]及[97,122])

- 數字字元對應的Unicode碼區間為[0030,0039](以十進位表示，則在[48,57])
- 「符號」字元對應的Unicode碼區間為[0021,002F]，[003A,0040]，[005B,0060]及[007B,007E](以十進位表示，則在[33,47]，[58,64]，[91,96]及[123,126])
- 「空白」字元對應的Unicode碼為[0020] (以十進位表示，則為32)
  - 詳細的Unicode編碼資訊，請參考「http://www.unicode.org/charts/PDF/」

- char型態資料的表示法有以下三種方式：
  1. 直接表示法:
     - 例，'0'、'A'、'a'、'—'、'{'等
  2. 以「(char) 十進位整數」方式，來表示所對應的字元
     - 例，(char) 48 表示'0'、(char)65 表示'A'、(char) 97 表示'a'、(char) 19968 表示' 一'、(char) 123 表示'{'
  3. 以「\u」開始，後面跟著十六進位的 Unicode碼來表示所對應的字元:
     - 例，'\u0041' (表示 'A')、'\u4e00'(表示 '一')
- [註] 對非ASCII字元集或鍵盤上沒有的字元或符號，可使用方式2或3來表示

- 若文字資料的內容超過一個字元，則稱此文字資料為string(字串)型態資料
- string 型態資料，必須放在一組雙引號「"」中
  - 例：「早安」應以「" 早安"」表示，「morning」應以「"morning"」表示
  - 字串相關說明，請參考「6-4　字串類別之屬性及方法」

2-1-4　布林型態

- 若資料的內容只能是「true」或「false」，則稱之為bool(布林)型態的資料
  - 系統會提供1 個位元組(byte)的記憶體空間給bool型態的資料存放
  - bool 型態的資料，是作為判斷條件是否為「true」(真)或「false」(假)之用

2-1-5　列舉型態(enum)

- Visual C# 中的enum 關鍵字，可提供使用者來制定一種稱為「列舉」的資料型態
  - 自訂列舉型態的目的，是定義一組常數整數值來限制列舉型態的範圍
    - 每一個常數整數值，是使用一個有意義的名稱來表示
    - 這一組有意義的名稱，是這個自訂列舉型態的列舉成員
- 自訂列舉型態，是以關鍵字enum 為首，其語法如下：

public enum 列舉名稱 {列舉成員1, 列舉成員2,… };

- 註：
  - 關鍵字public 表示列舉名稱中的列舉成員為公開的，任何地方都可使用
  - 列舉成員1，列舉成員2，...等之資料型態，都預設為int
  - 若未設定「列舉成員1」的值，則預設為0
  - 某個列舉成員的值等於前一個列舉成員的值加1
  - 列舉型態名稱定義的位置，若在命名空間內，則命名空間中的所有類別都可以存取該列舉型態名稱的列舉成員；若在類別或結構中，則只能在同一個類別或結構中存取該列舉型態名稱的列舉成員
  - 列舉型態名稱不可定義在方法中

- 列舉成員的使用語法，視需求可選擇下列8種方式之一
  - (sbyte)　　列舉型態名稱.列舉成員名稱
  - (byte)　　列舉型態名稱.列舉成員名稱
  - (short)　　列舉型態名稱.列舉成員名稱
  - (ushort)　　列舉型態名稱.列舉成員名稱
  - (int)　　列舉型態名稱.列舉成員名稱
  - (uint)　　列舉型態名稱.列舉成員名稱
  - (long)　　列舉型態名稱.列舉成員名稱
  - (ulong)　　列舉型態名稱.列舉成員名稱

- 例： 定義列舉型態Season，且其成員有Spring，Summer，Fall 及Winter，分別代表春、夏、秋及冬。定義敘述如下：

```
public enum Season {Spring, Summer, Fall, Winter};
```

  - 註：

    因Spring 沒設初始值，故

    (int) Season.Spring　　為0

    (int) Season.Summer　　為1

    (int) Season.Fall　　為2

    (int) Season.Winter　　為3

- 定義列舉型態名稱之後，就能宣告資料型態為列舉名稱的列舉變數
- 列舉變數的宣告語法，依是否要設定初始值，分成下列兩種方式：
  1. 列舉型態名稱 列舉變數;
  2. 列舉型態名稱 列舉變數 =

