



C/C++基礎程式設計

指標 (Pointer)

張傑帆

CSIE, NTU

瘋到自以為能改變世界的人，就能改變世界。

The people who are crazy enough to think they can change the world are the ones who do. -Steve Jobs

課程大綱

- 指標簡介
- 陣列與指標
- 動態記憶體配置
- 指標宣告進階



指標

- 用途：紀錄某資料的**記憶體位置**，透過位置**間接**使用該資料。
- 使用時機：當你需要用記憶體中的某一個資料，但這個資料沒有名稱可直接使用，可以使用**指標**這種特別的單位，**透過存放某資料的記憶體位置，來間接操作這些資料**
- 使用指標的注意事項：
 - 使用前要**先宣告** (要幫它取個名字)
 - 只能用來儲存**記憶體位置**
 - 三個重要的應用：
 - **動態記憶體配置**
 - **函式的設計**
 - **資料結構**



指標的概念

- 儲存另一個資料的位置，然後間接操作它

int *p;

int *

0x1000

名稱：p

p = &a;

想辦法取得0x1000位置!

***p**

間接找到資料並使用

int a = 70;

int

70

資料內容

名稱：a

位置：0x1000

沒有名稱可用：
透過指標!

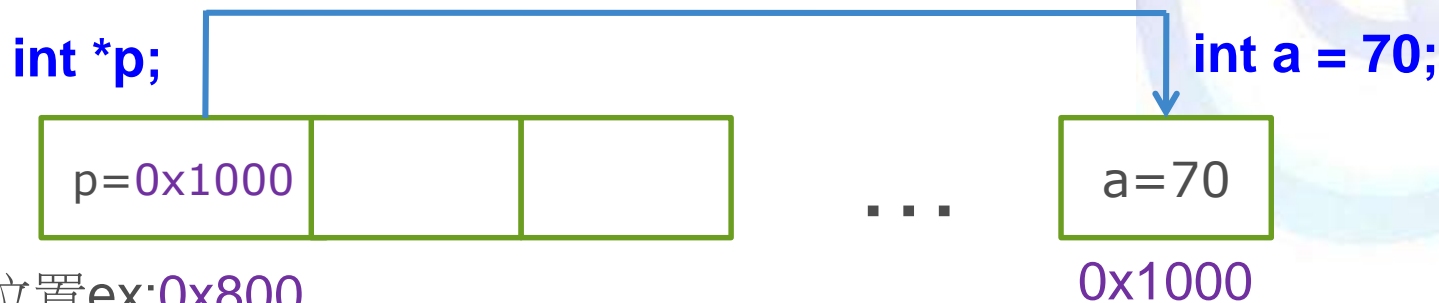


我要使用70那個整數...

有名稱可用：直接用**a**

指標簡介

- 指標 (pointer) 與 地址 (address)
 - 我們可宣告一變數為一指標，該變數的值為一記憶体的地址
 - 宣告語法: 資料型態 *指標名稱;
 - 例如: 宣告一個整數指標名稱為p: `int *p;`
 - p 為一指標，如果 p 的值为 `0x1000`，那麼我們可以說 p 指向記憶体其地址為 `0x1000`。

