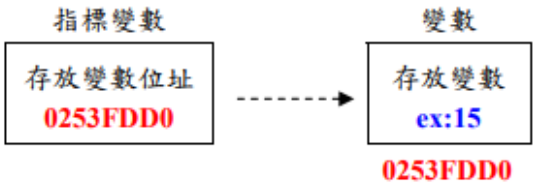
Chapter11 指標

一、指標變數

一般變數是由三個物件構成的：變數名稱、變數的資料、變數的記憶體位址。而指標 (Pointer) 是一種特殊的變數，用來存放變數在記憶體中的位址。C 語言中，無論指標指向何種資料型態，指標變數本身均佔有 4 個 bytes。



宣告格式：

型態 *指標變數;

範例：

```
int *ptri; /* 整數型態之指標變數 */
char *ptrch; /* 字元型態之指標變數 */
sizeof(ptri) -> 4 bytes
sizeof(ptrch) -> 4 bytes
```

注：我們通常會將指標變數簡稱為指標

二、指標運算子

1. 位址運算子&：用來取變數或陣列雲速在記憶體中的位址。
2. 依址取值運算子*：用來取得指標所指向的記憶體位址的內容。

```
int a = 10, b;
int *p;
p = &a;
b = *p;
*p = 20;
```

結果：a = 20, b = 10

三、指標的運算

1. 設定運算：將等號右邊的值設定給左邊的指標變數。
2. 加/減法運算：用各個資料型態的長度來處理位址的加減法運算。
3. 差值運算：計算兩個指標之間的距離，單位為資料型態的長度。

範例：

```
int a = 10, b;
int *p;
p = &a;
b = *p;
*p = 20;
int a = 10, b = 20;
int *p1, *p2;
char ch = 'a', *pch;
/* 設定運算 */
p1 = &a; /* 將 a 的位址存放於 p1 */
p2 = &b; /* 將 b 的位址存放於 p2 */
pch = &ch; /* 將 ch 的位址存放於 pch */
/* 加減法運算 */
p1++; /* 將 p1 中的位址值加上 4 bytes (int 型態的大小) */
pch--; /* 將 pch 中的位址值減去 1 byte (char 型態的大小) */
/* 差值運算 */
sub = p1 - p2; /* 計算 p1 和 p2 相差的距離 (以 int 為單位的距離)*/
```

四、指標的簡便運算式

```
int X;
int A[5] = {10,20,30,40,50};
int *p = A + 2;
```

運算式	同義	敘述及順序	執行後	
			X	*p
X = *(p++);	X = *p++;	X = *p; p = p + 1;	30	40
X = *(++p);	X = *++p;	p = p + 1; X = *p;	40	40
X = (*p)++;		X = *p; *p = *p + 1;	30	31

X = ++(*p);		*p = *p + 1; X = *p;	31	31
X = *(p--);	X = *p--;	X = *p; p = p - 1;	30	20
X = *(--p);	X = *--p;	p = p - 1; X = *p;	20	20
X = (*p)--;		X = *p; *p = *p - 1;	30	29
X = --(*p);		*p = *p - 1; X = *p;	29	29

五、指標與函數

1. 說明：函數的 return 敘述只能有一個回傳值，當程式需要傳遞兩個以上的值時，可以利用指標解決在函數間傳遞多個回傳值的問題。其做法是將指標當作引數傳入函數中，由於指標內的值是所指向變數的位址，因此不須經過 return 敘述即可更改變數的值。
2. 範例：

```
void swap(int *, int *); /* 函數原型，參數為兩個整數型態的指標*/
int main(void)
{
 int a=3, b=5;
 swap(&a, &b); /* 傳遞 a 和 b 的位址 */
 return 0;
}
void swap(int *x, int *y) /* 此函數用來交換 x、y 所指向的變數之值
*/
{
 int temp = *x;
 *x = *y;
 *y = temp;
}
```

六、指標與陣列

1. 說明：陣列可以看成是指標的一種分身，陣列元素的排列可以透過指標的運算去存取
2. 範例：

int a[3] = {5,7,9};

