【第二章】

資料型態、運算子與表示式

講師: 李根逸 (Ken-Yi Lee), E-mail: feis.tw@gmail.com

★ ★ ★ ★

★ ★ ★ ★

課程大綱

- 資料型態 [P.51]
  - C/C++ 內建的常見資料型態 [P.52]
  - 變數宣告 [P.53]
- 不同資料型態間的差異 [P.55]
  - 整數 (**int**) 的表示法 [P.57]
  - 浮點數 (**float/double**) 的表示法 [P.58]
  - **printf** 與 **scanf** 的格式字串 [P.59]
- 字面常數的型態 [P.61]
- 不同型態間的轉換 (隱性/顯性轉型) [P.62]
- 字元 (**char**) 的表示法 [P.65]
- 各種常見運算子：
  - 算數運算子: + - * / % [P.67]
  - 指定運算子: = [P.68]
  - 關係與等號運算子: <, >, <=, >=, ==, != [P.69]
  - 邏輯運算子 [P.70]
  - 運算子優先順序 [P.71]

資料型態 (Data type)

- 在高階語言中，為了能夠方便有效（省時省空間）的撰寫程式碼並做出各種複雜的運算，我們需要使用多種資料型態
  - 例如：整數, 小數和文字處理等 ...
- 電腦內部是使用位元 (Bit) 這個基本單位來表示資料並儲存於記憶單元 (記憶體) 或輔助記憶單元 (硬碟) 中。
  - 每個位元只可以表示 0 或 1 兩種值
- 任何資料型態的資料都可以轉換成由一串位元來表示
  - 換句話說，資料型態就是要告訴電腦要怎麼去解釋某一串位元資料

C 常見的內建資料型態

資料型態	名稱	大小 (bytes)	範例
整數 (Integer)	int	4*	32
長整數 (Long Integer)	long	4*	32
長長整數 (Long Long Integer)	long long	8	32
字元 (Character)	char	1	‘a’
單精度浮點數 (Single Precision Floating Point)	float	4	4.39
雙精度浮點數 (Double Precision Floating Point)	double	8	4.39
無	void	?	?

`sizeof()`

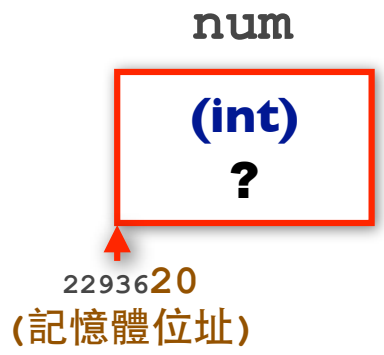
變數宣告

- 變數名稱在使用前，需要先進行宣告讓編譯器知道
- 基本語法：
  - 資料型態 變數名稱;

```
int num;
```

  - 資料型態 變數名稱 = 初始值;

```
int num = 0;
```



名稱通常為英文字母大小寫、數字和底線構成，數字不能開頭，大小寫不同也代表不同的名稱

- 保留字 **(keywords)**：
  - C 語言中下列單字 (保留字) 無法作為變數名稱
    - 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

【補充】使用 sizeof 看大小

- **`sizeof()` 是一個特殊的運算子，會得到某變數或資料型態在記憶體中佔有的大小。表示大小所使用的單位是位元組 (Bytes) [1 Byte = 8 Bits]**
  - 開啟範例檔 sizeof.cpp 並執行看看
- **有些資料型態佔記憶體的大小與系統有關，例如在 32-bit 系統中， `long` 的大小是 4 bytes，但是在 64-bit 系統中， `long` 的大小是 8 bytes**

不同資料型態間的差異

- 表示的資料意涵不同：
  - 整數 (int) 與字元 (char)
- 表示的原理不同：
  - 整數 (int) 與浮點數 (float)
- 可表示的範圍大小不同：
  - 整數 (int) 與長長整數 (long long)
- 可表示的精確度大小不同：
  - 單精度浮點數 (float) 與倍精度浮點數 (double)
- 有無正負數 (有號與無號)
  - 有號整數 (int) 與無號整數 (unsigned int)

