草之程式小記

將遇到的一些程式小事情記錄下來。

(C) 簡單搞懂指標(pointer)、指標陣列(pointers of array, int *foo[]) 與指向陣列的指標 (pointer to array, int (*bar)[])

3月 27, 2018

(一) 廢話

指標一直以來都是初學者的一個夢魘，有時後就算是老手也不一定會搞清楚，而且不常用的話也容易忘記，但指標在C理面是一個非常重要的核心技術，在很多公司的面試題都是必出的主題，因此能搞清楚指標對一個工程師而言是百利無一害的。

(二) 前言

在開始之前，我必須先強調以下看似廢話但其實是很重要的觀念，之後會提到他們是多麼重要，反正就先記起來吧，就算知道了也還是多想他一下。

什麼"資料型態(data type)"就存什麼樣的"型態資料"。因此指標變數就是存記憶體位址

真的很廢話，但在指標裡面會是很重要的觀念。

(三) 什麼是指標

指標(pointer)就是一個變數的記憶體的地址，在宣告的時候使用"*"放在變數型態之後。他可以被視為一種資料型別的修飾，因此若宣告了一個指標變數時，就代表其資料型態是指標類型，因此這個變數存的內容就是記憶體，不管他本身原本是什麼型態，只要在程式理面使用該變數(不額外加符號)，就是使用他的記憶體。就是上面強調的「什麼型態存什麼資料」。

(四) 基本符號

在指標裡，「*」是一個非常重要的符號，他同時可以是兩種意思，分別是：

1. 指標(pointer) - 這個符號**只有在宣告的時候**才會出現，表示該變數是**指標型態**，其存的內容就是記憶體位址；可以想像成是一個「**鎖**」。

2. 取值運算子(Dereference operator) - 在**宣告以外時所有**出現在變數前的都是這種(當然不包含乘法)，表示依照這個變數所存的地址，去取得該記憶體位址理面存的值；可以想像成是把該變數的鎖打開的「**鑰匙**」。

另外還有一個符號：

「 & 」: 取址運算子(Address-of operator) - 取得某一變數本身存放的記憶體位址。

(五) 宣告與使用

在使用指標之前，如果沒有進行記憶體配置的話，原則上初始值為NULL，要讓程式能動態配置一個記憶體，就必須使用**malloc**。
在Linux man pages中，malloc 的原型為以下所示：

void *malloc(size_t *size*);

傳入參數僅有一個，就是要配置的記憶體大小(單位為Bytes)，要配置時，必須搭配sizeof(型態)乘上要配置的大小才行。配置成功後，會回傳該長度記憶體的起始位址。
傳回型態為void*，不過指標變數存的就是"記憶體位址"，甚麼型態並不是那麼重要，且記憶體的長度受硬體架構所影響，同一個系統下都為固定值，因此只要在宣告時進行轉型就可以繼續使用了。這就是為什麼malloc前面都會加上(int*)之類的。

```
// 宣告一個int指標變數，其初始值為NULL
int *ptr;
// 配置 5個 int大小的記憶體位址 給ptr，並將第一個位址存進ptr中
ptr = (int*)malloc( sizeof(int) * 5 );
```

