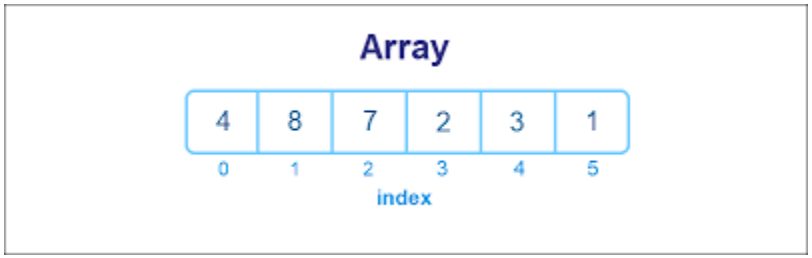


Chapter 02 陣列(Array)

陣列(Array) 是一種常見且基本的資料結構，將相同類型的資料存放於連續的記憶體中。

每個資料元素都有一個索引或稱為下標，透過這個索引可以直接存取對應的資料元素。

本章綱要

- 陣列(Array)的定義
- 陣列的表示法
- 陣列的操作
- 陣列元素位址的計算
- 魔術方陣

陣列(Array)的定義

- 陣列(Array) 是一種常見且基本的資料結構，將相同類型的資料存放於連續的記憶體中。

- 陣列也稱為線性串列又稱循序串列(sequential list)或有序串列(ordered list)。
- 其特性乃是每一項依據它在串列的位置，可以形成一個線性的排列次序，所以x[i-1]在x[i]之前， x[i+1]在x[i]之後。

陣列的特性

- 固定大小
  - 陣列的大小在創建時固定，在整個陣列的生命週期中不能改變。
- 元素的類型相同
  - 陣列中的所有元素都必須是相同資料類型(例如整數、浮點數或字符串)。
- 隨機存取
  - 陣列可以透過索引來直接存取元素，這是一個O(1)的操作(常數時間)，因此查找速度非常快。
- 連續記憶體分佈
  - 陣列中的元素在記憶體中是連續儲存的，這有助於提高存取效率，特別是在多次存取時。

陣列的表示(維度)

- 宣告一維陣列, 使用 1 對中括號[], 定義 1 個維度的大小：

```
int  list[5];
```

- 宣告二維陣列, 使用 2 對中括號[][], 定義 2 個維度的大小：

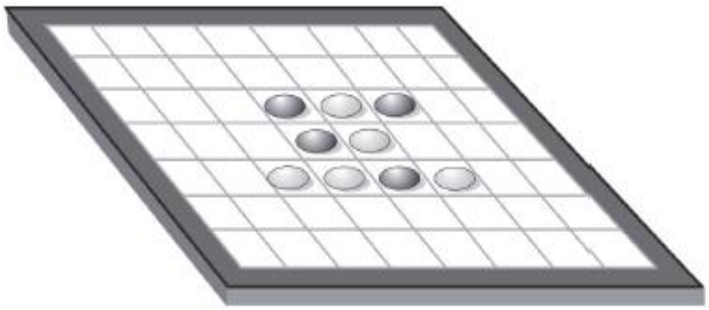
```
int  list[6][5];
```

- 宣告三維陣列, 使用 3 對中括號[][][], 定義 3 個維度的大小：

```
int  list[4][6][5];
```

二維陣列

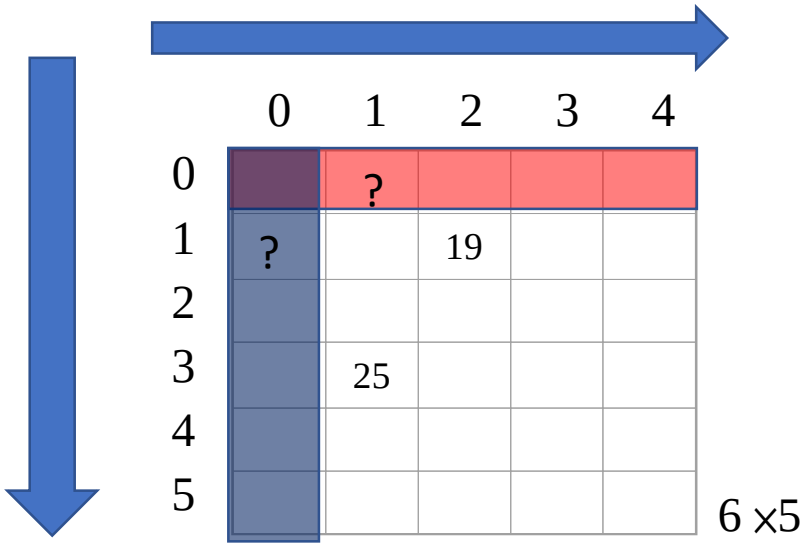
- int list[6][5]
  - 此二維陣列的所有元素在邏輯上可以排成一個有6列（row，橫者為列）、5行（column，直者為行）的平面格子



	第0行	第1行	第2行	第3行	第4行
第0列	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
第1列	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
第2列	[2][0]	[2][1]	[2][2]	[2][3]	[2][4]
第3列	[3][0]	[3][1]	[3][2]	[3][3]	[3][4]
第4列	[4][0]	[4][1]	[4][2]	[4][3]	[4][4]
第5列	[5][0]	[5][1]	[5][2]	[5][3]	[5][4]

