第二章

陣列

Array

本章內容

2-1 陣列是什麼

2-2 一維陣列的運算：用程式處理一維陣列

2-3 二維陣列(矩陣)的儲存與運算

2-4 字串—有結束符號的字元陣列

2-5 陣列元素位址的計算

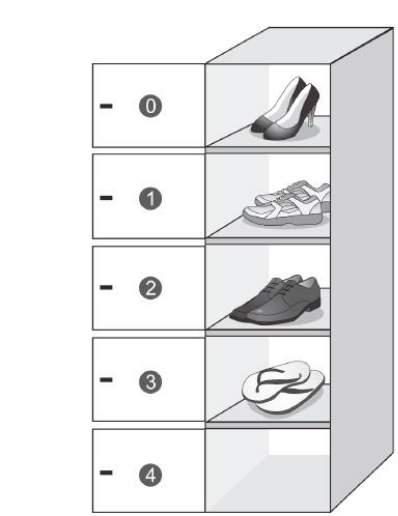
英雄、魔獸、怪物、武器、戰略、寶物、…，遊戲程式設計者是如何將同類元素放在一起, 在適當時機列出來讓玩家瀏覽選擇的？

答案是使用「陣列」(Array)。遊戲程式設計者將英雄等同類元素放在陣列中, 不只方便好管理, 更可以在遊戲程式進行中很有效率地對這些英雄進行動作, 例如改變其中某一個英雄的位置、或是列出所有英雄的生命值等。

2-1 陣列是什麼

- 「陣列」(Array) 是由多個相同型別元素所組成的串列，這些元素共用一個名稱來表示，並且加上編號（或稱註標，index）以區別個別的元素。

- 我們可以把陣列想像成一排有編號的格子，這些格子都存放相同類型的物品，例如鞋子。我們可以隨意打開某一個編號的格子，以查看裡面的內容；我們也可以把鞋子擺在某一個喜歡的鞋格內，這就是所謂的「隨機存取」(random access)。



list[0]	19
list[1]	21
list[2]	5
list[3]	115
list[4]	28

陣列的維度

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：

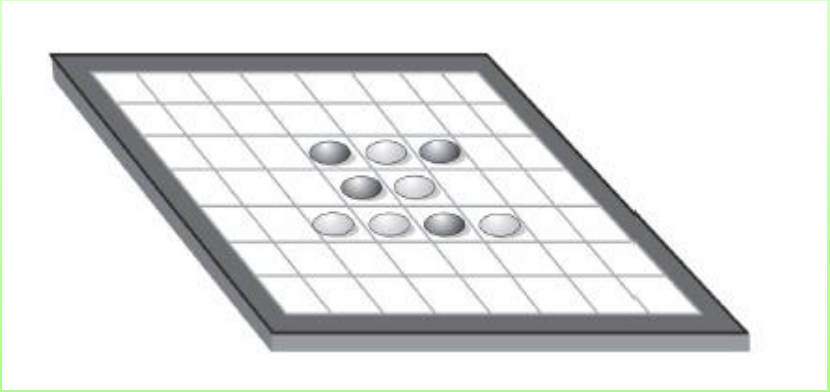
```
int  list[4][6][5];
```

二維陣列

int list[6][5]

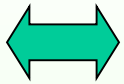
此二維陣列的所有元素在邏輯上可以排成一個有6列（row，橫者為列）、5行（column，直者為行）的平面格子

	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]

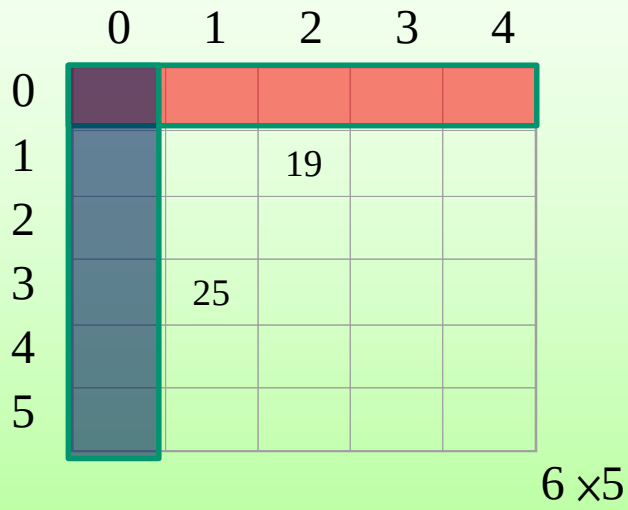


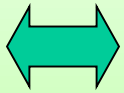
二維陣列

二維陣列在邏輯上表示為**平面**



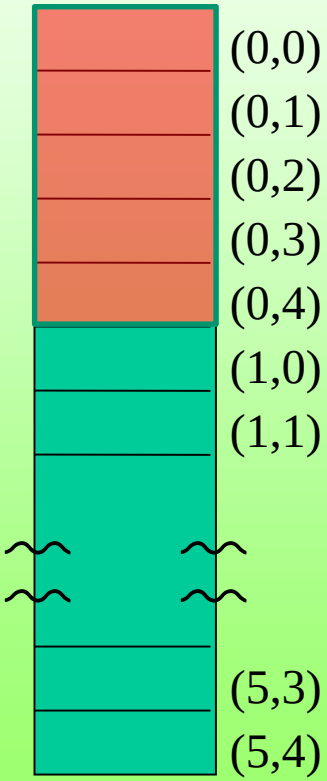
實體排列順序是**線性**的





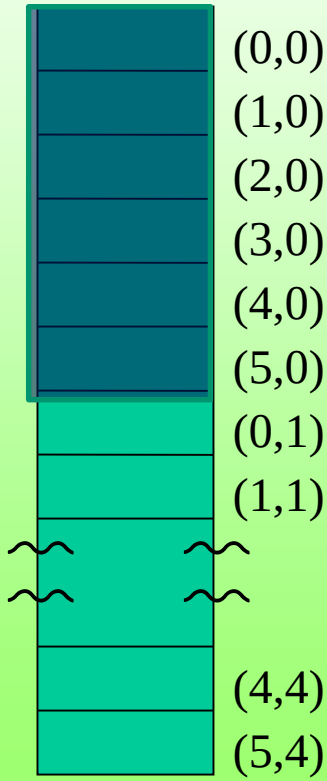
以列為主

(row major)



以行為主

(column major)

