

Course05

陣列與指標

一維陣列

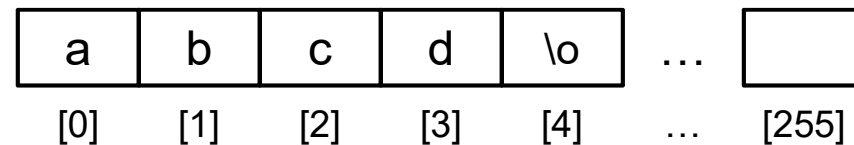
- 陣列宣告的語法：
資料型別 陣列名稱[大小];
- 資料型別：如：char、int、float 或是 double
- 陣列名稱
 - 命名規則同於變數的取名，後面的中括弧是必要的
 - 大小就是指定所要建立陣列的大小
 - 陣列當中每個位置的內容稱為陣列元素
- 陣列儲存空間是從 0 開始編號
 - 例如：int x[5];
 - 代表宣告一個名稱叫做 x 的陣列，總共佔用 5 個位置
 - 編號分別是 x[0]、x[1]、x[2]、x[3]、x[4]
 - 每一個位置都能儲存一個整數

- 陣列內容的存取：透過索引的方式來存取
 - 例如：`x[0] = 3;` 指定編號 0 的儲存位置內容為 3
 - 中括弧中的 0 就是索引值，代表儲存位置編號 0 的位置
 - 例如：將陣列位置編號 2 的內容乘 2 指定到編號 3 儲存位置
 - `x[3] = x[2] * 2;`
- 這種只有一個維度的陣列又稱為一維陣列

- 字元陣列(字串)

- `char str[255];`
- 宣告一個型態為字元的陣列，其大小為**255**個字元
- 宣告一個字串，最長可以儲存**255**個字元

`char str[255] = "abcd";`



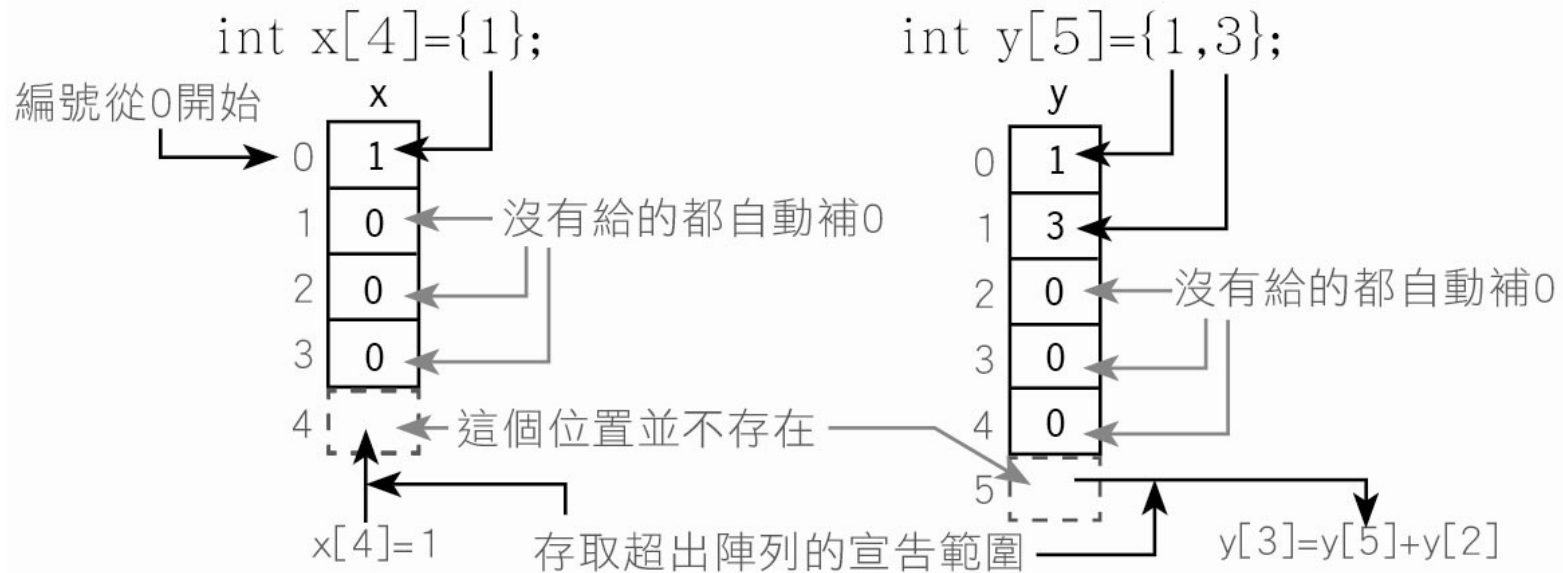
- 一維陣列

- 宣告時設定初始值

- 如果要在宣告時設定初始值，基本要求如下：

資料型別 陣列名稱[大小] = {內容0, 內容1, 內容2, ...};

- 初始值必須寫在「{ }」內，每個初始值必須以逗號區隔
 - 由左而右採一對一對應的方式存入
 - 當初始值個數小於陣列大小時
 - 不足的部分都會補上0
 - 字元陣列則補上 null character



• 存取的限制

- C 語言陣列的起始位置從 0 開始
- 不可以使用超出宣告範圍
 - 如：`int x[4];` 可以存取的陣列位置就是 `x[0]`、`x[1]`、`x[2]` 與 `x[3]`

程式

- 利用亂數模擬擲骰子 50000 次，然後計算 每一個數字出現的機率
- 思考
 - 變數 i 從0到 49999，每次都產生一個亂數
 - **i=0;i<5000;i++**
 - rand()%6+1產生1-6亂數
 - 最後再將出現的次數除上 50000 就是出現的機率
 - 判斷
 - 使用if-else-if
 - 使用switch-case

