

# 陣列

## 2.1 陣列表示法

在還沒有談到陣列(array)之前，讓我們先來看看線性串列(linear list)。線性串列又稱循序串列(sequential list)或有序串列(ordered list)，其特性乃是每一項資料是依據它在串列的位置，所形成的一個線性排列次序，所以  $x[i]$  會出現在  $x[i+1]$  之前。

線性串列經常發生的操作如下：

1. 取出串列中的第  $i$  項； $0 \leq i \leq n-1$ 。
2. 計算串列的長度。
3. 由左至右或由右至左讀此串列。
4. 在第  $i$  項加入一個新值，使其原來的第  $i, i+1, \dots, n$  項變為第  $i+1, i+2, \dots, n+1$  項。就是在  $i$  之後的資料都要退後一個位址。
5. 刪除第  $i$  項，使其原來的第  $i+1, i+2, \dots, n$  項變為第  $i, i+1, \dots, n-1$  項。就是在  $i$  之後的資料都會往前一個位址。

在 C# 程式語言中常利用陣列設置線性串列，以線性的對應方式將元素  $a_i$  置於陣列的第  $i$  個位置上，若要讀取  $a_i$  時，可利用  $a_i = a_0 + i*d$  來求得。其中  $a_i$  為相對位址， $a_0$  為陣列的起始位址， $d$  為每一個元素所佔的空間大小，但要注意的是 C# 的陣列是從 0 開始的喔！

以下所介紹的是陣列的表示方法：

## 2.1.1 一維陣列

假設一維陣列(one dimension array)是  $A(0 : u - 1)$ ，且每一個元素佔  $d$  個空間，則  $A(i) = l_0 + i*d$ ，其中  $l_0$  是陣列的起始位置。若每一元素，所佔的空間為  $d$ ，且起始的元素為 0，則陣列  $A$  的每一元素所對應的位址表示如下(假設  $d=1$ )：

$$A(i) = l_0 + i*d$$

陣列元素：	$A(0)$	$A(1)$	$A(2)$	...	$A(i)$	...	$A(u-1)$
位 址：	$l_0$	$l_0+1$	$l_0+2$	...	$l_0+(i)$	...	$l_0+(u-1)$

若陣列是  $A(t : u)$ ，則  $A(i) = l_0 + (i-t)*d$ 。

如陣列  $A(2 : 12)$ ，則  $A(6)$ 與  $A(2)$ 起始點相差 4 個單位(6-2)，相當於上述  $(i-t)$ 。

若陣列為  $A(1 : u)$ ，表示陣列的起始元素位置從 1 開始，則  $A(i) = l_0 + (i-1)d$ ，其中  $d$  為每一元素所佔的空間大小。因此，我們必需注意陣列起始元素的位址。

### 範例》

1. 有一陣列  $A(0 : 100)$ ，而起始位址  $A(0) = 100$ ， $d = 2$ ，則  $A(16) = ?$

**解** 由於  $A(i) = 100 + (i)*d$ ； $\therefore A(16) = 100 + 16*2 = 132$

假若陣列為  $(t : u)$ ，則  $A(i) = l_0 + (i-t)*d$

2. 如有一陣列  $A(-3 : 10)$ 且  $A(-3) = 100$ ， $d=1$ ，求  $A(5) = ?$

**解**  $A(5) = 100 + (5-(-3)) * 1$

$$= 108$$

## 2.1.2 二維陣列

假若有一二維陣列(two dimension array)是  $A[0 : u_1-1, 0 : u_2-1]$ ，表示此陣列有  $u_1$  列及  $u_2$  行；也就是每一列是由  $u_2$  個元素所組成。二維陣列化成一維陣列時，對應方式有二種：(1)以列為主(row - major)，(2)以行為主(column - major)。

