指標

9-1 位址

- 理解位址的概念
  - C語言中所謂的「位址」是使用在直接顯示記憶體上的位置，也就是記憶體上的「地址」。它就像是電腦裡的住址一樣，大多是使用16進位的數值，如0x1000、0x1004…等數值來表示。
- 變數的位址
  - 想要知道儲存變數的記憶體的位址，就得使用位址運算子（address operator）&，而這個符號要寫在變數的前面，就像這樣：

    & 變數名稱

- 範例：

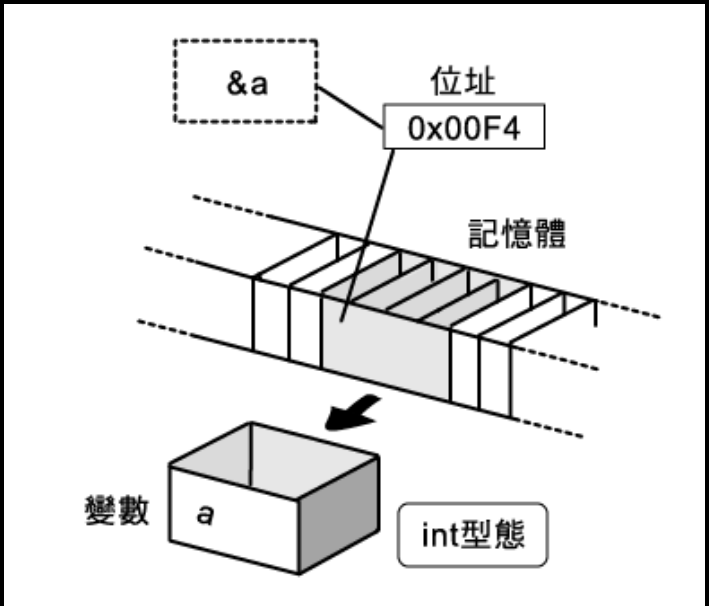
```
…
int a;
a = 5;
printf("變數a的值爲%d。\n", a);
printf("變數a的位址爲%p。\n", &a);
…
```

輸出變數a的位址

- 執行畫面：
  - 變數a的值爲5。
  - 變數a的位址爲0x00F4。

```
printf ( "變數a的位址為%p。\n" , &a ) ;
```

輸出位址時要使用**%p**作爲轉換規格



Sample1.c ▶ 輸出位址

```
#include <stdio.h>

int main(void)
{
    int a;

    a = 5;

    printf("變數 a 的值為 %d。\n", a);
    printf("變數 a 的位址為 %p。\n", &a);

    return 0;
}
```

輸出變數 a 的位址

9-2 指標

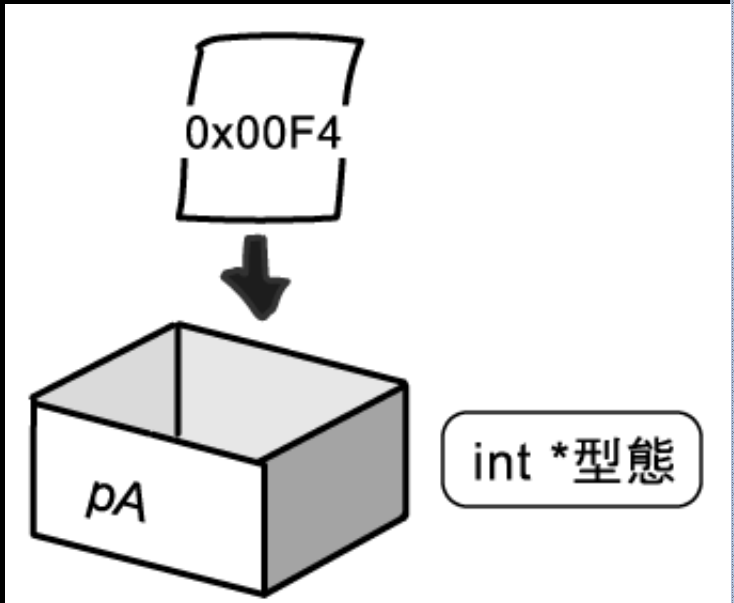
- 理解指標的原理
  - C語言有用來儲存位址的特殊變數，這種變數被稱爲指標(pointer)。
  - 指標的宣告：

    資料型態 *指標名稱;

  - 像這樣宣告指標的話：

    int *pA;

