指標
Pointers

指標 (Pointer)

- 指標 (pointer) 是 C 語言裡面非常重要的用法、同時也最讓初學者感到困擾的概念。

- 簡單地說，指標就是一個專門用來儲存位址的變數。

使用 **&** 符號取得位址

- 使用 **scanf( )**的時候，必須用位址來當作參數 (變數前面加 **&**)

- 在變數前面加上 **&**，會得到該變數的位址。譬如變數的名稱叫做 **y**，則 **&y** 就是這個變數的位址。我們可以把位址想成記憶體中的某個位置。

範例 E09_09.c

```c
#include <stdio.h>
int main(void)
{
    int y = 5;
    printf("%d %p\n", y, &y);
    return 0;
}
```

輸出：

```
5 0x22FF74
```

- 用 **%p** 以十六進位格式輸出 **y** 的位址 **&y**

使用 & 符號取得位址

- 之前提過每個 function 會有自己的 local variables，不會和外部衝突
- 參數傳入 function 的時候，是把值複製過去，所以在 function 裡任意修改參數並不會影響到外部的變數
- 下面的範例就是在說明這個特性
  - 範例 E09_10.c

指標 (Pointer)

- 寫一個 function，傳入兩個參數，然後在 function 裡面交換這兩個參數的值
  - 範例 E09_11.c

指標變數

- 指標變數專門用來儲存位址，例如：

```
ptr = &y;
```

- 可以改變 **ptr** 的值，拿它來記錄別的位址

```
ptr = &z;
```

- Dereferencing

```
x = *ptr;
```

- 這三行程式碼得到的效果相當於

```
x = z;
```

宣告指標變數

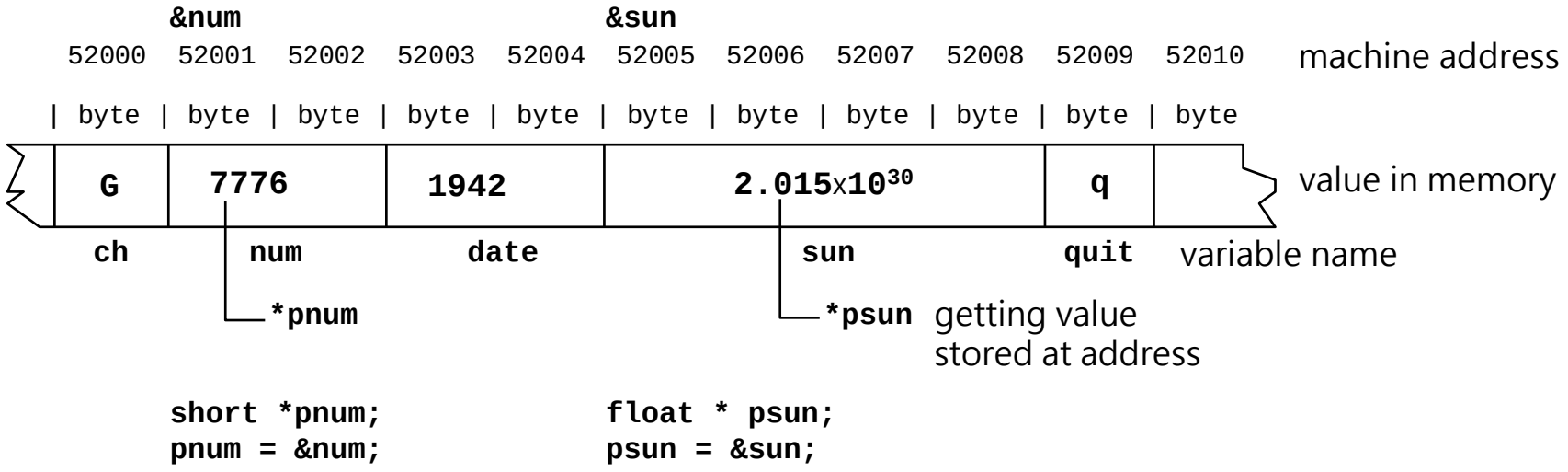
```
int * pi;
/* pi is a pointer to an integer variable */

char * pc;
/* pc is a pointer to a character variable */

float * pf, * pg;
/* pf and pg are pointers to float variables */
```

