CHAPTER

3

變數與基本資料型別

- 基本資料型別
- 延伸資料型別
- 字元、字串常值
- 整數、浮點數常值
- 符號常數
- 識別字與保留字
- 變數宣告
- 指定、算術、複合指定、遞增和遞減運算子
- 關係、邏輯運算子
- 運算子的優先順序
- 強制型別轉換
- 自動型別轉換
- 檢測模擬考題解析

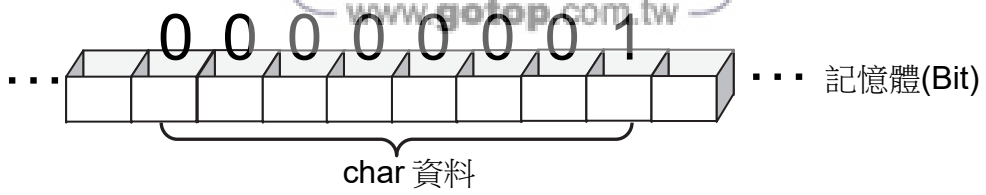
3.1 資料型別

電腦軟體就是用來處理各類的資料，以解決人類生活上的問題。生活中有各式各樣的資料，例如姓名、身高、年齡、數量、車牌號碼、編號…等。這些資料的內容有些是屬於文字，例如姓名、車牌號碼…等。有些是屬於數值內容，例如身高、年齡…等。在 C 語言中定義了一些基本資料型別，來處理各類的資料。在程式執行中為了方便快速存取資料，資料會存在記憶體中。所以在使用相關資料時，如果能根據資料的類別和可能的大小，配置合適的記憶體空間，如此才能順利運作又不浪費記憶體。例如學生人數是沒有小數的數值資料，平均分數是有小數的數值資料，政府總預算是有效位數較多的數值資料、身份證字號是長度為 10 的字串資料。

3.1.1 基本資料型別

1. **char** 字元資料型別

   使用 8 位元(1 Byte)來存放字元資料，有效範圍為-128 ~ 127。char 可以代表電腦系統中的字元，例如 A~Z、~、+、-、*、/ …等字元符號，這些字元是以 ASCII 碼(0~255)存入記憶體。有些系統會將 char 轉為無號數，增加可容納的範圍(0~255)。電腦中最小記憶體單位為 Bit，8 個 Bits 會組合成一個 Byte。



2. **int** 整數資料型別
   使用 32 位元(4 Bytes)來存放整數資料，有效範圍為 $-(2^{31})$ ~ $+(2^{31}-1)$。若資料超過此範圍，或有小數位就必須改成以浮點數來存放。

3. **float** 單精確度浮點數資料型別
   使用 32 位元(4 Bytes)來存放帶有小數位數的實數資料，有效範圍約為 -3.4×10^{-38} ~ 3.4×10^{38}，有效位數約 6~7 位數(10 進制)。

4. **double** 倍精確度浮點數資料型別
   使用 64 位元(8 Bytes)來存放帶有小數位數的實數資料，有效範圍約為-1.8×10^{308} ~ 1.8×10^{308}，有效位數約 15~16 位數(10 進制)。

在 C 語言中，每種資料型別都有規定的大小範圍，例如 int 整數值是使用 4 Bytes 來存放。如果整數資料超過最大值$+(2^{31}-1) = +2,147,483,647$ 時，就會發生資料溢位(Overflow)的現象。反之，若整數小於 $-(2^{31}) = -2,147, 483,648$ 也會發生資料不足位(Underflow)的現象。要特別注意，在 C 語言中如果發生資料溢位或不足位現象時，並不會產生錯誤訊息，所以程式設計者要自行留意。C 語言提供結構資料型別，來處理陣列、字串、結構，至於指標資料型別是用來存取記憶體某個位址的內容，這些較特殊的資料型別將在後面章節中陸續介紹。

3.1.2 延伸資料型別

C 語言為了要增加或降低基本資料型別的精確度和所佔用記憶體，可以使用 short、long、unsigned 等修飾詞，來宣告為延伸資料型別。

1. **short**
   使用 short 修飾詞，會減少資料的有效範圍。處理的資料若介於 -32,768 ~ +32,677 之間，使用 4 Bytes 的 int 來儲存會浪費記憶體，此時可以改用 short int 資料型別，長度僅為 2 Bytes 較節省記憶空間。short int 可以簡寫為 short。

2. **long**
   使用 long 修飾詞，會增加資料的有效範圍。long int 可以簡寫為 long，但是 long int 在編譯器為 32 位元的環境下，其長度與 int 都為 4 Bytes。

3. **unsigned**
   使用 unsigned 修飾詞會將資料改為正整數，整數值會由 0 開始。

資料型別	常值種類	長度	有效範圍
char	字元	1Byte	-128 ~ 127
short (int)	短整數	2Bytes	-32768 ~ 32767
int	整數	4Bytes	-2147483648 ~ 2147483647
long (int)	長整數	4Bytes	-2147483648 ~ 2147483647
unsigned char	無號字元	1Bytes	0 ~ 256
unsigned short (int)	無號短整數	2Bytes	0 ~ 65535
unsigned int	無號整數	4Bytes	0 ~ 4294967295
unsigned long (int)	無號長整數	4Bytes	0 ~ 4294967295
float	單精確浮點數	4Bytes	約-3.4×10^{-38} ~ 3.4×10^{38}(有效位數 6~7 位)
double	倍精確浮點數	8Bytes	約-1.8×10^{-308} ~ 1.8×10^{308}(有效位數 15~16 位)
long double	長倍精確浮點數	16Bytes	約-1.2×10^{-4932} ~ 1.2×10^{4932} (有效位數 18~19 位)