     列舉型態名稱.列舉成員名稱;
- 承上例，宣告資料型態為Season 的列舉變數s，其宣告語法如下：

```
Season s;
```

- 承上例，宣告資料型態為Season 的列舉變數s，且初始值為Fall，其宣告語法如下：

```
Season s = Season.Fall;
```

  - 若要知道列舉變數s代表列舉型態Season 中的哪一成員，及其所代表的常數值，則可使用下列語法輸出結果：

```
Console.Write("{0}所代表的常數值 = {1}",s , (int) s);
```

  - 結果: Fall所代表的常數值 = 1

2-2 常數與變數宣告

- 程式執行時，無論是輸入的資料或產生的資料，它們都是存放在電腦的記憶體中。但我們並不知道資料是放在哪一個記憶體位址，
- 如何存取記憶體中的資料呢？
  - 大多數的高階語言，都是透過常數識別字或變數識別字存取其所對應的記憶體中之資料
  - 程式設計者自行命名的常數(Constant)、變數(Variable)、方法(Method)、類別(Class)及介面(Interface)等名稱都稱為識別字(Identifier)

- 識別字命名規則如下：

1.識別字名稱必須以A~Z，a~z，_（底線），或「中文字」為開頭，但不建議以中文字開頭

2. 識別字名稱的第二個字（含）開始，只能是A~Z，a~z，_，0~9，或「中文字」，但不建議是中文字

- [註] 儘量使用有意義的名稱當作識別字名稱

3. 一般識別字命名的原則如下：

- 命名空間、介面、類別、結構、屬性、方法、事件等名稱的字首為大寫
- 若名稱由多個英文單字組成，則採用英文大寫駱駝式（upper camel case）的命名方式
- 例：CarStructure、TestType

- 常數名稱以大寫英文為主
- 其他識別字的字首為小寫。若名稱由多個英文單字組成，則採用英文小寫駱駝式（lower camel case）的命名方式
  - 例： getOptimalSoution、myAge

4. 識別字名稱有大小寫字母區分
   - 若英文字相同但大小寫不同，則這兩個識別字是不同的
5. 不能使能關鍵字當作其他識別字名稱
   - 關鍵字為編譯器專用的識別字名稱，每一個關鍵字都有其特殊的意義
   - 若要使用關鍵字作為識別字名稱，則必須在關鍵字前加上「@」

■ 表2-4　Visual C# 語言的關鍵字

abstract	**as**	**base**	**bool**	**break**	**byte**
case	**catch**	**char**	**checked**	**class**	**const**
continue	**decimal**	**default**	**delegate**	**do**	**double**
else	**enum**	**event**	**explicit**	**extern**	**false**
finally	**fixed**	**float**	**for**	**foreach**	**goto**
if	**implicit**	**in**	**int**	**interface**	**internal**
is	**lock**	**long**	**namespace**	**new**	**null**

object	**operator**	**out**	**override**	**params**	**private**
protected	**public**	**readonly**	**ref**	**return**	**sbyte**
sealed	**short**	**sizeof**	**stackalloc**	**static**	**string**
struct	**switch**	**this**	**throw**	**true**	**try**
typeof	**uint**	**ulong**	**unchecked**	**ushort**	**using**
virtual	**void**	**volatile**	**while**		

例：_a、b1、c_a_2及@if，為合法的識別字名稱

例：1a、%b1、c?a_2及if，為不合法的識別字名稱

- 常數識別字(Constant Identifier)與變數識別字(Variable Identifier)
  - 都是用來存取記憶體中之資料
  - 常數識別字儲存的內容是固定不變的，而變數識別字儲存的內容可隨著程式進行而改變

- C#是限制型態式的語言
  - 當我們要存取記憶體中的資料內容之前，必須要先宣告常數識別字或變數識別字，電腦會配置適當的記憶體空間給它們，接著才能對其所對應的記憶體中之資料進行各種處理；否則會出現類似以下的錯誤訊息：

    名稱 'xxx' 不存在於目前的內容中
    - 其中的xxx，代表常數識別字名稱或變數識別字名稱。

- 常數識別字的宣告語法如下：

[存取修飾詞] const 資料型態 常數名稱1 = 數值(或文字)運算式[,常數名稱2 = 數值(或文字)運算式 ,…] ;

- 註：
  - [存取修飾詞]，表示「存取修飾詞」可視需要填入適當的關鍵字
  - 常用的存取修飾詞有public，protected 及private
  - 若存取修飾詞為public，則常數識別字可在不同的類別中被使用

- 若存取修飾詞為protected，則常數識別字能在它所屬的類別，以及所屬類別的子類別中被使用
- 若存取修飾詞為private 或不寫，則常數識別字只能在它所屬的類別中被使用
- 常數識別字宣告的位置，若在類別或結構內，則類別或結構中都可使用該常數識別字；若在類別或結構中的方法內，則只能在類別或結構中的方法內使用該常數識別字
- 例：宣告一個常數識別字為PI，且值為3.14的常數
  - 解： **const** float PI =3.14f; // 或 3.14F