與大小有關

整數型態的可表示範圍

- 資料型態可表示的範圍與他佔記憶體的『大小』有關
  - 每個 Bit 可以表示兩種值 (0 或 1)
  - 每個 Byte 有 8 個 Bit
    - 每個 Byte 可以表示 $2^8 = 256$ 種值
    - 在可表示正負數 (即有號 [signed]) 的情況下，可表示的整數範圍會是從 -128 到 127 (共 256 個數字)
    - 無號 [unsigned] 時，可表示的整數範圍是從 0 到 255
  - 如果一個資料型態有 4 個 Byte，則可以表示：
    - $2^{8x4} = 2^{32} = 4{,}294{,}967{,}296$ 種值
    - 4 個 Byte 大的 **int** 的可表示範圍就是從 −2,147,483,648 到 2,147,483,647 (約十位數)
    - 4 個 Byte 大的 **unsigned int** 的可表示範圍就是從 0 到 4,294,967,295 (約十位數)

有號整數 (int) 的表示法

32 bits (4 bytes)

十進位表示法	二進位表示法			
1	00000000	00000000	00000000	00000001
2	00000000	00000000	00000000	00000010
5	00000000	00000000	00000000	00000101
255	00000000	00000000	00000000	11111111
256	00000000	00000000	00000001	00000000
0	00000000	00000000	00000000	00000000
2147483647	01111111	11111111	11111111	11111111
-1	11111111	11111111	11111111	11111111
-2	11111111	11111111	11111111	11111110
-255	11111111	11111111	11111111	00000001
-2147483648	10000000	00000000	00000000	00000000

負數用 2 的補數表示法：將正數表示法的 0 和 1 互換後再加 1

浮點數表示法

- 浮點數 **(floating point)** 是用來將實數數位化表示的一種表示法
  - 我們現在所用的是由 IEEE 制定的浮點數表示標準
- 簡單來看，浮點數的表示法將位元分成三個區塊
  - 符號位元 (1 Bit), 指數部分, 小數部分

(符號) (指數) (有效數字) [影響精確度]

$+ \quad 3$

$+ \ 0.1563 \times 10^{3}$

1563

float有效數字約7位，double約15位

(實際上格式比較複雜，這裡只是個概念的說明。細節可參考 http://goo.gl/imXGf)

printf 與 scanf 的格式字串

資料型態	名稱	符號
整數 (Integer)	int	`%d`
長整數 (Long Integer)	long	`%ld`
長長整數 (Long Long Integer)	long long	`%lld` 或 `%I64d`
字元 (Character)	char	`%c`
單精度浮點數 (Single Precision Floating Point)	float	`%f`
雙精度浮點數 (Double Precision Floating Point)	double	`%f`, `%lf` printf scanf

《範例》浮點數計算

- **請修改程式 (calc.cpp) 為 (calc_1.cpp) 讓使用者分別輸入三個整數後，算出三個整數的和、平均值、乘積並顯示給使用者看 (四捨五入到小數點後三位)**
  - 注意: 平均值可能具有小數而且使用者可能輸入的數值帶有小數
  - 提示:
    - 變數宣告時須改為用 float 宣告
    - scanf 時須使用 %f 來讀入 float
    - printf 時須使用 %f 來輸出 float
    - printf 的格式字串可以加上數字表示位數
      - %.3f 表示印出浮點數並四捨五入到小數點後第三位

字面常數的型態

字面常數	資料型態	名稱
3	整數 (Integer)	int
3.	雙精度浮點數 (Double Precision Floating Point)	double
3.f	單精度浮點數 (Single Precision Floating Point)	float
'3'	字元 (Character)	char