- `int dice[6];` 用來儲存每一個面出現的次數(六個面)
 - `dice[0]` 儲存出現 1 點的次數
 - `dice[1]` 儲存出現 2 點的次數
 - ...
 - `dice[5]` 儲存出現 6 點的次數
- 宣告時設定陣列元素初值都是 0
 - `int iDice[6] = {0};`
 - 只對設定陣列的所有內容都是 0 才會正確
- 以變數 `t` 取得出現的點數：`t = rand()%6+1;`
 - `t` 取得出現的面數 1 到 6 的其中之一
 - 以 `t` 的內容為點數，對 `dice` 陣列的相對應儲存位置內容遞增 1
 - 假設 `t` 內容為 3，就是 `dice[2]` 的內容要加 1
 - 寫成 `dice[2]++;`
 - » 將 `dice[2]` 看成是一個變數如：`x`，那就是 `x++`


```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(void)
{
    int dice[6] = {0};
    int i, t;
    srand( (unsigned)time(NULL) );
    for(i=1; i<=50000 ; i++) {
        t = rand()%6+1;
        if( t == 1 )
            dice[0]++;
        else if( t == 2 )
            dice[1]++;
        else if( t == 3 )
            dice[2]++;
        else if( t == 4 )
            dice[3]++;
        else if( t == 5 )
            dice[4]++;
        else
            dice[5]++;
    }
    for( i=0; i < 6 ; i++ ) {
        printf("#%d: %5d Times,", i+1, dice[i]);
        printf("Probability = %.3f\n", (float) dice[i] / 50000);
    }
    system("pause");
    return(0);
}

```

```

c:\ D:\jingting_Working\iSET_MyDocuments\Course\1002\c\SI
#1: 8339 Times, Probability = 0.167
#2: 8281 Times, Probability = 0.166
#3: 8415 Times, Probability = 0.168
#4: 8325 Times, Probability = 0.167
#5: 8457 Times, Probability = 0.169
#6: 8183 Times, Probability = 0.164
請按任意鍵繼續 . . .

```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  ▼ int main()
6  {
7      int dice[6] = {0};
8      int i;
9      srand((unsigned)time(NULL));
10
11     for(i=0;i<50000;i++)
12         dice[rand()%6]++;
13
14     for(i=0;i<6;i++)
15         printf("#%d: %d times, probability = %f\n ",i+1,dice[i],dice[i]/50000.0);
16     return 0;
17 }
18
```

二維與多維陣列

- 二維陣列宣告的語法：

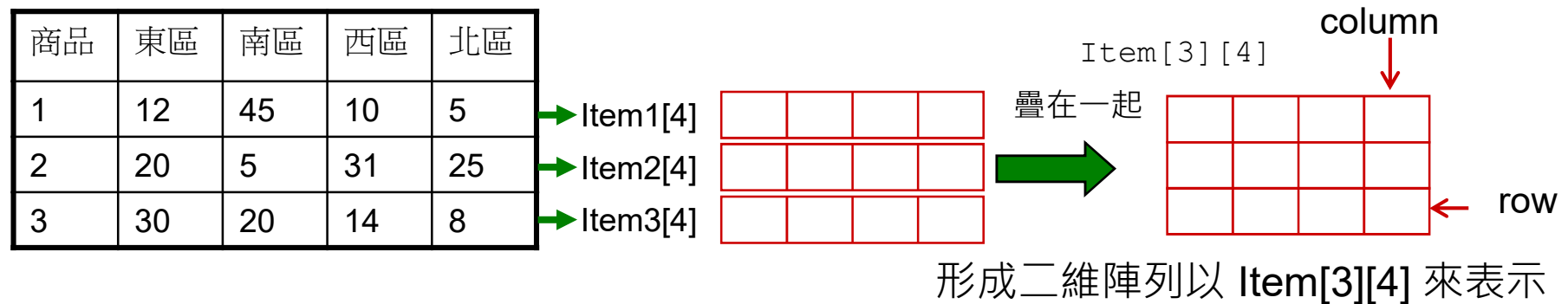
資料型別 陣列名稱 [列數] [行數];

- 範例

```
int iTable[9][9]; // 宣告一個 int 型別的陣列 iTable, 共有 9 列 9 行  
float fSales[10][4]; // 宣告 float 型別的陣列 fSales, 共有 10 列 4 行
```

- 基本的宣告規範都與一維陣列相同，差別在
 - 陣列的存取方式
 - 宣告時設定初始值

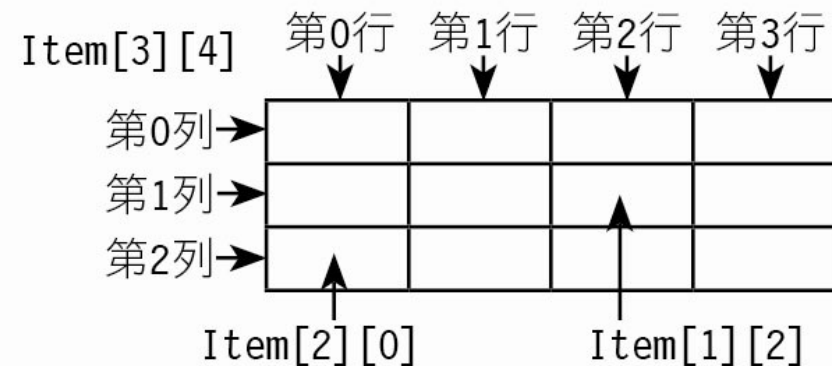
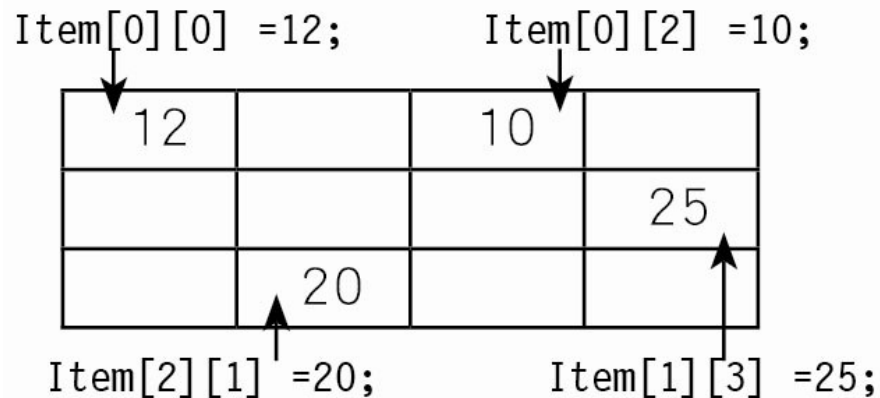
- 將一維陣列加以延伸就是二維陣列
- 三個區域的商品，以三個一維陣列或是一個二維陣列來儲存



- Item[3][4]
 - 第一個索引值，3 代表有三個 row (列)
 - 第二個索引值，4 代表有四個 column (行)

- 陣列的存取方式

- 二維陣列就是一張平面的表格
 - 水平為列，垂直為行
- 編號從 0 開始，水平從第 0 列開始，垂直也是從第 0 行開始
- 存取時指明要存取是哪一列哪一行的陣列內容就即可



程式

- 將表格儲存到二維陣列中
- 思考
 - 需要一個二維陣列
 - 3列4行，3x4
 - `int item[3][4]`
 - 需要使用兩層迴圈存取

| | 第1區 | 第2區 | 第3區 | 第4區 |
|-----|-----|-----|-----|-----|
| 商品1 | 12 | 45 | 10 | 5 |
| 商品2 | 20 | 5 | 31 | 25 |
| 商品3 | 30 | 20 | 14 | 8 |