- 變數識別字的宣告語法如下：
  - **方式1：資料型態 變數1[,變數2,…,變數n];**
  - **方式2：資料型態 變數1=初始值**
    [,變數2=初始值,…];
- [註] [ ]表示若同時宣告多個基本資料型態相同的變數，則必須利用「,」(逗號)將不同的變數名稱隔開；否則可去掉
  - 例：宣告兩個整數變數a及b
  - 解：int a,b;

- 例：宣告五個變數，其中i為整數變數，f為單精度浮點數變數，d為倍精度浮點數變數，c為字元變數，b為布林變數。且i的初值為0，f的初值為0.0f，d的初值為0.0，c的初值為'A'，b的初值為false。
  - 解：

```
int i=0;
//浮點數，C#語言預設double型態
//數字要設定為float型態，必須在數字後加上f或F
float f=0.0f;
double d=0.0;
char c='A';
bool b=false;
```

- 例：宣告三個整數變數b及h，且b的初值為6_{10}及h的初值為58_{10}。分別以二進位方式表示b的值及十六進位方式表示h的值。
  - 解：

```
int b=0b110; // 或int b=0B110;  6(10) 等於 110(2)
int h=0x3a;  // 或int h=0X3a;  58(10) 等於 3a(16)
```

- 宣告常數識別字或變數識別字的主要目的
  - 是告訴編譯器要配置多少記憶空間給常數識別字或變數識別字使用
  - 及以何種資料型態來儲存常數識別字或變數識別字的內容

- C#語言對記憶體配置方式有下列兩種：

1.靜態配置記憶體：是指在編譯階段時，就為程式中所宣告的變數配置所需的記憶體空間

例： double x = 3.14; // 靜態記憶體配置

- 宣告x 為倍精度浮點數變數時，編譯器會分配8bytes 的記憶體空間給變數x 使用，如上圖所示0x005e6888~0x005e6890(假設)

- C#語言對記憶體配置方式有下列兩種(續)：

  2.動態配置記憶體：是指在執行階段時，程式才動態宣告陣列變數的數量，並向作業系統要求所需的記憶體空間

  - 請參考「第七章 陣列」的「範例14」

2-3 資料運算處理

- 利用程式來解決日常生活中的問題，若只是資料輸入及資料輸出，而沒有做資料處理或運算，則程式執行的結果是很單調的
  - 為了讓程式每次執行的結果都不盡相同，程式中必須包含輸入資料，並加以運算處理
- 資料運算處理，是以運算式的方式來表示

2-3 資料運算處理

- 運算式，是由運算元(Operand)與運算子(Operator)所組合而成
  - 運算元可以是常數、變數、方法或其他運算式
  - 運算子包括指定運算子、算術運算子、遞增遞減運算子、比較(或關係)運算子、邏輯運算子，及位元運算子

- 運算子以其相鄰運算元的數量多寡來分類，有一元運算子(Unary Operator)、二元運算子(Binary Operator) 及三元運算子(Triple Operator)
- 結合算術運算子的運算式，稱之為算術運算式；結合比較(或關係)運算子的運算式，稱之為比較(或關係)運算式；結合邏輯運算子的運算式，稱之為邏輯運算式；…以此類推
- 例：a-b*2+c / 5 % 7 + 1.23*d
  - 其中「a」、「b」、「2」、「c」、「5」、「7」、「1.23」及「d」為運算元
  - 而「-」、「+」、「*」、「/」及「%」為運算子

2-3-1 指定運算子（＝）

- 指定運算子「＝」的作用，是將「＝」右方的值指定給「＝」左方的變數
  - 「＝」的左邊必須為變數，右邊則可以為變數、常數、方法或其他運算式
- 例：(程式片段)

```
sum=0; //將0指定給變數sum
//將變數a及變數b相加後除以2的結果，指定給變數
avg=(a+b)/2;
```

2-3-2 算術運算子

- 一般與數值運算有關的運算子有算術運算子、遞增運算子及遞減運算子三種
  - 假設a=-2，b=23

運算子	幾元運算子	作用說明	例子	結果	注意事項
+	二元運算子	將兩數字相加	a + b	21	數字可以是整數或浮點數
-	二元運算子	將兩數字相減	a - b	-25	數字可以是整數或浮點數