    就表示可以儲存int型態變數之位址的指標爲pA。也可以說是「指向int型態的指標pA」。



- 範例：

```
…
int main()
{
        int a;
        int *pA;
        a = 5;
        pA = &a;
        printf("變數a的值爲%d。\n", a);
        printf("變數a的位址爲%p。\n", &a);
        printf("指標pA的值爲%p。\n", pA);
       return 0;
}
```

1.宣告指標pA

2.將變數a的位址儲存到pA中

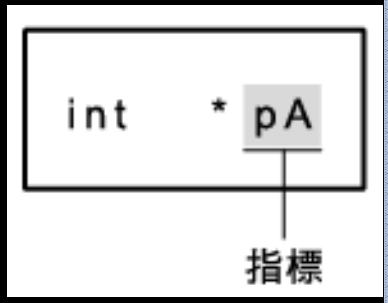
3.輸出pA的值 (變數a的位址)

- 執行畫面：

變數a的值爲5。

變數a的位址(&a)爲0x00F4。

指標pA的值爲0x00F4。←—指標的內容爲變數a的位址



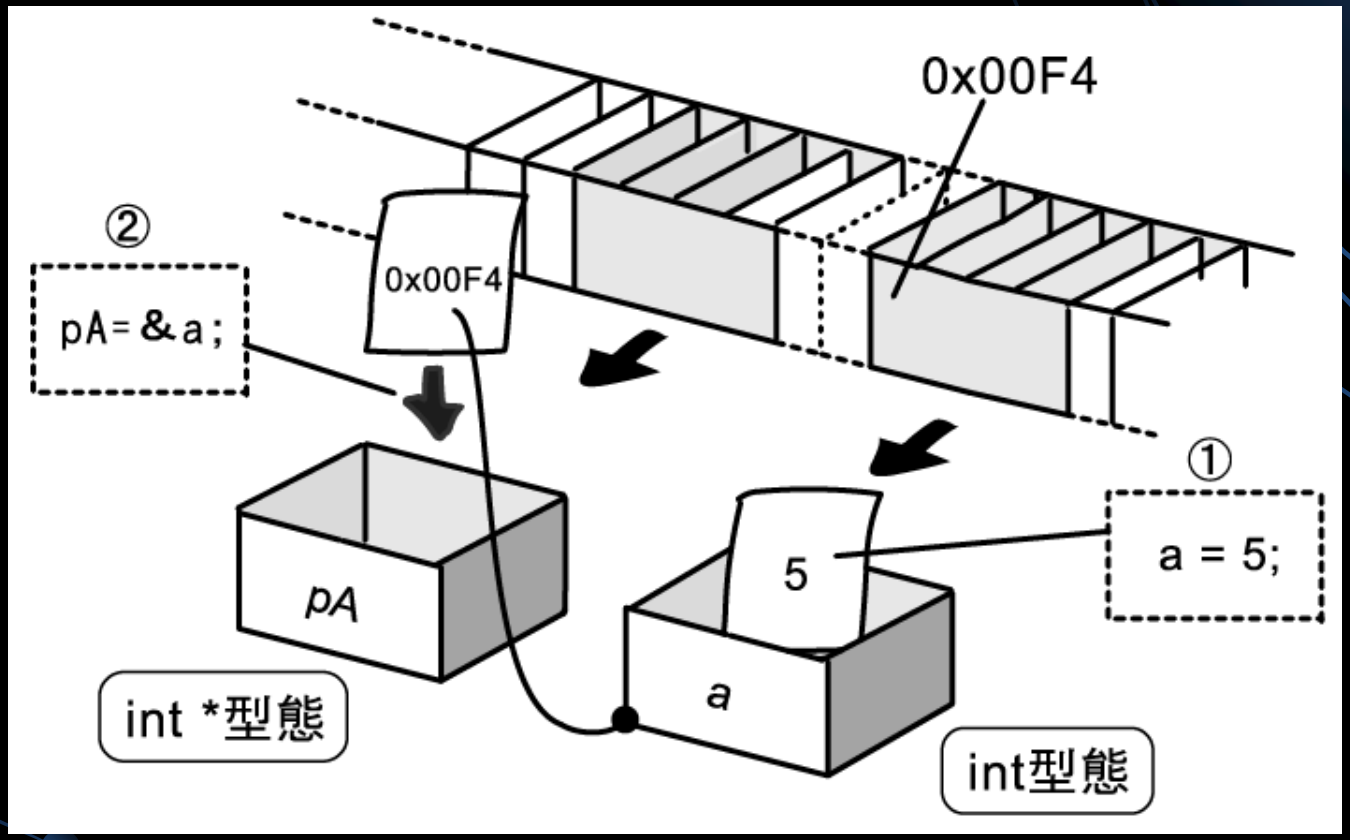
Sample2.c ▶ 將位址儲存於指標內

```
#include <stdio.h>

int main(void)
{
    int a;
    int *pA;

    a = 5;
    pA = &a;

    printf("變數 a 的值為 %d。\n", a);
    printf("變數 a 的位址為 %p。\n", &a);
    printf("指標 pA 的值為 %p。\n", pA);

    return 0;
}
```