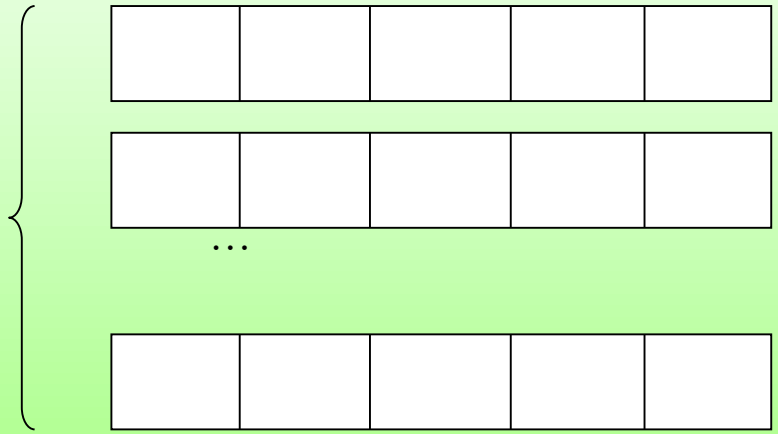


第1列第2行：List[1][2]值是 19

第3列第1行：List[3][1]值是 25

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，
因為6×5的平面是由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]



2-2 一維陣列的運算：用程式處理一維陣列

以下是陣列的六種基本運算 (operation) ：

1. 檢索（讀出）編號為 *i* 的元素內含值

   value = list[4] ;

2. 將新值 寫入 編號為 *i* 的元素位置

   list[4] = NewValue ;

3. 將一陣列的所有元素 輸出 到螢幕

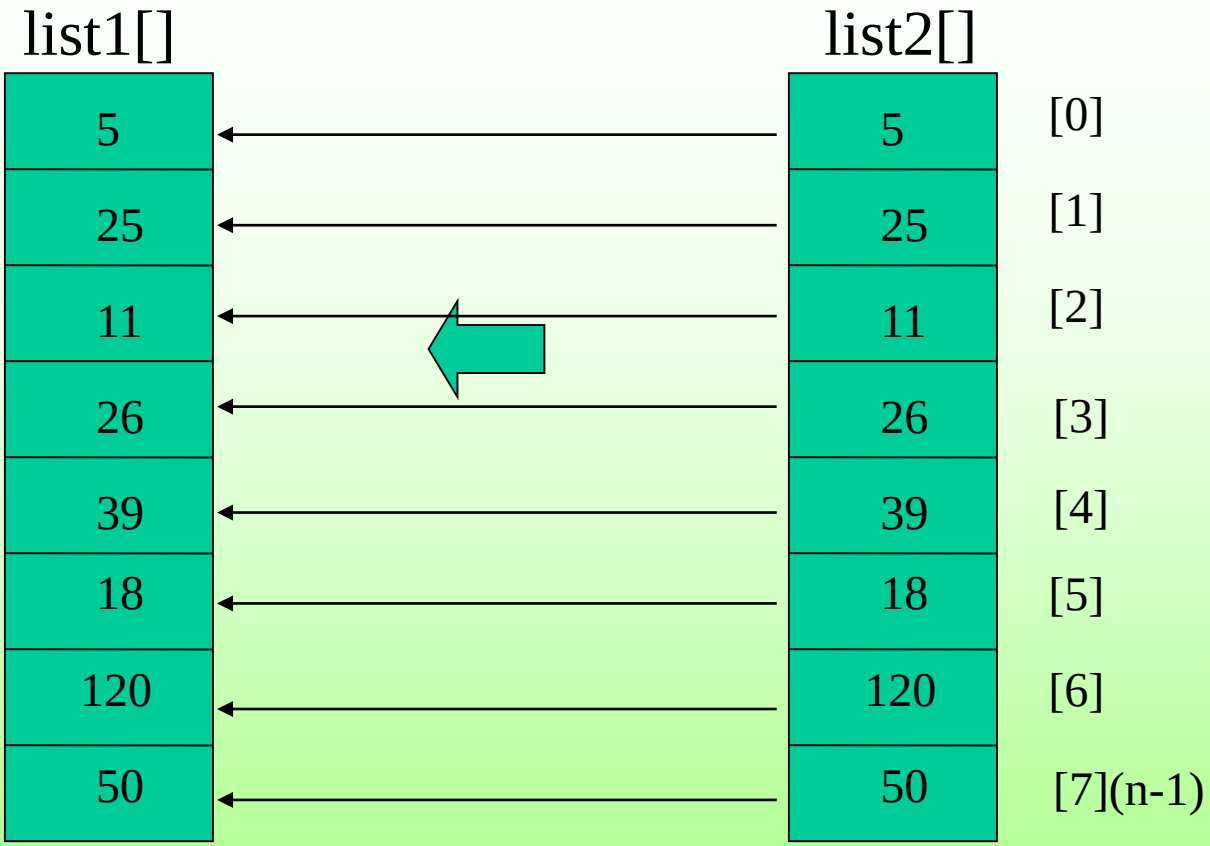
4. 將一陣列的所有元素 複製 到另一陣列（陣列的複製）

5. 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。如果陣列編號從 0 到 *n*-1，則 *n*-1 號元素可能將喪失

6. 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。第 *n*-1 號元素變成無意義（或填入固定值）