[註]上表的資料可能會因電腦系統而不同。

例 要知道常值或變數所占記憶體的長度，可用 sizeof()函式來查詢： (sizeof.c)

```
printf("char 資料常值占用的記憶體大小： %d \n", sizeof('A'));
printf("int 資料常值占用的記憶體大小： %d \n", sizeof(123));
printf("double 資料常值占用的記憶體大小： %d \n", sizeof(123.45));
```

```
char資料常值占用的記憶體大小： 4
int資料常值占用的記憶體大小： 4
double資料常值占用的記憶體大小： 8
```

3.2 常值與符號常數

所謂「常值」(Literal，或稱字面值)或稱「常數」(Constant)，是指資料本身的值不需要經過宣告，就直接以電腦能處理的數值或字元資料存在敘述中。例如數字 3、字串 "three"、字元'A'…等。而符號常數(或稱常數符號)是指宣告後到程式執行結束，該數的值不會隨著程式執行而改變。C 語言提供的常值有：

1. 字元常值是用來顯示單一字元。
2. 字串常值是用來顯示連續的字元。
3. 整數常值用來顯示整數。
4. 浮點常值是用來顯示帶有小數點數字的資料。

例 顯示基本資料型別常值。 (basic_type.c)

```
printf("%c 是 char 常值\n" , 'A');
printf("%d 是 int 常值\n" , 123);
printf("%f 是 double 常值\n" , 123.45);
```

```
A 是char常值
123 是int常值
123.450000 是double常值
```

3.2.1 字元常值

字元常值是指用單引號頭尾框住的字元，字元必須是屬於可顯示或可列印，例如：'A' ~ 'Z'、'a' ~ 'z'、'0' ~ '9' …等。在程式中，若必要時可用該字元的 ASCII 碼的十六進制表示。例如 'A' 的 ASCII 碼為 $65_{10} = 41_{16}$，所以使用 x41 會在螢幕上顯示大寫字母 A，數值前面加 x 是表示該數值為十六進制。另外，也可以該字元的 ASCII 碼以八進制表示，例如 'A' 的 ASCII 碼為 $65_{10} = 101_8$，其寫法為 101。

例 分別使用十六和八進制顯示'A'字元。 (char.c)

```
printf("十六進制 x41 的字元為：\x41");
printf("八進制 101 的字元為：\101");
```

有些字元已經被定義成其他的特殊功能，來當做程式中的控制字元，例如單引號 (')、雙引號（"）及反斜線（\），我們將這些字元稱為「逸出字元」(Escape Character)。使用 printf()函式顯示訊息時，如果有用到逸出字元時，必須在該字元的前面加上一個 '\' 反斜線構成逸出序列(Escape Sequence)。當編譯器掃描到這些逸出序列時，會將反斜線後面的字元，當成某種特殊意義來處理。詳細的逸出字元用法，請參考下一章的說明。

例 使用逸出序列的各種簡例： (escape.c)

```
01  printf("It\'s a book.\n");    ⇨It's a book.
02  printf("\"How are you?\" said Jerry.\n");⇨"How are you?" said Jerry.
03  printf("\\\\\\ 注意事項 ///\n");       ⇨\\\ 注意事項 ///
```

說明

1. 第 1、2 行：單引號 (')、雙引號 (") 前加反斜線 (\ )，才能正常顯示。
2. 第 3 行："\\\\\\" 六個反斜線只會顯示出三個反斜線。

3.2.2 字串常值

字串常值是由一個或一個以上的連串字元，頭尾使用雙引號「"」括住，例如："Welcome"、"反毒"、"1234"。另外，'a' 為字元常值，而 "a" 則為字串常值。

3.2.3 整數常值

整數常值是指沒有帶小數位數的整數數值，例如 3、123…。C 語言提供十進位制(Decimal)、八進位制(Octal)及十六進位制(Hexadecimal)三種方式，來處理整數常值。

1. **十進位制**：在日常生活中習慣使用十進制來做計數，程式中十進制數值的表示方式和一般習慣相同。
   ① 程式中一連串的數字，會視為十進制的 int 整數資料，例如 123。
   ② 數值後加 l(小寫 L)或 L 指定該值型別為 long 長整數，例如 123l 或 123L。
   ③ 數值後加 u 或 U 指定該值型別為 unsigned int 無號整數，例如 123u 或 123U。
2. **八進位制**：八進制是由數字 0 ~ 7 構成，程式中要使用八進制數值必須以數字 0(非字母 O)開頭，例如 0173。
3. **十六進位制**：十六進制是由數字 0 ~ 9 和字母 A ~ F 構成，程式中要使用十六進制數值必須以 0x 或 0X 開頭(為數字 0 開頭非字母 O)，例如 0x7b 或 0X7B。

