

第五章 運算式、運算子與敘述

運算式與運算子
常用的運算子
運算子的優先順序
資料型態轉換

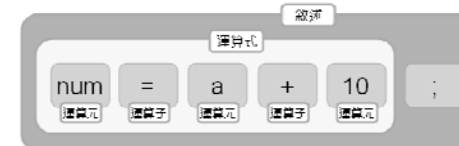
1

5.1 運算式與運算子

運算式、運算元與運算子

- 運算式由運算元與運算子組成
 - 運算式 (expression)
 - 運算元 (operand)：如變數 sum，或常數 10 等
 - 運算子 (operator)：如 +, -, *, / 與 = 等符號

運算式可以出現在設值敘述 (assignment statement) 裡：



運算式也可以出現在其他敘述裡：

```
if (a+5<2*b*c) {  
    ...  
}
```

```
for (i=a*b; i<5; i+=10) {  
    f = d = printf("%d", a/=c);  
}
```

2

設定運算子 (1/2)

- 「設定」運算子可將變數設值

表 5.1.1 設定運算子的說明

設定運算子	意義	範例	說明
=	設定	a=b	設定 a 的值等於 b

- 等號 (=) 是「設定」的意思，如下面的範例：



3

5.1 運算式與運算子

設定運算子 (2/2)

- 設定運算子簡單的範例：

```
01 /* prog5_1, 設定運算子「=」 */  
02 #include <stdio.h>  
03  
04 int main(void)  
05 {  
06     int age=14;  
07  
08     printf("age=%d\n", age);  
09     age=age+1; /* 將 age 加 1 後，再設回給 age 存放 */  
10     printf("將 age 加 1 之後，age=%d\n", age);  
11  
12     return 0;  
13 }  
  
/* prog5_1 OUTPUT---  
age=14  
將 age 加 1 之後，age=15  
-----*/
```

4

一元運算子 (1/2)

- 一元運算子 (unary operator) 只需要一個運算元
 - +3; /* 表示正3, 3 為運算元 */
 - a; /* 表示負a, a 為運算元 */
 - !a; /* NOT運算, 若a為0, 則!a為1, 若a不為0, 則!a為0 */

表 5.1.2 一元運算子的說明

元運算子	意義	範例	說明
+	正號	a+=5	同 a=5, 相當於設定 a 等於正 5
-	負號	a=-3	設定 a 等於 -3
!	NOT, 否	a=!b	把 b 的值取 NOT, 再設給 a 存放

一元運算子 (2/2)

- NOT運算的範例：

```
01 /* prog5_2, 「!」運算的用法 */
02 #include <stdio.h>
03
04 int main(void)
05 {
06     int a=0;
07     int b=6;
08     printf("a=%d, !a=%d\n", a, !a); /* 印出 a 及 !a 的值 */
09     printf("b=%d, !b=%d\n", b, !b); /* 印出 b 及 !b 的值 */
10
11     return 0;
12 }
```

/* prog5_2 OUTPUT--

a=0, !a=1
b=6, !b=0
-----*/

算數運算子

- 算數運算子的成員如下：

表 5.1.3 算數運算子的說明

算數運算子	意義	範例	說明
+	加法	2+4	計算 2+4
-	減法	3-6	計算 3-6
*	乘法	7*9	計算 7*9
/	除法	6.4/3	計算 6.4/3
%	取餘數	21%9	計算 21 除以 9 的餘數

+, -, *, / 這四個算術運算子都有兩個版本：整數的運算和浮點數的運算
因為資料格式不同, 運算的方式也不同

餘數運算子

- 下面的範例是餘數運算子「%」的練習：

- 要印出「%」符號, 可用格式碼「%%」

```
01 /* prog5_3, 餘數運算子的練習 */
02 #include <stdio.h>
03
04 int main(void)
05 {
06     printf("12%%4=%d\n", 12%4); /* 求出 12/4 的餘數 */
07     printf("12%%5=%d\n", 12%5); /* 求出 12/5 的餘數 */
08     printf("12%%16=%d\n", 12%16); /* 求出 12/16 的餘數 */
09
10     return 0;
11 }
```

-12 % 5 = -2

注意：和數論「同餘」運算的定義不一樣, 定義為 -(12%5)

/* prog5_3 OUTPUT---

12%4=0
12%5=2
12%16=12
-----*/

關係運算子與 if 敘述 (1/2)