①宣告指標 pA

②將變數 a 的位址儲存到 pA 中

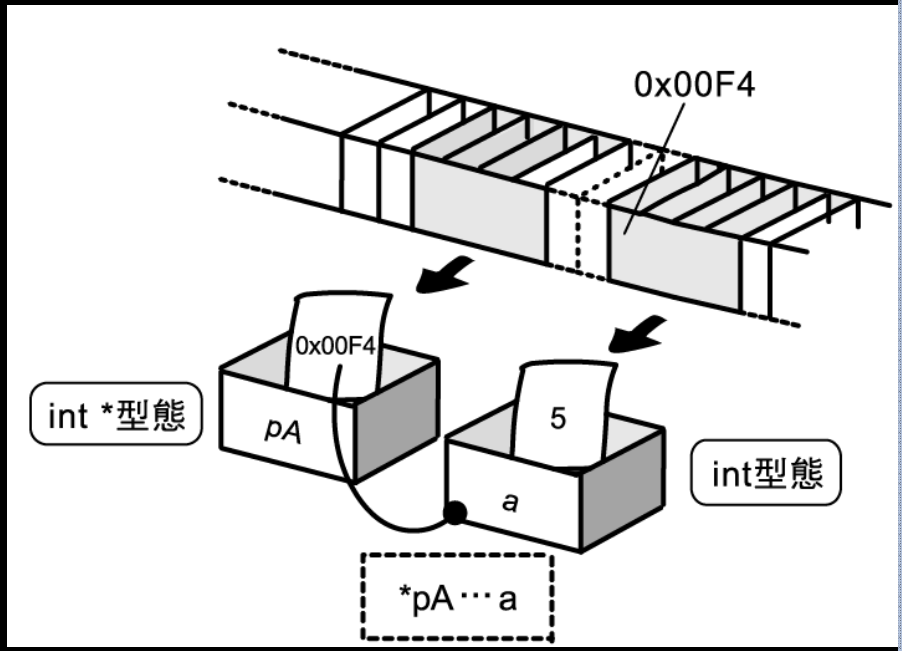
③輸出 pA 的值（變數 a 的位址）



- 1. 將5指派給int型態的變數a。
- 2. 將變數a的位址指派給int*型態的指標pA。

- 從指標理解變數的值
  - 如果將變數的位址儲存到指標的話，就可以從該指標倒推得知該變數原本的值。
  - 想要從指標來追溯變數的值時，就要對指標使用*這個運算子。*運算子就叫做間接參照運算子（indirection operator）。
  - 間接參照運算子的語法：

    ***指標名稱;**



- 範例：

```
…
int main()
{
    int a;
    int *pA;

    a = 5;
    pA = &a;
    printf("變數a的值爲%d。\n", a);
    printf("變數a的位址爲%p。\n", &a);
    printf("指標pA的值爲%p。\n", pA);
    printf("*pA的值爲%d。\n", *pA);
    return 0;
}
```

將變數a的位址儲存於pA當中

如果使用*符號，就可以得知指標所指向之變數的值

執行結果：

變數a的值爲5。
變數a的位址爲0x00F4。
指標pA的值爲0x00F4 。
*pA的值爲5。

輸出指標所指向之變數的值

Sample3.c ▶ 間接參照

```
#include <stdio.h>

int main(void)
{
    int a;
    int *pA;

    a = 5;
    pA = &a;

    printf("變數 a 的值為 %d。\n", a);
    printf("變數 a 的位址為 %p。\n", &a);
    printf("指標 pA 的值為 %p。\n", pA);
    printf("*pA 的值為 %d。\n", *pA);

    return 0;
}
```