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

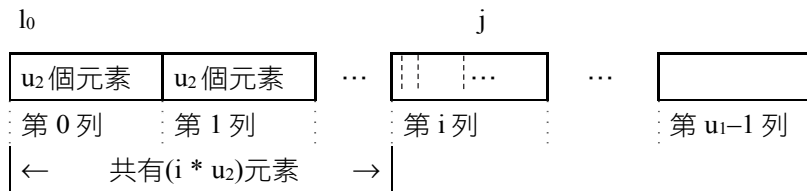
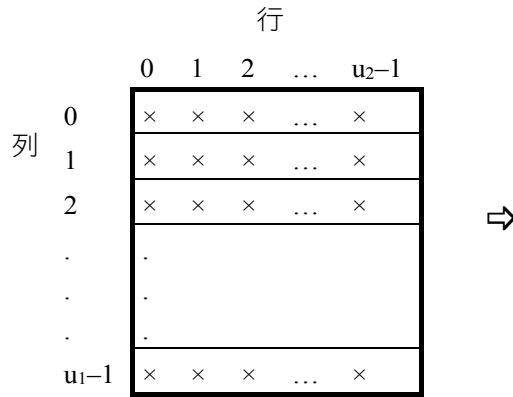


圖 2.1 以列為主的二維陣列循序表示

由上圖可知  $A(i, j) = l_0 + i * u_2 d + j * d$

2. 以行為主：視此陣列有  $u_2$  個元素  $1, 2, \dots, u_2$ ，每一元素有  $u_1$  個單位，每個單位佔  $d$  個空間。其情形如下圖所示：

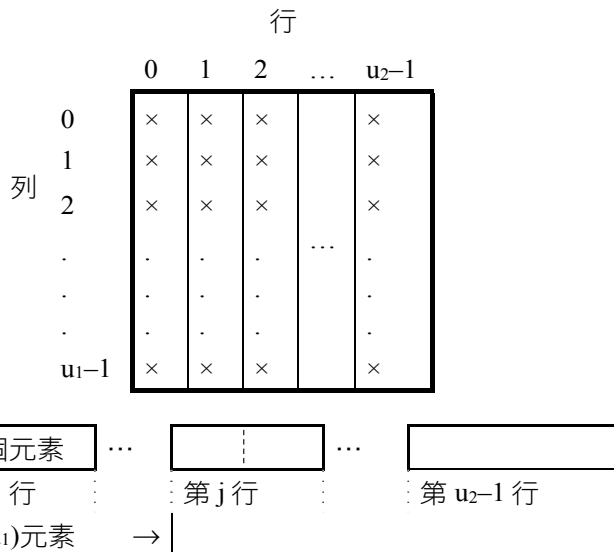


圖 2.2 以行為主的二維陣列循序表示

由上圖可知  $A(i, j) = l_0 + j * u_1 d + i * d$

假若陣列是  $A(l_1 : u_1, l_2 : u_2)$ ，則此陣列共有  $m = u_1 - l_1 + 1$  列， $n = u_2 - l_2 + 1$  行。  
計算  $A(i, j)$  的地址如下：

1. 以列為主

$$A(i, j) = l_0 + (i - l_1) * nd + (j - l_2)d$$

2. 以行為主

$$A(i, j) = l_0 + (j - l_2) * md + (i - l_1)d$$

### 範例》

假設  $A(-3 : 5, -4 : 2)$  且其起始位置為  $A(-3, -4) = 100$ ，以列為主排列，請問  $A(1, 1)$  所在的位址？( $d=1$ )

**解**  $m = 5 - (-3) + 1 = 9, n = 2 - (-4) + 1 = 7, l_1 = -3, l_2 = -4, i = 1, j = 1$

$$A(i, j) = l_0 + (i - l_1) * nd + (j - l_2)d$$

$$\begin{aligned} A(1, 1) &= 100 + (1 - (-3)) * 7 * 1 + (1 - (-4)) * 1 \\ &= 100 + 4 * 7 + 5 \\ &= 133 \end{aligned}$$

另一解法是將其化為標準式

$$A(-3 : 5, -4 : 2) \rightarrow A(0 : 8, 0 : 6), \text{ 得知 } u_1=9, u_2=7$$

$$A(-3, -4) \rightarrow A(0, 0), \text{ 得知 } A(1, 1) \rightarrow A(4, 5)$$

$$\therefore A(4, 5) = 100 + 4 * 7 + 5 = 100 + 28 + 5 = 133$$

## 2.1.3 三維陣列

假若有一三維陣列(three dimension array)是  $A(0 : u_1 - 1, 0 : u_2 - 1, 0 : u_3 - 1)$ ，如圖 2.3 所示：