*	二元運算子	將兩數字相乘	a * b	-46	數字可以是整數或浮點數
/	二元運算子	將兩數字相除	b / 2 b / 2.0	11 11.5	[註解]

[註解]

1. 整數相除，結果為整數
2.整數相除時，若分母為0，則編譯時會產生「除以常數零」的錯誤訊息
3. 數字為浮點數時，相除結果為浮點數
4.浮點數相除時，若分子與分母都為0.0，則結果為「不是一個數字」；若分子>0，分母為0.0，則結果為「正無限大」；若分子<0，分母為0.0，則結果為「負無限大」。

%	二元 運算子	兩數相除 之 餘數	b % 3 b % 2.5	2 0.5	數字可以是整數或浮點數
+	一元 運算子	將數字 乘以「+1」	+(a)	-2	數字可以是整數或浮點數
-	一元 運算子	將數字 乘以「-1」	- (a)	2	數字可以是整數或浮點數

2-3-3 遞增運算子(++)及遞減運算子(--)

- 遞增運算子「++」及遞減運算子「--」的作用，分別是對數字資料做「+1」及「-1」的處理
- 假設a=10

運算子	幾元運算子	作用說明	例子	結果	注意事項
++	一元運算子	將變數值+1	a++; ++a;	11 11	參考[註解]
--	一元運算子	將變數值-1	a--; --a;	9 9	

- [註解]
  - 若運算式中含有++ 及其他運算子，則++ 放在變數之前與之後，其執行的順序是不同的，且運算式的結果也不同
    - 請參考範例1 及範例2
  - 若運算式中含有-- 及其他運算子，則-- 放在變數之前與之後，其執行的順序是不同的，且運算式的結果也不同
    - 請參考範例3 及範例4

範例1：後置型的++(遞增)運算子應用

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace Ex1
8   {
9       class Program
10      {
11          static void Main(string[] args)
12          {
13              int a = 0, b = 1, c;
14              c = a++ + b; // 先處理c=a+b;，然後再處理a++;
15              Console.Write("a=" + a + " , c=" + c);
16              Console.ReadKey();
17          }
18      }
19  }
```

執行結果	a=1 , c=1

範例2：前置型的++(遞增)運算子應用

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace Ex2
8   {
9       class Program
10      {
11          static void Main(string[] args)
12          {
13              int a = 0, b = 1, c;
14              c = ++a + b; // 先處理++a;，然後再處理c=a＋b;
15              Console.Write("a=" + a + " , c=" + c);
16              Console.ReadKey();
17          }
18      }
19  }
```

執行結果	a=1 , c=2

範例3：後置型的--(遞減)運算子應用

範例4：前置型的--(遞減)運算子應用

(自行參考書籍中範例)

2-3-4 比較(或關係)運算子

- 比較運算子的作用是用來判斷資料間的關係
  - 即，何者為大，何者為小，或兩者一樣
- 若問題中提到條件或狀況，則必須配合比較運算子來處理
- 比較運算子通常撰寫在「if」選擇結構，「for」或「while」迴圈結構的條件中
  - 請參考「第四章 程式之設計模式-選擇結構」及「第五章 程式之設計模式——迴圈結構」

2-3-4 比較(或關係)運算子

■ 假設a=2，b=1

運算子	幾元運算子	作用	例子	結果
>	二元運算子	判斷「>」左邊的資料是否大於右邊的資料	a > b	true
<	二元運算子	判斷「<」左邊的資料是否小於右邊的資料	a < b	false
>=	二元運算子	判斷「>=」左邊的資料是否大於或等於右邊的資料	a >= b	true

<=	二元運算子	判斷「<=」左邊的資料是否小於或等於右邊的資料	a <= b	false
==	二元運算子	判斷「==」左邊的資料是否等於右邊的資料	a == b	false
!=	二元運算子	判斷「!=」左邊的資料是否不等於右邊的資料	a != b	true

- 各種比較運算子的結果不是「false」就是「true」
  - 「false」表示「假」；「true」表示「真」。

2-3-5 邏輯運算子

- 邏輯運算子的作用，是連結多個比較(或關係)運算式來處理更複雜條件或狀況的問題
- 若問題中提到多個條件(或狀況)要同時成立或部分成立，則必須配合邏輯運算子來處理
- 邏輯運算子通常撰寫在「if」選擇結構，「for」或「while」迴圈結構的條件中
  - 請參考「第四章 程式之設計模式-選擇結構」及「第五章 程式之設計模式-迴圈結構」