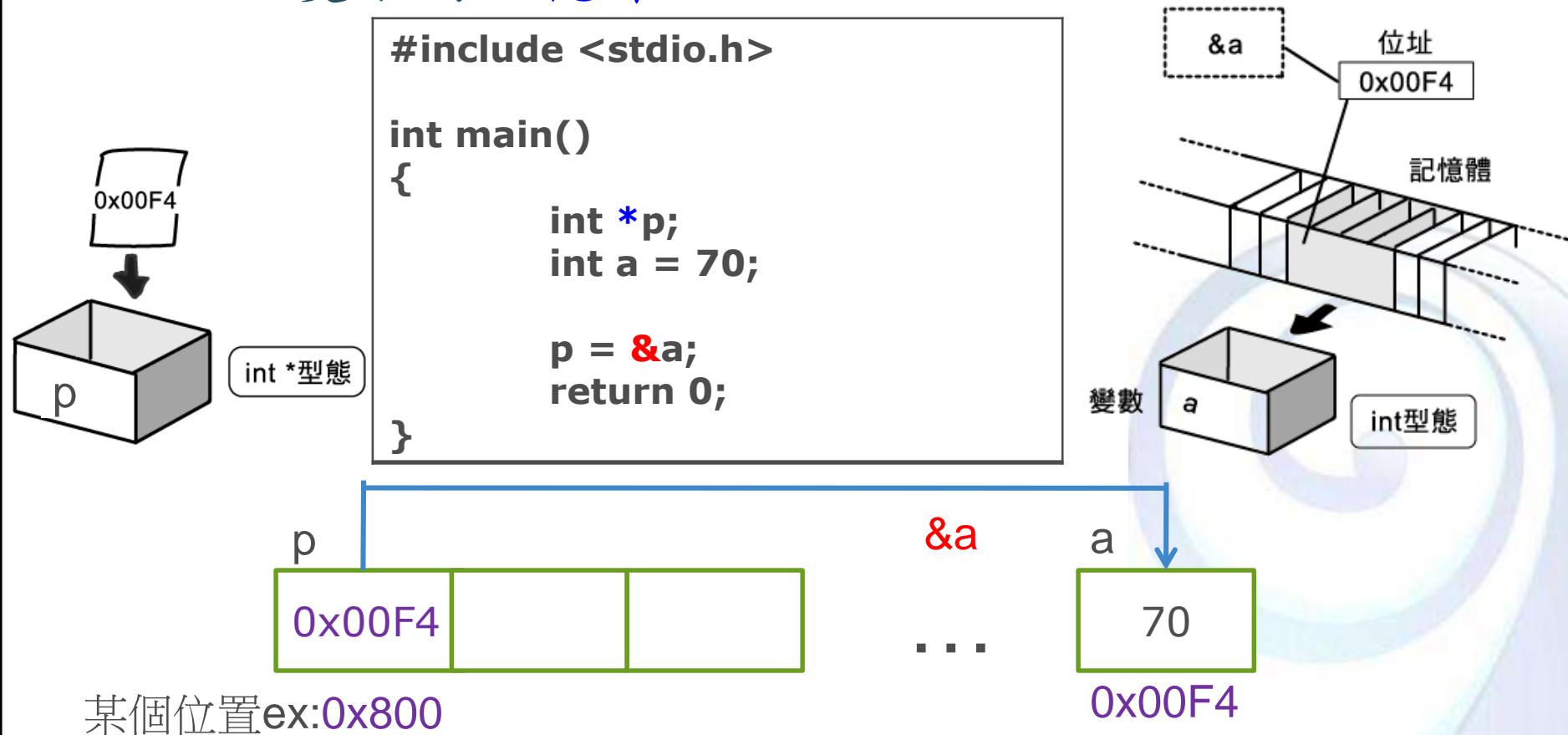


指標簡介

- 取址運算子 **&**

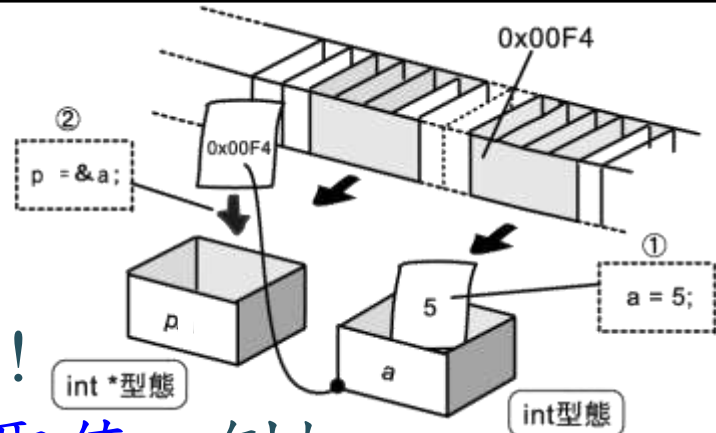
- “**&**”是可當做取址運算子，用來表示(或取得)一變數的記憶體地址

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



指標簡介

- **間接取值運算子 ***
 - 與宣告指標變數之 * 意義不同!
 - 我們可利用 指標變數來**間接取值**，例如:
- 



#include <stdio.h>

```
int main()
```

{

```
int *p;
```

/* p 為一指標變數 */

```
int a = 70, b;
```

p = &a;

