2. call by value? call by address (或call by pointer)? call by reference? –

swap(int\* a, int\* b) v.s. swap (int &a, int &b)  
  
JAVA中的reference與C++的reference意義上並不相同，卻使用同一個字，這也是reference容易造成混淆的原因。在此，我們暫不考慮JAVA中reference的觀念 (關於java中reference的觀念，請參考[Reference in JAVA -- 淺談java的指標](http://sandwichc-life.blogspot.com/2007/10/reference-in-java-java.html))，純粹把主題放在C/C++上。  
呼叫副函式時，call by value, address, 或reference是三種不同的參數傳遞方式。其意義如下：

* call by value  
  假設函式A呼叫函式B(p, q)，則B中的p和q是「複製」自函式A所傳入的參數，B中對p, q所做的任何運算都不會影響到A中的p和q，因為B執行完後，並不會把複製的p, q存回到A中。這種參數傳遞方式，我們稱之為call by value。  
  以swap這個常見的函式為例，若swap寫成下面的樣子：

void swap(int a, int b){  
int tmp = a;  
a = b;  
b = tmp;  
}

則呼叫

swap(x, y)

後，x和y的值並不會有變化。

* call by address (或call by pointer)  
  利用指標來做參數傳遞，這種方法骨子裡仍是call by value，只不過call by value的value，其資料型態為指標罷了。我們同樣看看用call by address來寫swap交換兩個integer的例子。

void swap(int\* a, int\* b){  
int tmp = \*a;  
\*a = \*b;  
\*b = tmp;  
}

呼叫swap時，要寫成swap(&x, &y)。呼叫swap時，x的指標 (x的儲存位置) 與y的指標 (y的儲存位置) 會被複製一份到swap中，然後把該位置內所記載的值做更換。swap結束後，&x (address of x) 和&y (address of y) 依然沒變，只是address of x所記錄之變數值與address of y所記錄之變數值交換了。因為&x 和&y 其實是利用call by value在傳，因此，call by address其實骨子裡就是call by value。

* call by reference  
  這是C++才加進來的東西，C本身並沒有call by reference。call by reference基本上是把參數做個別名 (alias)，以下面的swap為例：

swap(int &a, int &amp;b){  
int tmp = a;  
a = b;  
b = tmp;  
}

未來使用時，只要呼叫swap(x, y)，就可以讓x和y的值交換。在這個例子中，a 就是 x, b就是 y。這個觀念在上一節已經提過，在此不再贅述。