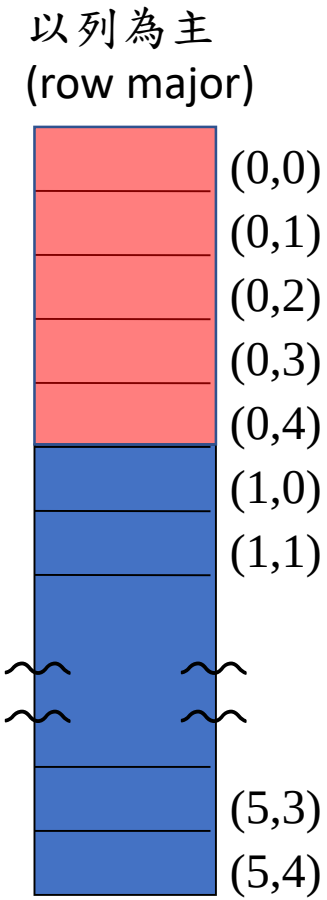
以列為主 VS 以行為主?

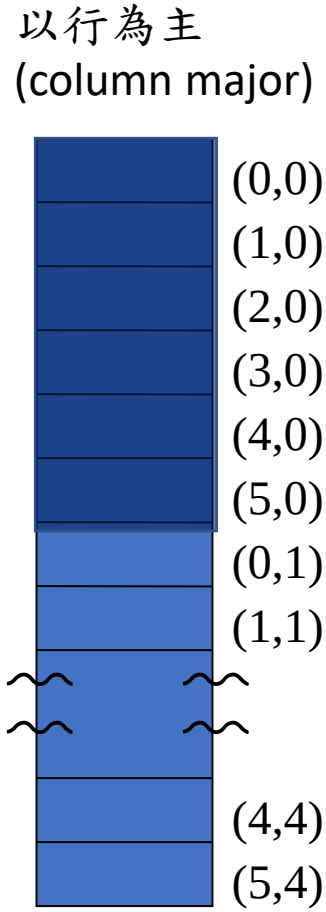
- 二維陣列在邏輯上為平面，在記憶體的排列順序是一維線性的



第1列 第2行：List[1][2]值是 19

第3列 第1行：List[3][1]值是 25

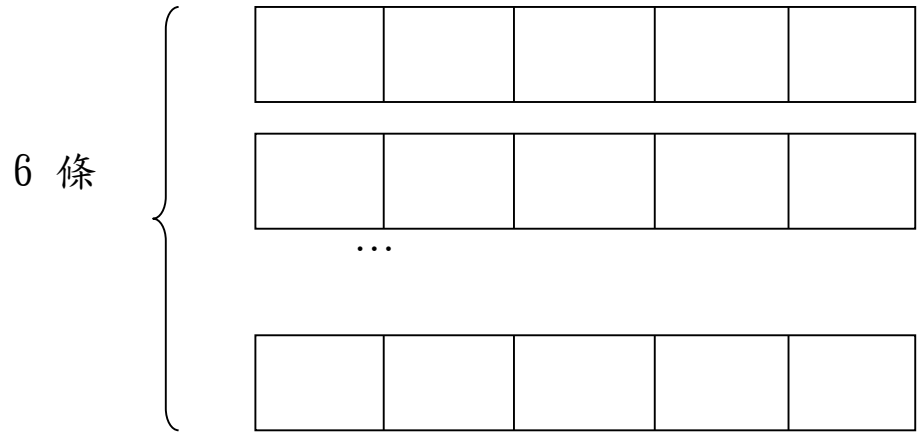




以列為主

- 若在以列為主的對應中
- list[6][5] 可以想像成有6個長度為5的一維陣列，
- 6×5的平面可由6條長度5的長條組成。

	第0行	第1行	第2行	第3行	第4行
第0列	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
第1列	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
第2列	[2][0]	[2][1]	[2][2]	[2][3]	[2][4]
第3列	[3][0]	[3][1]	[3][2]	[3][3]	[3][4]
第4列	[4][0]	[4][1]	[4][2]	[4][3]	[4][4]
第5列	[5][0]	[5][1]	[5][2]	[5][3]	[5][4]