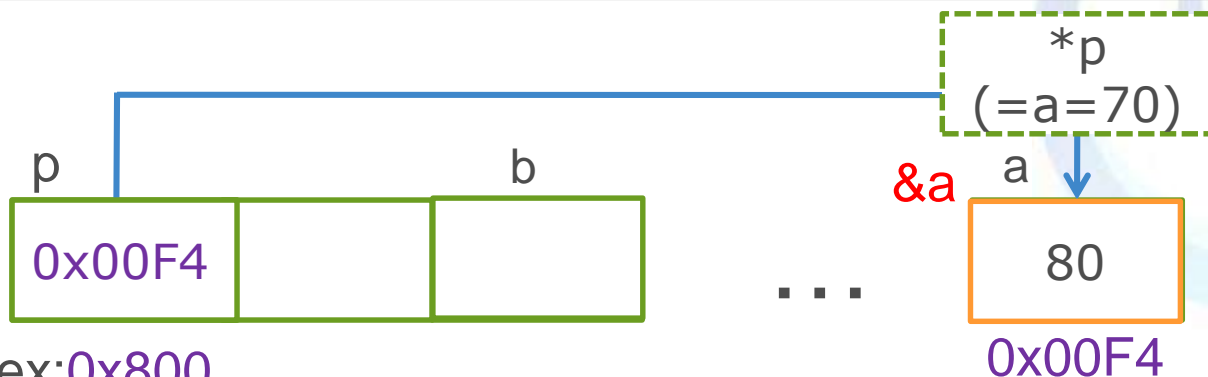
```
b = *p + 1;
```

/* 與 $b = a + 1$; 相同 */

***p = 80;**

```
return 0;
```

}

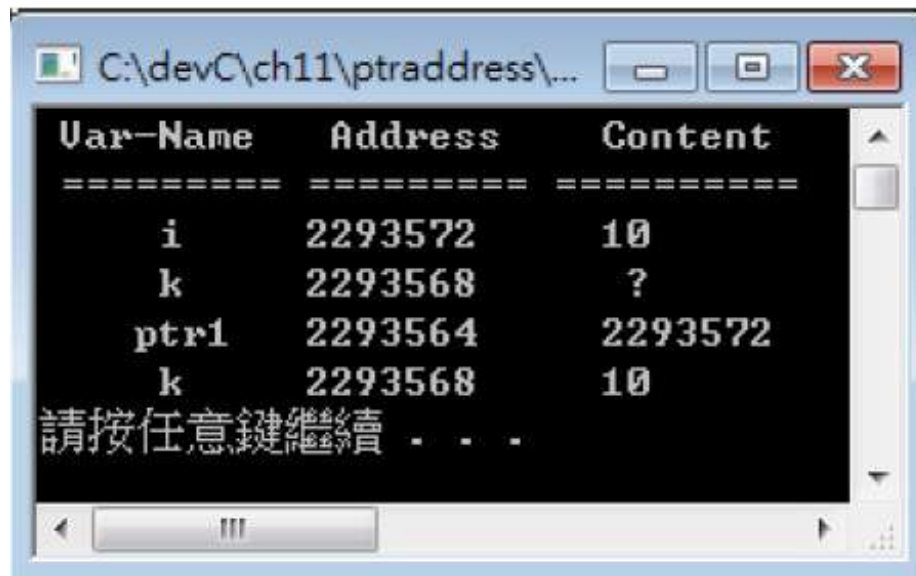


某個位置ex:0x800

0x00F4

小練習

- 命名三變數 `int i = 10; int k; int *ptr1;`
- 令 `ptr1` 取得 `i` 的記憶體位址
- 並令 `k` 透過 `ptr1` 取得 `i` 的值
- 且印出所有變數的記憶體位址與值



A screenshot of a Windows command prompt window titled "C:\devC\ch11\ptraddress\...". The window displays a table of memory addresses and values for variables `i`, `k`, and `ptr1`. The table has three columns: "Var-Name", "Address", and "Content". The content shows that `i` is at address 2293572 with value 10, `k` is at address 2293568 with value ?, and `ptr1` is at address 2293564 with value 2293572. Below the table, it says "請按任意鍵繼續 . . .".

Var-Name	Address	Content
i	2293572	10
k	2293568	?
ptr1	2293564	2293572
k	2293568	10

請按任意鍵繼續 . . .

錯誤操作範例

錯誤存取不知名記憶體位置

```
int i=10, k=0, *p;
```

```
*p = 20;
```

錯誤存取不特定記憶體位置

```
int i=10, k=0, *p;
```

```
p = 6470000;
```

```
*p = 5;
```

```
k = *p;
```

example



課程大綱

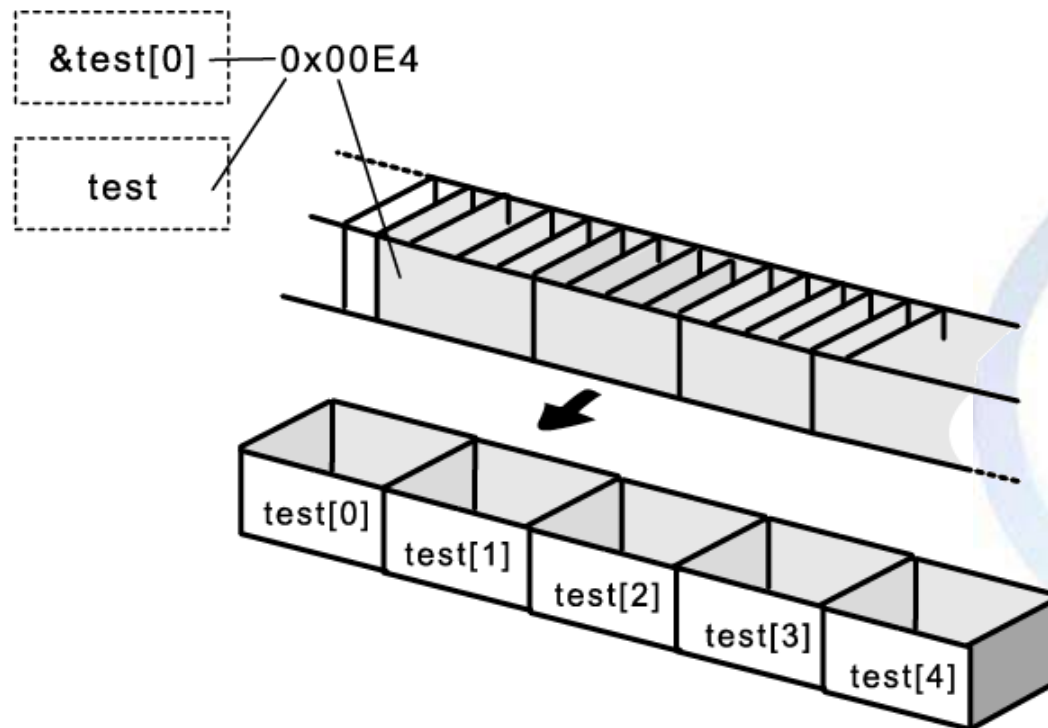
- 指標簡介
- **陣列與指標**
- 動態記憶體配置
- 指標宣告進階

- 作業系統記憶體限制
- [https://msdn.microsoft.com/en-us/library/aa366778\(VS.85\).aspx](https://msdn.microsoft.com/en-us/library/aa366778(VS.85).aspx)