將變數 a 的位址儲存於 pA 當中

使用 * 符號，可以得知指標所指向之變數的值

- 指標概念的整理
  - 首先是變數a和其位址的關係：

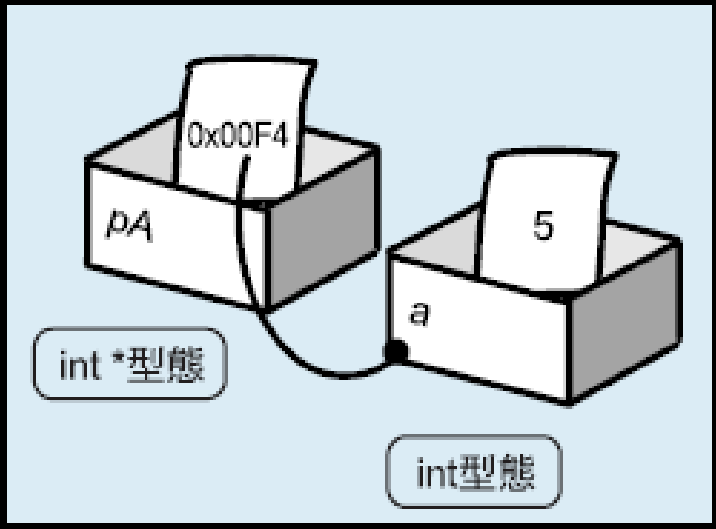
a	變數a
&a	變數a的位址

  - 若「pA = &a;」的話，其關係如下：

pA	儲存變數a位址的指標
*pA	儲存變數a位址的指標所指向的變數 → 變數a

  - 如果沒有進行「pA = &a;」的指派，後面的兩項關係便不會成立。

- 指標宣告的複習
  - 「int* pA;」就是「指標pA爲int*型態」的意思，也可以看成是「*pA爲int型態」。
  - 「int* pA;」和「int *pA;」的意思是一樣的。
  - 以逗號區隔宣告複數指標時，必須宣告爲「 int* pA, *pB; 」，而不是「 int* pA, pB; 」，因爲後者會等同於「 int* pA; int pB; 」。



- 指定其他的位址到指標中
  - 由於指標也是一種變數，所以也可以變更其中的值。
  - 範例：

```
...
int main()
{
    int a, b;
    int *pA;
    a = 5;
    b = 10;
    pA = &a;
    printf("變數a的值爲%d。\n ", a);
    printf("指標pA的值爲%p。\n “, pA);
    printf("*pA的值爲%d。\n “, *pA);
    pA = &b;
    printf("變數b的值爲%d。\n “, b);
    printf("指標pA的值變更爲%p。\n “, pA);
    printf("*pA的值爲%d。\n “, *pA);
    return 0;
}
```