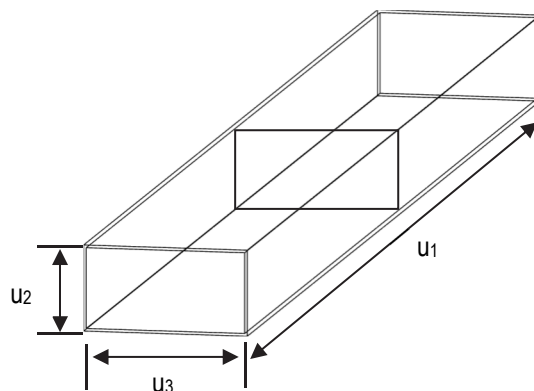
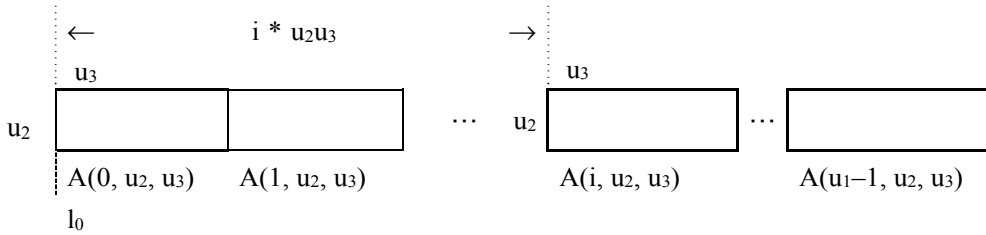


圖 2.3 三維陣列以  $u_1$  個二維陣列來表示

一般三維陣列皆先化為二維陣列後，再對應到一維陣列，對應方式也有二種：(1)以列為主，(2)以行為主。

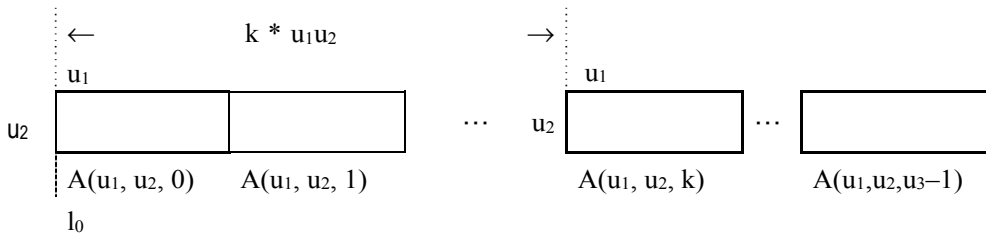
### 1. 以列為主

視此陣列有  $u_1$  個  $u_2 * u_3$  的二維陣列，每一個二維陣列有  $u_2$  個元素，每個  $u_2$  皆有  $u_3d$  個空間。



$$A(i, j, k) = l_0 + i * u_2 u_3 d + j * u_3 d + k * d$$

### 2. 以行為主



$$A(i, j, k) = l_0 + k * u_1 u_2 d + j * u_1 d + i * d$$

假設陣列為  $A(l_1 : u_1, l_2 : u_2, l_3 : u_3)$ ，則  $p = u_1 - l_1 + 1$ ， $q = u_2 - l_2 + 1$ ，

$r = u_3 - l_3 + 1$ 。

以列為主的公式為

$$A(i, j, k) = l_0 + (i - l_1) * qrd + (j - l_2) * rd + (k - l_3) d$$

以行為主的公式為

$$A(i, j, k) = l_0 + (k - l_3) * pqd + (j - l_2) * pd + (i - l_1) d$$

### 範例》

假設有一三維陣列  $A(-3 : 5, -4 : 2, 1 : 5)$  且其起始位置為  $A(-3, -4, 1) = 100$ ，以列為主排列，試求  $A(1, 1, 3)$  所在的位址？( $d=1$ )