例 使用十、八、十六進制顯示整數常值 123： (int_literal.c)

```
printf("十進制 123 顯示：%d\n",123);      ⇨123
printf("十進制 123L 顯示：%d\n",123L);    ⇨123
printf("十進制 123u 顯示：%d\n",123u);    ⇨123
printf("八進制 0173 顯示：%d\n",0173);    ⇨123
printf("十六進制 0X7B 顯示：%d\n",0X7B); ⇨123
```

3.2.4 浮點數常值

浮點數(Floating Point)常值又稱為實數常值，當我們需要用到帶小數點的數值時，就必須使用浮點數常值。在 C 語言中浮點數有兩種表示方式，一種是常用的小數點表示法，例如 3.14159；另一種則是標準 IEEE-754 科學記號，例如 1.2345e+2(123.45、1.2345×10^2)。例如數學式的 1.23×10^8 的數值，在 C 語言中可用 123000000.0 或 1.23E8 來表示。

在 C 語言中，浮點數資料型別有 32 位元的 float(單精確)、64 位元的 double(倍精確)和 128 位元的 long double(長倍精確)，三者的差異在位元越多能顯示的位數越多，但佔用的記憶體也越多。當在程式中直接使用帶有小數點的數字，數字最後面未加任何字元時，預設為倍精確度資料，例如：12.345。若在帶有小數點的數字後面緊跟 F 或 f，例如：12.345F 或 12.345f 則視為單精確度資料。若在帶有小數點的數字後面緊跟 L 或 l，例如：12.345L 或 12.345l 則視為長倍精確度資料。

例 使用小數點和科學記號顯示各種浮點數常值： (float_literal.c)

```
printf("123.45 顯示：%f\n",123.45);           ⇨123.450000
printf("0.123456789 顯示：%f\n",0.123456789); ⇨0.123457
```

```
printf("123.45F 顯示：%f\n",123.45F);          ⇨123.449997
printf("1.2345e2 顯示：%f\n",1.2345e2);        ⇨123.450000
printf("1.2345e8 顯示：%f\n",1.2345e8);        ⇨123450000.000000
printf("1.2345e-2 顯示：%f\n",1.2345e-2);      ⇨0.012345
```

3.2.5 符號常數

有些數值常值、字串常值…等資料，如果希望程式中維持固定不能更動時，就會將這些資料定義為符號常數，使用符號常數可增加程式的可讀性和容易維護。符號常數會在程式開頭定義數值常值、字串常值或公式供程式中全域使用，常數命名習慣上一律用大寫字母，單字之間以底線(_)連接。宣告符號常數的語法：

```
#define 符號常數名稱  符號常數值
或
const 資料型別  符號常數名稱 = 符號常數值;
```

宣告符號常數可以使用前置處理命令 #define 來定義，也可以在程式中使用 const 來定義。在程式中如果修改符號常數的內容值，會產生錯誤造成程式無法執行。

簡 例

利用符號常數宣告書名和單價，顯示 3 本書的售價。

程式碼

程式碼 (檔名：ex03\define\define.c)

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 #define BOOK_TITLE "C 語言基礎必修課"
04 const int PRICE = 500;
05 int main(int argc, char *argv[]) {
06    int num=3;
07    printf("%s %d 本 合計 %d 元\n", BOOK_TITLE, num, num * PRICE);
08    /*PRICE = 550;*/
09    system("PAUSE");
10    return 0;
11 }
```

結 果

說明

1. 第 3 行：使用前置處理命令 #define，定義 BOOK_TITLE 字串符號常數，其有效範圍為整個程式。
2. 第 4 行：使用 const 定義 PRICE 整數符號常數，其有效範圍為整個程式。
3. 第 7 行：直接使用符號常數來顯示或計算。
4. 第 8 行：在程式中若修改符號常數的內容值，會產生錯誤無法執行。

3.3 識別字與保留字

3.3.1 識別字

在現實的生活中，我們會為周遭的人、事、地、物賦予名稱，以方便說明和識別，例如「小明牽來福去河濱公園散步。」。在程式設計時，對程式中所使用的變數、常數、函式…等也會給於名稱，這些名稱就稱為「識別字」(Identifier，或稱識別項)。識別字在程式中必須是唯一的名稱，不允許重複定義。識別字的命名規則如下：

1. 識別字是由大小寫英文字母、阿拉伯數字和底線(_)所組成。但識別字的第一個字元限用英文字母或底線(_)當開頭，第二個字元以後才可以使用數字，至於空白字元或其他特殊字元是不被允許的。
2. C 語言將大小寫字母視為不相同的字元，所以 pass、Pass、PASS 為三個不同的識別字。
3. 不能使用保留字 (或稱關鍵字)當作識別字。
4. 識別字可以為任意長度，但是早期的 C 編譯器會限制長度。內部識別字名稱只認得最前面 31 字元，外部識別字則只有前 6 個字元有效。
5. ANSI C 標準保留兩個底線開頭的識別字名稱，例如：__asm、__volatile...等為系統保留字，專供編譯器使用。另外，以一個底線加上大寫字母開頭的識別項名稱，習慣上作為系統使用的識別字。所以請避免使用以兩個底線和一個底線加上大寫字母開頭的識別字名稱，以免編譯時產生錯誤。

例 下列為合法的識別字：