關於陣列名稱的機制

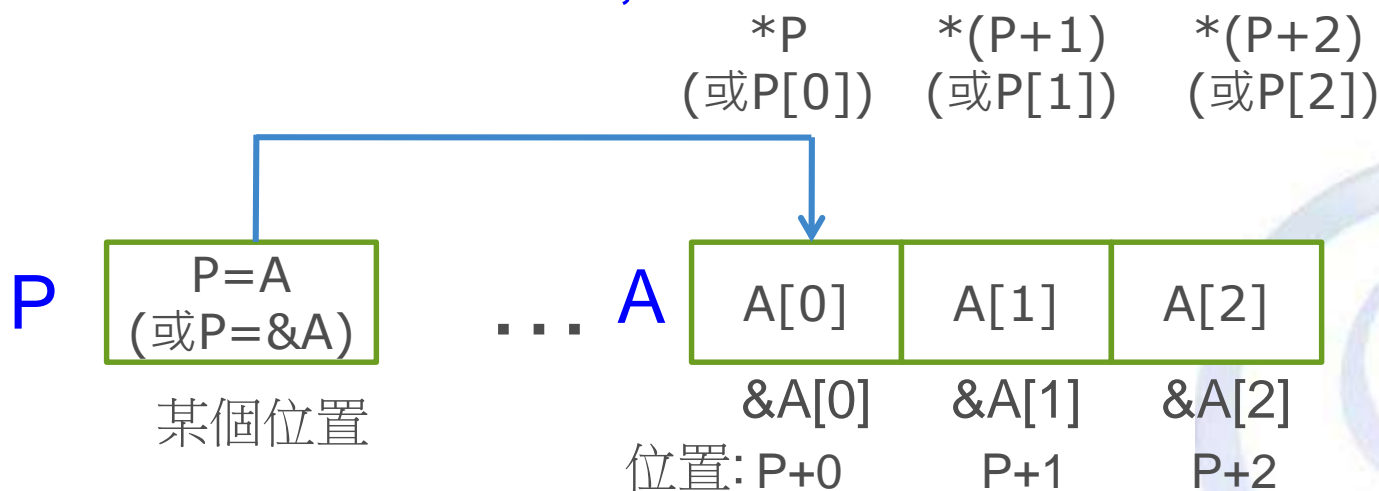
- 陣列名稱可以表示陣列最前面元素的位址。
 - `printf("test[0]的位址為%p。\\n", &test[0]);`
 - `printf("test的值為%p。\\n", test);`



陣列與指標

- 假如一個指標P指向陣列A，則P在語法上也可當陣列使用。

如下圖所示：
`int A[3];`
`int *P;`
`P = A;`



<https://goo.gl/ndjhx3>

<https://goo.gl/5RgrST> 中括號就是*號

陣列與指標

- 範例

```
#include <stdio.h>
```

```
int main()
```

```
{
```

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

```
    int *P, i;
```

```
    P=A; //或P=&A: 陣列名稱為記憶體位置
```

```
    for(i=0; i<10; i++)
```

```
    {
```

```
        printf("%d ", P[i]);
```

```
    }
```

```
    printf("\n");
```

```
    return 0;
```

```
}
```

陣列與指標

- 上頁範例也可以這樣寫：

```
#include <stdio.h>
```

```
int main()
```

```
{
```

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

```
    int *P, i;
```

```
    P=A; //或P=&A; 陣列名稱為記憶體位置
```

```
    for(i=0; i<10; i++)
```

```
    {
```

```
        printf("%d ", *P); //或 printf("%d ", *(P+i));
```

```
        P++;
```

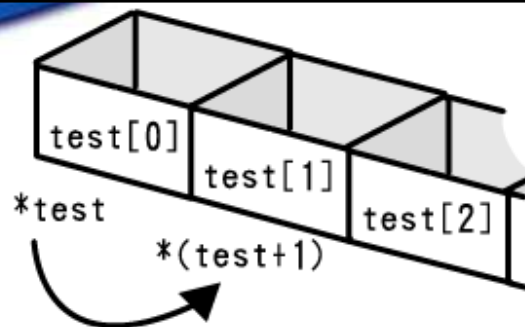
```
    }
```

```
    printf("\n");
```

```
    return 0;
```

```
}
```