指派變數**a**的位址

指派變數**b**的位址

執行畫面

變數a的值爲5。
指標pA的值爲0x00F4。
*pA的值爲5。
變數b的值爲10。
指標pA的值爲0x00F0。
*pA的值爲10。

一開始指向變數**a**

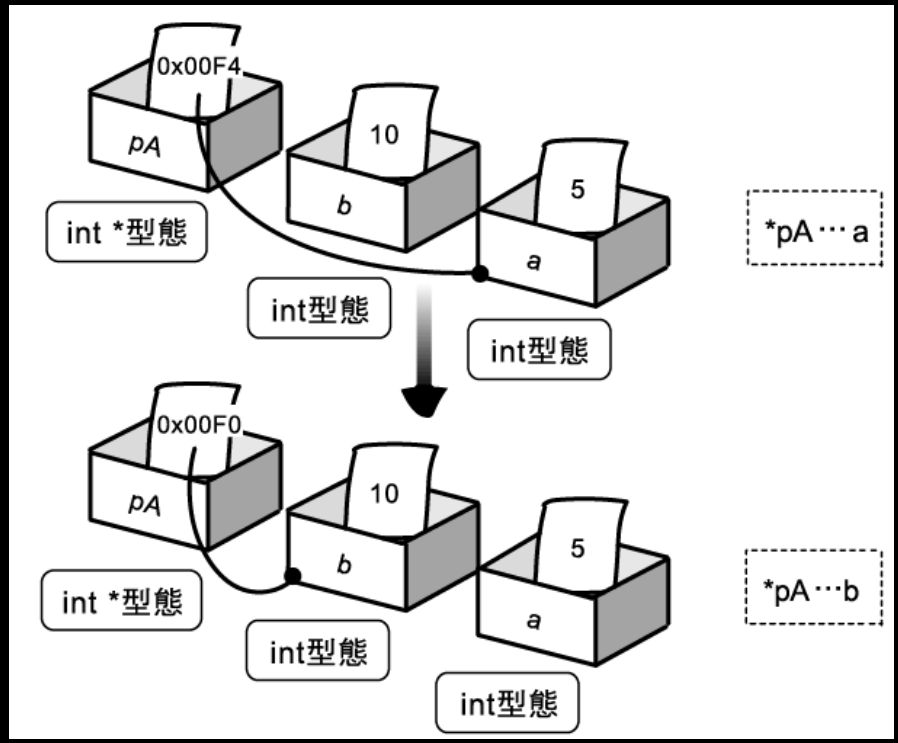
變更爲變數**b**的位址

變成指向變數**b**

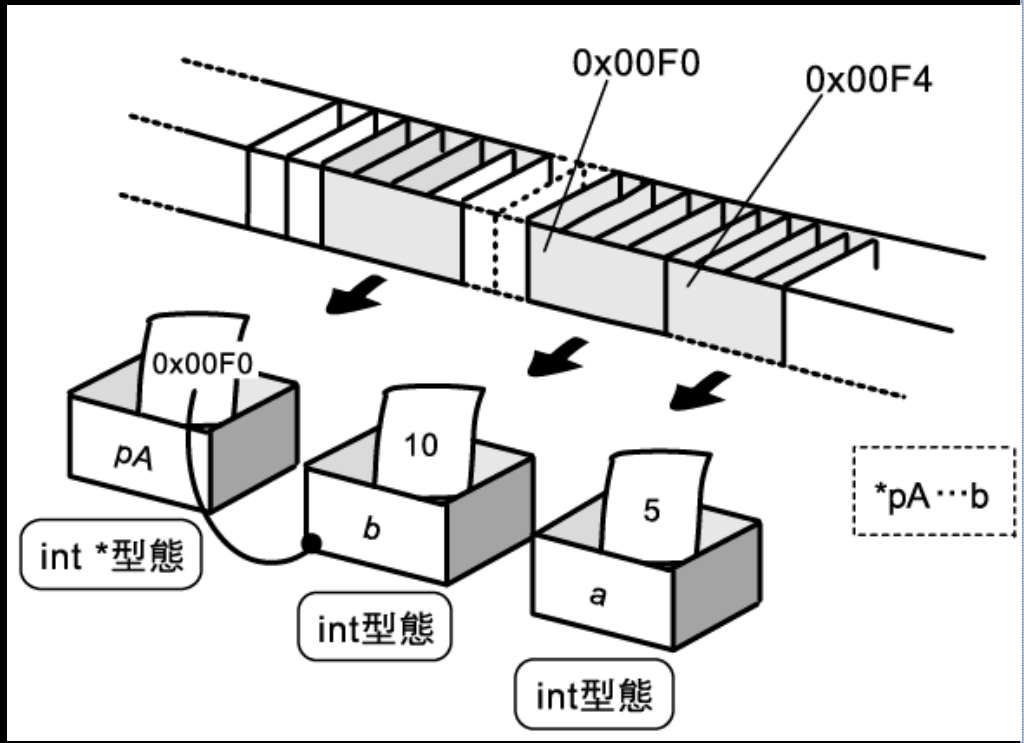
Sample4.c ▶ 變更指標的值

```
#include <stdio.h>

int main(void)
{
    int a, b;
    int *pA;

    a = 5;
    b = 10;

    pA = &a;          // 指定變數 a 的位址

    printf("變數 a 的值為 %d。\n", a);
    printf("指標 pA 的值為 %p。\n", pA);
    printf("*pA 的值為 %d。\n", *pA);

    pA = &b;          // 指定變數 b 的位址

    printf("變數 b 的值為 %d。\n", b);
    printf("指標 pA 的值變更為 %p。\n", pA);
    printf("*pA 的值為 %d。\n", *pA);

    return 0;
}
```

- 指派過程：
  - 先把變數a的位址指派到指標pA，此時輸出*pA的值，就會輸出與變數a相同的值「5」。
  - 再指派「pA = &b;」來變更指標的值，就會將變數b的位址儲存到指標pA。



- 再次輸出*pA，就會輸出與變數b相同的值「10」。
- 也就是說，變更指標的值就可以指向其他變數了。
- 原則上，指標只能儲存指定型態的值之位址。
- 請注意，必須先將變數的位址指派給指標，才能開始使用指標，否則會發生不可預期的錯誤。



- 如果不指定給指標的話？
  - 範例：

```
/* 這個程式碼是錯的 */
int a = 5;
int *pA;
```

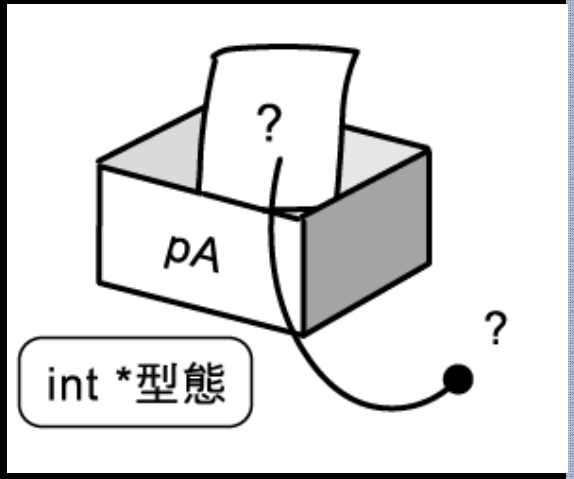
沒有指定給**pA**

```
printf("指標pA的值爲%p。\n", pA);
printf("*pA的值爲%d。\n", *pA);
…
```