```

#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int i, j;
    int Item[3][4];
    for( i = 0 ; i < 3 ; i++ ) {
        for( j = 0 ; j < 4 ; j++ ) {
            printf("Item # %2d , Area %2d : ", i+1, j+1);
            scanf("%d", &Item[i][j]);
        }
        printf("\n");
    }
    for( i = 0 ; i < 3 ; i++ ) {
        printf("Item # %2d :", i+1);
        for( j = 0 ; j < 4 ; j++ )
            printf("%2d ", Item[i][j]);
        printf("\n");
    }
    system("pause");
    return(0);
}

```

```

C:\D:\jt_Work\iSET_MyDocuments\Course\1001
Item # 1 , Area 1 : 12
Item # 1 , Area 2 : 45
Item # 1 , Area 3 : 10
Item # 1 , Area 4 : 5
Item # 2 , Area 1 : 20
Item # 2 , Area 2 : 5
Item # 2 , Area 3 : 31
Item # 2 , Area 4 : 25
Item # 3 , Area 1 : 30
Item # 3 , Area 2 : 20
Item # 3 , Area 3 : 14
Item # 3 , Area 4 : 8

Item # 1 :12 45 10 5
Item # 2 :20 5 31 25
Item # 3 :30 20 14 8
請按任意鍵繼續 . . .

```


- 二維陣列使用有關的重要性質：
- 兩個 **for** 的所形成的巢狀迴圈是二維陣列最常搭配的組合

```
for( i = 0 ; i < 3 ; i++ ) {  
    for( j = 0 ; j < 4 ; j++ ) {  
        printf("第 %2d 項商品在第 %2d 區的銷售量: ", i+1, j+1);  
        scanf("%d",&Item[i][j]); // 記得要加上 &  
    }  
}
```

- 兩個維度都有指定索引值的陣列寫法本質上就是**單一變數**
 - `scanf("%d",&Item[i][j])` 將輸入的結果儲存到第 *i* 列第 *j* 行
 - 將 `Item[i][j]` 看成是單一的變數 *x*
 - `scanf("%d",&Item[i][j])` 中的 `&Item[i][j]` 的寫法就很自然了

- 索引值從 0 開始所造成的困擾
 - 在某些應用上，陣列的索引值通常會拿來當成是輸出的編號
 - 而編號同常從1 開始，所以搭配索引值要記得 +1

```
for( i = 0 ; i < 3 ; i++ ) {  
    for( j = 0 ; j < 4 ; j++ ) {  
        printf("第 %2d 項商品在第 %2d 區的銷售量: ", i+1, j+1);  
        scanf("%d",&Item[i][j]); // 記得要加上 &  
    }  
}
```

初值

- 二維陣列在宣告時設定初始值
 - 基本規則與一維陣列相同，指定內容會以**列為基礎**依序的填入
 - 指定的內容個數**小於**陣列大小時
 - **不足的部分都會補上0**
- 二維陣列多了一個維度，可以利用「{ }」針對個別的列指定內容

只用一對「{ }」來指明初始值

`int Ary[2][3] = {0};`

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |

Ary的內容

`int Ary[2][3] = {1};`

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 0 | 0 |

Ary的內容

`int Ary[2][3] = {1,2,3,4};`

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 0 | 0 |

← 以 row major 的方式依序填入，
不足部分自動補0

利用 { } 分別指明第0列與第1列

`int Ary[2][3] = {{1},{2}};`

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 0 | 0 |

Ary的內容

利用 { } 分別指明第0列與第1列

`int Ary[2][3] = {{1,3,5},{2,4}};`

| | | |
|---|---|---|
| 1 | 3 | 5 |
| 2 | 4 | 0 |

Ary的內容

不足部分自動補0

每一列用一組{ }來設定初始值

程式

- 矩陣加法運算

$$A = \begin{bmatrix} 3 & 2 & 4 \\ 6 & 1 & 5 \end{bmatrix}, B = \begin{bmatrix} 2 & -1 & 3 \\ 3 & -4 & -2 \end{bmatrix} \quad A + B = \begin{bmatrix} 3 & 2 & 4 \\ 6 & 1 & 5 \end{bmatrix} + \begin{bmatrix} 2 & -1 & 3 \\ 3 & -4 & -2 \end{bmatrix} = \begin{bmatrix} 5 & 1 & 7 \\ 9 & -3 & -3 \end{bmatrix}$$

- 思考
 - 需要兩個二維矩陣 2x3