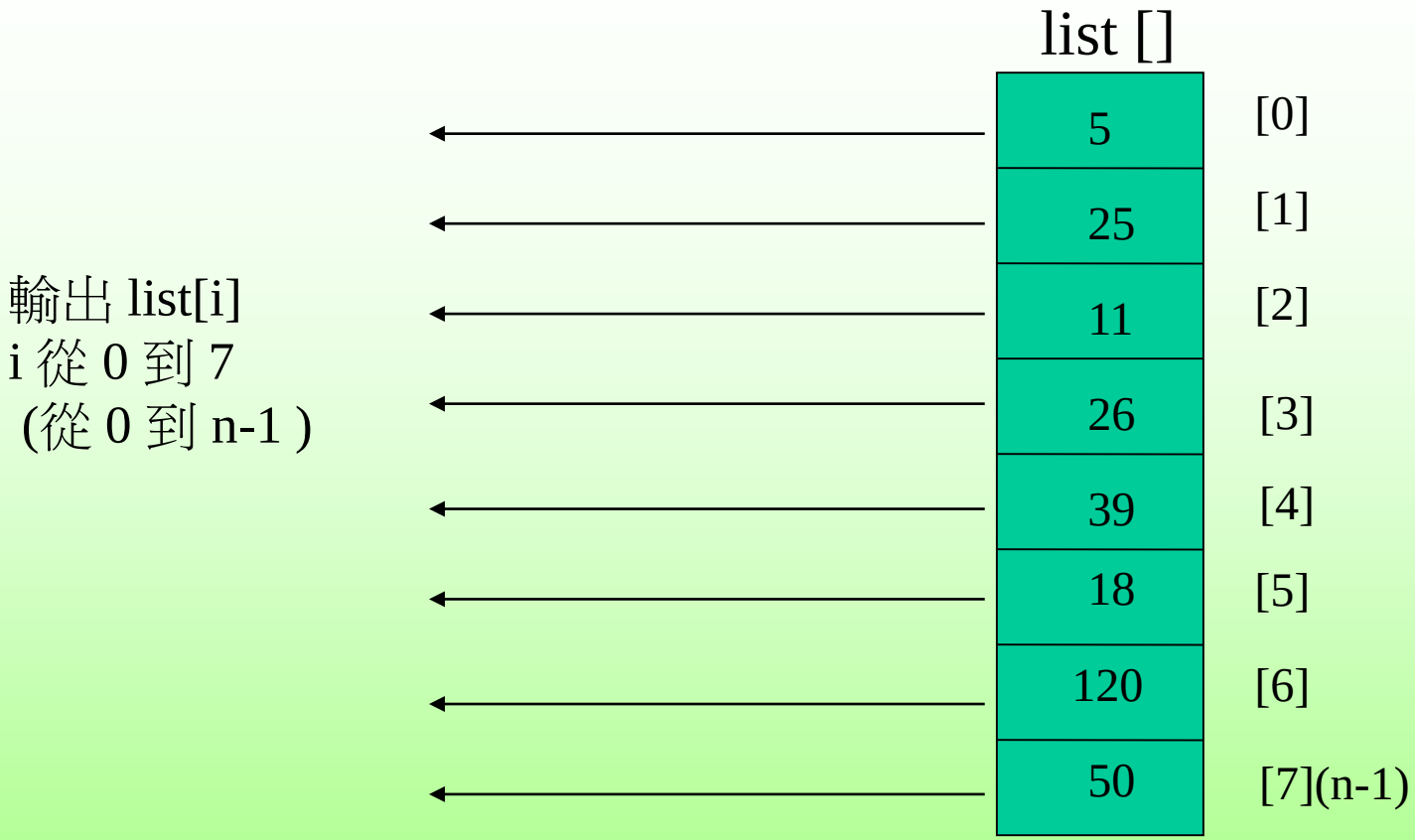
陣列的複製

list1[i] = list2[i]
i 從 0 到 7
(從 0 到 n-1)



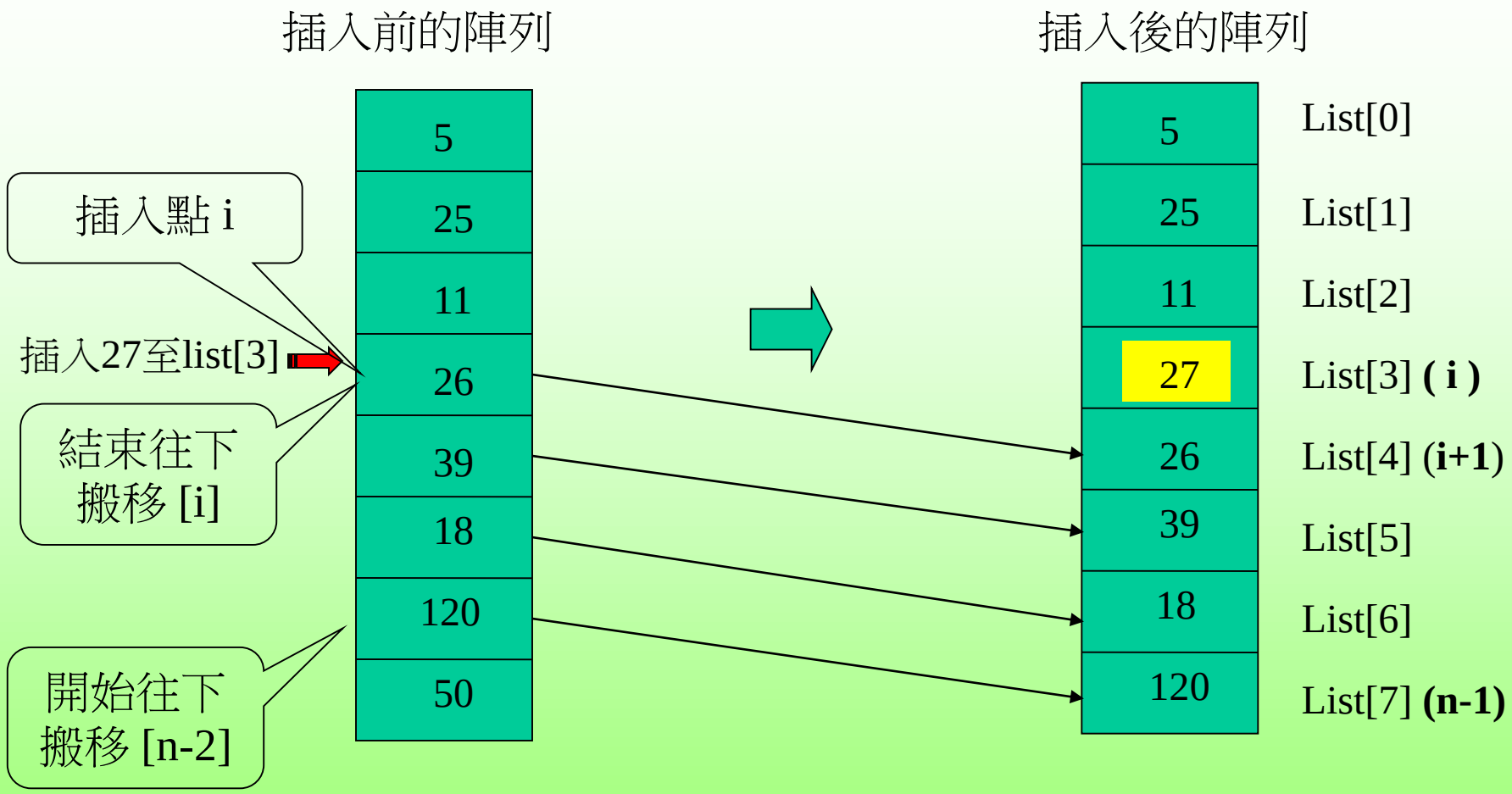

```
1.  void  ArrayCopy( int list1[], int list2[], int n)
2. {        int  i;
3.          for (i = 0 ; i < n ; i++)
4.               list1[i ] = list2[i] ;
5. }
```