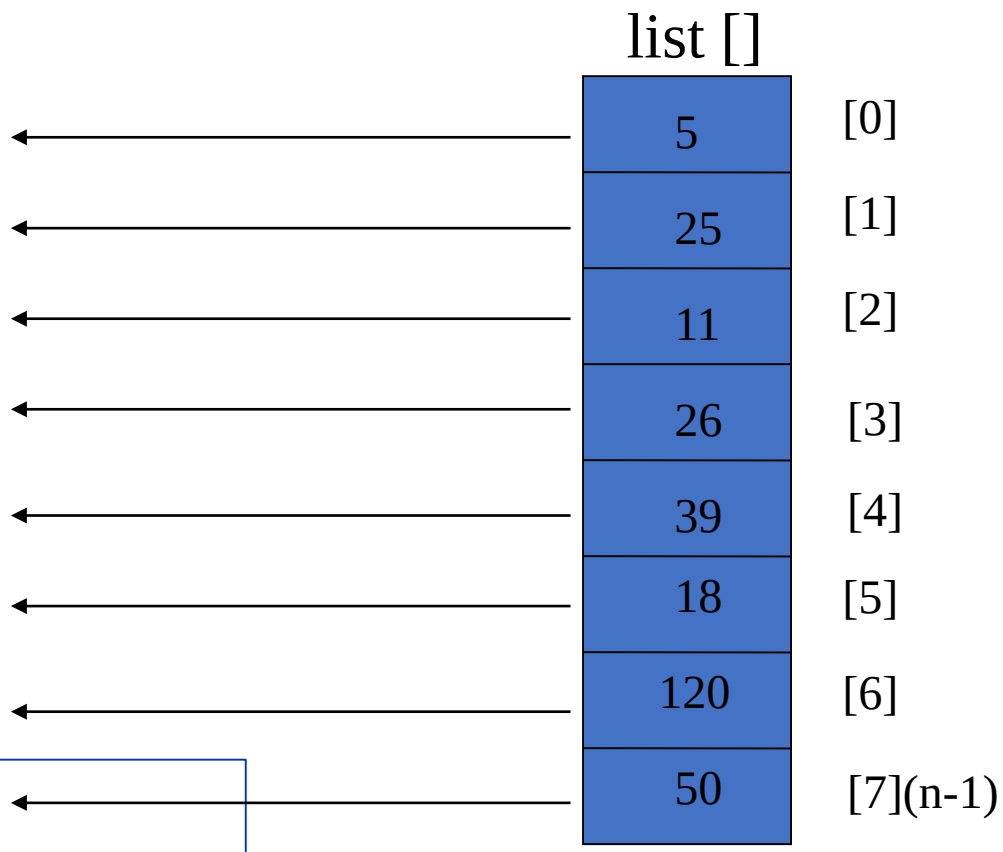
陣列的操作

1. 讀出編號為 *i* 的陣列元素內容
   value = list[i] ;
2. 將值 寫入 編號為 *i* 的元素位置
   list[i] = NewValue ;
3. 將一陣列的所有元素 輸出
4. 在編號 i 的位置 插入 一新元素，原來 *i* 和之後的元素各往後挪一個位置
5. 刪除 編號 *i* 的元素，原來 *i* +1 和之後的元素各往前挪一個位置

陣列的輸出

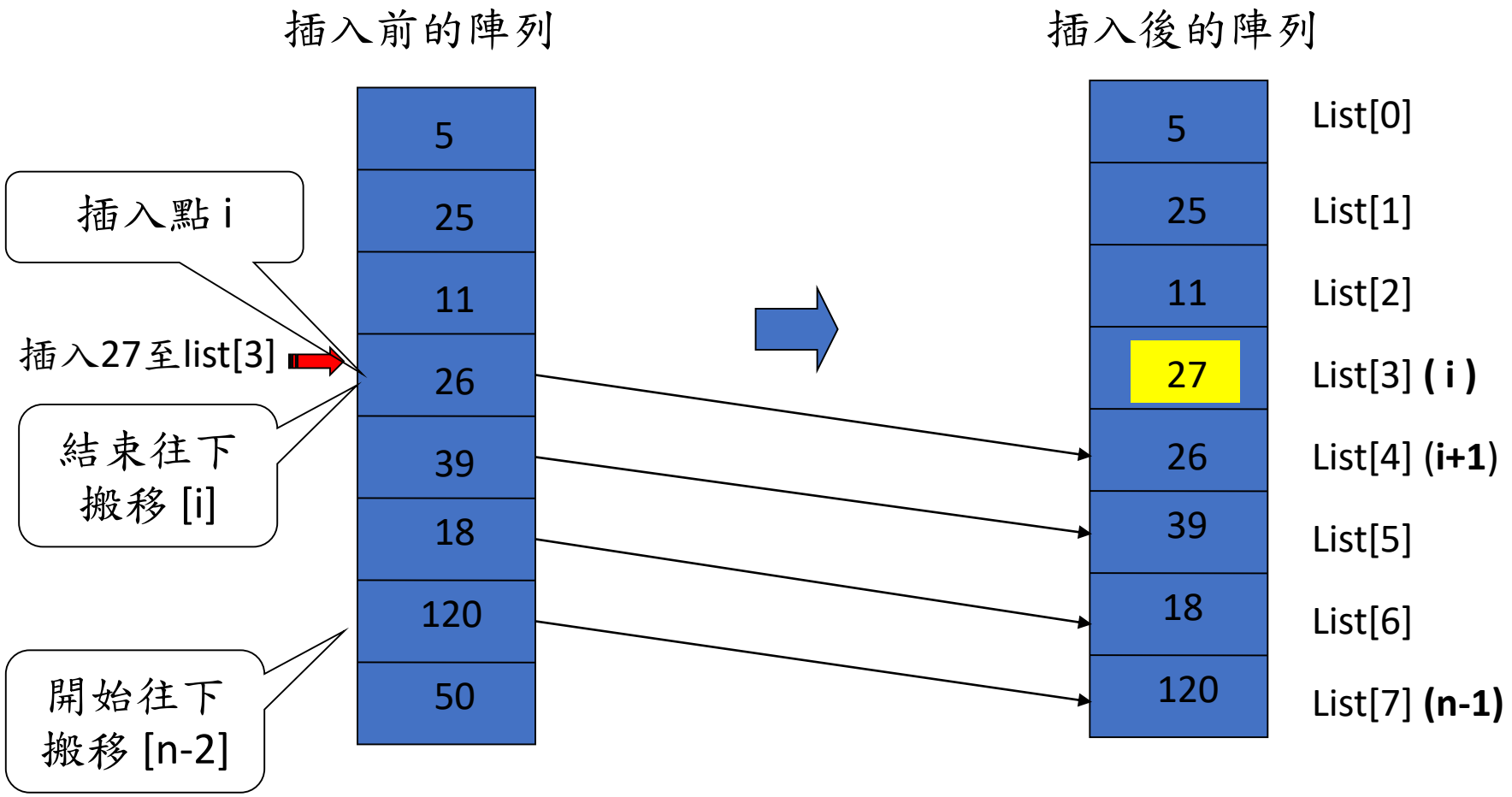
輸出 list[i]

i 從 0 到 7
(從 0 到 n-1)