■ 假設a=1，b=2，c=3

運算子	幾元運算子	作用說明	例子	結果
&&	二元運算子	判斷「&&」兩邊的運算式結果，是否都為「true」	a>3 && b<2	false
||	二元運算子	判斷「||」兩邊的運算式結果，是否有一個為「true」	a>3 || b<=2	true
^	二元運算子	判斷「^」兩邊的比較運算式結果，是否一個為「true」一個為「false」	(a>3) ^ (b<2)	true

- 假設a=1，b=2，c=3

運算子	幾元運算子	作用說明	例子	結果
!	一元運算子	判斷「!」右邊的運算式結果，是否為「false」	!(a>3)	true

- **運算式，可以是比較運算式或邏輯運算式**
- 各種邏輯運算子的結果，不是「false」，就是「true」
  - 「false」表示「假」；「true」表示「真」

■ 真值表是比較運算式在邏輯運算子「&&」，「||」，^或「!」處理後的所有可能結果

表2-9 &&，|| ，^及！運算子之真值表

&&(且)運算子		
A	B	A && B
true	true	true
true	false	false
false	true	false
false	false	false

^（互斥或）運算子		
A	B	A ^ B
true	true	false
true	false	true
false	true	true
false	false	false

||(或)運算子		
A	B	A || B
true	true	true
true	false	true
false	true	true
false	false	false

!(否定)運算子	
A	!A
true	false
false	true

2-4 運算子的優先順序

- 不管哪一種運算式，式子中一定含有運算元與運算子
- 運算處理的順序是依照運算子的優先順序為準則
  - 運算子的優先順序在前的，先處理
  - 運算子的優先順序在後的，後處理

表2-11 常用運算子的優先順序

運算子 優先順序	運算子	說明
1	( )	括號
2	+，-，++， --，!，~	取正號，取負號，前置型遞增， 前置型遞減，邏輯否，位元否
3	*，/，%	乘，除，取餘數
4	+，-	加，減
5	<<，>>	位元左移，位元右移
6	>，>=， <，<=，	大於，大於或等於， 小於，小於或等於

運算子 優先順序	運算子	說明
7	==，!=	等於，不等於
8	&	位元且
9	^	位元互斥或
10	|	位元或
11	&&	邏輯且
12	||	邏輯或
13	=，+=，-=，*=，/=， %=，&=，^=，|=，<<=， >>=	指定運算及各種複合指定運算
14	++，--	後置型遞增、後置型遞減

2-5 資料型態轉換

- 當不同型態的資料放在運算式中，資料是如何運作?
- 資料處理的方式有下列兩種方式：
  - 自動轉換資料型態(或隱式型態轉換：Implicit Casting)：
    - 由編譯器來決定轉換成何種資料型態
    - C#編譯器會將數值範圍較小的資料態型轉換成數值範圍較大的資料型態
    - 數值資料型態的範圍由小到依序為int (sbyte、byte 、 char、short及ushort ，也都歸類在int型態)、long (uint也歸類在long型態)、float、double

■ 自動轉換資料型態例子(程式片段)

```
char c='A';
int i=10;
float f=3.6f;
double d;
d=c+i+f ;
Console.Write(d);
//將c值轉換為整數65，再執行65+i → 75
//將75的值轉換為單精度浮點數75.0，
//再執行75.0+f → 78.6
//最後將單精度符點數78.6轉換為倍精度浮點數78.6
//並指定給d，結果d=78.5999999046326
//(輸出浮點數，有時會有誤差)
```

- 強制轉換資料型態(或顯式型態轉換：Explicit Casting)：
  - 由設計者自行決定轉換成何種資料型態
  - 當問題要求的資料型態與執行結果的資料型態不同時，設計者就必須對執行結果的資料型態做強制轉換

■ 強制轉換資料型態：

(1) 將一般型態的資料，強制轉換成其他一般態型的資料。

語法如下：

(指定的資料型態名稱) 變數(或運算式)

■ 強制轉換資料型態例子(程式片段)

```
int a=1,b=2,c=1;
float avg;
avg=(float)(a+b+c)/3;
//將a+b+c的值轉換為浮點數型態，再除以3
```

(2)使用指定的結構所提供Parse 方法，將字串型態的資料強制轉換成指定結構型態的資料。語法如下：

指定的結構名稱.Parse(字串變數(或運算式))

- 請參考「3-2-1　標準輸入方法」之範例2

(3)將參考型態的資料，強制轉換成其他參考型態的資料。語法如下：

(指定的參考型態名稱) 物件變數

- 物件變數，請參考「第九章　自訂類別」
- 請參考「第十一章　抽象類別和介面」之範例1