(六) 指標與陣列

用到指標的時機，大多都是為了配置**動態陣列**才會用到，因此我們先來探討兩者之間的差異。

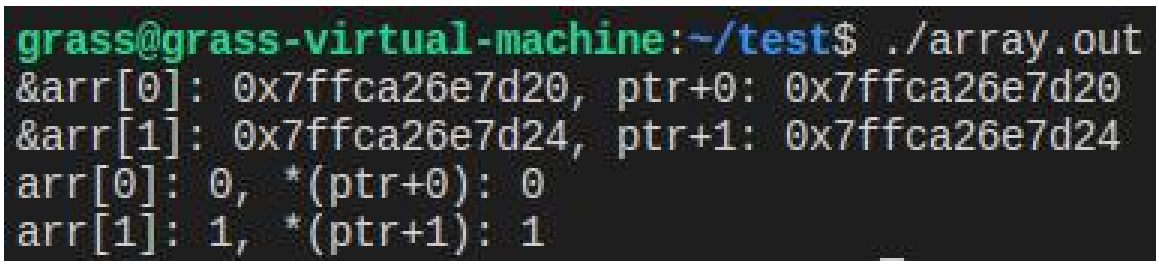
1. 一維陣列、指標：

這邊直接對陣列及指標做比較：

```
int size = 2, i;
int arr[size];
int *ptr;
ptr = &arr;     //將ptr指向array
// 1.記憶體位址
for(i=0; i<size; i++)
    printf("&arr[%d]: %p, ptr+%d: %p\n", i, &arr[i], i, ptr+i);
```

```
// 2.值的存取
arr[0] = 0;
*(ptr+1) = 1;
for(i=0; i<size; i++)
    printf("arr[%d]: %d, *(ptr+%d): %d\n", i, arr[i], i, *(ptr+i));
```

以上的輸出會如下：



上面提過指標存取值，必須透過一把鑰匙，即用**"*"**去取值。我們如果要取第一個值，很簡單的就是直接加上一個*即可，不過第二個以後的呢?
在前言有提到，指標變數本身存的就是記憶體位址，因此要取下一個的值，就必須要「**位移該變數型態的長度**」，就如上圖所示，0和1之間記憶體位址差了4 bytes(因為int佔4 bytes)，那我們要透過指標變數取值時，記憶體位址不是應該要ptr+4*n嗎?
但其實C會根據變數本身型態，去做記憶體位址的位移，**因此int型態的指標寫+n時，他會自動位移為4*n bytes，並且再加上取值運算子*去取值，其他的型態亦同。**

由上可以整理出陣列與指標的轉換：
記憶體位址： **&arr[i] == ptr+i**
變數值存取： **arr[i] == *(ptr+i)**

另外要注意++/--的位置，因為++/--的優先權(precedence)是高於*(dereference)的，因此:

a) *ptr++是表示*(ptr++)的意思，就是ptr指向下一個位置所存的值(同arr[i++])。
b)*++ptr是代表ptr先做位址往前移，再取值(同arr[++i])。
c) ++*ptr是表示++(*ptr)之意，即為ptr當下位置的值+1(同++arr[i])。

ps. 陣列變數，因為存的是值而不是位址，因此不存在++arr與arr++之類的表示法。

2. 二維陣列、指標的指標：

指標的指標(pointer to pointer)或雙層指標，顧名思義就是用兩個**去上鎖(宣告)。至於指標的指標，還是指標，因此在該記憶體中存的就仍是指標(記憶體位置)。說起來很繞口，其實

我們可以把上面藍色的指標先改名為"A"，也就是該記憶體裡存的是A的記憶體位址。但如果還是不懂呢? 那我們看一個例子：

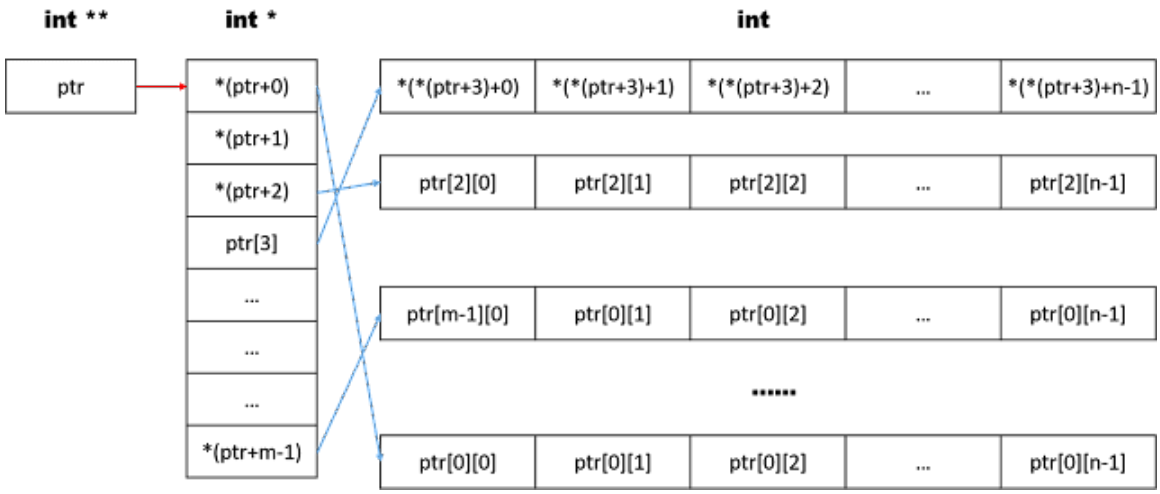
```
int a = 10;
int *ptr1 = &a;
int **ptr2 = &ptr1;
```