執行 **constant.cpp**

不同型態間的自動轉換

- **進行內建運算時，運算元的型態必須相同。當型態不同時，編譯器會試著幫你做自動轉換 (隱性轉型)。**
  - 不同內建型態間的自動轉換 (隱性轉型) 通常以『可表示範圍大』的為準
    - 例如：`4 / 3` 與 `4 / 3.`
      - **4 / 3** 時，計算結果的資料型態會是 **int**
      - **4 / 3.** 時，**4** 是 **int** 而 **3.** 是 **double**。計算時會先將 **4** 轉換成 **double** 後再除以 **3.**，計算結果資料型態是 **double**。
  - 你也可以用強制的方式轉換 (顯性轉型)：
    - 例如：`(double) 4 / 3`
      - **4** 會先被強制轉換為 **double** 型態 (即 **4.**)，再試著去除以 **3**，此時 **3** 也被動的隱性轉型成 **double** 型態 (即 **3.**)。計算結果的資料型態會是 **double**

《範例》隱性轉型與格式

- 請開啟 **casting.cpp**，並猜測執行的結果
- 執行 **casting.cpp**，檢查是否與預期的相同
- 提示：
  - A = B 是指將 B 的值指定給 A，此時如果 B 的型態與 A 不同則可能會造成無法編譯或發生隱性轉型將 B 轉為 A 的形態
    - 轉型時，浮點數轉為整數是無條件捨去，通常值會變得不精確
  - 要小心的是，使用 **printf** 或 **scanf** 時，輸入的參數並不會自動的轉型
    - 例如: printf("%d", 3.);
      - 會因為 **3.** 是 **double** 卻當成 **int** 印而失敗，產生不易預期的結果
    - **printf** 跟 **scanf** 這算是特例中的特例，但是我們常常使用到。

《範例》大數計算

- 試寫一程式 **(bignum)** 輸入兩個五位數整數後輸出他們的乘積
  - **int** 型態約可表示 $\log(2^{31}) \sim 9$ 位有效整數
  - **long long** 型態約可表示 $\log(2^{63}) \sim 19$ 位有效整數
  - 在 Dev C++ 中，請使用 %I64d 來列印 long long 型態數值

ASCII 值 十進位	十六進位	字元符號	ASCII 值 十進位	十六進位	字元符號	ASCII 值 十進位	十六進位	字元符號	ASCII 值 十進位	十六進位	字元符號
000	00	(null)	032	20		064	40	@	096	60	
001	01	☺	033	21	!	065	41	A	097	61	a
002	02	●	034	22	"	066	42	B	098	62	b
003	03	♥	035	23	#	067	43	C	099	63	c
004	04	♦	036	24	$	068	44	D	100	64	d
005	05	♣	037	25	%	069	45	E	101	65	e
006	06	♠	038	26	&	070	46	F	102	66	f
007	07	•	039	27	'	071	47	G	103	67	g
008	08	◘	040	28	(	072	48	H	104	68	h
009	09	○	041	29	)	073	49	I	105	69	i
010	0A	◙	042	2A	*	074	4A	J	106	6A	j
011	0B	♂	043	2B	+	075	4B	K	107	6B	k
012	0C	♀	044	2C	,	076	4C	L	108	6C	l
013	0D	♪	045	2D	-	077	4D	M	109	6D	m
014	0E	♫	046	2E	.	078	4E	N	110	6E	n
015	0F	☼	047	2F	/	079	4F	O	111	6F	o
016	10	►	048	30	0	080	50	P	112	70	p
017	11	◄	049	31	1	081	51	Q	113	71	q
018	12	↕	050	32	2	082	52	R	114	72	r
019	13	‼	051	33	3	083	53	S	115	73	s
020	14	¶	052	34	4	084	54	T	116	74	t
021	15	§	053	35	5	085	55	U	117	75	u
022	16	▬	054	36	6	086	56	V	118	76	v
023	17	↨	055	37	7	087	57	W	119	77	w
024	18	↑	056	38	8	088	58	X	120	78	x
025	19	↓	057	39	9	089	59	Y	121	79	y
026	1A	→	058	3A	:	090	5A	Z	122	7A	z
027	1B	←	059	3B	;	091	5B	[	123	7B	{
028	1C	∟	060	3C	<	092	5C	\	124	7C	|
029	1D	↔	061	3D	=	093	5D	]	125	7D	}
030	1E	▲	062	3E	>	094	5E	^	126	7E	~
031	1F	▼	063	3F	?	095	5F	_	127	7F	⌂

在輸入、儲存和運算時，char 都是使用整數(ASCII值)格式，只有在輸出(顯示或列印) 的時候會依照該 ASCII 值所對應的文字套用字型後輸出

'A'+1 = ?