```
p、sum、_ok、stu_score、lucky7、GoodMoring
```

例 下列為不合法的識別字：

7Eleven	⇨不能以數字開頭	A B	⇨不能使用空白字元
B&Q	⇨不能使用&字元	int	⇨不能使用保留字
__inline	⇨編譯器系統保留字	總計	⇨不能使用中文

3.3.2 保留字

保留字(Reserved Word)又稱關鍵字(Keyword)，是 C 語言定義具特定用途供程式設計使用的識別字。透過這些保留字，配合運算子(Operator)、分隔符號(Seperator)..等，就可以撰寫出各種敘述(Statement)。因為保留字已經有特定用途，所以不允許使用保留字來做識別字。下表為 ANSI C 所定義的保留字：

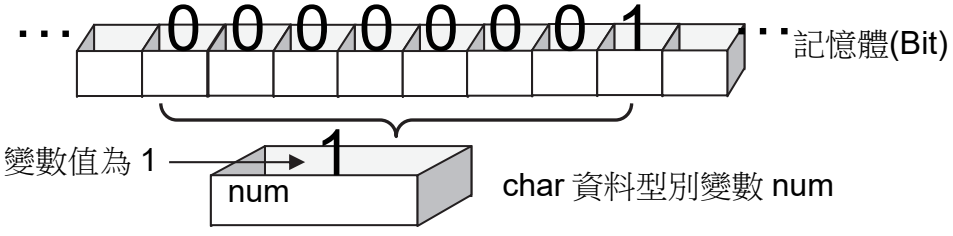
auto	break	case	char	const	continue
default	do	double	else	enum	extern
float	for	goto	if	int	long
register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void
volatile	while				

3.4 變數

常值不必經過宣告就可直接在程式中使用，而變數(Variable)的內容值會隨程式執行而改變。在下面敘述中，其中 a、b 為變數，而 1 為整數常值。

```
a = b + 1;
```

變數使用前必須先行宣告(Declare)，宣告變數時要給予一個名稱，並指定資料型別。程式經過編譯後，系統就會根據宣告的資料型別，配置對應的記憶體空間給該變數使用。要存取變數時，在低階語言要指定記憶體位址，但在高階語言是利用一個變數名稱來指定會較簡便。例如宣告一個 num 變數，資料型別為 char 初值為 1 的示意圖如下：



宣告變數時要選擇適當的資料型別，才不會佔用太大的記憶空間，或超出範圍造成溢位或不足位而產生錯誤結果。例如學生人數可以用無號整數資料型別，因為人數不會為負，如果人數少於兩百五十人可以使用 unsigned char 型別，最節省記憶體。又例如要將-3.14 數值存入變數中，因為是屬於浮點數所以應使用 float 或 double 資料型別，不可以使用 int 資料型別。另外，使用 32 位元的 float 型別已經足夠儲存-3.14，雖然也可以使用 double 型別，但是會浪費較大的記憶體空間。變數的宣告語法如下：

方法 1　僅宣告未設定初值
　資料型別 變數名稱 1 [, 變數名稱 2 …]；
方法 2　宣告並設定初值　⇦ 建議使用
　資料型別 變數名稱 1 = 初值 1 [, 變數名稱 2 = 初值 2 …]；

說明

1. 使用方法 1 僅宣告變數但未設定初值時，C 語言會先從記憶體中配置記憶空間給該變數使用，當程式執行到指定變數初值時，才將初值置入對應的記憶體位址內。萬一所配置的記憶體空間已經有資料，若未指定初值就直接讀取變數值時，可能造成不可預期的結果。所以應該儘量在變數宣告後馬上給予初值，或使用方法二宣告變數。
2. 使用方法 2 時，可以在宣告變數的同時指定變數的初值。
3. 如果同資料型別有兩個(含)以上的變數要宣告時，可以在同一行敘述中一起宣告，變數間以「,」逗號加以區隔即可。

變數的名稱除了要遵循識別字的命名規則外，要使用易懂而且有意義的名稱，以提高程式的可讀性。不要貪圖一時方便，使用簡單且無意義的字母當做變數名稱，會造成以後維護程式的困擾。變數的命名通常依照下列方式來命名：

1. **駝峰式命名法(Camel Case)**
   用有意義的英文單字串連來命名，一般第一個單字會以小寫字母開頭，第二個單字起第一個字母改用大寫表示，就像駱駝的駝峰一般。如果變數名稱是單一單字，例如：「單價」可以使用 price 變數名稱。如果是多個單字的組合，例如：「特價」可以使 priceForVip 變數名稱。

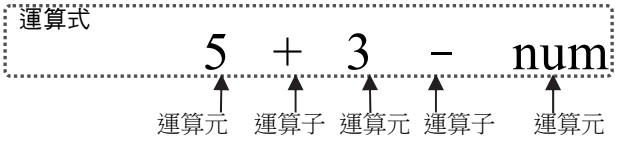
2. **匈牙利命名法**
   在變數名稱前面加上資料型別縮寫的字首(可以用三個字或一個字)，可以馬上知道該變數得資料型別，縮寫時一般是省略 a、e、i、o、u 這些母音。例如：chrEnd(字元變數)、intNum、iNum(整數變數)、fltAverage (浮點變數)、dTaxes(倍精確變數)。

例 下列為宣告變數的範例：