**解**  $p = 5 - (-3) + 1 = 9$ ， $q = 2 - (-4) + 1 = 7$ ， $r = 5 - 1 + 1 = 5$ ， $l_1 = -3$ ， $l_2 = -4$ ， $l_3 = 1$ ，  
 $i=1$ ， $j=1$ ， $k=3$

$$A(1, 1, 3) = 100 + (1 - (-3)) * 7 * 5 * 1 + (1 - (-4)) * 5 * 1 + (3 - 1) * 1 = 267$$

另一種解決為將式子全化為標準式來看

$$A(-3 : 5, -4 : 2, 1 : 5) \rightarrow A(0 : 8, 0 : 6, 0 : 4) \quad \text{————— ①}$$

從上式得知  $u_1=9$ ， $u_2=7$ ， $u_3=5$

$$A(1, 1, 3) \rightarrow A(4, 5, 2) \quad \text{————— ②}$$

此乃第①式中全部化為標準式，分為加 3，加 4 及減 1 的原故， $\therefore$  在第②式中也要分別做加 3，加 4 及減 1 的動作。同時  $A(-3, -4, 1) \rightarrow A(0, 0, 0)=100$ (已知)

$$\begin{aligned} \therefore A(4, 5, 2) &= 100 + 4*7*5 + 5*5 + 2 \\ &= 100 + 140 + 25 + 2 \\ &= 267 \end{aligned}$$

與上一種解法答案是相同。

## 2.1.4 n 維陣列

假若有一 n 維陣列(n dimension array)為  $A(0 : u_1-1, 0 : u_2-1, 0 : u_3-1, \dots, 0 : u_n-1)$ ，表示 A 陣列為 n 維陣列，同樣 n 維陣列亦有二種表示方式：(1)以列為主，(2)以行為主。

1. 以列為主：若 A 陣列以列為主，表示 A 陣列有  $u_1$  個 n-1 維陣列， $u_2$  個 n-2 維陣列， $u_3$  個 n-3 維陣列， $\dots$  及  $u_n$  個一維陣列。假設起始位址為  $l_0$ ，則

$$\begin{aligned} A(0, 0, 0, \dots, 0)\text{-之位址為} & \quad l_0 \\ A(i_1, 0, 0, \dots, 0)\text{-之位址為} & \quad l_0 + i_1 * u_2 * u_3 \dots u_n \\ A(i_1, i_2, 0, \dots, 0)\text{-之位址為} & \quad l_0 + i_1 * u_2 * u_3 \dots u_n \\ & \quad + i_2 * u_3 * u_4 \dots u_n \\ \dots & \\ A(i_1, i_2, i_3, \dots, i_n)\text{-之位址為} & \quad l_0 + i_1 * u_2 * u_3 \dots u_n \\ & \quad + i_2 * u_3 * u_4 \dots u_n \\ & \quad + i_3 * u_4 * u_5 \dots u_n \\ \dots & \\ & \quad + i_{n-1} * u_n \\ & \quad + i_n \end{aligned}$$

上述可歸納為：

$$A(i_1, i_2, i_3, \dots, i_n) = l_0 + \sum_{m=1}^n i_m * a_m, \text{ 其中}$$

$$\begin{cases} a_m = \prod_{p=m+1}^n u_p, 1 \leq m < n \\ a_n = 1 \end{cases} \quad (\text{此處的 } \pi \text{ 是連乘的意思})$$

2. 以行為主：若 A 陣列以行為主，表示 A 陣列有  $u_n$  個  $n-1$  維陣列， $u_{n-1}$  個  $n-2$  維陣列， $\dots$ ， $u_j$  個  $j-1$  維陣列及  $u_2$  個一維陣列。假設起始位址亦是  $l_0$ ，則

$$\begin{array}{ll}
 A(0, 0, 0, \dots, 0) & \text{之位址為 } l_0 \\
 A(0, 0, 0, \dots, i_n) & \text{之位址為 } l_0 + i_n * u_1 u_2 \dots u_{n-1} \\
 A(0, 0, 0, \dots, i_{n-1}, i_n) & \text{之位址為 } l_0 + i_n * u_1 u_2 \dots u_{n-1} \\
 & \quad + i_{n-1} * u_1 u_2 \dots u_{n-2} \\
 \dots & \\
 A(i_1, i_2, i_3, \dots, i_n) & \text{之位址為 } l_0 + i_n * u_1 u_2 \dots u_{n-1} \\
 & \quad + i_{n-1} * u_1 u_2 \dots u_{n-2} \\
 & \quad + i_{n-2} * u_1 u_2 \dots u_{n-3} \\
 & \dots \\
 & \quad + i_2 * u_1 \\
 & \quad + i_1
 \end{array}$$