(char) (int)(???)

'A'-1 = ?

(char) (int)(???)

'A'+'1' = ?

(char)(char)(???)

[備註] char 做算術運算時當 int 用

請參考 **char.cpp**

《範例》大小寫轉換

- 試寫一程式（**tolower**）輸入一大寫英文字元，顯示相對應的小寫英文字元
  - 提示：

```
char input = ?;
char output = input - 'A' + 'a';
```

範例輸入一 ：	範例輸入二 ：	範例輸入三 ：
A	B	Z
範例輸出一 ：	範例輸出二 ：	範例輸出三 ：
a	b	z

算術運算子: +-*/%

- 算術運算子，運算的結果與運算元的值跟型態有關
  - 運算子是有優先順序的 (***/%** 優先於 **+-**)
  - 算術運算子優先順序相同時，在左邊的先
- 每次執行一個運算子時就會產生一個中間結果，我們應該試著去熟知這個中間結果的『值』與『型態』

```
A = 3 * ( 2 + 1 ) + 7 / 2 + 9 * 3.;
A = 3 * 3         + 7 / 2 + 9 * 3.;
A = 9             + 7 / 2 + 9 * 3.;
A = 9             + 3     + 9 * 3.;
A = 9             + 3     + 27.   ;
A = 12                    + 27.   ;
A = 39.                             ;
```

指定運算子：=

- = 為指定運算子
  - 指定運算子會將右方的值指定給**左方的變數**
    - 指定運算子的左方一定要放置某個變數名稱。
  - 指定運算子的運算結果就是左方變數最後的值跟型態
- 運算優先順序
  - 指定運算子 (=) 的運算優先順序是全部裡面最低的而且運算順序是由右至左 (特別！)

assign.cpp

```
int A,B;
double C, D;
A = 3;
3 = A;
A = B = 3;
A = C = B = D = 3 + 7 / 2. ;
```

是非真假

運算結果	意	義	
不是 **0**	真	正確	成立
0	假	錯誤	不成立

- 0 表示假、錯誤和不成立的意思
- 1 表示真、正確和成立的意思
- 2 表示真、正確和成立的意思
- -1 表示真、正確和成立的意思

關係與等號運算子

- 關係與等號運算子的運算結果有 **0** 與不是 **0** 兩種
  - 不是 0 代表的意思是真 (成立)
  - 0 代表的意思是假 (不成立)

運算意義	運算符號
大於	>
小於	<
大於等於 (不小於)	>=
小於等於 (不大於)	<=
等於	==
不等於	!=

```
4   >   3
4   <   3
4   ==  3
4   !=  3
4   >   3   >   2
```

邏輯運算子

- 邏輯運算子的運算結果有 **0** 與不是 **0** 兩種可能
  - 不是 0 代表的意思是真 (成立)
  - 0 代表的意思是假 (不成立)
- 一般在 **C** 習慣上還是會避免直接使用 **and** 與 **or** 來表示邏輯運算而是使用符號：

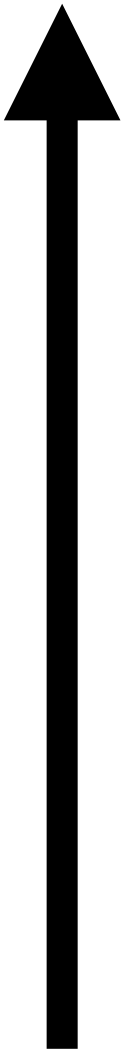
運算意義	運算符號
而且 (and)	&&
或者 (or)	||
非 (not)	!

```
1 && 1
1 && 0
1 || 1
1 || 0

4 > 3 && 3 > 2
4 > 3 && 4 < 3
4 > 3 || 4 < 3

!1
```

運算子優先順序表



運算符號	平手時運算順序
( )	由左至右
!	由左至右
* / %	由左至右
+ -	由左至右
< > <= >=	由左至右
== !=	由左至右
=	由右至左
&&	由左至右
||	由左至右

if ...