陣列的輸出



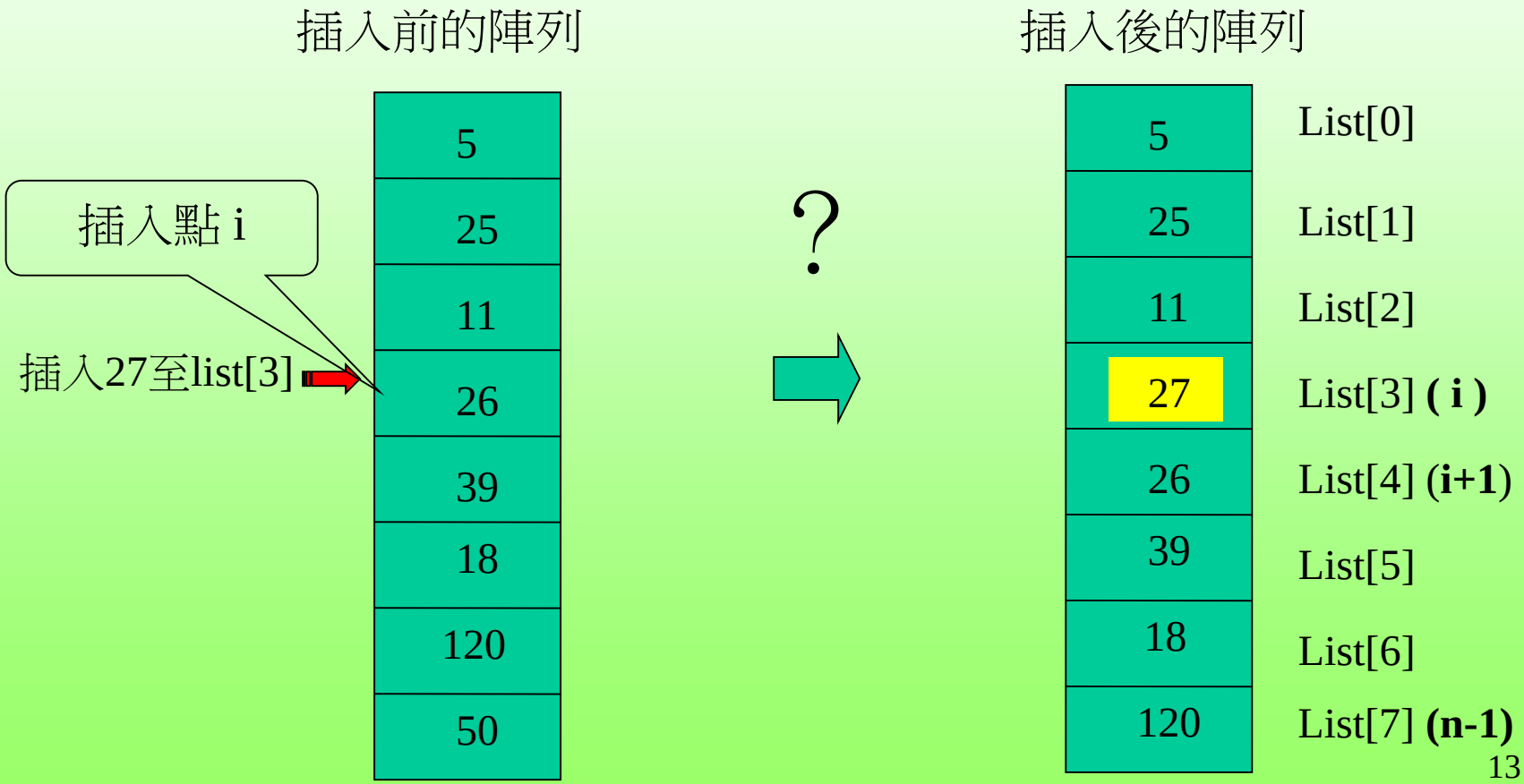

```
1.  void PrintArray( int list [], int n)
2.  {       int  i;
3.          for (i = 0 ; i < n ; i++)
4.                  cout << list[i] << endl ;
            //C: printf("%d\n", list[i]);
5.  }
```