現在一般int變數，叫做a，該變數存了一個int值，5。
ptr1是指標變數，因此該變數存了a的記憶體位址，&a。
又如上面所說，什麼資料型態就存什麼資料下，ptr2是指標的指標變數，因此該變數存了ptr1(指標)的記憶體位址(的指標)，就是&ptr2。

```
printf("a = \t%d\n", a);
printf("*ptr1 = \t%d\n", *ptr1);     \\一把鑰匙打開
printf("**ptr2 = \t%d\n", *ptr2);    \\兩把鑰匙打開
```

因此上面的3行都會印出5。

當然上述的用法並不常見，多層以上的指標大多利用其特性用於動態陣列上。其原理是分配連續多個int*的記憶體給該變數，組成一個指標的指標連續記憶體，不小心又太繞口了，看以下的圖：




```
int i;
int **ptr;

ptr = (int **)malloc( m * sizeof(void *));
for (i = 0; i < m; i++)
    ptr = (int *)malloc( n * sizeof(int *));
```

不過這張圖的宣告可以看出一個缺點，就是記憶體過於破碎(memory fragment)，因此盡量避免該宣告方式。最好的方法是先直接宣告 一個m*n的一維陣列，再把他指定給ptr，如下：

```
int **ptr, *tmp;
int i;
ptr = (int**)malloc( m * sizeof(int *) );
tmp = (int*)malloc( m * n * sizeof(int) );
for(i = 0; i < m; tmp += n)
    ptr[i++] = tmp;
```

這樣在free時只需要free ptr(上圖int*部分)與ptr[0](上圖int部分)即可。

這邊要注意的是，雖然雙層指標可以理解成二維陣列，然而二維陣列，卻不能直接賦予給雙層指標。也就是以下是不行的：

```
int array[3][2] = {1,2,3,4,5,6};
int **ptr= array;
```

ptr的資料型態是pointer，存的就是記憶體位址，他如果被分配至一動態二維陣列foo時，存的位置就是該二維陣列foo[0][0]的初始位址；然而array**並不**代表指向array[0][0]的記憶體位址，也就是上面的例子是根本不具任何意義且錯誤的。

若真的需要宣告一個變數去接二維陣列的話(但這比較沒意義又麻煩)，可以用下面的方式：

```
int array[3][2] = {1,2,3,4,5,6};
int *ptr[3] = {array[0], array[1], array[2]};
```