```
char cYesNo = 'y';                  /* 宣告 cYesNo 為字元變數並設定初值為'y' */
/* 宣告 passScore、maxScore、minScore 為無號短整數變數，並設定初值 */
unsigned short passScore = 60, maxScore = 100, minScore = 0;
int iIncome, iPay = 500;            /* 宣告 iIncome、iPay 為整數變數，iPay 設初值 */
iIncome = 50000;                    /* 指定 iIncome 變數值為 50000 */
float fHeight, fWeight;             /* 宣告 fHeight、fWeight 為浮點數變數 */
float total = 1.23456E+6;           /* 宣告 total 為浮點數變數，並設初值 1234560*/
double dValue;                      /* 宣告 dValue 為倍精確度變數 */
```

3.5 運算子

運算子(Operator)是指對運算元做特定運算的符號，例如+、-、*、/ ...等。運算元(Operand)是運算的對象，運算元可以為變數、常值或是運算式。而運算式(Expression)是由運算元與運算子所組成的計算式。



運算子若按照運算所需要的運算元數目來分類，可以分成：

1. 一元運算子 (Unary Operator)：-(負)、++(遞增)、++(遞減)，如：-5、++x、y--。
2. 二元運算子 (Binary Operator)：+、-、*、/、+= ...等，如：x + y、x / y。
3. 三元運算子 (Ternary Operator) ：：? 條件運算子，如：max = x > y ? x : y ;

C 語言所提供的運算子，如果按照運算子的性質可分為七大類：

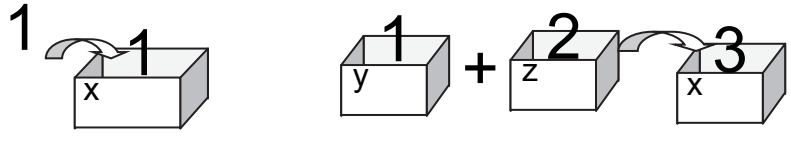
1. 指定運算子 (Assignment Operator)
2. 算術運算子 (Arithmetic Operator)
3. 複合指定運算子 (Shorthand Assignment Operator)
4. 遞增運算子 (Increment Operator)
5. 遞減運算子 (Decrement Operator)
6. 關係運算子 (Relational Operator)
7. 邏輯運算子 (Boolean Logical Operator)

3.5.1 指定運算子

宣告變數指定初值，或是要改變數值時，可以使用指定運算子「=」。指定時可以將一個常值、變數或運算式的結果，指定為變數的變數值，其語法為：

```
變數名稱 =  常值 | 變數 | 運算式;
```

例如將變數 x 指定變數值為 1，寫法為：x=1;。又例如變數 x 指定變數值為 y、z 變數的和，其寫法為：x = y + z ;。上面兩個例子的示意圖如下：



如果多個變數要同時指定相同的變數值時，可以將指定運算子串接。例如同時指定整數變數 x、y 和 z 的變數值為 1，其寫法如下：

```
x = y = z = 1 ;
```

例 下列為指定變數值的範例：

```
x = 5;     /* 指定 x 變數值為常值 5 */
x = y;     /* 指定 x 變數值為變數 y 的值 */
x = 5 + y;/* 指定 x 變數值為運算式 5+y 的結果 */
x = y = 5;/* 指定 x、y 變數值都為常值 5 */
```

3.5.2 算術運算子

算術運算子可以用來執行數學運算，包括加法、減法、乘法、除法、取餘數…等。下表為 C 語言常用的算術運算子：

運算子	功能說明	範例	結果(假設 y=4)
+	加法	x = y + 2	x 變數值為 6
-	減法	x = y - 2	x 變數值為 2
*	乘法	x = y * 2	x 變數值為 8
/	除法	x = y / 2	x 變數值為 2
%	取整數的餘數	x = y % 2	x 變數值為 0
>>	右移位元	x = y >> 2	x 變數值為 1(等於除以 4)
<<	左移位元	x = y << 2	x 變數值為 16(等於乘以 4)

例 下列為算術運算子的範例：

```
int length = 10 + 21 + 16;      ⇨求三角形的周長，結果為 47
float area = 12.5 * 8.7;        ⇨求長方形的面積，結果為 108.75
int x = 24 % 5;                 ⇨求兩整數的餘數，結果為 4。
```

簡 例

已知圓的半徑為 6.4，請算出圓的面積、圓周長和四分之一圓的面積。

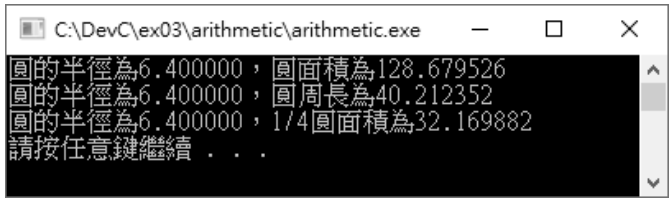
程式碼

程式碼 (檔名：ex03\arithmetic\arithmetic.c)

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 #define PI 3.14159
04 int main(int argc, char *argv[]) {
05    double r=6.4;
06    printf("圓的半徑為%f，圓面積為%f\n",r, PI*r*r);
07    printf("圓的半徑為%f，圓周長為%f\n",r, PI*r*2);
08    printf("圓的半徑為%f，1/4 圓面積為%f\n",r, PI*r*r/4);
```

```
09    system("PAUSE");
10    return 0;
11 }
```