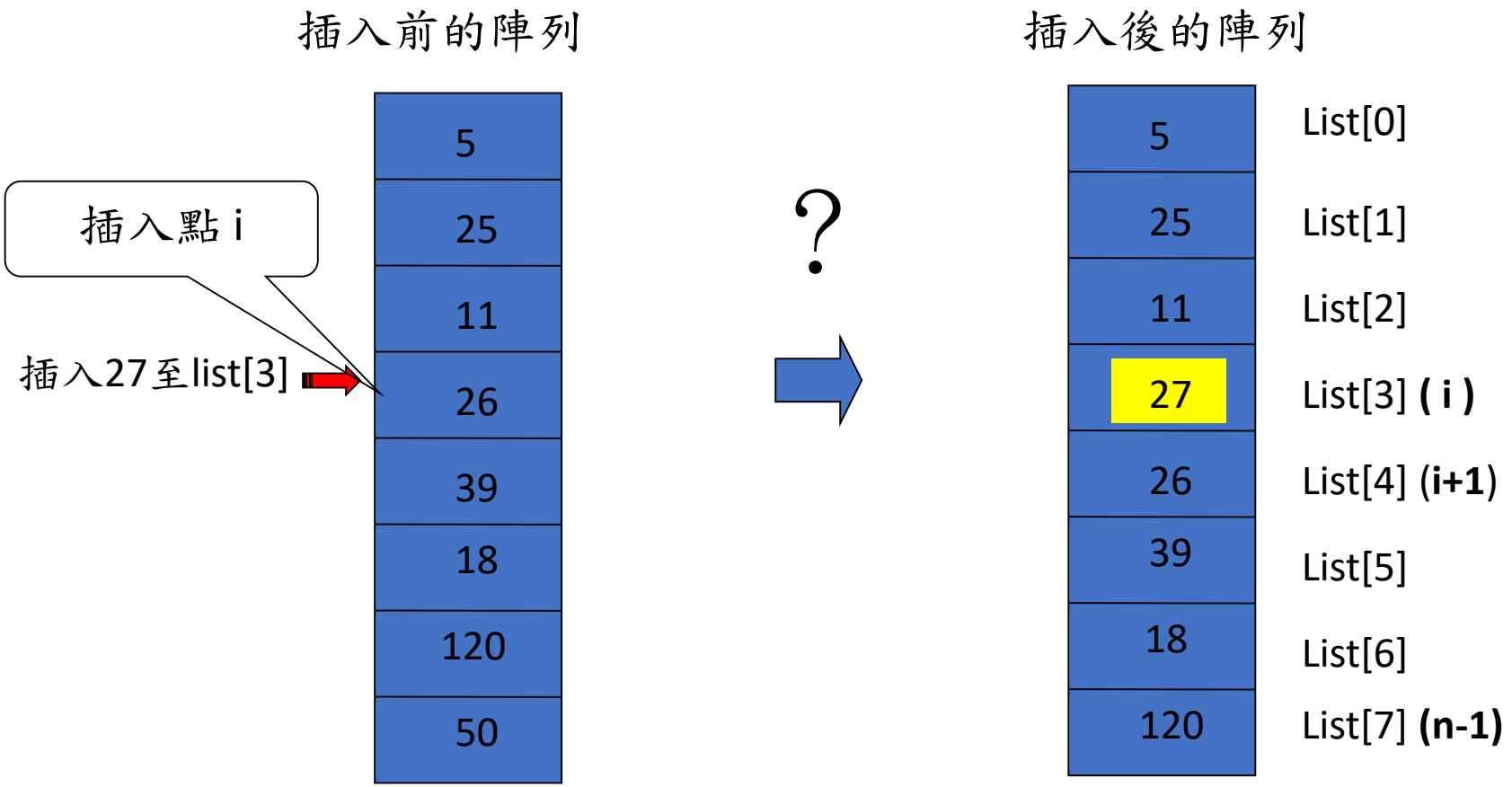

```
1.  void PrintArray( int list [], int n)
2.  { int  i;
3.     for (i = 0 ; i < n ; i++)
4.                  cout << list[i] << endl ;
           //C: printf("%d\n", list[i]);
5.  }
```