上述可歸納為：

$$A(i_1, i_2, i_3, \dots, i_n) = l_0 + \sum_{m=1}^n i_m * a_m, \text{ 其中}$$

$$\begin{cases} a_m = \prod_{p=1}^{m-1} u_p, 2 \leq m < n \\ a_1 = 1 \end{cases}$$



### 練習題

1. 有一二維陣列如下： $A(1 : u_1, 1 : u_2)$ ，若分別寫出(a)以列為主(b)以行為主的  $A(i, j) = ?$
2. 假設  $A(-3 : 5, -4 : 2)$  且其起始位置  $A(-3, -4) = 100$ ，以行為主排列，請問  $A(1, 1)$  所在位址？(d=1)

若有一陣列為  $A(1 : u_1, 1 : u_2, 1 : u_3)$ ，試問  $A(i, j, k)$  分別以列為主，和以行為主各為何？

假設有一三維陣列  $A(-3 : 5, -4 : 2, 1 : 5)$  且其起始位置為  $A(-3, -4, 1) = 100$ ，以行為主排列，則  $A(2, 1, 2)$  所在位址為何？(d=1)

3. 假設有一  $n$  維陣列  $A(1 : u_1, 1 : u_2, 1 : u_3, \dots, 1 : u_n)$ ，試分別寫出以列為主和以行為主的  $A(i_1, i_2, i_3, \dots, i_n)$  的位址為何？

## 2.2 C# 語言的陣列表示方法

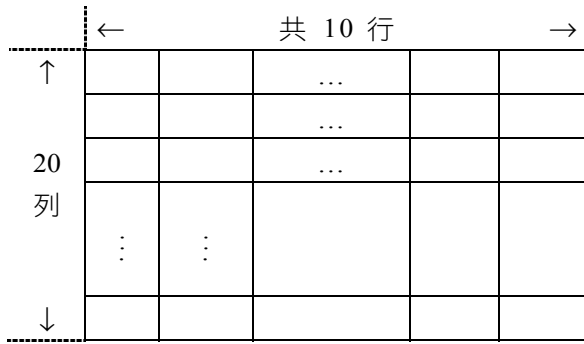
C# 語言的一維陣列宣告如下：

```
int[] A = new int[20];
```

表示 A 陣列有 20 個整數元素，從 A[0]到 A[19]，此處陣列的元素乃以大括號表示之。注意！C# 語言的陣列註標起始值為 0，而二維陣列宣告方法為

```
int[,] A = new int[20, 10];
```