```

#include <stdio.h>
#include <stdlib.h>
#define ROW 2
#define COL 3
int main(void)
{
    int i,j;
    int A[ROW][COL]={{3,2,4},{6,1,5}};
    int B[ROW][COL]={{2,-1,3},{3,-4,-2}};
    int C[ROW][COL]={0};
    for( i = 0 ; i < ROW ; i++ )
        for( j = 0 ; j < COL ; j++ )
            C[i][j] = A[i][j] + B[i][j];
    printf("Matrix C = A + B = \n");
    for(i=0;i<ROW;i++) {
        for(j=0;j<COL;j++)
            printf("%3d",C[i][j]);
        printf("\n");
    }
    system("pause"); return(0);
}

```

```

c:\ D:\jt_Work\SET_MyDocuments\Course
Matrix C = A + B =
 5  1  7
 9 -3  3
請按任意鍵繼續 . . .

```

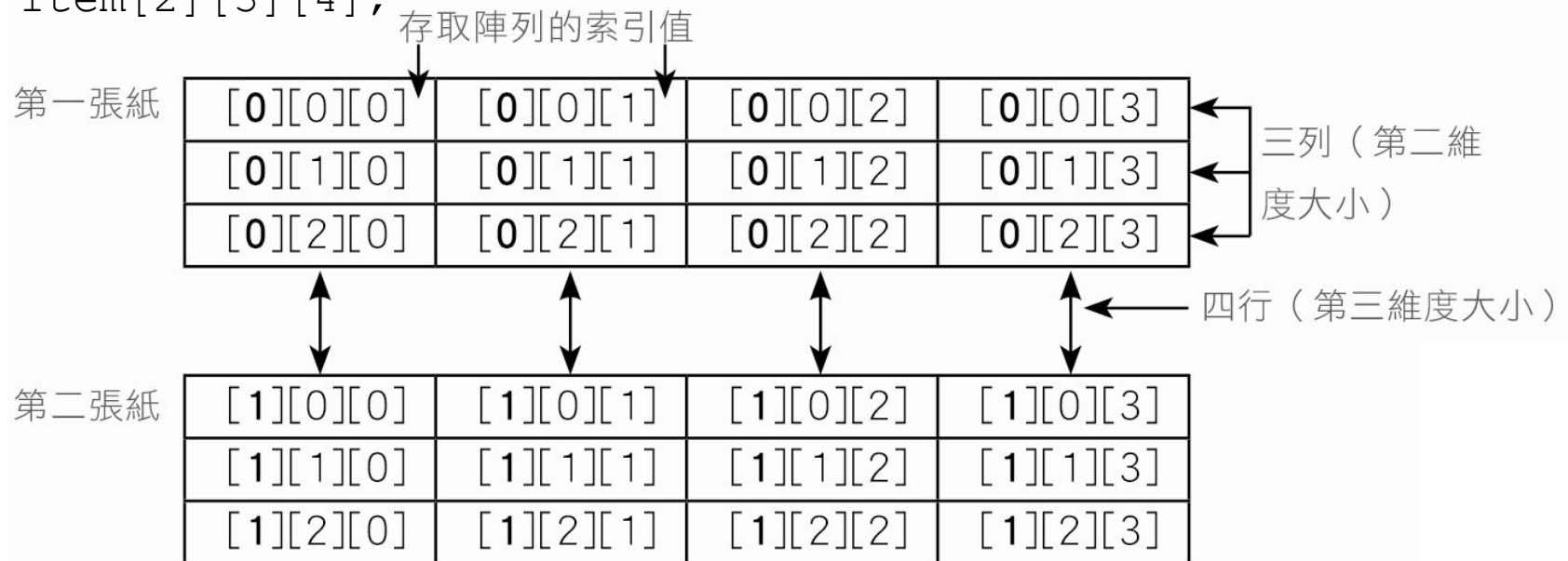
n維陣列

- 三維陣列的宣告就是延伸二維陣列的宣告，只是再增加一個維度而已
- 三維陣列宣告的語法：

資料型別 陣列名稱 [第一維度大小] [第二維度大小] ... [第n維度大小] ;

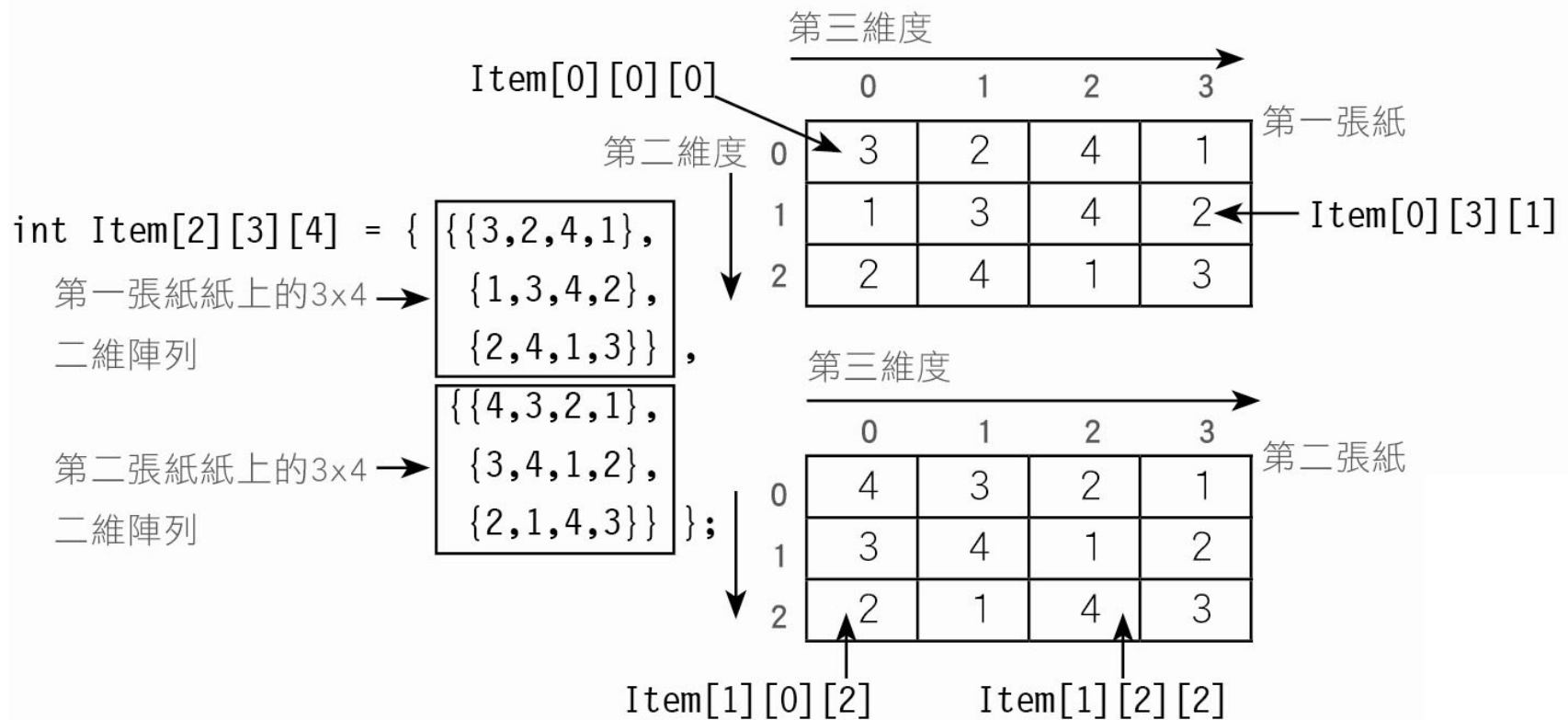
- 想成一疊連續的紙張，每一張紙上都有相同列數（第二維度的大小）與行數（第三維度的大小）
- 將三維拆成許多個二維陣列來看