陣列元素的插入

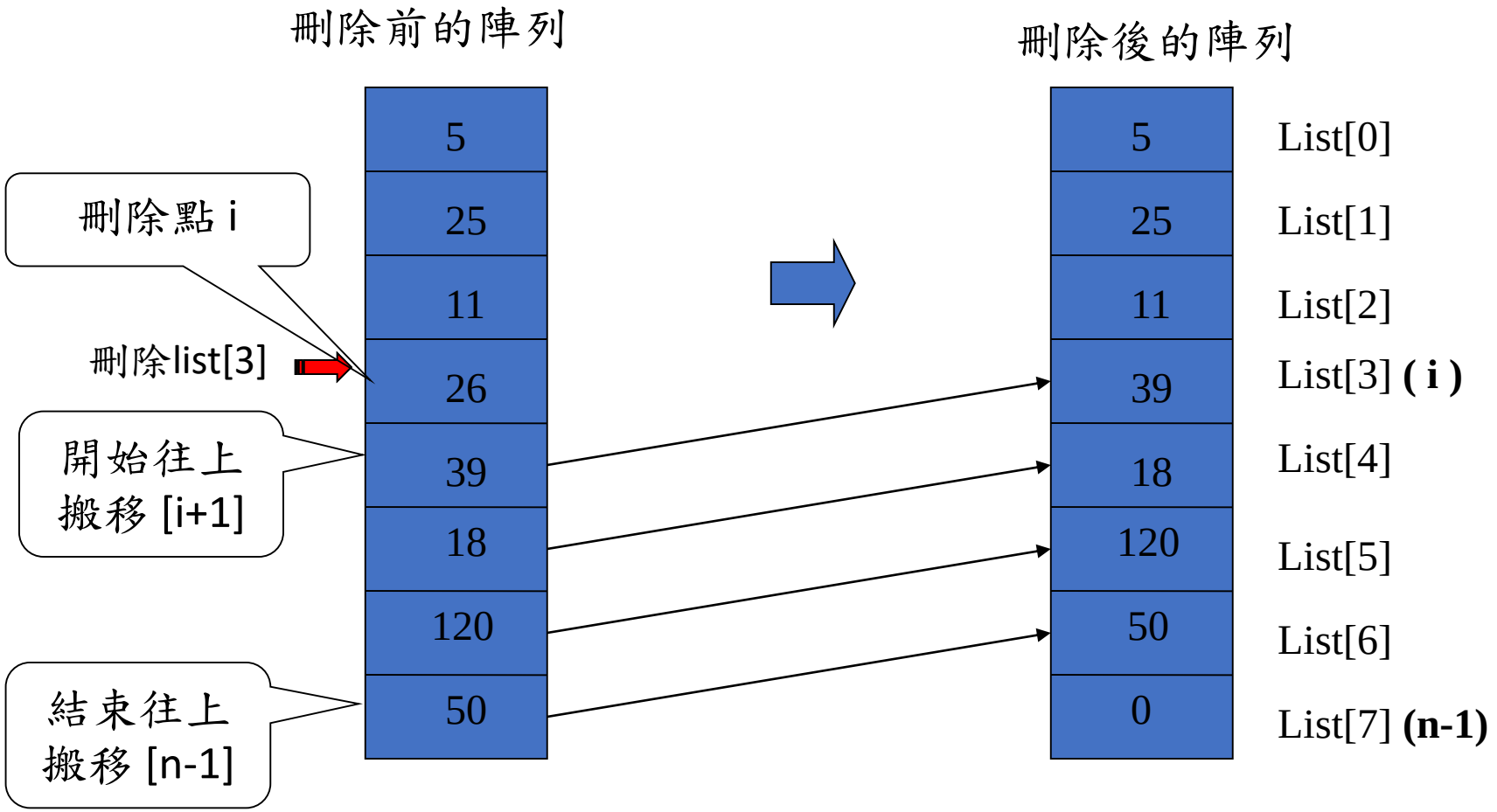


從倒數第二個資料開始(list[n-2])，到插入點為止(list[i]) 每個元素往下移一個位置

如果只是執行 list[3] = 27; 會不會有插入運算的效果？為什麼？



陣列元素的刪除



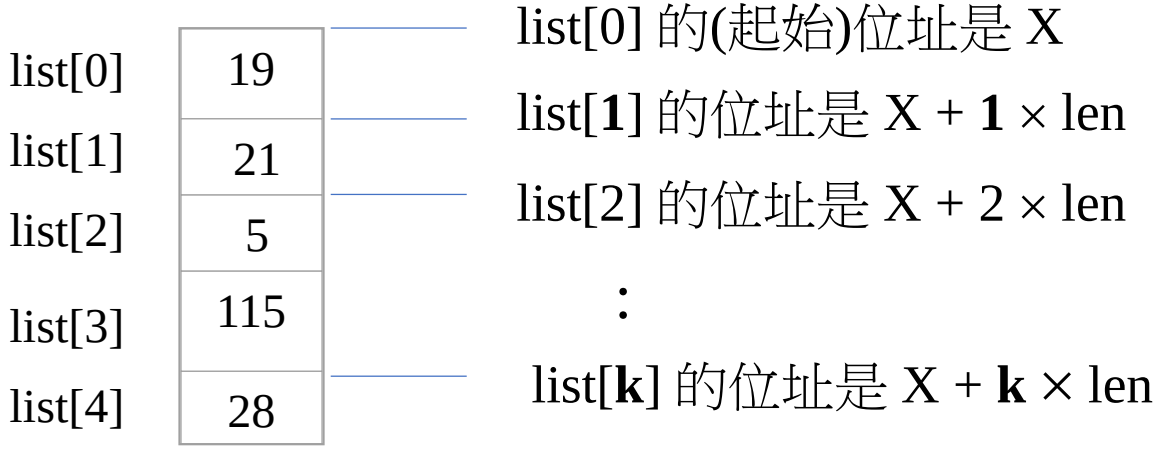
從刪除點的下一個資料開始(list[i+1])，到最後一個資料為止(list[n-1])
每個元素往上移一個位置

問題思考

- 陣列元素的插入與刪除，是否需要做資料搬移的動作?
- 若需要搬移，是否那些資料先搬移，那些資料後搬移的考量，還是沒有關係!