至於**int *ptr[3](**指標陣列, array n of pointer to type)是甚麼，在下面繼續介紹。

3. 指標陣列? 指向陣列的指標? 傻傻分不清楚：

有種頭痛的指標變數，就是同時擁有Indirection(*)、Array subscripting ([]) 甚至是function call(())，如下：

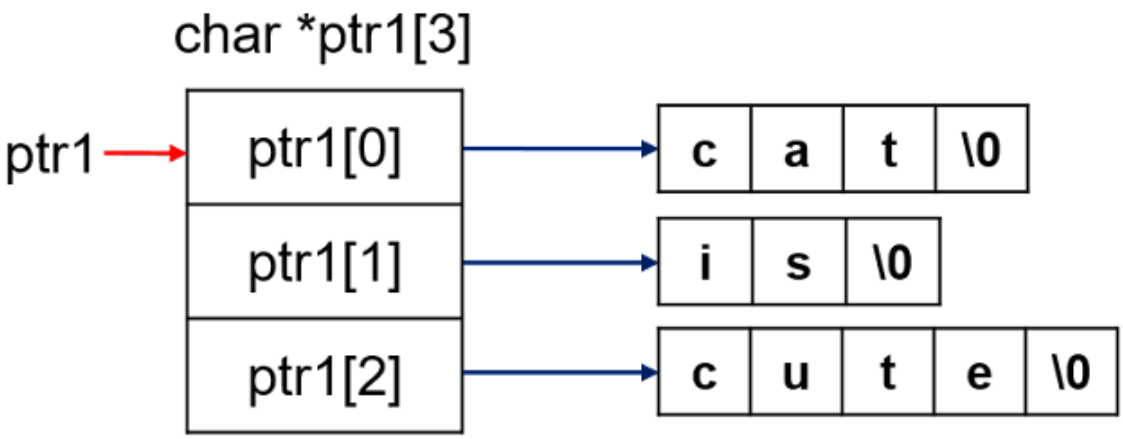
```
char *ptr1[3] = {"cat", "is", "cute!"};

char arr[4] = {"Cat"};  //注意這邊因為包含\0，所以長度是4，或是都不給值直接用arr[
char (*ptr2)[4] = &arr;
```

若沒有好好弄懂，真的很容易搞混他們之間的差異。
這邊又要提到operator之間的precedence了。在C的優先表中，**Array subscripting ([])是大於Indirection(*)的**，因此：

(a) char *ptr1[3];

又稱為指標陣列(array of pointer)，表示陣列的元素，皆指向某特定資料型態的指標。從優先表來看，可以解讀成長度為3的陣列(先)，每個皆存放指標(後)。結構如下：

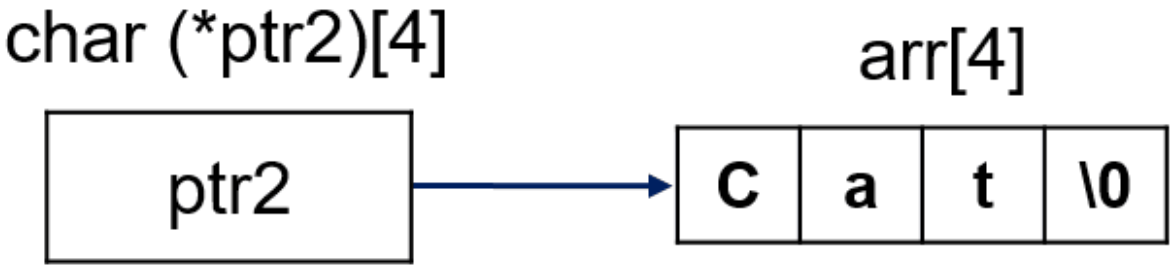


ptr1代表第一個元素ptr1[0]的記憶體位址。另外ptr1[0]則代表存"cat"的c的記憶體位址；ptr1[0]存"is"的i的位址；ptr1[2]就是存"cute"的c的位址。所以上面所分配給ptr1的記憶體大小為3*8bytes = 32 bytes (64bits下pointer大小為8bytes)(不包含cat is cute這12bytes)。

因此，本例中的指標陣列可以視為一個二維陣列 char arr1[3][];

(b) char (*ptr2)[4];