結 果



3.5.3 複合指定運算子

在程式中若需要將某個變數值運算後，再將運算結果指定給該變數時，可以利用複合指定運算子來簡化敘述。例如將 x 變數值加 5，再指定給 x 變數寫法為：

```
x = x + 5;
```

因為指定運算子(=)的左右邊都有相同的變數 x，此時可以使用複合指定運算子來簡化敘述，程式寫法改為：

```
x += 5;
```

下表是 C 語言常用的複合運算子：

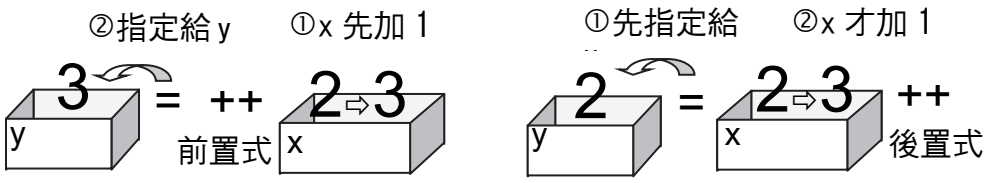
運算子	功能	範例	結果(x 變數值原為 3)
=	指定	x = 3	x 變數值為 3
+=	相加後再指定	x += 2 (x = x + 2)	x 變數值為 5
-=	相減後再指定	x –= 2 (x = x - 2)	x 變數值為 1
*=	相乘後再指定	x *= 2 (x = x * 2)	x 變數值為 6
/=	相除後再指定	x /= 2 (x = x / 2)	x 變數值為 1
%=	相除取餘數後再指定	x %= 2 (x = x % 2)	x 變數值為 1

3.5.4 遞增和遞減運算子

++遞增運算子(Increment Operator)和--遞減運算子(Decrement Operator)都屬於一元運算子，用來對指定的變數值作加 1 和減 1 的運算。運算子若在變數前面稱為「前置式」如：++x 或--x，會先做遞增減動作後才指定變數值。反之，若運算子在變數後面則為「後置式」如：x++或 x--，會先指定後才做遞增減動作。

遞增、減運算式	一般運算式寫法	結果(若 x=2)
y = ++x ;	x = x + 1 ; y = x ;	x = 3 , y = 3
y = x++ ;	y = x ; x = x + 1 ;	x = 3 , y = 2

遞增、減運算式	一般運算式寫法	結果(若 x=2)
y = --x ;	x = x – 1 ; y = x ;	x = 1 , y = 1
y = x-- ;	y = x ; x = x – 1 ;	x = 1 , y = 2



3.5.5 關係運算子

關係運算子(Relational Operator)又稱為「比較運算子」是屬於二元運算子，可以對兩個運算元作比較，並傳回比較結果。如果比較的結果是成立，傳回結果為真(true)C 語言以 1 表示。若不成立，傳回結果為假(false)以 0 表之。關係運算子常配合 if 等選擇結構，來決定程式的流向。下表是 C 語言常用的關係運算子：

關係運算子	功能	數學表示式	範例	結果(若 x=1、y=2)
==	等於	x = y	x == y	0(假)
!=	不等於	x ≠ y	x != y	1(真)
>=	大於等於	x ≥ y	x >= y	0(假)
<=	小於等於	x ≤ y	x <= y	1(真)
>	大於	x > y	x > y	0(假)
<	小於	x < y	x < y	1(真)

簡 例

練習各種關係、遞增和遞減運算子的運算。

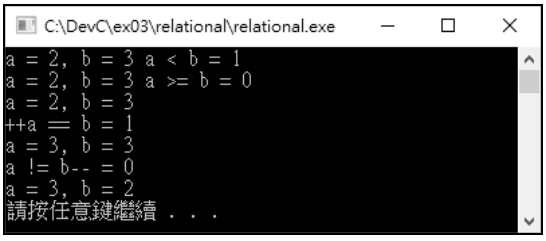
程式碼

程式碼 (檔名：ex03\relational\relational.c)

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 int main(int argc, char *argv[]) {
04    int a = 2, b = 3;
05    printf("a = %d, b = %d a < b = %d\n", a, b, a < b);
06    printf("a = %d, b = %d a >= b = %d\n", a, b, a >= b);
07    printf("a = %d, b = %d\n", a, b);
08    printf("++a == b = %d\n", ++a == b);
09    printf("a = %d, b = %d\n", a, b);
```