```
int Item[2][3][4];
```



- 給值

- 與二維陣列相類似，只是現在多一個維度
- 如前述，想成一疊紙張，每次設定一個二維陣列的內容



排序

氣泡排序法

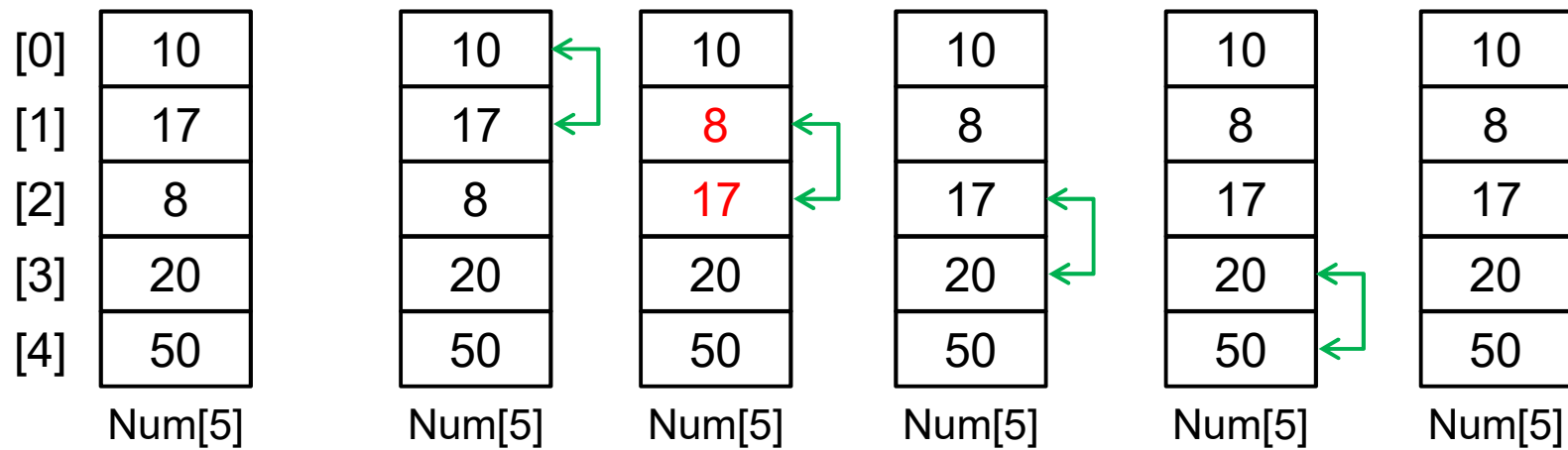
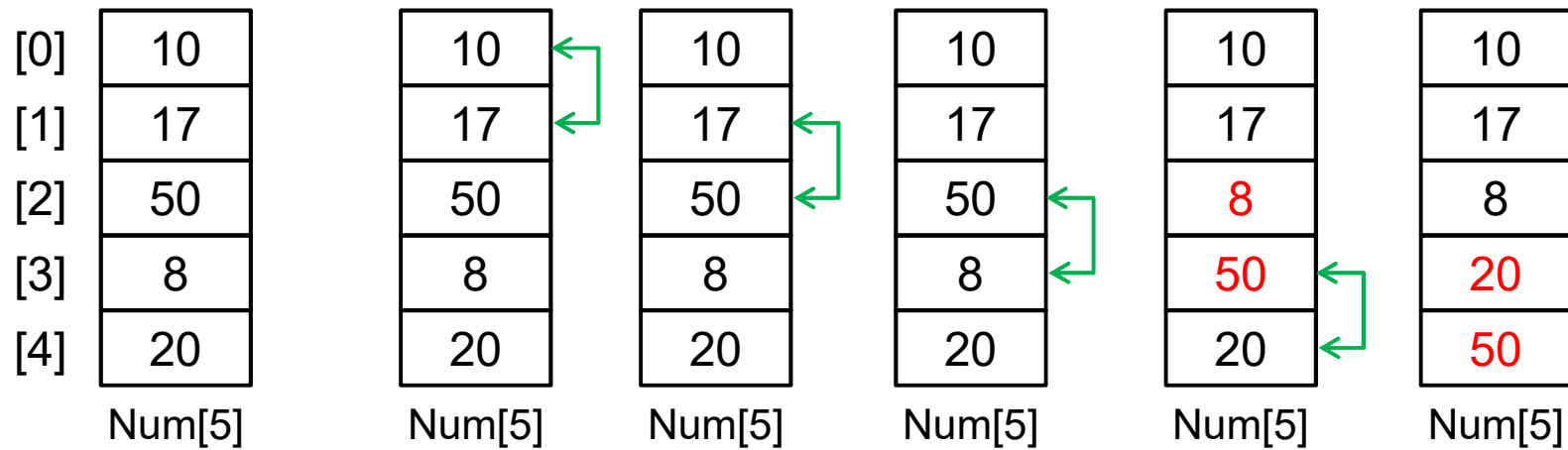
- 氣泡排序法的精神就是「交換」
 - 每次從頭到尾將所有的數值依序的檢視一次
 - 檢視的過程依照下面的規則
 - 相鄰位置的資料相互比較大小，如果「**順序**」不對就彼此交換位置
 - 如果是由小而大，就是小在前大在後
 - 如果是由大而小，就是大在前小在後
 - 在某次的檢視過程中，如果所有的數值都不需要交換，就表示序列已經排好

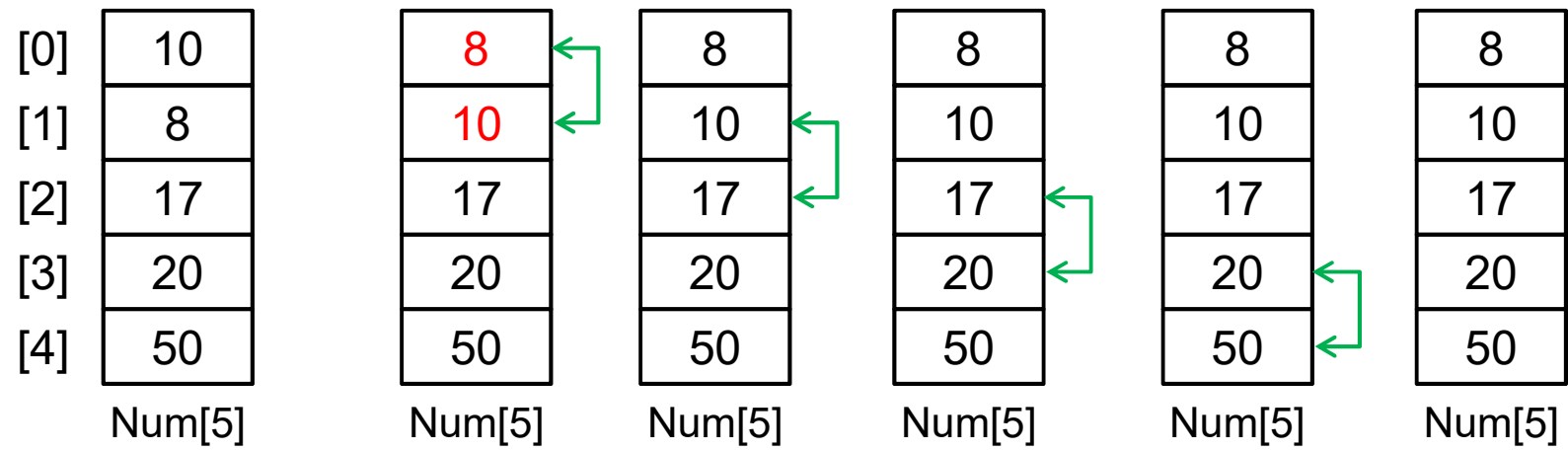
程式

- 以 `int num[5]={10,17,50,8,20}` 為例，從小排到大

| | |
|--------|----|
| [0] | 10 |
| [1] | 17 |
| [2] | 50 |
| [3] | 8 |
| [4] | 20 |
| Num[5] | |