- if 敘述與關係運算子

if (判斷條件)
敘述主體;

if 敘述的格式

表 5.1.4 關係運算子的說明

關係運算子	意義	範例	說明
>	大於	a>b	判斷 a 是否大於 b
<	小於	a<b	判斷 a 是否小於 b
>=	大於等於	a>=b	判斷 a 是否大於等於 b
<=	小於等於	a<=b	判斷 a 是否小於等於 b
==	等於	a==b	判斷 a 是否等於 b
!=	不等於	a!=b	判斷 a 是否不等於 b

9

關係運算子與 if 敘述 (2/2)

- if 敘述與關係運算子的使用範例：

```
01 /* prog5_4, 關係運算子的練習 */      /* prog5_4 OUTPUT---
02 #include <stdio.h>                      5>2 成立
03                                          此行一定會被執行
04 int main(void)                          -----*/
05 {
06     if(5>2) /* 判斷 5>2 是否成立 */
07         printf("5>2 成立\n");
08
09     if(1) /* 1 代表 true, 所以 if 的判斷結果會成立 */
10         printf("此行一定會被執行\n");
11
12     if(3==8) /* 判斷 3 是否等於 8 */
13         printf("3==8 成立\n");
14
15     return 0;
16 }
```

10

遞增與遞減運算子

- 遞增與遞減運算子的成員：

遞增與遞減運算子	意義	範例	說明
++	遞增，變數值加 1	a++	a 加 1 後再設定給 a 存放
--	遞減，變數值減 1	a--	a 減 1 後再設定給 a 存放

- a++ 會先執行整個敘述後，再將 a 的值加 1
a = 10; b = 20 + a++;
printf("a=%d b=%d\n", a, b); // a = 11, b = 30
- ++a 則是先把 a 的值加 1 後，再執行整個敘述
a = 10; b = 20 + ++a;
printf("a=%d b=%d\n", a, b); // a = 11, b = 31
- 請避免使用 a = 10; b = ++a + a * 2; 這樣子的敘述, C 語言中未明確定義執行的順序, 不同版本編譯器可能有不同結果

11

遞增與遞減運算子

- 下面的程式是使用遞增運算子的範例： **請避免使用**
執行結果與編譯器有關

```
01 /* prog5_5, 遞增運算子「++」 */      b=10;
02 #include <stdio.h>                      printf("%d %d", b, ++b); ==> 11 11
03
04 int main(void)
05 {
06     int a=3, b=3;                      b=10;
07                                          printf("%d %d", b, ++b); ==> 11 10
08     printf("a=%d", a);
09     printf("a 的傳回值為%d", a); /* 計算 ++a, 並印出其傳回值 */
10     printf("a=%d\n", a);
11
12     printf("b=%d", b);
13     printf("b 的傳回值為%d", b); /* 計算 ++b, 並印出其傳回值 */
14     printf("b=%d\n", b);
15
16     return 0;
17 }
```

/* prog5_5 OUTPUT
a=3, a 的傳回值為 3, a=4
b=3, ++b 的傳回值為 4, b=4
-----*/

12

邏輯運算子 (1/2)

• AND、OR 與真值表

表 5.1.6 邏輯運算子的說明

邏輯運算子	意義	範例	說明
&&	AND, 且	a&& b	計算 a AND b 的結果
	OR, 或	a b	計算 a OR b 的結果

AND 及 OR 真值表

AND	T	F
T	T	F
F	F	F

OR	T	F
T	T	T
F	F	F

13

邏輯運算子 (2/2)

• 邏輯運算子的應用範例：

成績輸入錯誤 需要補考 成績輸入錯誤

0 49 60 100

```

01 /* prog5_6, 邏輯運算子的應用 */
02 #include <stdio.h>
03
04 int main(void)
05 {
06     int score;
07     printf("請輸入成績:");
08     scanf("%d",&score);
09
10     if ((score<0) || (score>100)) /* 若成績超出 0 到 100 之間 */
11         printf("成績輸入錯誤!!\n");
12
13     elseif ((score<60) && (score>49)) /* 若成績介於 50 到 59 之間 */
14         printf("需要補考!!\n");
15     return 0;
16 }

```

/* prog5_6 OUTPUT---

請輸入成績:54
需要補考!!

score is -1 if (b==0||a/b>10) ...
if (b!=0&&a/b>10) ...

score is 70

Short circuit evaluation (minimal evaluation)