表示 A 陣列有 20 列、10 行，如下圖所示：

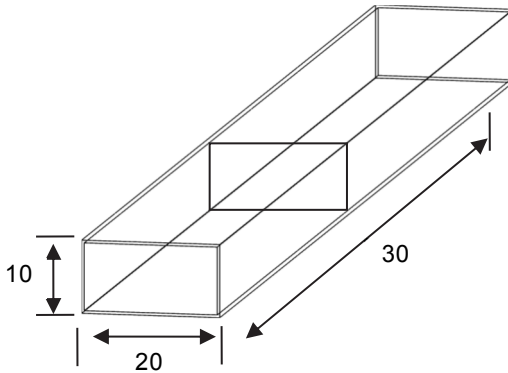


從 A[0, 0]，A[0, 1]，...，A[19, 9]等 200 個元素。

以此類推，三維陣列就是由三個中括號表示之，如

```
int[,,] A = new int[30, 20, 10];
```

圖形就像三度空間：



總共有 6000 個元素。

 **練習題**

1. 將第一章的動動腦時間的第 2 題實際以 C# 語言執行，並檢查你做的答案是否相符合。



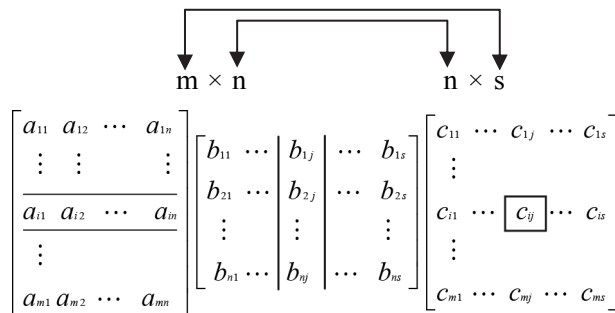
## 2.3 矩陣

### 1. 矩陣相乘

假設  $A = (a_{ij})$  是一  $m \times n$  的矩陣，而  $B = (b_{ij})$  為  $n \times s$  的矩陣，則  $AB$  的乘積為  $m \times s$  的矩陣

$$(AB)_{ij} = \sum_{k=1}^n a_{ik}b_{kj}$$

如下圖所示：



#### 範例》

$$\begin{bmatrix} 2 & 1 & -3 \\ -2 & 2 & 4 \end{bmatrix}
 \begin{bmatrix} -1 & 2 \\ 0 & -3 \\ 2 & 1 \end{bmatrix}
 =
 \begin{bmatrix} 2(-1)+1(0)+(-3)2 & 2(2)+1(-3)+(-3)1 \\ (-2)(-1)+2(0)+4(2) & (-2)2+2(-3)+4(1) \end{bmatrix} \\
 =
 \begin{bmatrix} -8 & -2 \\ 10 & -6 \end{bmatrix}$$

矩陣相乘的片段程式如下：

#### C# 片段程式》

```

static void accessMatrix(int[,] A, int[,] B)
{
    int i=0, j=0, k=0;

    /* 將 A 矩陣每一列元素與 B 矩陣每一列元素相乘之和放入 C 矩陣之中 */
    for ( i = 0; i < N; i++ )
        for ( j = 0; j < N; j++ ) {
            sum = 0;
            for ( k = 0; k < N; k++ )
                sum = sum + A[i, k] * B[k, j];
            C[i, j] = sum;
        }
}

```

其中 `sum` 為 `int` 的資料型態。有關矩陣相乘的程式實作，請參閱 2.8 之程式集錦。

## 2. 稀疏矩陣

若一矩陣中有大多數元素為 0 時，則稱此矩陣為稀疏矩陣(sparse matrix)。到底要多少個 0 才算是疏稀，則沒有絕對的定義，一般而言，大於 1/2 個就可稱之，如下列矩陣為一稀疏矩陣。

$$\begin{bmatrix} 0 & 15 & 0 & 0 & -8 & 0 \\ 0 & 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 18 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 16 \\ 72 & 0 & 0 & 0 & 20 & 0 \end{bmatrix}$$

上列矩陣共有 36 個元素，但只有 7 個非為 0，因此 0 的元素佔了 80%左右，若是將 1000×1000 矩陣以二維陣列，將這些元素儲存之，勢必會浪費許多空間，因為有許多的 0 根本不必存起來，只要存非零的元素即可，因此，我們可以下列的資料結構表示之。

(i, j, value)，其中 i 表示列數，j 表示行數，而 value 表示要儲存的值，若將此矩陣的非零值存於二維陣列為 A(0 : n, 1 : 3)的結構中，n 表示共有多少個非零的數目，則如下所示(假設二維陣列的第一個元素為 A(1, 1))：

	1)	2)	3)
A(0,	6	6	7
A(1,	1	2	15
A(2,	1	5	-8
A(3,	3	4	-6
A(4,	4	3	18
A(5,	5	6	16
A(6,	6	1	72
A(7,	6	5	20

其中 A(0, 1)=6 表示有 6 列，A(0, 2)=6 表示有 6 行，而 A(0, 3)=7 表示有 7 個非零值，而 A(1, 1)=1，A(1, 2)=2 及 A(1, 3)=15，表示第一列第 2 行的值為 15，餘此類推。