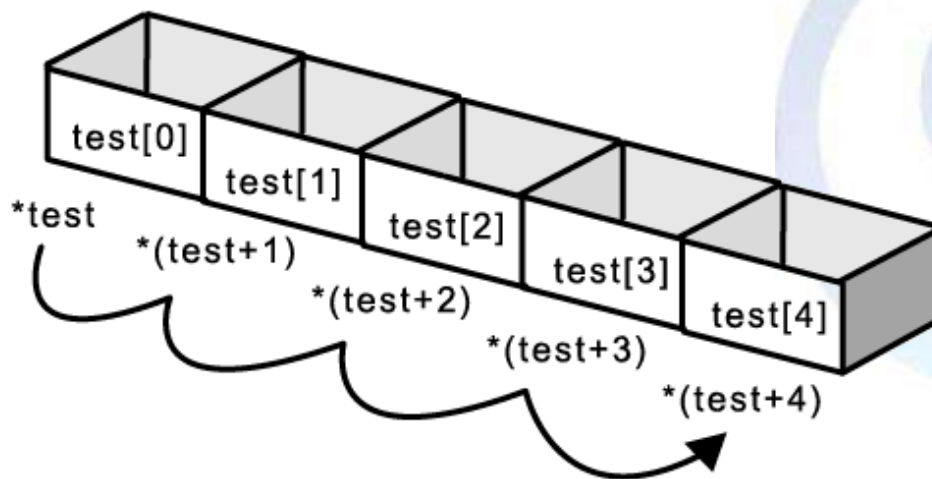

指標的運算



- 「**指標運算**」是指對目前指標(變數)內所存放的**記憶體位址**做**運算**，由於指標中所存放的是某個資料的**記憶體位址**，因此**更改指標的內容**相當於**變更資料的位址**，更改後指標會**指到新的記憶體位址**，這是指標如何在記憶體中**連續存取**資料的方法。
- 指標運算時，指標**一次移動的單位**是以目前**指標所指向的資料型別**所**佔用 Bytes 數**來計算。

陣列與指標

- 所以 $P+i$ 指的是 P 指到的這個位址後第*i*個整數的記憶體所在位置
- $*(P+i)$ 指的就是在 $*P$ 之後第*i*個整數的值
- $*(P+i)$ 可以寫成 $P[i]$
- 所以，指向一大塊空間的指標，可以把它當成陣列語法來用



陣列與指標

- 陣列與指標**相同**處：
 - 陣列名稱代表某資料的記憶體位置
 - 指標名稱代表某資料的記憶體位置
- 陣列與指標**不同**處：
 - 使用陣列名稱**不能**存放資料，索引值用來**直接**存取該內容
 - 使用指標名稱**可以**存放資料，索引值用來**間接**存取該內容

小練習

- 宣告一陣列A，長度為5
- 使用一指標P存取此陣列 ex: 「*(P+i)」
- 讓使用者輸入5個數字，並找到最大值

<https://jgirl.ddns.net/problem/0/1050>



課程大綱

- 指標簡介
- 陣列與指標
- 動態記憶體配置
- 指標宣告進階

動態記憶體配置

- 動態記憶體配置則是：
 - 當程式執行到一半，發現它需要一塊記憶體空間來存放資料，才向系統索取一塊記憶體空間。
 - 當此記憶體空間用不到時，也可隨時將之釋放供其它程式使用
- 動態記憶體配置的特色：
 - 由於寫程式時無法預知使用者需要多少資料，因此可設計成在使用者輸入數字個數後，再動態配置所需的記憶體空間來存放數值。
 - 所需使用的空間很大(2MB)的時，可用動態記憶體配置。

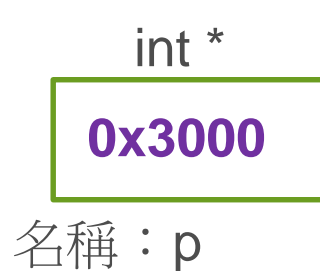
動態記憶體配置

- 動態記憶體配置：
 - 先準備好一個指標
 - 跟系統要求空間, 用指標間接操作

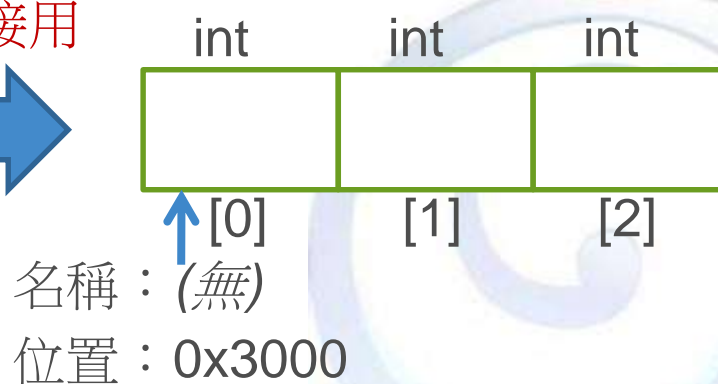
使用指標做記憶體配置：

```
p = (int*)malloc(sizeof(int)*3);
```

系統：**0x3000**這位置有你要的!



不能直接用, 只好間接用



我突然需要使用三個整數當陣列用!

動態記憶體配置的語法

- 向系統索取記憶體區塊主要是

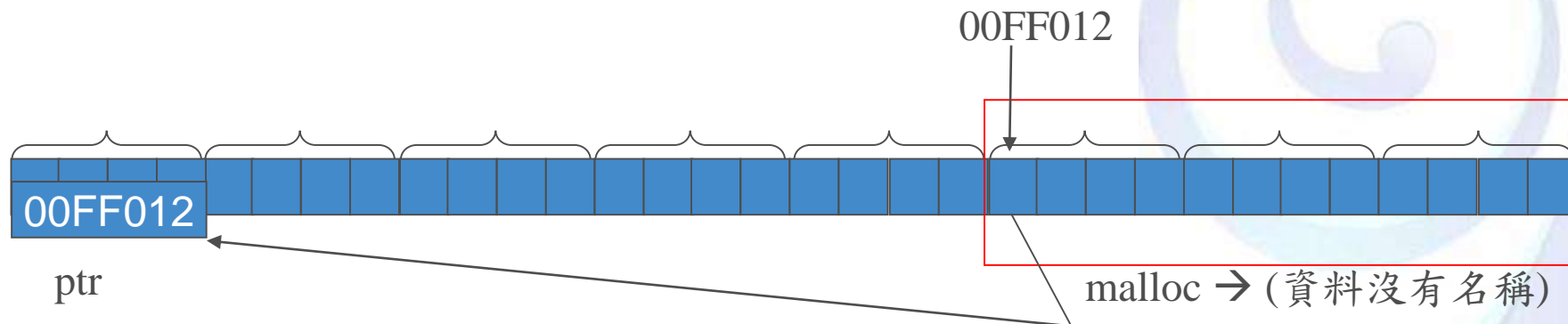
- 透過 `malloc ()` 函式來做
- 此函式的原型宣告放在 `stdlib.h`
- 呼叫的語法如下：