```
if(num[i]>=num[i+1])  
    SWAP(num[i],num[i+1]);
```

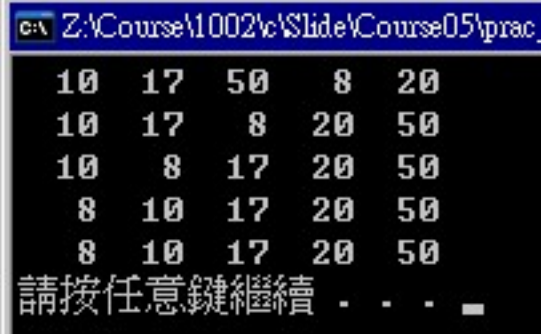




```

#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int num[5]={10,17,50,8,20};
    int i, j, t;
    int flag;
    for( j = 0; j < 5 ; j++ )
        printf("%4d",num[j]);
    printf("\n");
    do {
        flag = 0;
        for( i = 0 ; i < 4 ; i++ ) {
            if( num[i] >= num[i+1] ) {
                t = num[i];
                num[i] = num[i+1];
                num[i+1] = t;
                flag = 1;
            }
        }
        for( j = 0; j < 5 ; j++ )
            printf("%4d",num[j]);
        printf("\n");
    } while( flag );
    system("pause");
    return (0);
}

```