### 練習題

將此節所看到的稀疏矩陣，利用 C# 程式加以掃描，將非零的元素一一存在一個名為 sm 的二維陣列中。

## 2.4 多項式表示法

有一多項式  $p = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$ ，我們稱  $A$  為  $n$  次多項式， $a_i x^j$  是多項式的項 ( $0 \leq i \leq n, 1 \leq j \leq n$ ) 其中  $a_i$  為係數， $x$  為變數， $j$  為指數。一般多項式可以使用線性串列來表示其資料結構，也可以使用鏈結串列來表示(在第四章討論)。

多項式使用線性串列來表示有兩種方法：

1. 使用一個  $n+2$  長度的陣列，依據指數由大至小依序儲存係數，陣列的第一個元素是此多項式最大的指數，如  $p = (n, a_n, a_{n-1}, \dots, a_0)$ 。
2. 另一種只考慮多項式中非零項的係數，若有  $m$  項，則使用一個  $2m+1$  長度的陣列來儲存，分別存每一個非零項的指數與係數，而陣列中第一個元素是此多項式非零項的個數。

例如有一多項式  $p = 8x^5 + 6x^4 + 3x^2 + 12$  分別利用第 1 種和第 2 種方式來儲存，其情形如下：

$$(1) \quad p = (5, 8, 6, 0, 3, 0, 12)$$

$$(2) \quad p = (4, 5, 8, 4, 6, 2, 3, 0, 12)$$

假若是一個兩變數的多項式，那如何利用線性串列來儲存呢？此時需要利用二維陣列，若  $m, n$  分別是兩變數最大的指數，則需要一個  $(m+1) * (n+1)$  的二維陣列。如多項式  $p_{xy} = 8x^5 + 6x^4 y^3 + 4x^2 y + 3xy^2 + 7$ ，則需要一個  $(5+1) * (3+1) = 24$  的二維陣列，表示的方法如下：

$$\begin{array}{c}
 y^0 \quad y^1 \quad y^2 \quad y^3 \\
 \begin{array}{l}
 x^0 \\
 x^1 \\
 x^2 \\
 x^3 \\
 x^4 \\
 x^5
 \end{array}
 \left[ \begin{array}{cccc}
 7 & 0 & 0 & 0 \\
 0 & 0 & 3 & 0 \\
 0 & 4 & 0 & 0 \\
 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 6 \\
 8 & 0 & 0 & 0
 \end{array} \right]
 \end{array}$$

兩多項式  $A, B$  相加的原理很簡單，比較兩多項式大小時，有下列三種情況：

- (1)  $A$  指數 =  $B$  指數
- (2)  $A$  指數 >  $B$  指數
- (3)  $A$  指數 <  $B$  指數，

多項式相加的片段程式如下：

### C# 片段程式》

```
static void Padd(int[] a, int[] b, int[] c)
{
    int p, q, r, m, n;
    char result;
```

```

m = a[1]; n = b[1];
p = q = r = 2;
while ((p <= 2 * m) && (q <= 2 * n))
{
    /*比較 a 與 b 的指數*/
    result = compare(a[p], b[q]);
    switch (result)
    {
        case '=':          /* a 的指數等於 b 的指數 */
            c[r + 1] = a[p + 1] + b[q + 1]; /*係數相加*/
            if (c[r + 1] != 0)
            {
                c[r] = a[p]; /*指數 assign 給 c */
                r += 2;
            }
            p += 2; q += 2; /*移至下一個指數位置*/
            break;
        case '>':          /* a 的指數大於 b 的指數 */
            c[r + 1] = a[p + 1];
            c[r] = a[p];
            p += 2; r += 2;
            break;
        case '<':          /* a 的指數小於 b 的指數 */
            c[r + 1] = b[q + 1];
            c[r] = b[q];
            q += 2; r += 2;
            break;
    }
}
while (p <= 2 * m)
{ /*將多項式 a 的餘項全部移至 c */
    c[r + 1] = a[p + 1];
    c[r] = a[p];
    p += 2; r += 2;
}
while (q <= 2 * n)
{ /*將多項式 b 的餘項全部移至 c */
    c[r + 1] = b[q + 1];
    c[r] = b[q];
    q += 2; r += 2;
}
c[1] = r / 2 - 1; /*計算 c 總共有多少非零項*/
}

static char compare(int x, int y)
{
    if (x == y)
        return '=';
    else if (x > y)
        return '>';
}