又稱為指向陣列的指標(pointer to array)，表示指向一個長度為4的char陣列的指標。也就是可以解讀成這是一個指標(先)，存的是長度為4的陣列(後)之位址。結構如下：



ptr2 毫無任何太大的用處，就只是存著arr這個長度為4的陣列位址。分配給ptr2的記憶體大小也就只有8bytes而已。

欸?不過似乎有些怪怪的? 好像有某東西長得很像所謂的指向陣列的指標，又說不上來是吧? 所以我們看一下以後的程式碼：

```
int *p;    // 指向int資料型態的指標
int (*ptr)[5];           // 指向長度為5的陣列的指標
int arr[5];  // 長度為5的陣列

p = arr;  // 將p指向arr的第1個元素(arr[0])
ptr = &arr;  // 將ptr指向"整個"arr陣列

printf("p = %p, ptr = %p\n", p, ptr);

p++;
ptr++;

printf("After Plus one element...\n");
printf("p = %p, ptr = %p\n", p, ptr);
```

以下是執行結果：

```
grass@grass-virtual-machine:~/test$ ./a.out
p = 0x7ffd5a3c1d80, ptr = 0x7ffd5a3c1d80
After Plus one element...
p = 0x7ffd5a3c1d84, ptr = 0x7ffd5a3c1d94
```

看懂了嗎?ptr是指向整個陣列，如果做++就等**同於移動整個arr陣列大小**的記憶體位址(0x14 = 20bytes = 5*4bytes)；而p僅僅是指向arr中的第一個元素，做++則是**往下一個元素移動**而已。

其實如同上面一再強調的什麼資料型態就存什麼資料，ptr是存長度為5的int陣列指標，當然每次移動都是增加等量，即5個單位的陣列大小。而p是存int指標，所以每次++就是加1個單位拉~。

因此，本例中的指向陣列的指標可以視為一個二維陣列 char arr2[][4];

(c) 公式(?) 參考參考

根據C白皮書**K&R**中所言，一個宣告包含basic types(即int char之類)與derived types，而後者被分為以下三種：

* 稱之為 **pointer to ...**
[] 稱之為 **array n of ...**
() 稱之為 **function returning ...**

根據precedence tables，[]與()總是先於*。
以下是步驟：

1. 解碼(公式)方法，總是以**變數名稱起頭**：

foo is ...

2. 且永遠是以**basic type為結尾**：

foo is ... **int**

3. 中間的...則以**"go right when you can, go left when you must"**為原則進行填充
以下直接以一個實際範例來解說：

double **foo[5]

1. 前置動作，頭尾先解碼：
double ****foo**[5]：**foo is ... double**

2. 根據**"go right when you can"**與precedence tables，先處理[5]：
~~double~~ ****~~foo~~[5]**：foo is **array 5 of** ... double

3. 最右邊處理完了，該處理左邊中的兩個*。又因go right，先處理右邊的*：
~~double~~ ~~**~~foo[5]~~：foo is array 5 of **pointer to** ... double

4. 最後一個*處理：
~~double~~ ~~*~~*foo[5]~~：foo is array 5 of pointer to **pointer to** double

更難的部分就不贅述了，這邊提供**線上轉換的網站**，直接輸入程式碼就會將答案顯示出來。

(七) 寫在最後

本篇文章僅做較基本的探討，太過繁複的部分可以請各位看官再去詳細深入，希望本篇可以給讀者更進一步的對於指標的認識，並且不會害怕看到他(但如果看到char *(*(*a[])())()之類的，也別沮喪，這部分就更進階了)，同時我在反覆編寫時也可以做為複習並釐清各種觀念。算是教學相長吧! 若本文有任何錯誤與疑問還敬請指教。函數指標部份等有空時再看要不要整理XD。

至於超進階部分，這邊我推薦我心中的超神 Jserv老師 的影片與教材:

你所不知道的C語言：指標篇

看完保證功力大增唷~

西撒 *2019年2月3日 上午8:35*

printf("a = \t%d\n", a);
printf("*ptr1 = \t%d\n", *ptr1); \\一把鑰匙打開
printf("**ptr2 = \t%d\n", *ptr2); \\兩把鑰匙打開

因此上面的3行都會印出5。

我覺得最後一行不會印出5
而是印出ptr2的記憶體位置
我測試的結果也沒印出5
https://onlinegdb.com/SJNS_5NEN
而且範例a=10 為什麼說明又換成5?