宣告指標變數

- 用底下的圖來模擬一下指標的宣告和使用過程中記憶體的狀態變化：



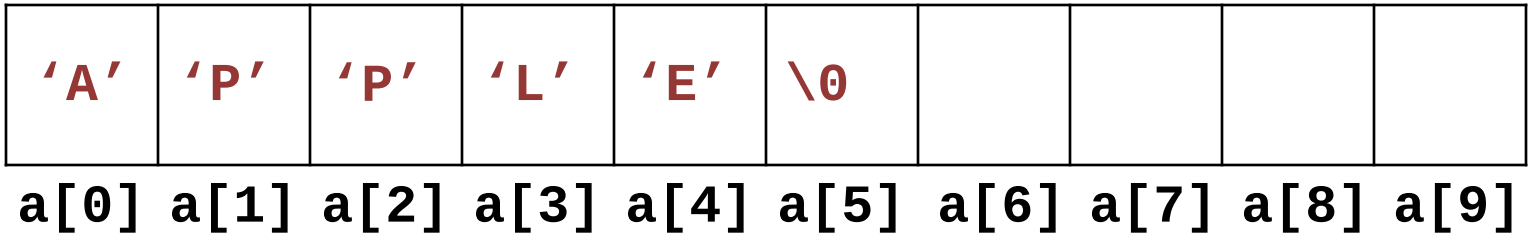
指標變數

- 指標變數可以用來記住記憶體的位址
- C 語言裡，程式設計者可以用 **&** 符號來取得某個變數的位址
- 試試看透過記憶體位址，達到真正交換兩個變數值的效果
- 範例 E09_12.c

陣列

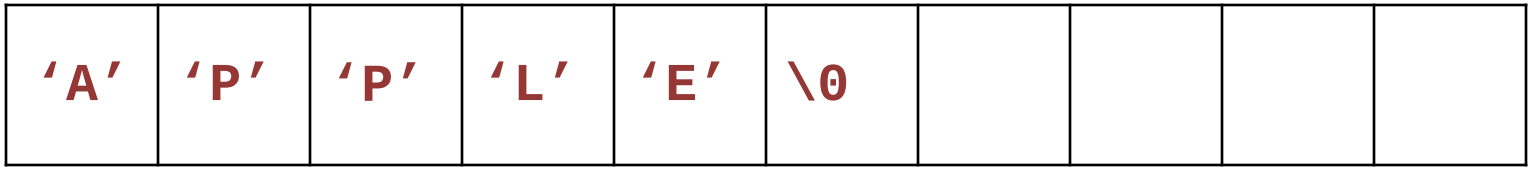
```
char a[10];
```

a



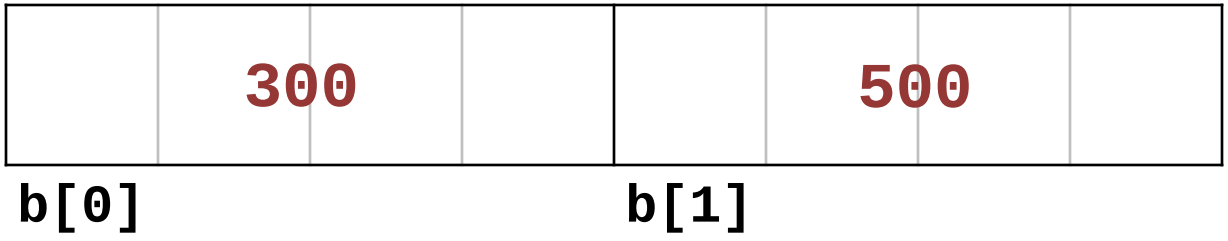
```
char a[10];
```

a




```
int b[2];
```

b



二維陣列

```
int a[3][4];
```

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

二維陣列

```
int a[3][4] = { {1,2,3,4}, {5,6,7,8}, {9,10,11,12} };
```

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

二維陣列

```
int a[3][4] = { {1,2,3,4}, {5,6,7,8}, {9,10,11,12} };
```

1	2	3	4
5	6	7	8
9	10	11	12

二維陣列

```
int a[3][4] = { 1,2,3,4, 5,6,7,8, 9,10,11,12 };
```

1	2	3	4
5	6	7	8
9	10	11	12

指標與陣列

- 陣列的名稱，同時也可以代表陣列的第一個元素的位址：

```
int a[10];
printf("%p %p %d\n", a, &a[0], a==&a[0]);
```

- **%p** 格式專門用來顯示指標變數所儲存的的值(代表某個位址)
- 範例 E10_07.c
- 範例 E10_08.c

函數、指標與陣列

- 假設把陣列當作參數傳給某個 function，然後在 function 中計算陣列元素的總和
- 假設主程式裡的陣列叫做

```
int a[10] = {1, 1, 2, 3, 5, 8, 13, 21, 34, 55};
```

- 希望在主程式裡使用下面的方式呼叫 function 取得陣列總和，再把結果儲存在變數 total 裡面

```
total = sum(a);
```

函數、指標與陣列

- 由於我們知道陣列的名稱可以用來代表整個陣列的起始位址，所以 function 的 prototype 的合理宣告方式大概是

```
int sum(int *ap);
```

- 範例 E10_09.c

函數、指標與陣列

- 上面例子裡，底下四種宣告方式是一樣的

```
int sum(int * ap, int n);
int sum(int *, int);
int sum(int a[], int n);
int sum(int [], int);
```