(資料型態 *)`malloc(sizeof(資料型態) * 個數);`

- 使用完畢後，用 `free` (指標名稱) 的語法將配置的記憶體釋放

- EX:**

- `int *ptr = (int*) malloc(sizeof(int) * 3);`



[例]

```
int *ptr, n=4;  
ptr=(int*)malloc(sizeof(int)*n);
```

說明

1. sizeof(int)

取得一個整數變數的大小，假設為 4 Bytes。

2. sizeof(int)*n

取得 n 個整數變數的大小，n=4 則 16 Bytes。

3. malloc(sizeof(int)*n)

系統配置連續 16 個記憶體位址給這四個整數使用。並傳回 void 指標指向此塊配置記憶體的起始位址。

4. ptr=(int*)malloc(sizeof(int)*n);

由於 ptr 為指向整數變數的指標，因此使用(int*)將 malloc()函式傳回的 void 指標轉換成 int 型別的整數指標，最後再指定給 ptr 整數指標。

動態配置記憶體

- **int *ptr, n=4;**
/* 動態配置記憶體 */
ptr=(int*)malloc(sizeof(int)*n);
/* 釋放記憶體 */
free(ptr);

動態記憶體配置

- 範例:使用者先輸入一班有幾個學生，再一一輸入學生的考試成績

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int students;
    int *score;
    int i;

    printf("How many students?");
    scanf("%d",&students);
    score=(int *)malloc( sizeof(int)*students );
    for ( i=0; i < students; i++ )
    {
        printf("student %d=",i);
        scanf("%d",&score[i]);
    }
    free(score);
    score = NULL;
    return 0;
}
```

<https://goo.gl/ctC6Hb>

小練習 (回家作業一)

- 修改上述範例，算出全班平均
- 學生成績 一維動態 (HW)

使用者先輸入一班有幾個學生，再一一輸入學生的考試成績，並求出它們的平均值

印出不及格的同學之號碼與分數(若無不及格的同學也會印出fail:)

印出其中最高分同學之分數(若有多筆最高分同分者，請列出索引值最小的)

Sample Input