Unknown *2019年3月6日 下午6:43*

我覺得作者打錯了
他應該是要打：
printf("a = \t%d\n", a);
printf("*ptr1 = \t%d\n", *ptr1); \\一把鑰匙打開
printf("**ptr2 = \t%d\n", *×ptr2); \\兩把鑰匙打開

另外，a=10是正確的，作者在這裡也是錯把10打錯5了

Unknown *2019年12月1日 清晨6:14*

我覺得最後一行會印出
ptr1的位置
因為ptr2已經用*指向ptr1的位置

寵物鼠 *2019年12月1日 清晨6:16*

我覺得最後一行會印出
ptr1的位置
因為ptr2已經用*指向ptr1的位置

回覆

西撒 *2019年2月3日 上午8:38*

作者已經移除這則留言。

回覆

西撒 *2019年2月3日 上午8:43*

```
int i;
int **ptr;

ptr = (int **)malloc( m * sizeof(void *));
for (i = 0; i < m; i++)
ptr = (int *)malloc( n * sizeof(int *));
```

.....
範例 最後一行
是不是少了i相關的運算?
看不懂怎麼執行

還有i疑問ptr = (int **)malloc(m * sizeof(void *));
為什麼是用?sizeof(void *)
我還以為是用ptr = (int **)malloc(m * sizeof(int *));

睏寶 *2019年3月3日 清晨7:13*

我不是作者，也不肯定回覆是否正確
但是sizeof(void*)與sizeof(int*)正常來說兩者的value是一樣的

在我的理解中
C/C++中的指標不論指向何種type
所佔用的空間都是固定的大小
不會因為今天是指向char, int抑或是任意型態(void)就改變

回覆

豆花 *2019年5月8日 凌晨4:32*

謝謝 受益良多!

回覆

加油好嗎 *2019年8月9日 晚上10:45*

小錯誤很多，麻煩校正後再發佈，不然只是誤人子第

回覆

小豆 *2022年1月13日 下午6:32*

int *ptr;
//ptr =&arr //error
ptr = arr; //ptr指向整個陣列
or
ptr =&arr[0] //陣列元素0的首地址=陣列起始地址

回覆

kalamajadee *2022年3月4日 上午10:10*

Ritzy Vegas: The Ultimate Casino For The Super - Dr. MCD
The ultimate casino 순천 출장마사지 for the Super 화성 출장안마 Slots; •••• ••••• •••• ••• 울산광역 출장안마 ••• ••• ••• ••• ••• •••. Rating: 3.8 보령 출장마사지 · 7 votes 수원 출장마사지

回覆

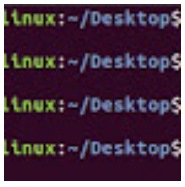
如要留言，請點按下方的按鈕使用 Blogger 帳戶登入。

使用 BLOGGER 帳戶登入

這個網誌中的熱門文章

(c/c++) Function Pointer函式指標兩三事 (Function Pointer 的 typedef 與 Array of Function Pointer)

7月 07, 2017



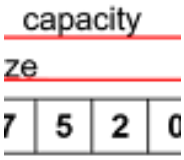
Function Pointer 定義: Function Pointer (中文直譯「函式指標」)，即為儲存某一個函式起始memory address的 變數 ，此變數可以提供我們在之後進行呼叫。 乍聽之下，function pointer就只是多一個別名 ...

閱讀完整內容

(C++) 別再用dynamic array與pointer了! 趕快學STL的vector與iterator！

4月 04, 2018

(一) 廢話 別再用array與pointer?那我不是在自打上一篇的嘴巴嗎? 其實並不衝突。因為在C之下，pointer還是非常重要的，本篇所著重的是C++。 不知道大家在學C++時，老師或書中有沒有教或學到STL(Standard Template Library)呢? STL是C++下非常非常好用 ...



閱讀完整內容