不知道是指向何處

這裡少了「pA = &a;」這段程式碼。也就是說，在指標pA當中並未儲存任何位址。所以指標顯示爲未指向任何一處的狀態。即使寫成*pA，也無法得到有意義的值。

一定要指定位址後再使用指標。



使用指標來變更變數

- 範例：

```
…
int main()
{
  int a;
  int *pA;
  a = 5;
  pA = &a;
   printf("變數a的值爲%d。\n", a);
  *pA = 50;
   printf("把50指定給*pA。\n");
   printf("變數a的值爲%d。\n", a);
   return 0;
}
```

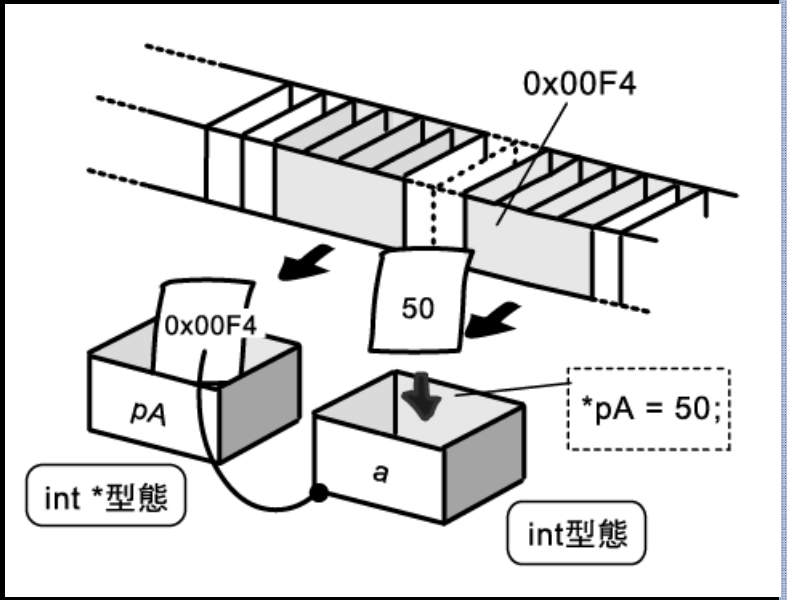
執行畫面

```
變數a的值爲5。
將50指派給*pA。
變數a的值爲50。
```

變數a的值被變更了

將值指派給***pA**，也就是變數**a**

試著將變數**a**的值輸出看看



Sample5.c ▶ 使用指標改變變數的值

```
#include <stdio.h>

int main(void)
{
    int a;
    int *pA;

    a = 5;
    pA = &a;

    printf("變數 a 的值為 %d。\n", a);

    *pA = 50;

    printf("把 50 指定給 *pA。\n");
    printf("變數 a 的值為 %d。\n", a);

    return 0;
}
```