注意：陣列 a 以指標的方式表示時，a 會被視為指標常數，所以不可寫成 a++。

七、指標陣列

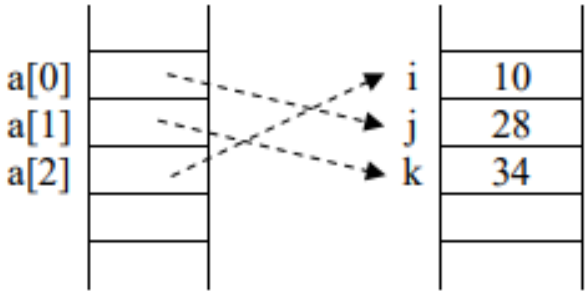
若陣列中所存放的變數為指標變數，我們稱之為指標陣列。

宣告格式：

型態 *陣列名稱[個數];

範例：

```
int i=10,j=28,k=34;
int* a[3];
a[0] = &j;
a[1] = &k;
a[2] = &i;
```



【比較】字串陣列 V.S. 指標陣列

(1) 字串陣列：char name[3][10] = {"David", "Jane Wang", "Tom Lee"};

name[0]	D	a	v	i	d	\0				
name[1]	J	a	n	e		W	a	n	g	\0
name[2]	T	o	m		L	e	e	\0		

(2) 指標陣列：char *name[3] = {“David”, “Jane Wang”, “Tom Lee”};

name[0]	D	a	v	i	d	\0				
name[1]	J	a	n	e		W	a	n	g	\0
name[2]	T	o	m		L	e	e	\0		

注：指標陣列可節省浪費的記憶體空間

八、雙重指標：指向指標的指標

指標變數中若是存放另一個指標變數的位址，這種指向指標的指標稱為雙重指標。

宣告格式：

型態 **指標變數;

九、動態記憶體配置

1. 介紹：若要放入資料結構的資料數是無法預期的，我們就無法先準備好記憶體來儲存這些資料，而是要等有一筆資料加入後，再向系統要求記憶體空間，這個機制就是動態記憶體配置。而我們常用的函式有 malloc、free、calloc。
2. 函式原型：

   void *malloc(size_t size);

   void *calloc(size_t n, size_t size);

   void free(void *ptr);

   void *realloc(void *mem_address, unsigned int newsize);
3. 範例：

   (1) malloc：

```
int *p = malloc(sizeof(int) * 1000);
```

   (2) calloc：

```
int *p = calloc(1000, sizeof(int));
```

   (3) free：

```
free(p);
```

   (4) realloc：重新配置記憶體空間

```
int *arr = realloc(p, sizeof(int) * size * 2);
```

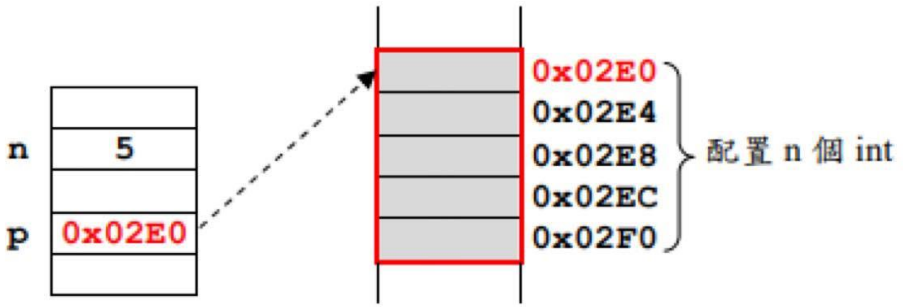
   注：realloc 重新配置後的位址不保證相同，realloc 會複製資料來改變記憶體的大小。

   若原位址有足夠的空間，使用原位址調整記憶體的大小。

   若空間不足，會重新尋找足夠的空間來進行配置，在這個情況下，realloc 前舊位址的空間會被釋放掉。

   若 realloc 失敗會傳回空指標（null），因此最好對位址進行檢查。
4. 一維配置：

```
int n = 5;
int *p;
p = (int*) malloc( n * sizeof(int) );
```



5. 二維配置：

```
int row = 3,col = 5,i;
int** array;
array = (int**) malloc(row * sizeof(int*));
for (i = 0; i < row; i++)
    array[i] = (int*) malloc(col * sizeof(int));
```