函數、指標與陣列

- 另一種作法，用到的概念是傳入兩個**指標變數**當參數，其中一個儲存的是陣列的開頭位址，另一個用來儲存陣列的結尾位址
- 範例 E10_10.c

函數、指標與陣列

- 指標的各種運算和使用方式回顧
- 範例 E10_11.c

函數、指標與陣列

- 未做初始化的指標，絕對不能對牠做 dereferencing 的動作
- 例如：

```
int *pt;
*pt = 5;
```

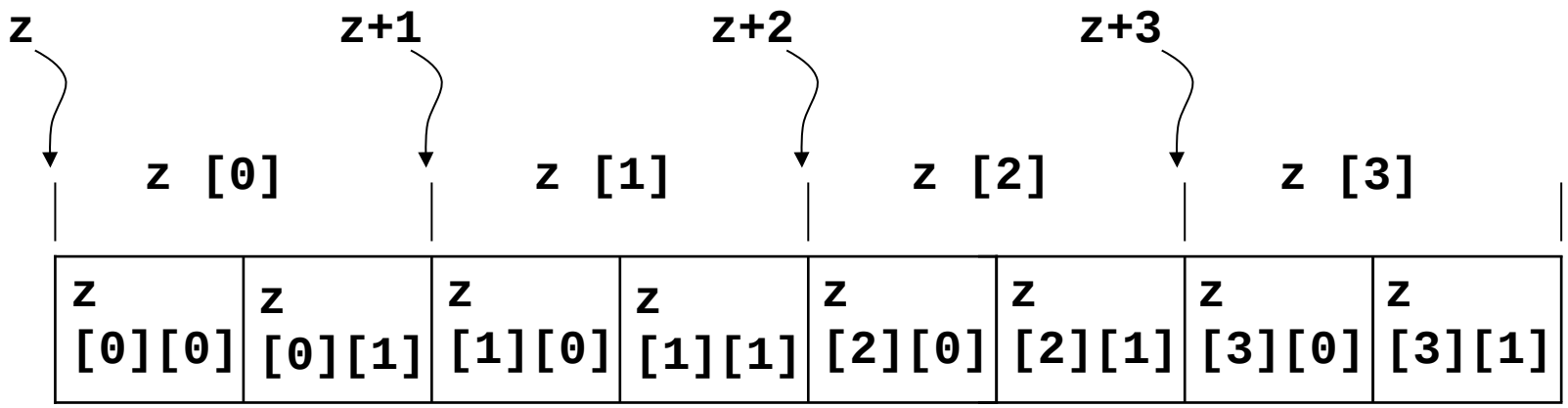
指標和二維陣列

- 假設我們宣告了一個二維陣列：

```
int z[4][2] = {{1,2}, {3,4}, {5,6}, {7,8}};
```

- **z** 的值和 **&z[0]**是一樣的
- 但是 **z** 和 **z[0]**是不同型別的指標，所以對他們做位址加一的動作，在意義上或是造成的效果，都不相同
- 如果計算 **z+1**，在這個例子裡得到的位址變化是增加八個 bytes，也就是兩個整數的大小
- 如果計算 **z[0]+1**，則位址變化則只會增加 4 bytes

指標和二維陣列



指標和二維陣列

- 宣告一個可以指向二維陣列的指標
- 用上一個範例為例

```
int (*pz)[2];
```

- 意思是說，**pz** 是一個指到整數陣列的指標，而牠所指到的陣列包含了兩個整數元素。要注意上面的括號一定要加，因為如果不加的話，代表的是另一種意義：

```
int *py[3];
```

- 上面表示 **py** 是一個「有三個元素的陣列」，其中每個元素都是「指向整數的指標」

如何傳二維陣列給 function

- 範例 E10_13.c
- 必須要使用上面的範例裡的 function 宣告方式
- 由於一個二維陣列可以解讀成「由一維陣列組成的一維陣列」，但是傳遞參數時，我們能夠做的事情只是把位址 (某個大小為 4 bytes 的值) 傳給 functions，所以傳遞二維陣列其實應該是要傳遞位址給某個指標變數

指標陣列

- 範例 E10_14.c

動態取得記憶體

- stdlib.h 裡面提供了兩個常用來管理記憶體的 functions，叫做 **`malloc()`**和 **`free()`**，我們可以靠它們來取得和釋放記憶體
- 範例 E10_15.c

動態取得記憶體

- 傳給 **malloc()**的參數是我們想要取得的記憶體大小，以 byte 為單位
- **malloc()**傳回來的是指向 **void** 型別的指標，所以我們必須依照想要的型別，對指標做強制轉換

```
ptd = (double*)malloc(array_size* sizeof(double));
```

動態取得記憶體

- 當**malloc()** 呼叫失敗，沒辦法取得記憶體，則會傳回 **NULL**
- 最後當取得的記憶體不需要再使用，就用 **free(ptd);** 把它釋放
- 程式如果不斷取得記憶體，但沒有正確地用 **free()**釋放不再用到的空間，一但原本用來記錄位址的指標變數的值被改變，例如指到了別的地方，則原本那塊記憶體就沒有人能找得到，也就沒人能再使用它，這會造成所謂的 memory leak 的問題