```

C:\Z:\Course\1002\c\Slide\Course05\prac_
10 17 50 8 20
10 17 8 20 50
10 8 17 20 50
8 10 17 20 50
8 10 17 20 50
請按任意鍵繼續 . . .

```

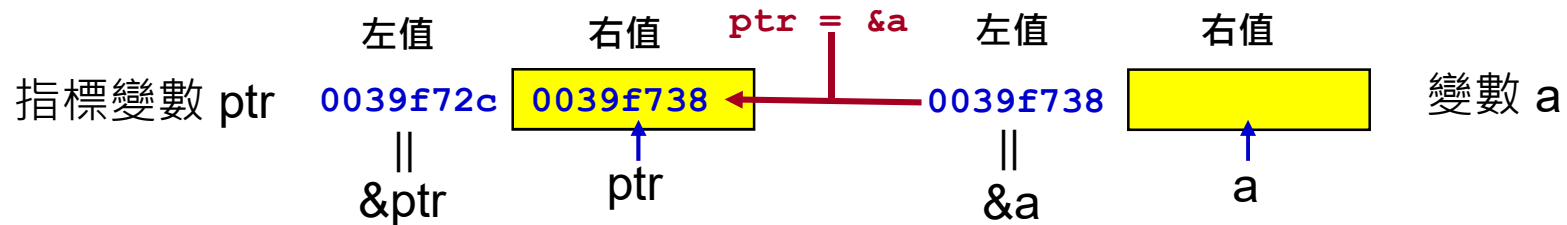
指標

觀念與宣告

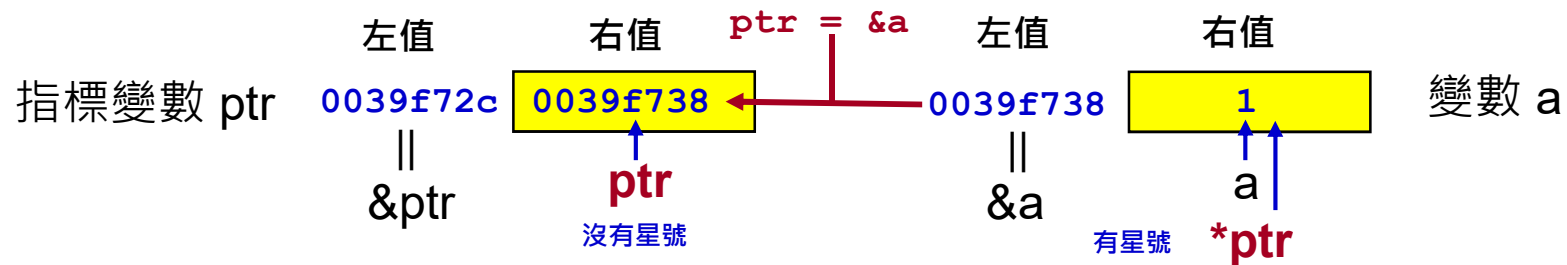
- 指標(Pointer)
 - Point + er 代表的就是指向東西的人
- 變數的左值與右值
 - 右值代表的是變數的內容
 - 左值指向變數，也就是儲存變數所使用的**記憶體位址**
- 指標最主要的目的就是**指向變數所使用記憶體的位址**
- 指標變數(pointer variable)的宣告語法如下：

資料型別 *指標變數名稱;
 加上 * 就成為指標變數

- 指標的用語
 - 當指標變數的右值被設定成某一個變數的位址時
 - 習慣上會說成指標變數指向該變數（請熟記這句話！）
 - 如：`ptr = &a`；說成變數 `ptr` 指向變數 `a`
- 指標變數的使用規則 → 指向一般變數
 - 只使用指標變數名稱不加星號：代表指標變數本身的右值
 - 例如：`int a; int *ptr;`
 - `ptr = &a` 就是將變數 `a` 的位址儲存到指標變數 `ptr` 的右值
 - 也就是 `ptr` 指向變數 `a`



- 指標變數的使用規則 — 存取被指向變數的內容
 - 在指標變數前面加上 *，就是代表被指向變數的內容
 - 如：int a=1, *ptr; ptr = &a;
 - *ptr 就是變數 a 的右值



- 指標變數兩個右值的使用 — 指標變數前面有沒有 * 號是關鍵的所在
 - 指標變數名稱前面沒有星號 — 指標變數本身的右值(位址)
 - 指標變數名稱前面有星號 — 被指向變數的右值

程式

```
1  #include <stdio.h>
2
3  ▼ int main()
4  {
5      int a=1, *ptr;
6      ptr = &a;
7      printf("a Left: %p, Right: %d\n",&a, a);
8      printf("prt Left: %p, Right: %p\n",&ptr, ptr);
9      printf("ptr value: %d\n",*ptr);
10     return 0;
11 }
12
```

```
# a Left: 0x7ffee4a14a58, Right: 1
# prt Left: 0x7ffee4a14a50, Right: 0x7ffee4a14a58
# ptr value: 1
# 按 <RETURN> 鍵來關閉此視窗 ...
#
```

記憶體配置的方式

- 記憶體配置：
 - 靜態配置：
 - 在編譯時期（ `compile-time` ），編譯器會配置記憶體空間給變數
 - 缺點為記憶體使用無彈性，可能會配置過多或過少
 - 動態配置：
 - 程式執行時期（ `run-time` ），才向系統要求記憶體空間
 - 可以有效的配置記憶體
- 取得→使用→歸還
 - 有借有還，再借不難

取得記憶體

- malloc
 - 記憶體配置

malloc() 的語法

指標變數 = (指標變數所指向的型態 *) malloc(所需的記憶體空間)

將 malloc() 所傳回的位址強制
轉換成指標變數所指向的型態

- 配置可存放3個整數的記憶體空間：

```
int *ptr;           /* 宣告指向整數的指標ptr */
ptr = (*int) malloc(12);           /* 較不好的寫法 */
ptr = (*int) malloc(3*sizeof(int)); /* 較好的寫法 */
```

將程式抽象化及通用化!!

- **new**
 - c++專屬的記憶體配置

new 的語法

指標變數 = new 指標變數型態 [所需的資料個數];

- 配置可存放3個整數的記憶體空間：

