Appendix A

基數

在文明發展的過程中，使用過許多表示數字的系統。有一些系統，好比說羅馬數字，對於算術來說較為不方便。在另外一方面，北印度（Hindi）數字系統經過修改並傳至歐洲以後，成為阿拉伯數字，並且這個數字系統對於數學家、科學家與商人等所需要的計算，都相當的適合。現代電腦數字系統基於佔位符（placeholder）的概念，使用首次出現在北印度數字系統的符號 0。然而，他們將這樣的使用原則歸納統合至其它的基數上。雖然現今我們使用的標記都是以 10 作為基數（下一節會加以說明），但在電腦的世界中，常常還是使用以 8（八進位）, 16（十六進位）和 2（二進位）為基數的數字。

十進位數字（以 10 為基數）

我們書寫數字的方法是以 10 的次方為基礎，例如數字 2468，2 代表 2 個 1000，4 代表 4 個 100，6 代表 6 個 10，8 代表 8 個 1：

$$2468 = 2\times1000 + 4\times100 + 6\times10 + 8\times1$$

1000 等於 10×10×10，或 10 的三次方，並且可以寫成 10^3。以此記法，上面運算式可表示成：

$$2468 = 2\times10^3 + 4\times10^2 + 6\times10^1 + 8\times10^0$$

因為我們皆以 10 的次方為基礎表示數值，所以稱之基數 10 的表示法，或十進位（decimal）表示法。其實也可以選擇其它數字作為基數，C++提供基數 8（八進位）和基數 16（十六進位）表示整數（注意，10^0 等於 1，此定理適用任何非零數字的 0 次方）。

八進位整數（以 8 為基數）

八進位（octal）數字是以 8 的次方為基礎，所以基數 8 表示法使用數字 0–7 表示數字。C++以前置詞 0 表示八進位表示法。因此，0177 為一八進位數值，你可以用 8 的次方轉換成基數 10 的同等值：

八進位	十進位
0177	= 1×82 + 7×81 + 7×80
	= 1×64 + 7×8 + 7×1
	= 127

UNIX 作業系統一般以八進位表示數值，所以 C++和 C 提供八進位表示法。

十六進位數值（以 16 為基數）

十六進位（hexidecimal）數值是以 16 的次方為基礎，意思是十六進位的 10 表示的值是 16 + 0，或 16。為了表示 9 到 16 之間的值，我們需要更多的數字，標準的十六進位表示法，是採用字母 a–f。C++ 接受大小寫的 a–f 字元，如表格 A.1 所示。

表 A.1 十六進位的數字

十六進位數字	十進位值
a 或 A	10
b 或 B	11
c 或 C	12
d 或 D	13
e 或 E	14
f 或 F	15

C++以 0x 或 0X 表示十六進位數值。因此 0x2B3 為十六進位數值。從十六進位轉成十進位值，方法如下：

十六進位	十進位
0x2B3	$= 2\times16^2 + 11\times16^1 + 3\times16^0$
	$= 2\times256 + 11\times16 + 3$
	$= 691$

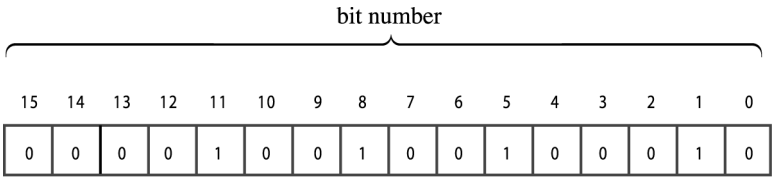
硬體文件一般皆以十六進位表示，如記憶體位置和埠（port）的位置。

二進位數值（以 2 為基數）

無論是以十進位，八進位，或十六進位表示整數，電腦內部均存成二進位（binary）或基數 2 的值。二進位表示法只使用兩種數字 0 或 1。例如，10011011 為一個二進位數字。注意 C++ 並未提供數值的二進位表示法。二進位數值是以 2 的次方為基礎：

二進位	十進位
10011011	$= 1\times2^7 + 0\times2^6 + 0\times2^5 + 1\times2^4$
	$+ 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$
	$= 128 + 0 + 0 + 16 + 8 + 0 + 2 + 1$
	$= 155$

二進位表示法與電腦記憶體的用法十分相符，其中每個獨立單元稱之**位元**（bit），每個位元可以設成開或關。一般 0 代表關，1 代表開。記憶體一般的組成單位為**位元組**（byte），每個位元組有 8 個位元。位元在位元組內的位置正好對應成 2 的次方，因此最右位元是位元 0，下一位元是位元 1，以此類推。例如，圖 A.1 表 2-位元組的整數。



$$\text{值} = 1 \times 2^{11} + 1 \times 2^{8} + 1 \times 2^{5} + 1 \times 2^{1}$$
$$= 2048 + 256 + 32 + 2$$
$$= 2338$$

圖 A.1 一個為 2-位元組的整數值

二進位和十六進位

十六進位表示法經常是看二進位資料的方便方法，如記憶體位址或是儲存位元旗標指定的整數。原因是每個十六進位數字會對應至 4 個位元。表 A.2 顯示這種對應關係。

表 A.2 十六進位的數字和等值的二進位

十六進位數字	等值之二進位
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100

十六進位數字	等值之二進位
D	1101
E	1110
F	1111

要將十六進位值轉成二進位，只要將十六進位數字取代成，對應等值的二進位即可。例如，十六進位值 0xA4 對應的二進位是 1010 0100。同樣的，二進位值也可以容易的轉成十六進位表示法，作法是以 4 位元為單位，將它轉成等值的十六進位數字。例如二進位值 1001 0101 會轉成 0x95。

何謂 Big Endian 和 Little Endian？

說也奇怪，兩個都使用整數的二進位表示式的電腦平台，但也許不會代表相同的數字。例如，Intel 機器利用 Little Endian 架構儲存位元組，而 Motorola 處理器，IBM 大型主機，SPARC 處理器，以及 ARM 處理器都使用 Big Endian 架構（但是，最後兩個系統可以指定使用兩者中的任何一種方法）。

名詞 Big Endian 和 Little Endian 是衍生自“Big End In”和“Little End In”——這是記憶體字組（一般是 2-位元組單元）之位元組的參考順序。在 Intel 電腦上（Little Endian），是先儲存低順位（low-order）的位元組。這意思是十六進位值，如 0xABCD 會以（0xCD 0Xab）儲存在記憶體中。Motorola（Big Endian）機器則以相反的順序儲存相同的值，所以 0xABCD 以（0xAB 0xCD）儲存在記憶體中。

對於這些專有名詞，Jonathan Swift 在 *Gulliver's Travels* 書中作了很完整的解釋。Swift 虛構了一個小人國，並且在裡面有兩個愛爭論的派系，來挖苦許多政治爭論的不合理事物：在 Big Endian 那裡，主張打破雞蛋的位置要在較為粗圓的端點；而 Little Endian 則是支持雞蛋要由較細小的端點那裡打破。

你是一個軟體工程師，應該瞭解你所使用之平台的字組順序。除此以外，它會影響傳送於網路上之資料的解釋方式，以及資料如何儲存在二進位檔案中。在上面的範例中，2-位元組記憶體樣式 0xABCD 在 Little Endian 機器上會解釋為十進位值 52,651，而在 Big Endian 機器上解釋為十進位值 43,981。