一維陣列位址計算

- 宣告 int list[5];
- 如果每個元素大小為 len 個 bytes



- 陣列的元素在記憶體中是連續的位置。
  list[0] 是陣列 list 的頭一個元素。
  list[1] 是編號 1 號元素，它緊接著 0 號元素。同樣的，list[i+1] 的位置是緊接著 list[i] 的。

一維陣列位址計算(例)

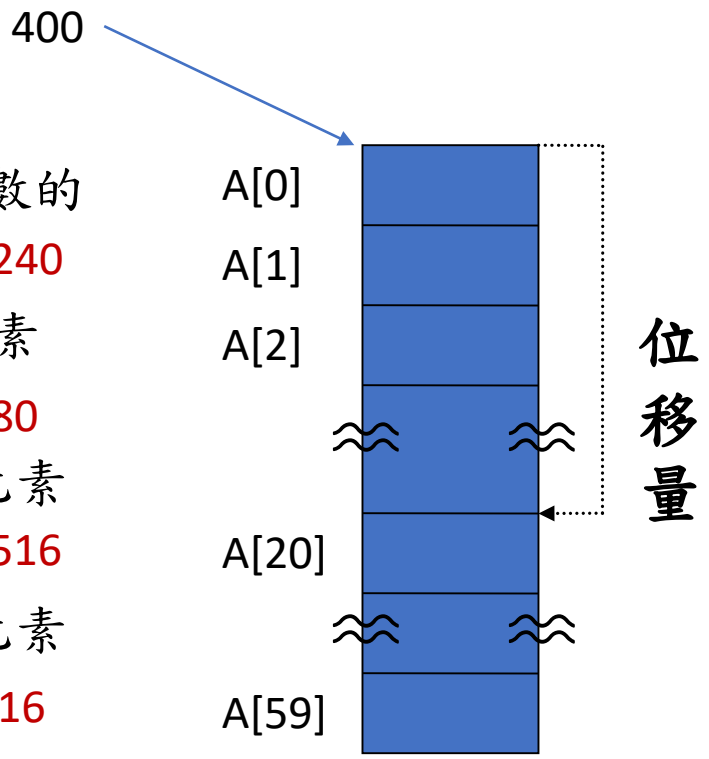
例2.4 int A[60];

(1) 此陣列共佔多少位元組（假設每個整數的大小 sizeof(int) = 4）？ =4*60=240

(2) 若 A[0]在記憶體中的位址為 400, 則元素 A[20] 的位址為何？ =400+(20-0)*4=480

(3) 若 A[10]在記憶體中的位址為 400, 則元素 A[39] 的位址為何？ =400+(39-10)*4=516

(4) 若 A[30]在記憶體中的位址為 400, 則元素 A[9] 的位址為何？ =400+(9-30)*4=316



二維陣列位址計算(例)

- **例2.5** 二維浮點數陣列宣告為 float A[8][10], 則：

(1) 此陣列共佔多少位元組（假設 sizeof(float) = 4）？ =8*10*4=320

(2) 若 A[0][0] 在記憶體中的位址為 100, 則元素 A[0][9] 的位址為何？

(3) 元素 A[6][5] 的位址為何？

Low major or column major

陣列位址計算

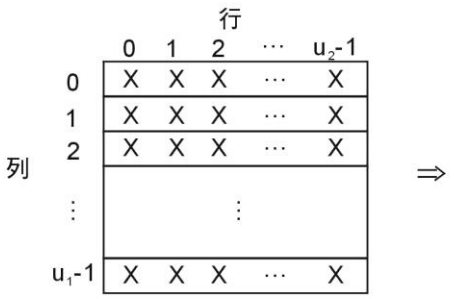
- 一維陣列（One Dimension Array）
- 若陣列是A[0 : u-1]，並假設每一個元素佔d個空間，則A[i]=α+i*d，其中α是陣列的起始位置。

陣列位址計算–二維陣列

- 假若有一陣列是A[0 : u_1-1, 0 : u_2-1]，表示此陣列有u_1列及u_2行；每一列是由u_2個元素組成。
- 二維陣列化成一維陣列時，對映方式有二種：一種以列為主（row-major），二為以行為主（column-major）。