```
int *ptr;  
ptr = new int [3];
```

```
/* 宣告指向整數的指標ptr */
```

```
int *ptr;  
ptr = new int[100];
```

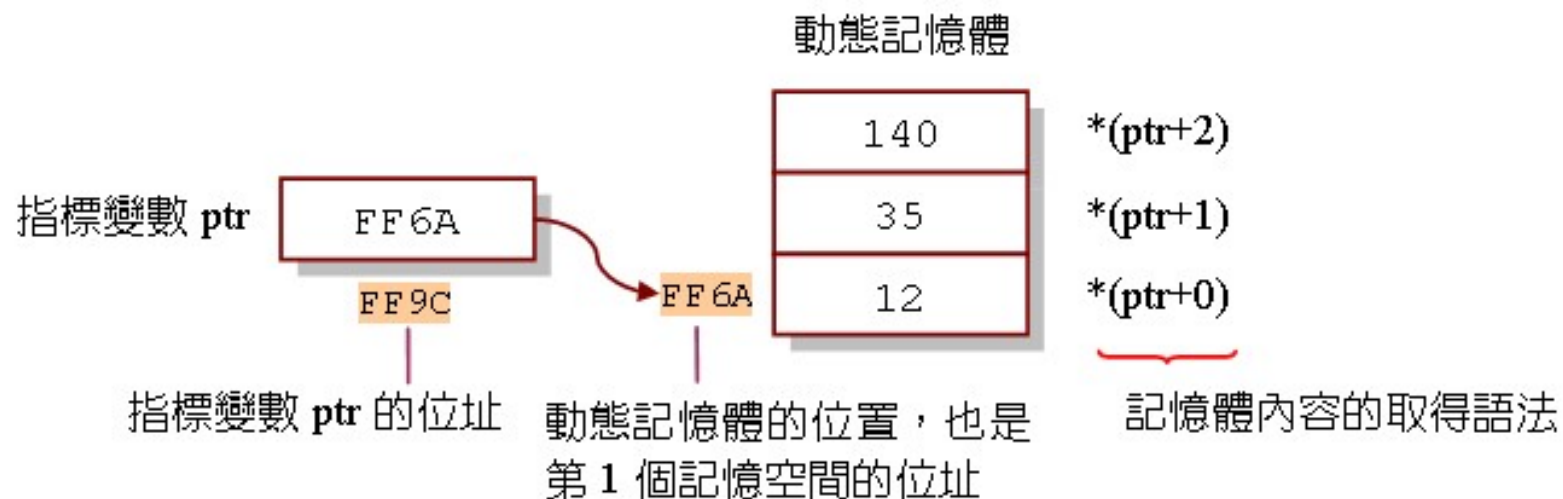
```
int *ptr;  
ptr = (*int) malloc(100*sizeof(int));
```

使用記憶體

設定與取得記憶體的內容

- 第 k 個記憶空間的內容為 $*(ptr+k-1)$

```
int *ptr;  
ptr=(int) malloc(3*sizeof(int));  
*ptr=12;           /* 將ptr所指向的第1個記憶空間設值為12 */;  
*(ptr+1)=35;       /* 將第2個記憶空間設值為35 */;  
*(ptr+2)=140;      /* 將第3個記憶空間設值為140 */;
```



歸還記憶空間

- `free()`;
- `delete[]`

`free()` 與 `delete[]` 的語法

```
free(指標變數); /* 釋放由指標變數所指向的記憶空間 */  
delete [] (指標變數); // c++專屬, 與new一起使用
```

- 記憶體洩漏 (`memory leakage`)
 - 沒有用 `free()` 歸還記憶空間
- 記憶空間分割失敗 (`segmentation fault`)
 - 記憶空間已歸還，卻還嘗試著去使用那塊記憶空間

程式

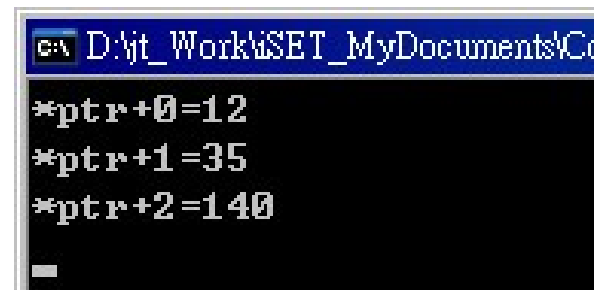
- 試寫一個程式，利用指標儲存三個變數

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

int main(int argc, char *argv[]) {
    int *ptr, i;
    ptr = (int *)malloc(sizeof(int) * 3);

    *ptr = 12;
    *(ptr+1) = 35;
    *(ptr+2) = 140;

    for (i=0; i<3; i++)
        printf(" *ptr+%d=%d\n", i, *(ptr+i));
    free(ptr);
    getch();
}
```



```
D:\jt_Work\SET_MyDocuments\Co
*ptr+0=12
*ptr+1=35
*ptr+2=140
```

函數指標

- Function Pointer
- 指向函數的指標
 - 可以存放函數的起始位址
 - 讓程式抽象化

```
void bubbleSort(int *num, int n);  
void selectSort(int *num, int n);
```

```
void (*sort)(int *,int);
```

← 宣告指向函數的指標變數
必須和函數型態一致

用法:

```
sort = bubbleSort;  
sort(num,n);
```

動態切換使用bubbleSort或是selectSort

```
1  #include <stdio.h>
2
3  void bubbleSort(int *num, int n)
4  {
5      int flag;
6      int t;
7      printf("bubble sort\n");
8      do {
9          flag = 0;
10         for(int i=0;i<n-1;i++)
11             if(*(num+i)>*(num+i+1))
12             {
13                 t = *(num+i);
14                 *(num+i) = *(num+i+1);
15                 *(num+i+1) = t;
16                 flag = 1;
17             }
18     } while(flag);
19 }
20
21 void selectSort(int *num, int n)
22 {
23     int t,min;
24     printf("select sort\n");
25     for(int i=0;i<n-1;i++)
26     {
27         min = i;
28         for(int j=i+1;j<n;j++)
29             if(*(num+min)>*(num+j))
30                 min = j;
31         t = *(num+min);
32         *(num+min) = *(num+i);
33         *(num+i) = t;
34     }
35 }
36
37 void (*sort)(int *,int);
```

```
38
39 int main()
40 {
41     int num[5] = {10,17,50,8,20};
42     int method;
43     printf("method:");
44     scanf("%d",&method);
45     switch(method)
46     {
47         case 1: sort = bubbleSort;
48                 break;
49         case 2: sort = selectSort;
50                 break;
51     }
52     sort(num,5);
53     for(int i=0;i<5;i++)
54     {
55         printf("%d ",num[i]);
56     }
57     return 0;
58 }
59
```