陣列元素的插入

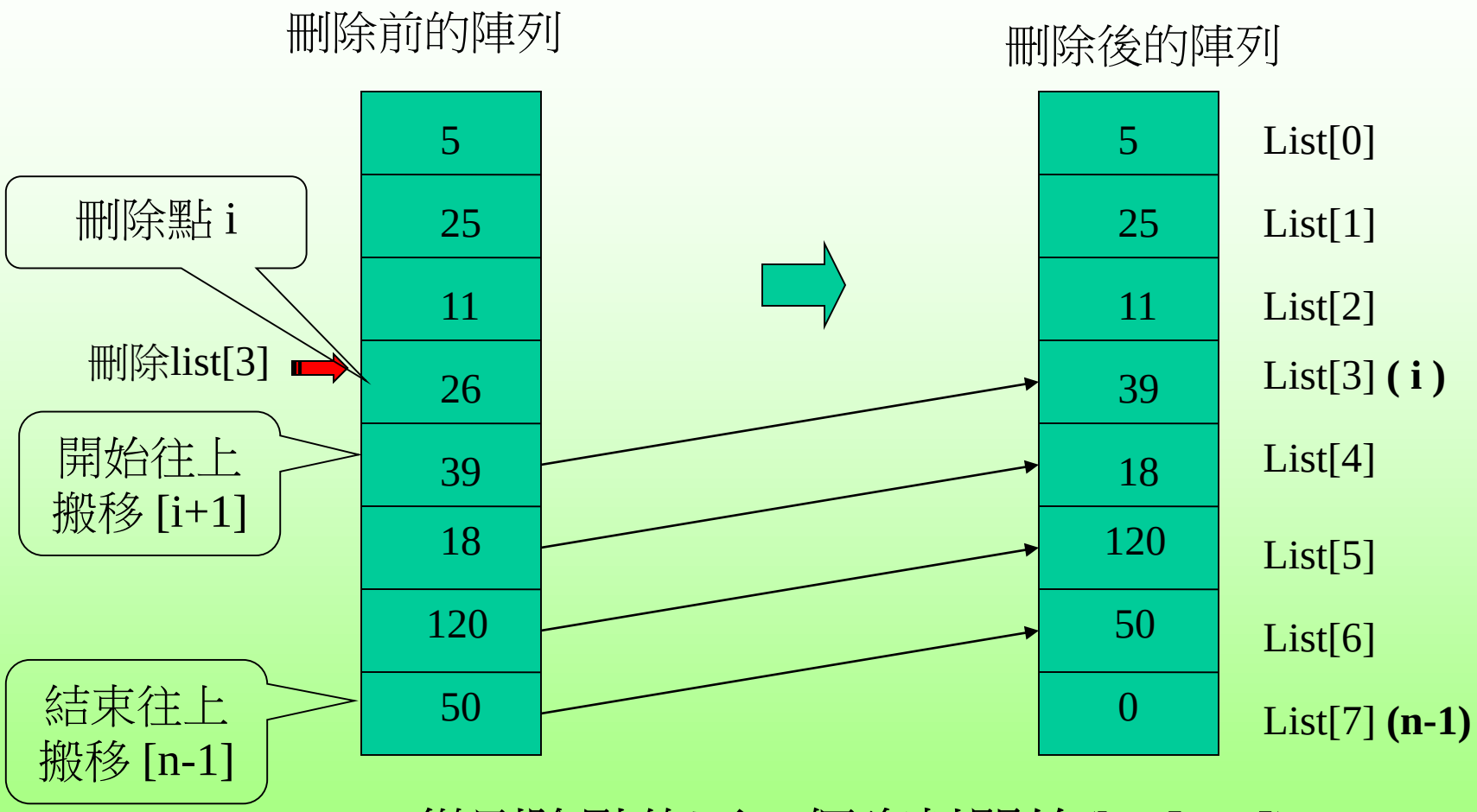


從倒數第二個資料開始(list[n-2])，
到插入點為止(list[i])
每個元素往下移一個位置

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

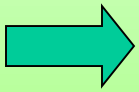
2-3 二維陣列(矩陣)的儲存與運算

二維陣列可以用來儲存數學的矩陣及行列式，因此可以用來解決諸如“解連立方程組” 等線性代數問題

數學習慣表示法

4行

3列 $\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \end{pmatrix}$ 3×4

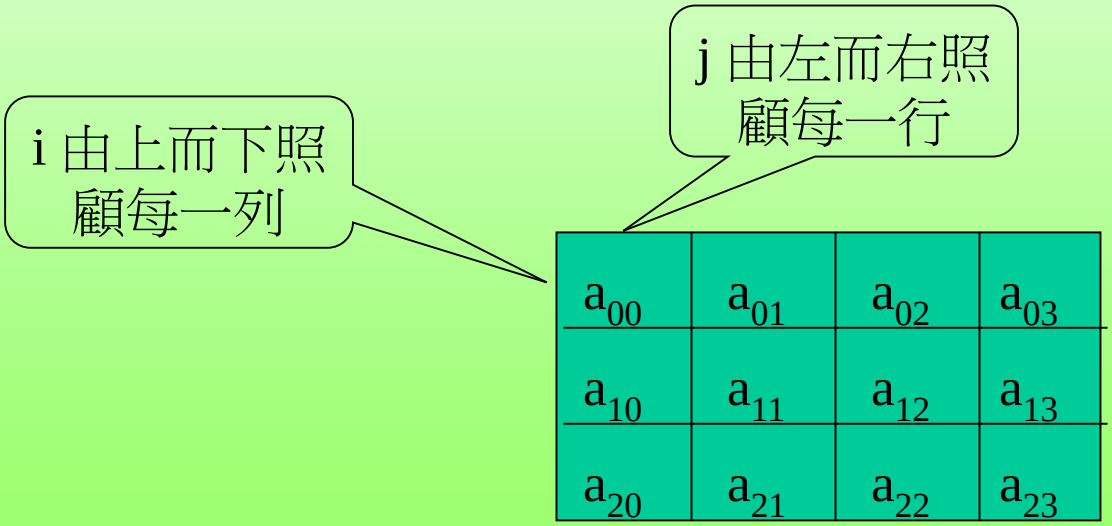


資料結構示意圖

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}

矩陣的輸出

```
for (i = 0 ; i < m ; i++)          //照顧矩陣每一列
{       for (j = 0 ; j < n ; j++) //照顧矩陣第i列的每一個元素
                cout << A[i][j] << “  ” ;
                //C: printf(“%d  ”, A[i][j]);
        cout << endl ; //C: printf(“\n”); //印完一列要換行
 }
```



矩陣的相加

矩陣相加：對應的元素相加

特徵：C[i][j] = A[i][j] + B[i][j]

$$\begin{pmatrix} a_{00} & a_{01} & \cdots & a_{0(n-1)} \\ a_{10} & a_{11} & \cdots & a_{1(n-1)} \\ \vdots & \vdots & & \\ a_{(m-1)0} & \cdots & & a_{(m-1)(n-1)} \end{pmatrix}_{m \times n} + \begin{pmatrix} b_{00} & b_{01} & \cdots & b_{0(n-1)} \\ b_{10} & b_{11} & \cdots & b_{1(n-1)} \\ \vdots & \vdots & & \\ b_{(m-1)0} & \cdots & & b_{(m-1)(n-1)} \end{pmatrix}_{m \times n}$$

$$= \begin{pmatrix} a_{00}+b_{00} & a_{01}+b_{01} & \cdots & a_{0(n-1)}+b_{0(n-1)} \\ a_{10}+b_{10} & a_{11}+b1_{01} & \cdots & a_{1(n-1)}+b_{1(n-1)} \\ \vdots & \vdots & & \\ a_{(m-1)0}+b_{(m-1)0} & a_{(m-1)1}+b_{(m-1)1} & \cdots & a_{(m-1)(n-1)}+b_{(m-1)(n-1)} \end{pmatrix}_{m \times n}$$

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

4×3 +

0	1	2
3	4	5
6	7	8
9	10	11

4×3 =

1	3	5
7	9	11
13	15	17
19	21	23

4×3

```
for( i=0 ; i < rows ; i++ )
        for( j=0 ; j < cols; j++ )
                C[i][j] = A[i][j] + B[i][j] ;
```

矩陣的相乘

$$A_{m\times n} \times B_{n\times t} = C_{m\times t}$$

C_{00} 是 A 矩陣的第 0 列向量乘上 B 矩陣的第 0 行向量

$$c_{00} = [a_{00}\ a_{01}\ \cdots\ a_{0(n-1)}] \bullet \begin{pmatrix} b_{00} \\ b_{10} \\ \cdots \\ b_{(n-1)0} \end{pmatrix}^t$$