把值指定給 *pA，也就是變數 a

試著將變數 a 的值輸出看看

9-3 引數與指標

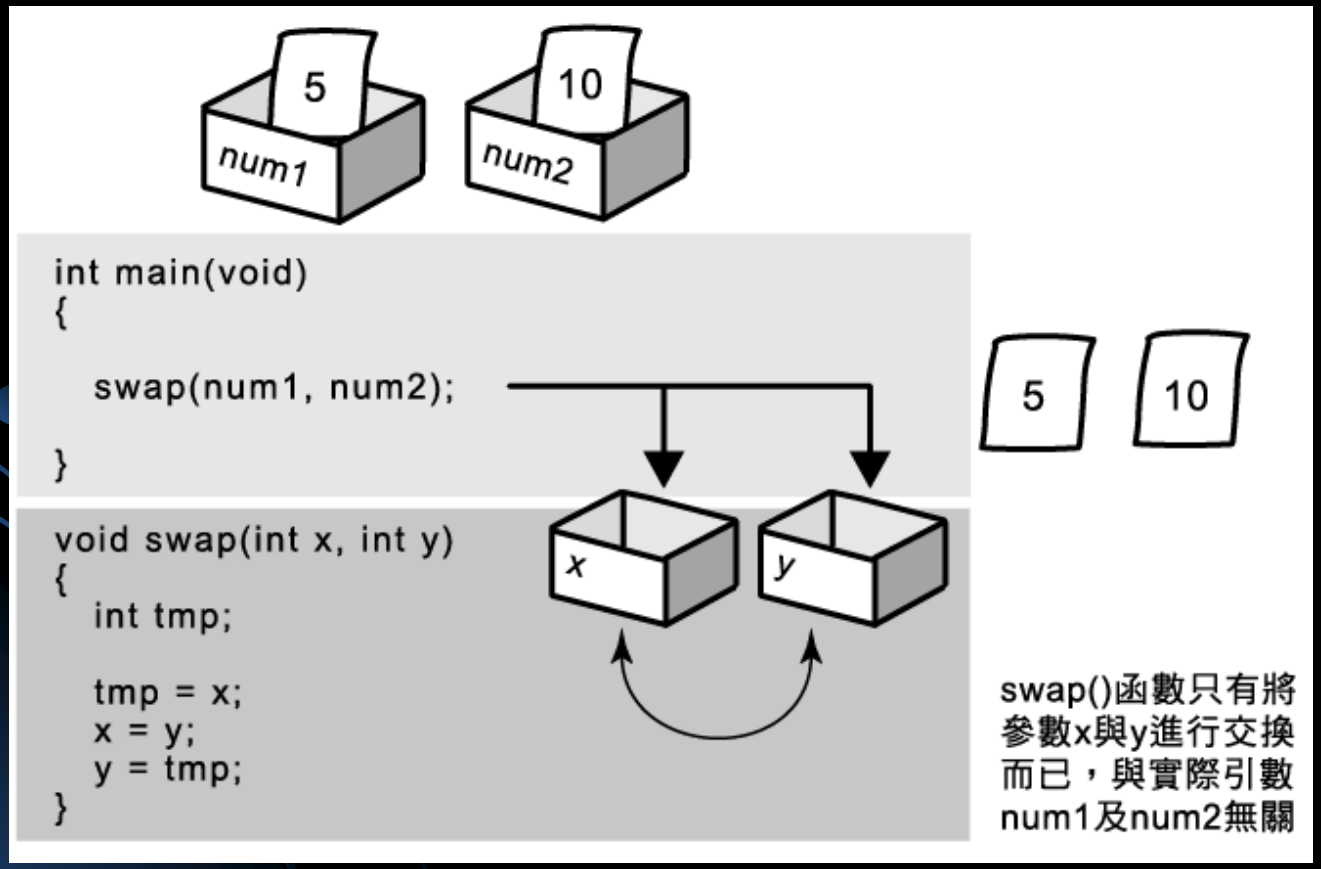
- 將參數傳遞至函數的方法
  - 當函數傳遞引數的時候，只有引數的「值」會傳到函數當中，以上的引數傳遞方式就稱爲傳值（pass by value）。
  - 如果把指標當作函數的參數來使用的話，就可以變更引數方的值了。此外，當函數被呼叫出來的時候，引數的位址會被傳遞到函數當中，這個動作就稱爲傳址（pass by reference）。
  - 如果不想在函數內變更引數，可以在參數上加上const。