```
10    printf("a != b-- = %d\n", a != b--);
11    printf("a = %d, b = %d\n", a, b);
12    system("PAUSE");
13    return 0;
14 }
```

結 果



說明

1. 第 5 行：因為 a=2、b=3，所以 a < b 的值為 1(真、成立)。
2. 第 6 行：因為 a=2、b=3，所以 a >= b 的值為 0(假、不成立)。
3. 第 8 行：因為++a 為前置式，a 會先加 1 變數值為 3，所以 a==b 的值為 1。
4. 第 9 行：第 8 行敘述執行後，a=3、b=3。
5. 第 10 行：因為 b--為後置式，所以會先執行 a != b 傳回值為 0，然後 b 才減 1 變數值為 2。

3.5.6 邏輯運算子

邏輯運算子(Logical Operator)屬於二元運算子，可以對兩個運算元作邏輯運算，並傳回運算結果。邏輯運算子運算的結果，只有 1(真)和 0(假)兩種。邏輯運算子可以用來測試較複雜的條件，常常用來連結多個關係運算子。例如(score >= 0) && (score <= 100)，其中(score >= 0)和(score <= 100)為關係運算子，兩者用&&(且)邏輯運算子連接，表示兩個條件都要成立才為真，所以上述表 score 要介於 0~100。邏輯運算子常結合關係運算子，在 if 等選擇結構中決定程式的流向。下表為 C 語言提供的邏輯運算子：

邏輯運算子	功能	範例	結果(若 x=1、y=0)	說明
&&	且	x && y	0(假)	兩者皆真結果才為真
||	或	x || y	1(真)	只要一個為真結果就為真
!	非	!x	0(假)	相反(原為真結果為假)

下表列出邏輯運算子的各種運算結果：

x	y	x && y	x || y	!x	!y
1	1	1	1	0	0
1	0	0	1	0	1
0	1	0	1	1	0
0	0	0	0	1	1

例 下列為邏輯運算子的範例：

```
(1 < 2) && ('A' == 'a') ⇨結果 0(因為前者為真後者為假，&&必須兩者皆真才為真)
(-1 < 0) || (-1 > 100)  ⇨結果 1(因為前者為真，||一個真就為真)
!( 'A' != 'a')    ⇨結果 0(因為'A' != 'a'為真，所以做!運算後結果為假)
```

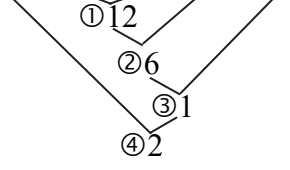
3.5.7 運算子的優先順序

程式中的運算式可能非常複雜，如果同時有多個運算子時，C 語言就必須根據一套規則，才能計算出正確的結果。基本原則為由左至右依序運算，但有些運算子優先權較高必須要優先處理。下表為常用運算子的優先執行順序：

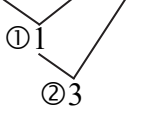
優先次序	運算子(Operator)	運算次序
1	()(呼叫函數)、[](陣列註標)、++(後置)、--(後置)、()	由左至右
2	!(非)、+(正號)、-(負號)、++(前置)、--(前置)	由右至左
3	*(乘)、 / (除)、 %(取餘數)	由左至右
4	+(加)、 -(減)	由左至右
5	<< (左移) 、 >> (右移)	由左至右
6	< 、 <=、 >、 >= (關係運算子)、	由左至右
7	== (相等) 、 != (不等於)	由左至右
8	&& (且邏輯運算子)	由左至右
9	|| (或邏輯運算子)	由左至右
10	？： (條件運算子)	由右至左
11	=、 +=、 -=、 *=、 /=、 %=、<<=、>==、&=、^=、!= (指定、複合指定運算子)	由右而左
12	, (逗號)	由左至右

例如運算式 1 + 2 * 3，因為*運算子的優先順序高於+，所以會先計算 2*3 結果為 6，然後 1 + 6 所以結果為 7。運算式使用()左右括號，不但可以減少運算錯誤增加可讀性，而且可以改變運算的順序。例如上面運算式改為(1 + 2) * 3 時，因為 1 + 2 用()括住所以要先計算結果為 3，然後 3 * 3 結果為 9。

例 1 + 4 * 3 / 2 % 5 = 2 (operator.c)



例 (2 == 2) + 2 = 3



例 int x=2, y=3;

```
x >= y && x != y || x * 2 > y = 1
 ①0    ②1    ④4
   ③0        ⑤1
       ⑥1
```

例 int x=2, y=3, z=4;

```
z += ++x - --y
   ①3  ②2
     ③1
     ④5
```

3.6 資料型別轉換

C 語言在宣告變數時要注意資料型別的範圍，宣告範圍太大會浪費記憶體空間；範圍太小時則會造成溢位的錯誤。為避免超出範圍產生的錯誤，可以使用強制型別轉換 (Cast) 來自行轉型，和利用系統的「自動型別轉換」(Automatic Type Conversion)來轉型。前者是採外顯方式(Explicit)方式，而後者是採隱含方式(Implicit) 轉型。

3.6.1 強制型別轉換

在程式當中，必要時可以將變數的資料型別做轉換，例如將整數轉型為浮點數，或將浮點數變數轉型為整數。強制型別轉換是採外顯方式語法如下：

```
(新資料型別) 變數名稱;
```

小範圍資料型別轉型為較大範圍型別時，變數值沒有問題。但是大範圍轉型為較小範圍資料型別時，變數值就會失真。例如浮點數強制轉型為整數時，會將小數部分直接捨棄，不會做四捨五入的運算。

例