14

括號運算子

• 括號運算子「()」用來改變運算子執行的優先順序

表 5.1.8 括號運算子的說明

括號運算子	意義
()	提高括號中運算式的優先順序

■ 3+5*4*6-7; /* 未加括號的運算式 116 */

■ (3+5*4)*(6-7); /* 加上括號的運算式 -23 */

15

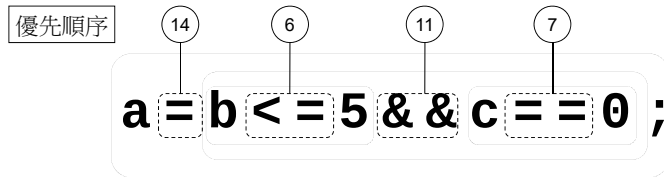
運算子的優先順序

優先順序	運算子	類別	結合性
1	()	括號運算子	由左至右
1	[]	方括號運算子	由左至右
2	!、+ (正號)、- (負號)	一元運算子	由右至左
2	~	位元邏輯運算子	由右至左
2	++、--	遞增與遞減運算子	由右至左
3	*、/、%	算數運算子	由左至右
4	1、\	算數運算子	由左至右
5	<<、>>	位元左移、右移運算子	由左至右
6	>、>=、<、<=	關係運算子	由左至右
7	--、!-	關係運算子	由左至右
8	& (位元運算的 AND)	位元邏輯運算子	由左至右
9	^ (位元運算的 XOR)	位元邏輯運算子	由左至右
10	(位元運算的 OR)	位元邏輯運算子	由左至右
11	&&	邏輯運算子	由左至右
12		邏輯運算子	由左至右
13	?:	條件運算子	由右至左
14	=、+=、-=、*=、/=	設定運算子	由右至左
15	=、=	設定運算子	由右至左

16

運算子的優先順序

- 運算子優先順序的範例：



1. 先計算 $b \leq 5$ ($\leq$ 的優先順序為 6)
2. 再計算 $c == 0$ ($==$ 的優先順序為 7)
3. 然後進行 $\&\&$ 運算 ($\&\&$ 的優先順序為 11)
4. 最後再把運算結果設給變數 a 存放 ($=$ 的優先順序為 14)

運算式與簡潔運算子

- 下面的例子均為運算式 (運算式 為運算子與運算元組成)：

- $\text{sum} = \text{sum} + 3;$
- $-5 * (12 - 4);$

- 簡潔運算子可簡化運算式

表 5.3.1 簡潔的運算子與使用說明

運算子	範例用法	說明	意義
$+$	$a + b$	$a + b$ 的值存放到 a 中	$a = a + b$
$-$	$a - b$	$a - b$ 的值存放到 a 中	$a = a - b$
$*$	$a * b$	$a * b$ 的值存放到 a 中	$a = a * b$
$/$	a / b	a / b 的值存放到 a 中	$a = a / b$
$\%$	$a \% b$	$a \% b$ 的值存放到 a 中	$a = a \% b$

簡潔運算子

- 簡潔運算子的使用範例：

```

01 /* prog5_7, 簡潔運算式 */
02 #include <stdio.h>
03
04 int main(void)
05 {
06     int a=3, b=5;
07     printf("計算前: a=%d, b=%d\n", a, b);
08     a+=b; /* 計算 a+=b, 即 a=a+b */
09     printf("計算後: a=%d, b=%d\n", a, b);
10
11     return 0;
12 }

```

/* prog5_7 OUTPUT

計算前: a=3, b=5

計算後: a=8, b=5

*/

運算式的型態轉換 (1/3)

- 型態轉換發生在運算子左右兩邊的運算元型態不同時
- 自動型態轉換：
 - 算術運算子
 - $+$, $-$, $*$, $/$ 只要有一個運算元是浮點數，就是浮點數的運算，另外一個運算元會自動轉換為浮點數
 - 整數運算 $+$, $-$, $*$, $/$ 會把兩邊運算元都轉換成 int ($\text{char} / \text{short} \Rightarrow \text{int}$)
 - 浮點運算 $+$, $-$, $*$, $/$ 會把兩邊運算元都轉換成 double (表示範圍較小的型態轉換成表示範圍較大的型態)
 - 例如： $\text{int} + \text{float}$ 相加， int 會被轉成 double
 $\text{char} * \text{int}$ 相加， char 會被轉成 int
 - 設定運算子，函數參數
 - 除非資料內容可能有損耗，自動換成左邊變數(函數參數)的型態

運算式的型態轉換 (2/3)