```
3
40 60 50
```

Sample Output

```
avg = 50.00
fail:
1: 40
3: 50
highest:
2: 60
```


補充

- 動態記憶體配置初始化為0
- memset()
- `string.h`

- 配置後即為0
- `void* calloc (size_t num, size_t size);`
- <http://www.cplusplus.com/reference/cstdlib/calloc/>

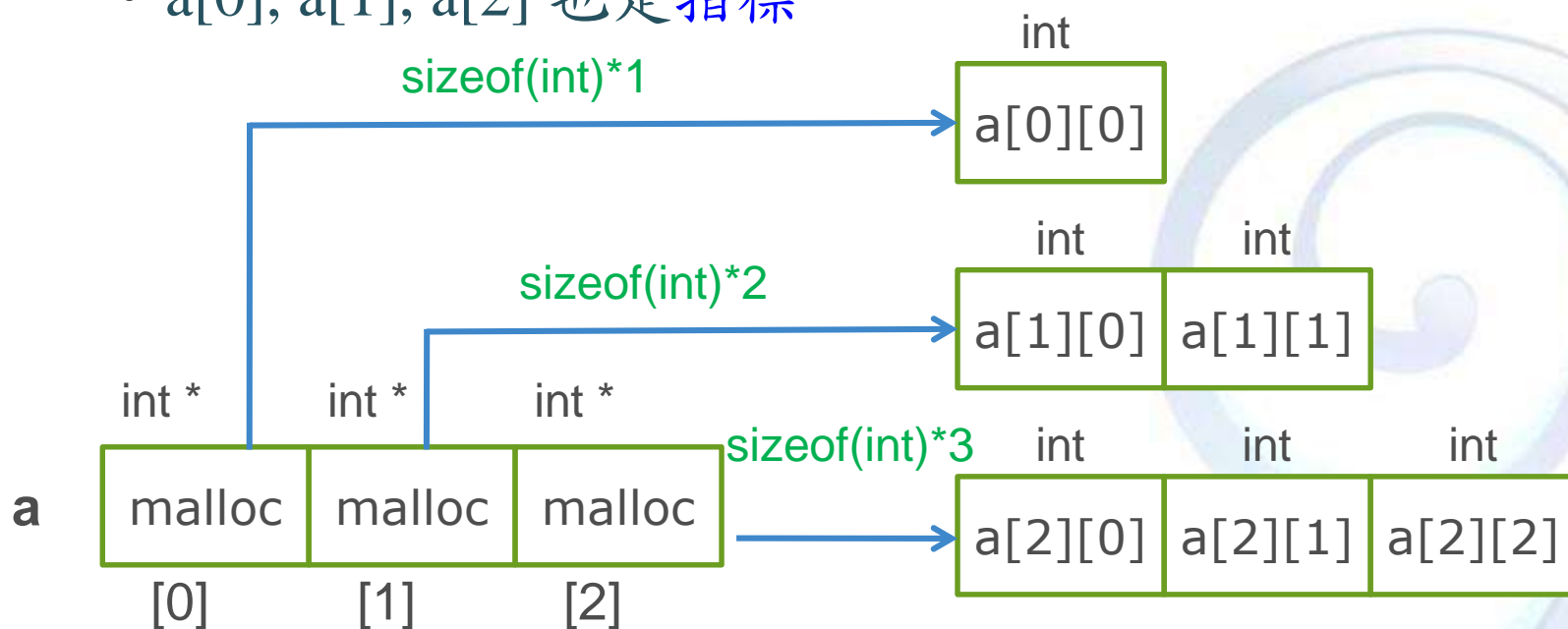
課程大綱

- 指標簡介
- 陣列與指標
- 動態記憶體配置
- 指標宣告進階



指標陣列

- 將指標宣告成陣列 (很多個指標)
 - 用指標陣列配置多個指標.
 - 應用: 配置列數固定，行數不固定的二維陣列
- 範例: **int *a[3];**
 - a[0], a[1], a[2] 也是指標



指標陣列

- 用多個指標配置出不同行數之二維陣列
- 常見應用: 固定三個班級，每班人數不同，輸入成績

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int *a[3]; // 宣告3個可以指向整數的指標變數

    a[0] = (int *)malloc( sizeof(int) );
    a[1] = (int *)malloc( sizeof(int)*2 );
    a[2] = (int *)malloc( sizeof(int)*3 );

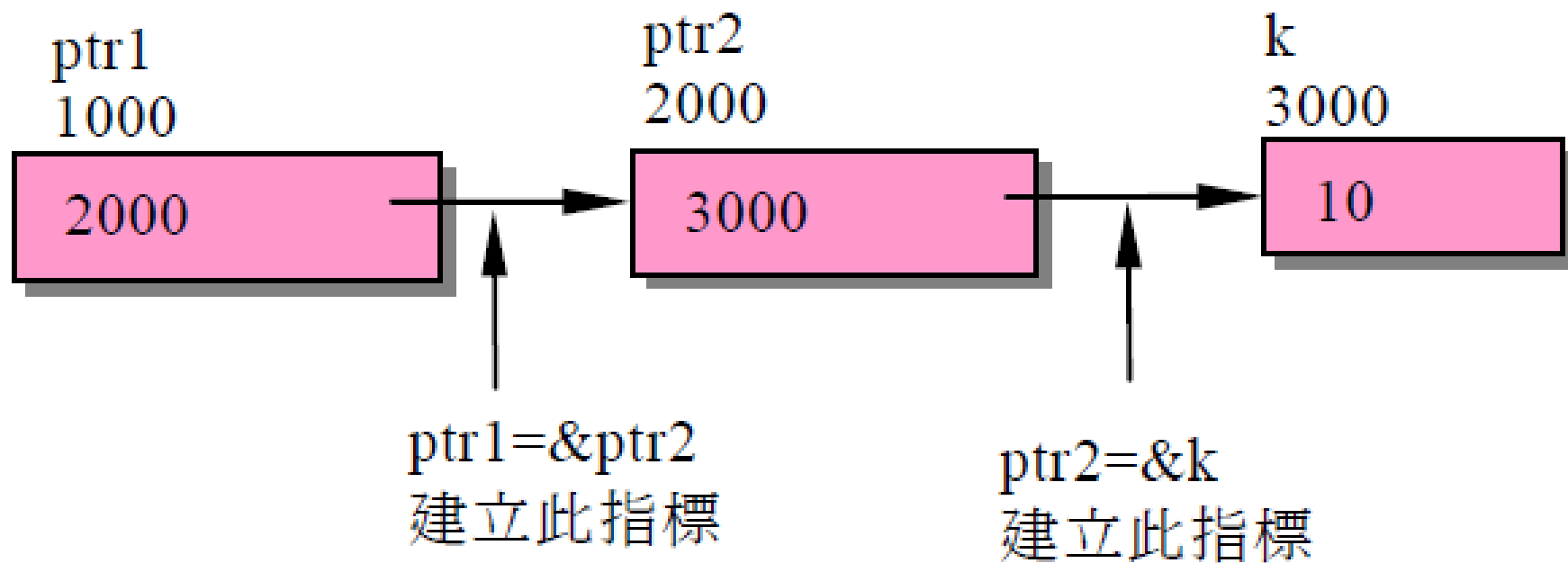
    a[0][0] = 10;
    a[1][0] = 20;
    a[1][1] = 30;
    a[2][0] = 40;
    a[2][1] = 50;
    a[2][2] = 60;

    return 0;
}
```

[0]	10		
[1]	20	30	
[2]	40	50	60
	[0]	[1]	[2]

使用多重指標

- 指標也可以再指向另一個指標，這種指標的指標稱之為「多重指標」



上圖中透過指標中的指標來存取資料
有下面兩種寫法：

方式 1：

```
int k=10;  
int *ptr2=NULL;  
int **ptr1=NULL;  
ptr2=&k;  
ptr1=&ptr2;  
printf("%d \n",**ptr1);
```