```
int i =(int) f;   ⇨變數 f 先轉型為 int，才指定給整數變數 i。
(float) i * 2.5   ⇨變數 i 先轉型為 float，才和 2.5 做乘法運算
```

簡 例

體驗溢位產生的錯誤結果，以及強制資料型別轉換。

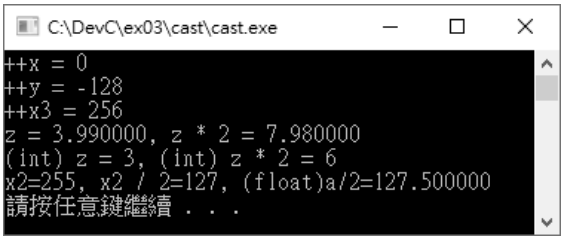
程式碼

程式碼 (檔名：ex03\cast\cast.c)

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 int main(int argc, char *argv[]) {
04    unsigned char x = 255;
05    printf("++x = %d\n", ++x);
06    signed char y =127;
```

```
07    printf("++y = %d\n", ++y);
08    unsigned char x2 = 255;
09    int x3 = (int) x2;
10    printf("++x3 = %d\n", ++x3);
11    double z = 3.99;
12    printf("z = %f, z * 2 = %f\n", z, z * 2);
13    printf("(int) z = %d, (int) z * 2 = %d\n",(int) z,(int) z * 2);
14    printf("x2=%d, x2 / 2=%d, (float)a/2=%f\n",x2, x2 / 2, (float) x2 / 2);
15    system("PAUSE");
16    return 0;
17 }
```

結 果



說明

1. 第 5 行：x 變數值為 255 加 1 後變數值應為 256，但 x 的資料型別為 unsigned char 範圍為 0~255，會產生溢位結果值為 0。要注意編譯時，並不會有任何錯誤提示。
2. 第 7 行：y 變數值為 127 加 1 後變數值應為 128，但 y 的資料型別為 signed char 範圍為 -126~127，會產生溢位結果值為-128。編譯時一樣不會有任何錯誤提示。
3. 第 9 行：x2 資料型別為 unsigned char 變數值為 255，用(int)強制轉型為整數，然後指定給整數變數 x3。
4. 第 10 行：x3 變數值為 255 加 1 後，因為 int 範圍大不會溢位所以值為 256。
5. 第 13 行：z 變數值為 3.99，用(int)強制轉型為整數，小數部分會無條件捨棄，所以變數值為 3，會造成運算結果失真。
6. 第 14 行：x2 變數值為 255，因為型別為 unsigned char 除以 2 後結果 127，小數位數被捨棄了。如果希望提高精確度，可以先將 x2 強制轉型為 float，運算後的結果為 127.5。

3.6.2 自動型別轉換

當運算式中如果有資料型別不同的數值要做運算時，除非主動使用強制型別轉換外，否則系統會做自動型別轉換，將資料型別轉成一致後再進行運算。自動型別轉換是屬於採隱含方式，如果兩個不同資料型別的資料需要做運算時，是將型別長度較小的資料先轉成型別長度較大者，兩者調整為相同的資料型別才做運算。其轉型規則如下：

char ⇨ int ⇨ long ⇨ float ⇨ double

資料型別長度短 ⟹ 資料型別長度長

編譯器在編譯運算式時，會檢查運算元的資料型別。如果相同時，就會直接進行運算。如果不相同時，會將資料型別長度短的資料轉換成較長的資料型別，然後才進行運算。

所以在 C 語言中做除法運算時，如果有一個運算元為浮點數資料型別，就會自動轉型為浮點數。此時，除法運算會採用浮點數除法就是我們熟悉的除法運算，會計算到小數部份。但是如果兩個運算元都是整數，則會採用整數除法。整數除法只會計算整數部份，小數部份會無條件捨去。為避免運算結果失真，應該使用前面介紹的強制型別轉換。

例

```
printf("%f\n", 3/10);     ⇨執行結果為 0.000000 不是期望的 0.3
printf("%d\n", 3/2);      ⇨執行結果為 1 不是 1.5，因為小數部分被無條件捨棄
printf("%f\n", 3.0/10); ⇨執行結果為 0.300000，因為 3.0 是浮點數常值
printf("%f\n", (float)3/2);       ⇨執行結果為 1.500000，因為用(float)強制轉型
```

簡 例

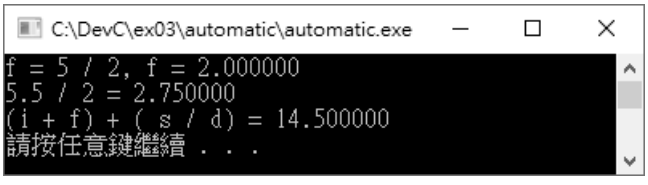
練習運算式的自動型別轉換。

程式碼

程式碼 (檔名：ex03\automatic\automatic.c)

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 int main(int argc, char *argv[]) {
04    float f = 5 / 2;
05    printf("f = 5 / 2, f = %f\n", f);
06    printf("5.5 / 2 = %f\n", 5.5 / 2);
07    int i=2; short s=25; double d=2.5;
08    f=2.5;
09    printf("(i + f) + ( s / d) = %f\n", (i + f) + ( s / d));
10    system("PAUSE");
11    return 0;
12 }
```

結 果



說明

1. 第 4 行：5 / 2 運算式的兩個運算元 5 和 2 資料型別都是整數，所以就直接用整數相除結果為整數 2。指定給 float 型別變數 f 時，會轉型為 float 結果變數值為 2.000000。
2. 第 6 行：5.5 / 2 運算式的兩個運算元 5.5 和 2，資料型別分別為 double 和整數，所以 2 會先轉型為 double，才做相除運算結果為 2.75 資料型別為 double。