二維陣列位址計算–以列為主

- 以列為主
  視此陣列有u_1個元素0, 1, 2, ..., u_1-1，每一元素有u_2個單位，每個單位佔d個空間。
  其情形如圖所示：



- 由圖可知
  $A[i,j]=\alpha+i*u_2d+j*d$，其中α為此陣列第一個元素的位址

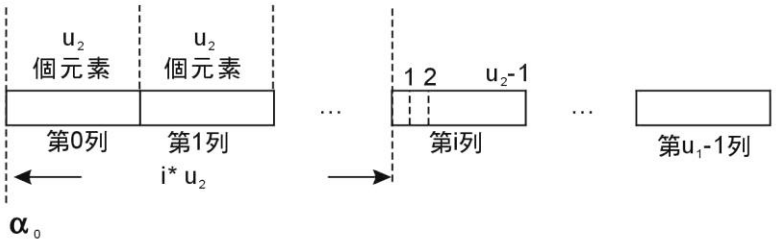


圖 2-1　以列為主的二維陣列循序表示

二維陣列位址計算–以行為主

- 以行為主：
  視此陣列有u_2個元素0, 1, 2, …, u_2，其中每一元素含有u_1個單位，每單位佔d個空間，其情形如圖所示：

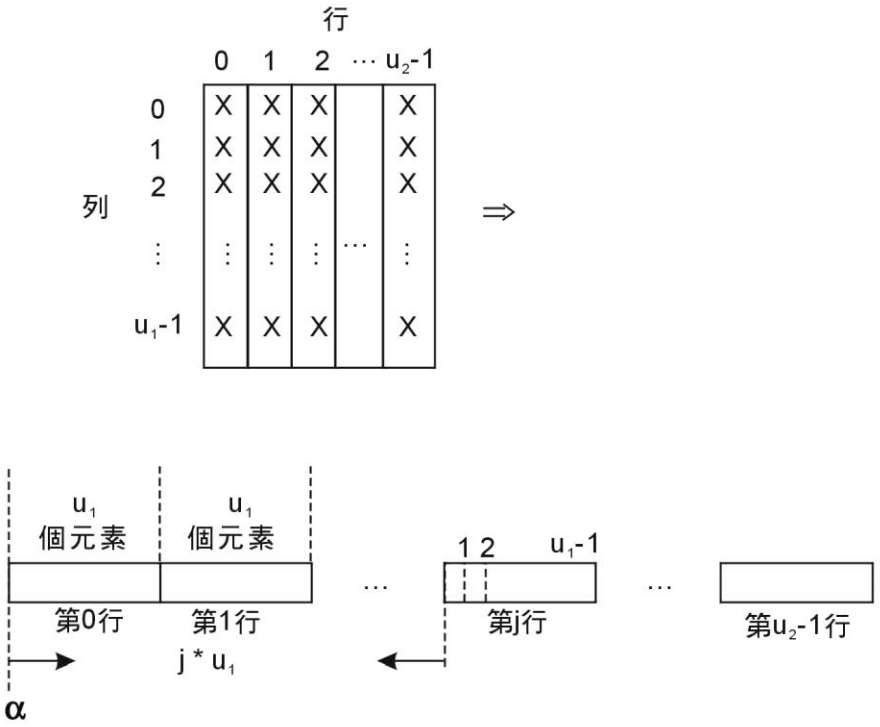
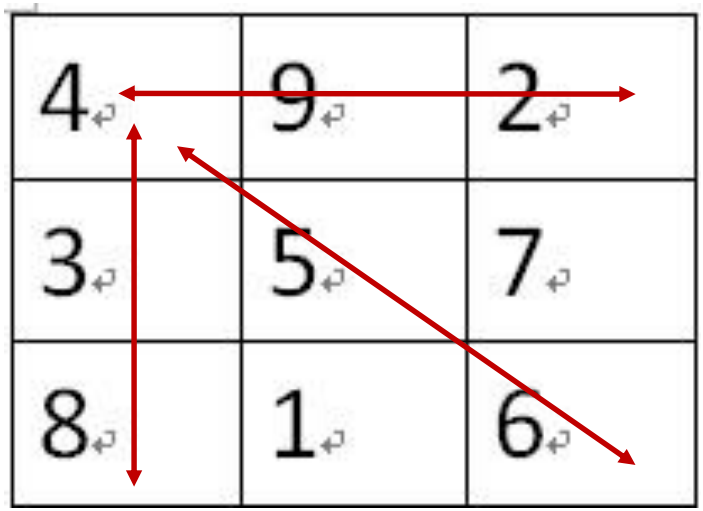


圖 2-2　以行為主的二維陣列循序表示

- 由圖可知
  $A[i,j]=\alpha+j*u_1d+i*d$

魔術方陣(1/10)

- 有一n×n的方陣，其中n為奇數?，請你將n×n的魔術方陣，將1到n^2的整數填入其中，使其各列、各行及對角線之和皆相等。



魔術方陣(2/10)

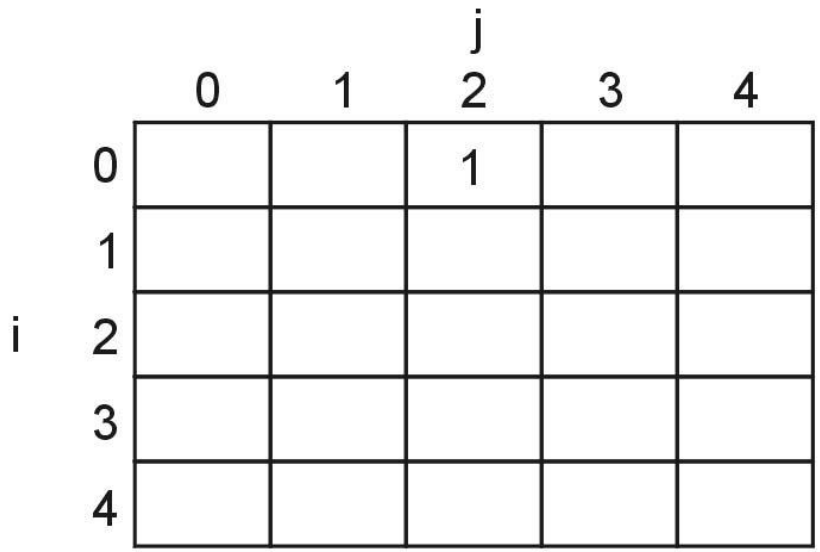
- 做法很簡單，首先將1填入最上列的中間格，然後往左上方走，(1)以1的級數增加其值，並將此值填入空格；(2)假使方格已填滿，則在原地的下一方格填上數字，並繼續做；(3)若超出方陣，則往下到最底層或往右到最右方，視兩者那一個有方格，則將數目填上此方格；(4)若兩者皆無方格，則在原地的下一方格填上數字。

