3. pointer to pointer, reference to pointer (int\*\* v.s. int\*&)

當我們用call by pointer (或address) 來傳遞參數時，被呼叫的函式複製一份pointer的值過去。但是，當我們想在函式內改變pointer的值 (而非pointer所指向之變數的值)，而且改變的效果要能在函式外看得到時，call by pointer就不足夠用了。此時應該用的是"call by pointer to pointer"或"call by reference to pointer"。我們先看下面的例子：

int g\_int = 0;  
void changePtr(int\* pInt){  
pInt = &g\_int;  
}  
void main(){  
int localInt = 1;  
int\* localPInt = &localInt;  
changePtr(localPInt);  
printf("%d\n", \*localPInt);  
}

在這個例子中，印出來的數字仍然會是localInt的1，因為changPtr中的pInt是由localPInt「複製」過去的，對pInt做改變並不會反應到localPInt身上。  
我們先用pointer to pointer對localPInt做改變，請看下例。

int g\_int = 0;  
void changePtr(int\*\* pInt){  
\*pInt = &g\_int;  
}  
void main(){  
int localInt = 1;  
int\* localPInt = &localInt;  
changePtr(&localPInt);  
printf("%d\n", \*localPInt);  
}

本例中，印出來的數字會是g\_int的0。changePtr函式中的pInt是由&localPInt複製所得，因此對pInt做改變並不會影響main中的&amp;localPInt (資料型態：pointer to pointer to integer)。但在changePtr函式中我們改變的對象是pInt所指向的內容，因此這項改變在main中會顯示出來。  
  
同樣的功能，我們也可改用reference to pointer來完成。但同樣切記，reference是C++才有的功能，因此reference to pointer也只能在支援C++的環境中使用。

int g\_int = 0;  
void changePtr(int\* &refPInt){  
refPInt = &g\_int;  
}  
void main(){  
int localInt = 1;  
int\* localPInt = &localInt;  
changePtr(localPInt);  
printf("%d\n", \*localPInt);  
}

這段程式印出來的數字會是0。因為在changePtr中，我們宣告的參數型態為int\* &，即：reference to pointer to integer。因此，main中的localPInt與changePtr函式中的refPInt其實是「同一件東西」。  
  
另一種常見的混淆是pointer array (指標陣列) 與pointer to pointers，因為兩種都可以寫成\*\*的型式。如，int\*\*可能是pointer to pointer to integer，也可能是integer pointer array。但pointer array的觀念相對來講要簡單且直觀許多，這裡我們就暫不花篇幅敘述。常見的例子：main(int argc, char\*\* argv)其實應該是main(int argc, char\* argv[])。