- 型態轉換的範例：

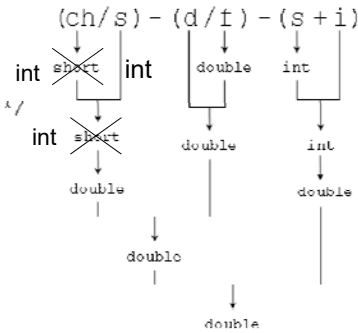
```
01 /* prog5_8, 運算式的型態轉換 */
02 #include <stdio.h>
03
04 int main(void)
05 {
06     char ch='a';
07     short s=-2;
08     int i=3;
09     float f=5.3f;
10     double d=6.28;
11     printf("(ch/s)-(d/f)-(s+i)=%f\n", (ch/s)-(d/f)-(s+i));
12     printf("size=%d\n", sizeof((ch/s)-(d/f)-(s+i)));
13
14     return 0;
15 }
```

/* prog5_8 OUTPUT

(ch/s)-(d/f)-(s+i)=-50.184906

size=8

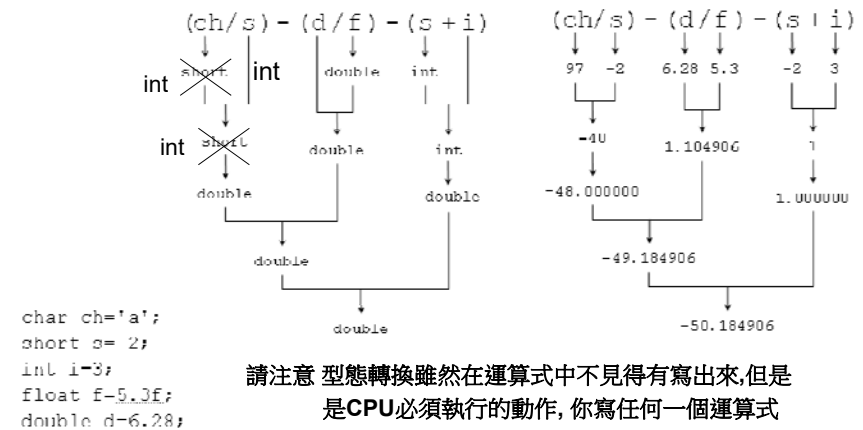
-----*/



21

運算式的型態轉換 (3/3)

- 運算式中，變數型態的轉換過程：



```
char ch='a';
short s= -2;
int i=3;
float f=5.3f;
double d=6.28;
```

請注意 型態轉換雖然在運算式中不見得有寫出來,但是是CPU必須執行的動作,你寫任何一個運算式時需要清楚地知道什麼時間點做了哪些轉換

22

Sequence Point (1/2)

- C/C++ 程式執行時，一個敘述一個敘述順序執行，單一敘述中的運算式有時候有規定執行的先後順序 (括號，優先順序...)
- 但是單一敘述中有時候沒有規定運算式執行的先後順序, 另外有 side effects 運算式並沒有說在那個時間點產生效果，語法上只規定一定要完成所有 side effects 的敘述 (statement): 時間點稱為 sequence point

- 為什麼你需要知道這個?

結果不確定

確定

✓ printf("%d%d", a, a++);

✓ a = 10; a = a++;

✓ (b + c++) + c

✓ fun1(&a) + fun2(a++)

✓ if (fun1(a++) && fun2(&a))

✓ a++, b = a + c;

✓ int b=10, a=++b, c = b;

```
a = b + c * d; while (...) {
fun(...); ...
if (...) { ...
else { ...
}
```

23

Sequence Point (2/2)

- && 或是 ||：先左(後右) $a++ == 10 \ \&\& \ a == 11$
- comma operator： $a = b++$, $c = b$; (請注意定義變數時的逗點和分隔函式參數的逗點都不是 comma operator)
- Ternary operator： $a++ \neq 10 ? c = a : c = a+1$;
- Before the first statement of a function：fun(c++)
- Between each declarator: $\text{int } x = a++, y = a++$;

- 很多運算式之間有定義先後順序，但是沒有定義 sequence point
- Sequence point 主要是定義 side effects 是否完成，但是同時也順便定義出前後運算式的執行順序
- Side effects 在兩個 sequence points 中間完成，並沒規定在 sequence point 的時間點才完成，多個 side effects 之間的發生順序也沒有定義
- https://en.wikipedia.org/wiki/Sequence_point

24