Sample6.c ▶ 嘗試使用錯誤的函數

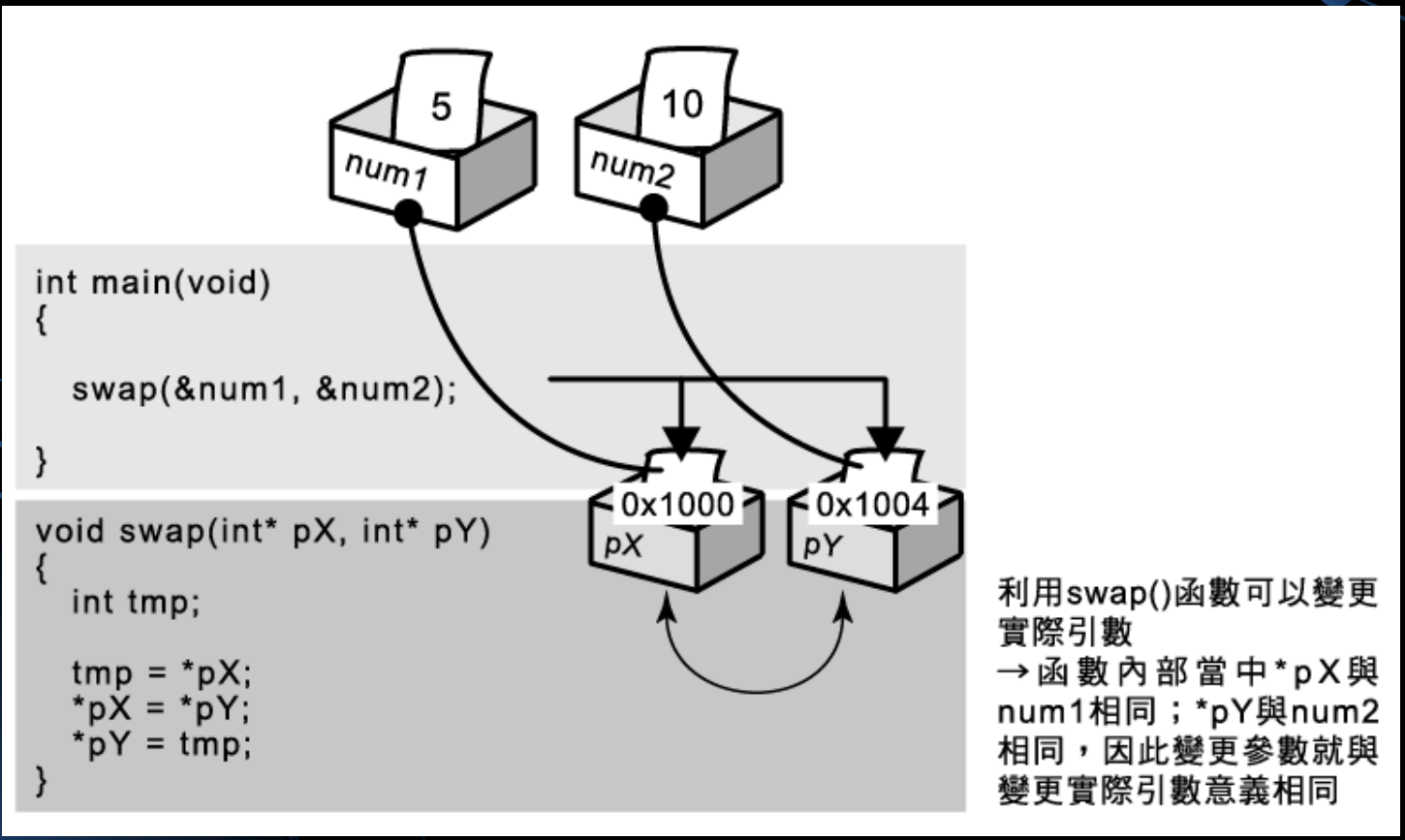
```
#include <stdio.h>

/* 錯誤的 swap 函數的宣告 */
void swap(int x, int y);

int main(void)
{
    int num1 = 5;
    int num2 = 10;

    printf("變數 num1 的值為 %d。\n", num1);
    printf("變數 num2 的值為 %d。\n", num2);
    printf("將變數 num1 與 num2 的值進行交換。\n");

    swap(num1, num2);

    printf("變數 num1 的新值為 %d。\n", num1);
    printf("變數 num2 的新值為 %d。\n", num2);

    return 0;
}

/* 錯誤的 swap 函數的定義 */
void swap(int x, int y)
{
    int tmp;

    tmp = x;
    x = y;
    y = tmp;
}
```

已經把 swap()函數呼叫出來了，但是…

- 範例：
  - 若是想要定義交換引數x與y的swap函數，就不能使用傳值的方式來傳遞引數，因爲傳值只是傳遞引數中的值而已，並不會對引數本身產生改變，所以只會交換參數。



- 範例：
  - 如果使用傳址的方式定義swap函數，也就是用指標來作爲參數，就會將變數的位址作爲引數來傳遞，所以此函數可以正確地交換引數。



Sample7.c ▶ 將指標用作函數的引數

```
#include <stdio.h>

/* swap函數的宣告 */
void swap(int *pX, int *pY);

int main(void)
{
    int num1 = 5;
    int num2 = 10;

    printf("變數num1的值為%d。\n", num1);
    printf("變數num2的值為%d。\n", num2);
    printf("把變數num1與num2的值進行交換。\n");

    swap(&num1, &num2);

    printf("變數num1的新值為%d。\n", num1);
    printf("變數num2的新值為%d。\n", num2);

    return 0;
}

/* swap函數的定義 */
void swap(int *pX, int *pY)
{
    int tmp;

    tmp = *pX;
    *pX = *pY;
    *pY = tmp;
}
```

呼叫出新的 swap()函數（傳遞位址）