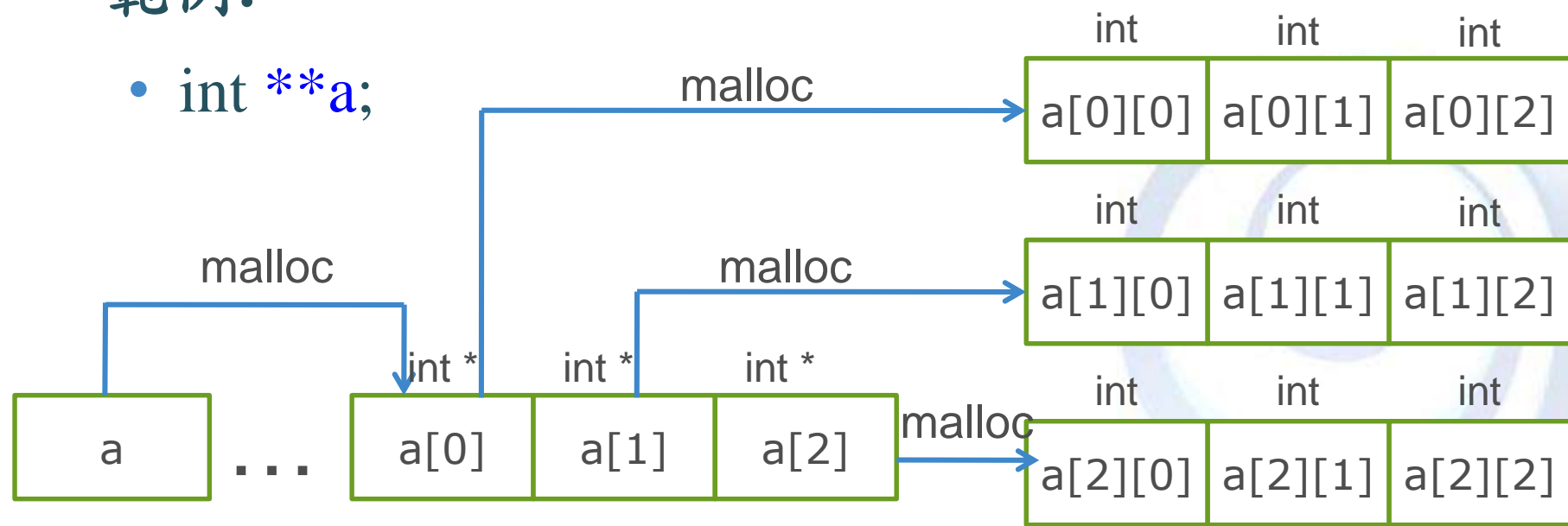
方式 2：

```
int k=10;  
int ptr2=&k;  
int **ptr1=&ptr2;  
printf("%d \n",**ptr1);
```


指標的指標

- 用指標的指標配置多個指標 (為了產生多個指標)
- 用指標的指標配置多個指標。
 - 應用：配置列數不固定，行數也不固定的二維陣列
- 範例：

- `int **a;`



某個位置

指標的指標

- 動態配置**m**班，每班**n**人，輸入成績後計算平均分數

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main()
{
```

```
    int i, j;
    double sum=0, aver;
    int **student;
    int m, n;
```

```
    printf("請輸入班級數目: ");
    scanf("%d", &m);
    printf("請輸入每班人數: ");
    scanf("%d", &n);
```

```
//動態配置m班各n人之記憶體
```

```
student = (int **)malloc(sizeof(int *) * m);
for ( j=0; j < m; j++ )
    student[j] = (int *)malloc(sizeof(int) * n);
```

```
// 分別讀入m班, 各n個同學成績
```

```
for ( j=0; j < m; j++ )
{
    printf("班級%d:\n", j+1);
    for ( i=0; i < n; i++ )
    {
        printf("學生%d: ", i+1);
        scanf("%d", &student[j][i]);
    }
}
```

```
// 計算總和
```

```
for ( j=0; j < m; j++ )
    for ( i=0; i < n; i++ )
        sum+=student[j][i];
```

```
// 求平均值
```

```
aver=sum/(m*n);
printf("全校平均為: %lf\n", aver);
return 0;
}
```

Sample Input

回家作業二

- 修改上述例子，增加輸出各班平均，與回收記憶體

```
2
3
70 80 90
50 60 70
```

Sample Output

```
class 1
1: 70
2: 80
3: 90
avg: 80.00
class 2
1: 50
2: 60
3: 70
avg: 60.00
avg: 70.00
```

補充說明

- 當系統記憶體耗盡或沒權限獲得更多記憶體時
- malloc() 要記憶體就會失敗
- `ptr=(int*)malloc(sizeof(int)*n);`
- 此時 ptr 就會得到 NULL
- `ptr=(int*)malloc(sizeof(int)*n);`
- `if(ptr==NULL){`
 - `puts("out of mem");`
 - `return 8; //ERROR_NOT_ENOUGH_MEMORY`
- `}`

示範

- 靜態陣列的使用限制 2MB (stack的限制)
- 動態記憶體配置記得一定要回收記憶體
不回收的話會發生什麼事？