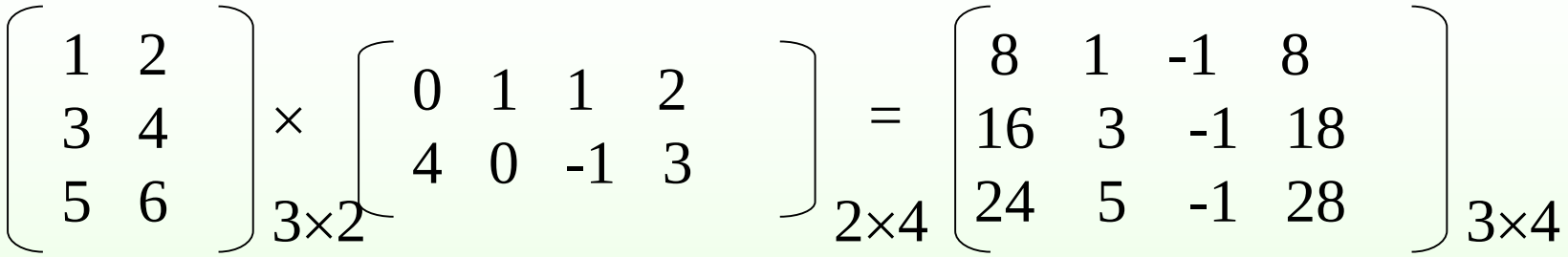
$$= a_{00}\times b_{00} + a_{01}\times b_{10} + \cdots + a_{0(n-1)}\times b_{(n-1)0} \quad (\text{n個乘法})$$

$$= \sum_{k=0}^{n-1} a_{0k}\times b_{k0}$$

C_{ij} 是 A 矩陣的第 i 列向量乘上 B 矩陣的第 j 行向量

特徵：

$$c_{ij} = \sum_{k=0}^{n-1} a_{ik}\times b_{kj}$$



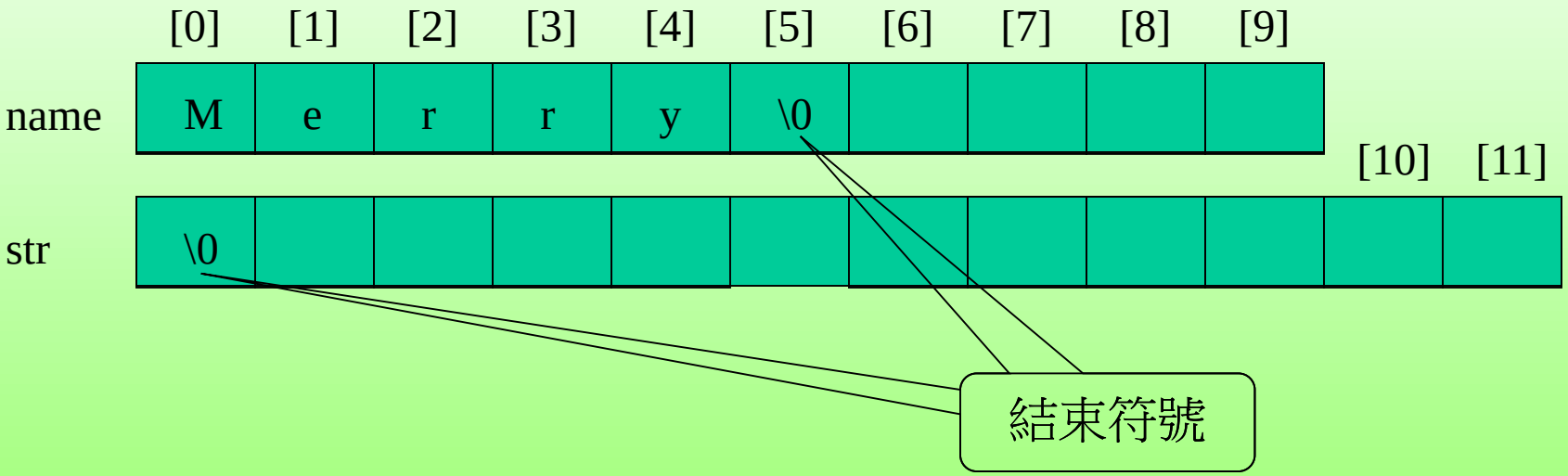
矩陣相乘：C中每個元素是兩個向量相乘(一串乘法的相加)

```
for( i = 0 ; i < m ; i++ )
    for( j = 0 ; j < t ; j++ )
    {     C[i][j] = 0 ;
          for( k = 0 ; k < n ; k++ )
               C[i][j] = C[i][j] + A[i][k] * B[k][j] ;
    }
```

2-4 字串

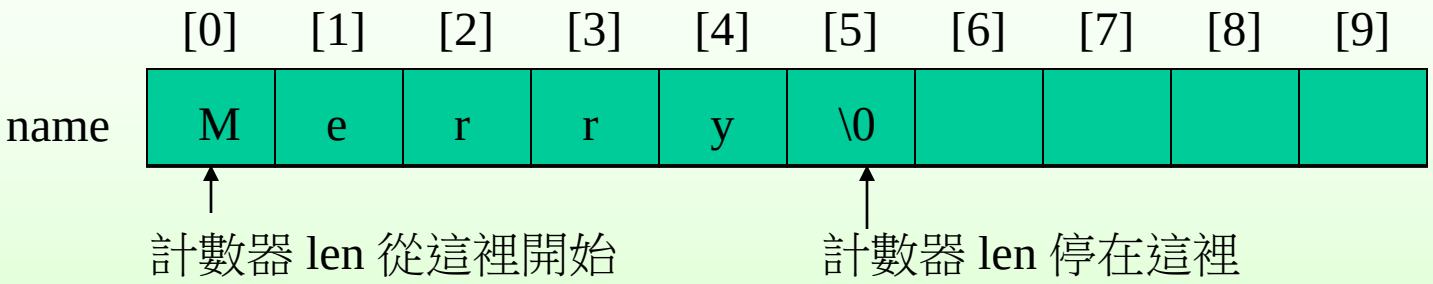
◎ 在 C/C++ 語言中，字串就是有結束符號的字元陣列

```
char    name[10] = { “Merry” } ;
char    str [12] ;
```



◎ 字串的結束符號是 ‘\0’ (空字元)，就是 8 個 bit 都是 0 的 byte

◎ 字串的長度計算 strlen()