魔術方陣(3/10)

- 例如有一5x5的方陣，其形成魔術方陣的步驟如下，並以上述(1)、(2)、(3)、(4)規則來說明。

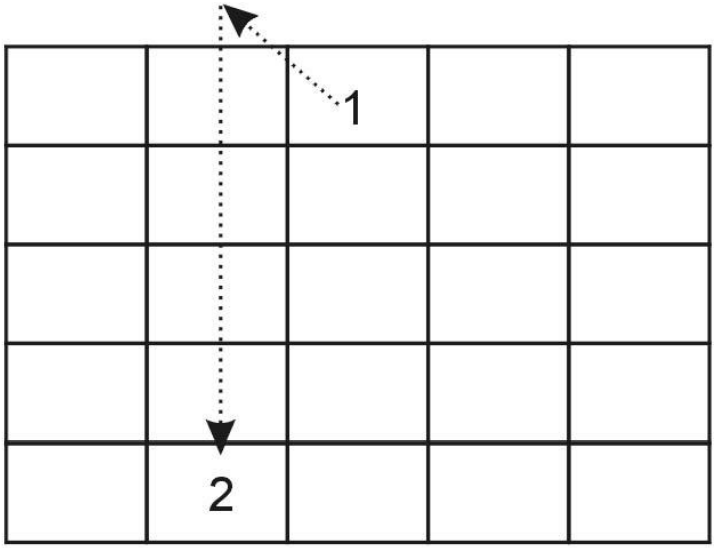
魔術方陣(4/10)

1. 將1填入此方陣最上列的中間方格，如下所示：



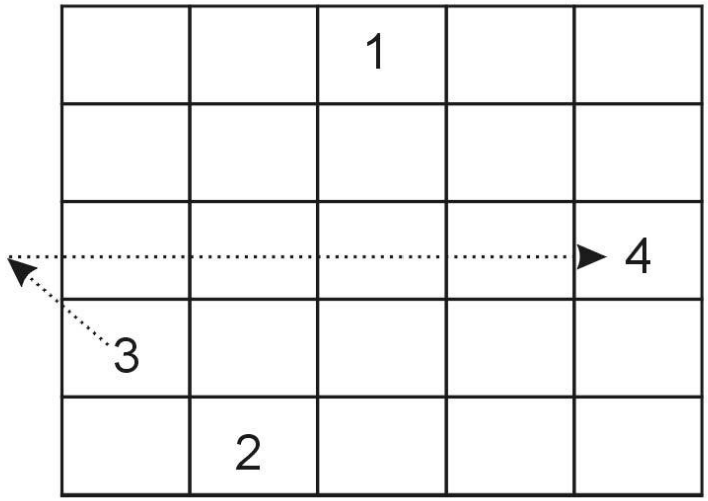
魔術方陣(5/10)

2. 承1.往左上方走，由於超出方陣，依據規格(3)發現往下的最底層有空格，因此將2填上。如下所示：



魔術方陣(6/10)

3. 承2.往左上方，依據規格(1)將3填上，然後再往左上方，此時，超出方陣，依據規則(3)將4填在最右方的方格，如下所示：



魔術方陣(7/10)

4. 承3.往左上方，依據規則(1)將5填上，再往左上方時，此時方格已有數字，依據規則(2)往5的下方填，如下所示：

		1		
			5	
			6	4
3				
	2			

魔術方陣(8/10)

5. 餘此類推，依據上述四個規格繼續填，填到15的結果如下：

15	8	1		
	14	7	5	
		13	6	4
3			12	10
9	2			11

魔術方陣(9/10)

6. 承5.此時往左上方，發現往下的最底層和往右的最右方皆無空格，依據規則(4)在原地的下方，將此數字填上，如下所示：

15	8	1		
16	14	7	5	
		13	6	4
3			12	10
9	2			11

魔術方陣(10/10)

7. 繼續往下填，並依據規則(1)、(2)、(3)、(4)最後的結果如下：

15	8	1	24	17
16	14	7	5	23
22	20	13	6	4
3	21	19	12	10
9	2	25	18	11

- 此時可以算算各行、各列及對角線之和是否皆相等，答案是肯定的，其和皆為65。

問題思考

- 魔術方陣的規則是往"左上方"走，請問往右上方走是否也可以?

陣列的優缺點

- 優點
  - 快速的隨機訪問：由於元素在記憶體中是連續存放的，因此可以通過索引來迅速存取任何元素，時間複雜度是O(1)。
  - 簡單的結構：陣列的結構比較簡單，易於實現和理解。
- 缺點
  - 固定大小：一旦創建，陣列的大小不能改變。如果預先設置的大小不夠大，就無法儲存更多的元素；如果設置過大，則可能浪費空間。
  - 插入與刪除的低效：在陣列中插入或刪除元素需要移動其他元素，最壞情況下的時間複雜度是O(n)，其中n是陣列的長度。
  - 記憶體分配：陣列必須在記憶體中有足夠的連續空間來存儲所有元素，如果可用的連續記憶體不足，則會有問題。

本章完結