- if (表示式) { ... }
  - 如果 表示式 為真就 ...
- 真或假
  - 非零或零

```
if (80 >= 60)
{
   printf("PASSED !\n");
}
if (80 < 60)
{
   printf("FAILED !\n");
}
```

```
int grade = 80;
if (grade >= 60)
{
   printf("PASSED !\n");
}
if (grade < 60)
{
   printf("FAILED !\n");
}
```

《範例》比較兩數大小

- 試寫一程式（**cmp**）讓使用者輸入兩個數字後顯示其中比較大的給使用者看：

```
請輸入第一個整數 : 3
請輸入第二個整數 : 4
比較大的整數是   : 4
```

- 提示：(程式片段)

```
int max;
if (num1 >= num2) {
  max = num1;
}
if (num1 < num2) {
  max = num2;
}
printf("比較大的整數是 %d\n", max);
```

《練習》簡易版猜數字

- 試寫一個程式 **(guess)**，在程式內部預設一個整數作為猜數字遊戲的答案。當使用者執行程式後，需要輸入一個整數，如果該整數與程式預設的答案不同，請顯示是比較大或者比較小；如果該整數與程式預設的答案相同，請恭喜使用者：

```
請輸入你的猜測 ： 3
太小了！

請輸入你的猜測 ： 5
太大了！

請輸入你的猜測 ： 4
答對了！
```

【補充】位元運算子

運算意義	運算符號
Bitwise AND	&
Bitwise OR	|
Bitwise XOR	^
Bitwise NOT	~
Bitwise Left Shift	<<
Bitwise Right Shift	>>

習題 (1)

- **[E0201]** 試寫一程式，印出下面這個變數值：
  - **double** x = 3000000000000000.5;
- **[E0202]*** 試寫一程式，印出下面式子的計算結果：
  - 3000000000000000.5+0.05
- **[E0203]** 試寫一程式，輸入英哩換算後印出公里 **(四捨五入至小數點後一位) [公里 = 英哩 * 1.6]**
- **[E0204]** 試寫一程式印出 **129263*54628** 的結果
  - 與 [E0110] 相同
- **[E0205]*** 試寫一程式算出 **1292635428** 三次方的值 **(2159872744519190546127922752)**

習題 (2)

- **[E0206]** 試寫一程式，輸入一有號整數，顯示該整數是正整數 **(>=0)** 或負整數 **(< 0)**
- **[E0207]** 試寫一程式，輸入一字元，顯示該字元是數子 **(0-9)**、英文字元 **(a-zA-Z)** 或其他符號
  - 字元 (**char**) 請在 **scanf** 內用 **%c** 讀入，在 **printf** 內用 **%c** 印出
- **[E0208]** 試寫一程式，輸入一個英文小寫字元，將字元轉換為大寫印出
- **[E0209]** 試寫一程式，輸入一個英文字元，將字元的大寫印出 **(**不限制輸入的字元為大寫或小寫**)**

習題 (3)

- **[E0210]** 試寫一程式，輸入兩個整數，將兩個整數由小到大印出。

  範例輸入一： 3 6　　範例輸出一： 3 6

  範例輸入二： 6 3　　範例輸出二： 3 6

- **[E0211]** 試寫一程式，讓使用者輸入一八位整數然後將數字直排顯示

```
請輸入一個八位數整數： 38603456
3
8
6
0
3
4
5
6
```

習題 (4)

- **[E0213]** 試寫一程式，讓使用者輸入身分證字號的前九碼後顯示該身分證字號的第十碼 (驗證碼) `%c%d`
  - 我們的檢查碼計算方式：(A: 10, B:11, C:12, ..., Z:36)

A	1	2	3	4	5	6	7	8
1	0	1	2	3	4	5	6	7	8
x1	x9	x8	x7	x6	x5	x4	x3	x2	x1

  - 總和 = 1x1 + 0x9 + 1x8 + 2x7 + 3x6 + 4x5 + 5x4 + 6x3 + 7x2 + 8x1 = **121**
  - **121**%10 = **1**, (10-**1**)%10 = **9**