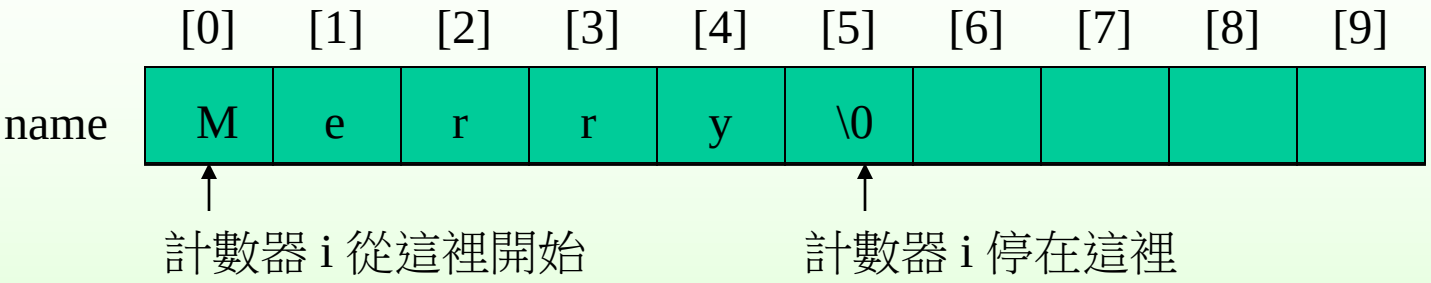


呼叫 strlen(name) 將會得到 5

```
int         strlen( char S[] )
{           int         len ;
            for ( len = 0 ; S[len] != '\0' ;  len++) ;
            return ( len ) ;
}
```

技巧是在迴圈中不斷將計數器(len) 的值加 1，
直到碰到空字元時就結束

◎ 字串的複製 strcpy()

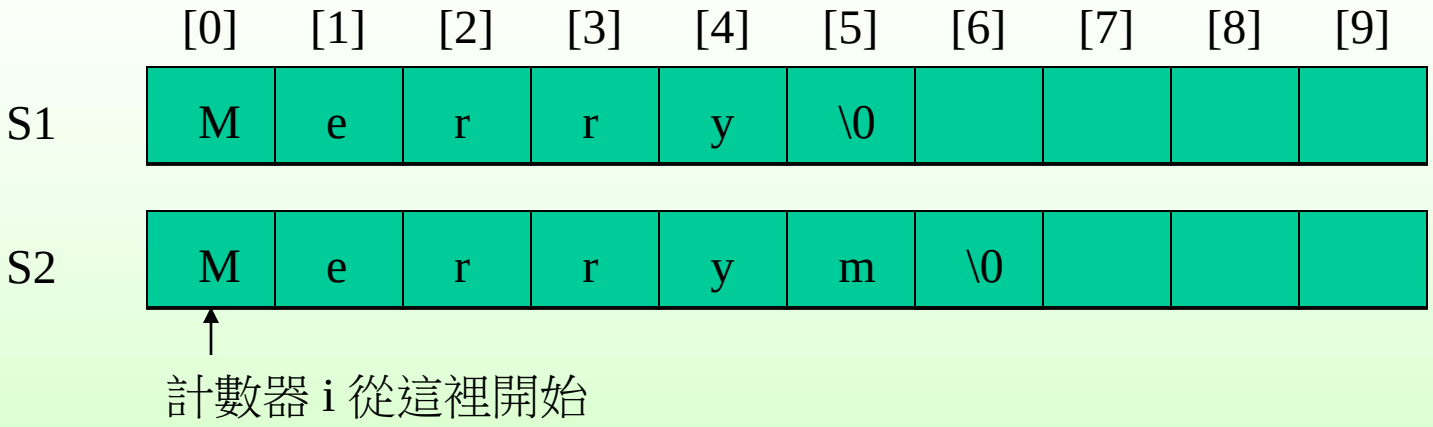


呼叫 strcpy(S, name) 會將字串name的內容複製到字串 S

```
int     strcpy( char S1[], char S2[])
{       int     i ;
        for ( i = 0; S2[i] != '\0'; i++)
                S1[i] = S2[i] ;
        S1[i] = '\0' ;
}
```

技巧是不斷做逐字元拷貝，直到碰到空字元就停止

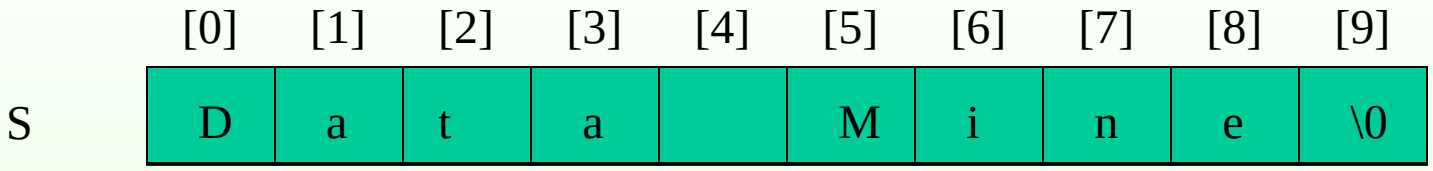
◎ 字串的比較 strcmp



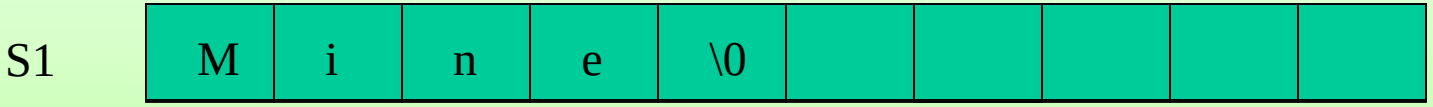
呼叫 strcmp(S1,S2) 會將S1和S2從頭一個字元開始比起，直到分出大小、或兩者比完後完全相等為止。

```
int     strcmp( char S1[], char S2[])
{       int       i ;
        for ( i = 0; S1[i] ==S2[i]; i++)
            if (S1[i]=='\0' && S2[i] =='\0')
        return (0);
        if ( S1[i] > S2[i])    return (–1) ;
        else                   return (1) ;
}
```

◎ 子字串的截取 substr



呼叫 substr(S1,S,5,4) 會從 S 字串的第5個字元開始截取4個字，並且放入S1字串中。

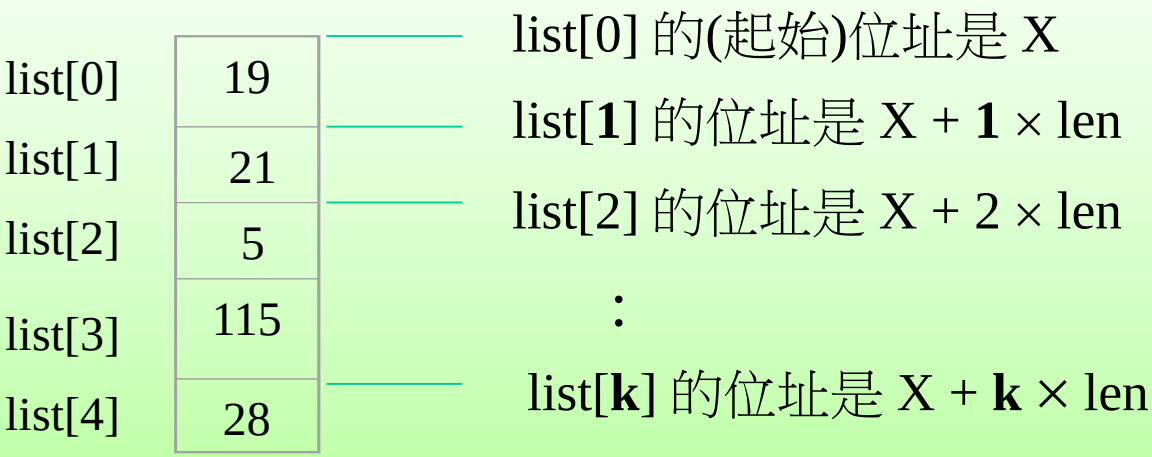



```
void    substr( char S1[], char S[], int i, int n )
{       int       len, end, j ,k ;
        len = strlen( S ) ;  end = i + n ;
        if ( (end>len) || ( n==0))     end = len ;
        for ( j = i, k=0 ; j < end ; j++, k++)
                  S1[k]= S [j];
        S1[k] = '\0' ;
}
```

2-5 陣列元素位址的計算

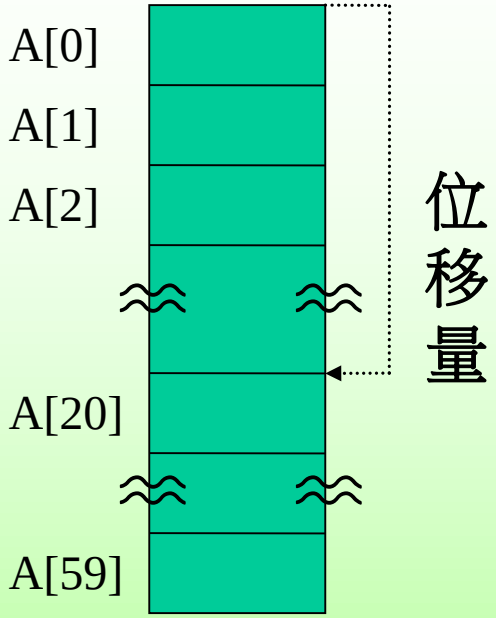
宣告　　int list[5];

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）？

(2) 若 A[0]在記憶體中的位址為 400, 則元素 A[20] 的位址為何？

(3) 若 A[10]在記憶體中的位址為 400, 則元素 A[39] 的位址為何？

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

(1) A 陣列共佔 $60 \times 4 = 240$ (bytes)

(2) A[20] 的位址 ＝ A[0] 的位址 + 位移量
＝ $400 + (20 - 0) \times 4$
＝ 480

(3) A[39] 的位址 ＝ A[10] 的位址 + 位移量
＝ $400 + (39 - 10) \times 4$
＝ 516

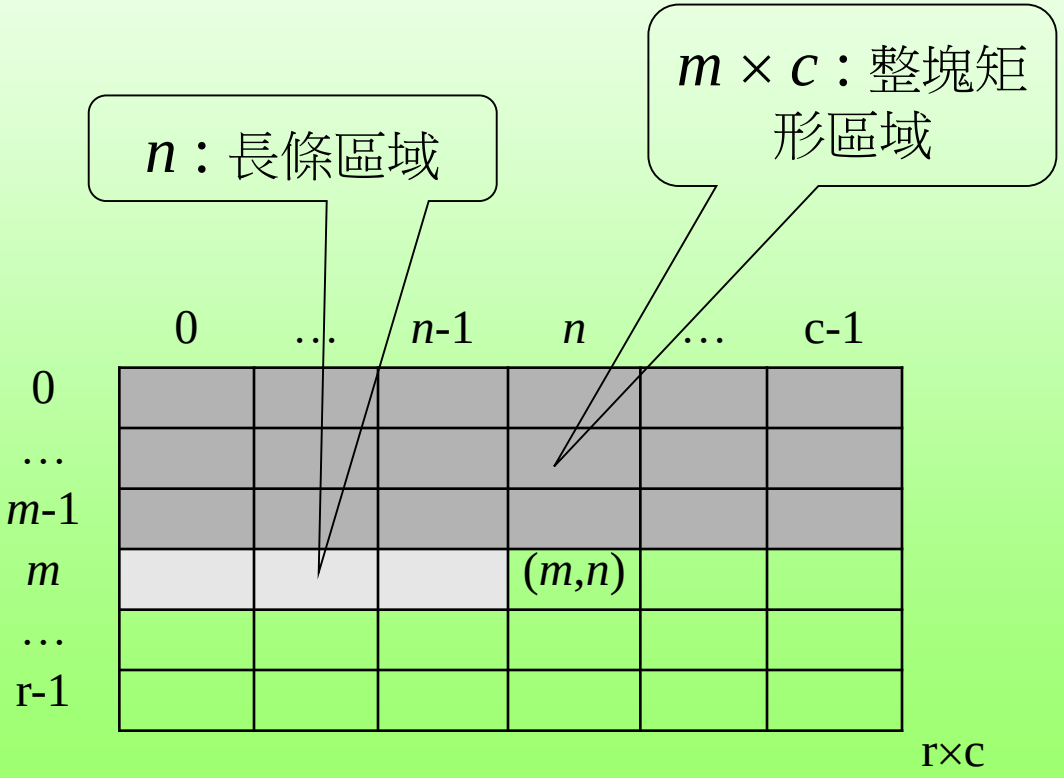
(4) A[9] 的位址 ＝ A[30] 的位址 + 位移量
＝ $400 + (9 - 30) \times 4$
＝ 316

如果每個元素大小為 len 個 bytes，而且 list[0][0] 的位址為 X，則：

◎ list[0][3] 的位址是 X + (3 – 0) × len ＝X + 3 × len

◎ list[4][3] 的位址是 X + ((4 - 0) × 5 +(3 - 0)) × len ＝ X + (4 × 5 +3) × len

因為 list[4][3] 之前有 4 整列（每列有5個元素），加 3 個元素



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）？

(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？

答：

(1) 陣列 A 共佔 $8 \times 10 \times 4 = 320$ (bytes)

(2) A[0][9] 的位址 = A[0][0] 的位址 + 位移量

$= 100 + 9 \times 4$

$= 136$

在 A[0][9] 之前有 9 個元素 (A[0][0] ~ A[0][8])

(3) A[6][5] 的位址 = A[0][0] 的位址 + 位移量

$= 100 + (6 \times 10 + 5) \times 4$

$= 360$

在 A[6][5] 之前有 6 整列加 5 個元素 $= 6 \times 10 + 5 = 65$ 個元素