```

```

else
    return '<';
}

```

### 》程式解說

在程式中， $a[1]$ 表示  $a$  多項式非零項的個數， $b[1]$ 為  $b$  多項式非零項的個數，一開始先將  $p$ 、 $q$ 、 $r$  指到陣列的第 2 個元素，而當  $p \leq 2*m$  及  $q \leq 2*n$  的狀況下，才做多項式的比較動作。

多項式的比較動作是比較指數而非係數，因此在 `while` 敘述中， $p+=2$  與  $q+=2$  的目的，是為了取得多項式的指數。最後，`while (p <= 2*m)` 敘述，是當  $b$  的多項式已結束，則將  $a$  多項式的餘項搬到  $c$  多項式；若 `while (q <= 2*n)` 條件成立，表示  $a$  多項式已結束，將  $b$  多項式的餘項搬到  $c$  多項式中，最後計算  $c$  多項式中非零項的個數。

有關多項相加的程式實作，請參閱 2.8 之程式集錦。

### 練習題

1. 有一多項式  $p_x=6x^7+8x^5+5x^4+3x^2+7$  請利用課文中提到的兩種方法表示之。
2. 有一多項式  $p_{xy}=6x^5+3x^4y^3+2x^3y^2-8x^2y+9x+3$  試利用二維陣列表示之。

## 2.5 上三角形和下三角形表示法

若一矩陣的對角線以下的元素均為零時，亦即  $a_{ij} = 0, i > j$ ，則稱此矩陣為上三角形矩陣(upper triangular matrix)。反之，若一矩陣的對角線以上的元素均為零，亦即  $a_{ij} = 0, i < j$ ，此矩陣稱為下三角形矩陣(lower triangular matrix)，如圖 2.4 所示：

$$\begin{array}{cc}
 \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix} & 
 \begin{bmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \\
 \text{(a) 上三角形矩陣} & \text{(b) 下三角形矩陣}
 \end{array}$$

圖 2.4 上、下三角形矩陣

由上述得知一個  $n * n$  個的上、下三角形矩陣共有  $[n(n+1)] / 2$  個元素，依序對應至  $D(1 : [n(n+1)] / 2)$ 。

## 1. 以列為主

一個  $n \times n$  的上三角形矩陣其元素分別對應至 D 陣列，如下所示：

$$\begin{array}{cccccccccccc} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{22} & a_{23} & a_{24} & \cdots & a_{ij} & \cdots & a_{nn} \\ \hline D(1) & D(2) & D(3) & D(4) & \cdots & D(n+1) & D(n+2) & D(n+3) & \cdots & D(k) & \cdots & D([n(n+1)]/2) \end{array}$$

$\therefore a_{ij} = D(k)$ ，其中  $k = n(i-1) - [i(i-1)]/2 + j$

例如圖 2.4 之(a)的  $a_{34}$  元素對應  $D(k)$ ，而

$$k = 4(3-1) - [3(3-1)]/2 + 4 = 8 - 3 + 4 = 9$$

讀者可以這樣想： $a_{34}$  表示此元素在第 3 列第 4 行的位置，因此上面有二列的元素，而每列 4 個位置，共 8 個空間，由於此矩陣是上三角形矩陣，因此有些位置不放元素，所以必需減掉這些不放元素的空間。有  $1+2 = 3$  ( $[i(i-1)]/2$ ,  $i=3$ )，然後再加上此元素在那一行( $j$ )。

假使是一個  $n \times n$  的下三角形矩陣，其元素分別對應至 D 陣列，如下所示：

$$\begin{array}{cccccccccccc} a_{11} & a_{21} & a_{22} & a_{31} & a_{32} & \cdots & a_{ij} & \cdots & a_{nn} \\ \hline D(1) & D(2) & D(3) & D(4) & D(5) & \cdots & D(k) & \cdots & D([n(n+1)]/2) \end{array}$$

$\therefore a_{ij} = D(k)$ ，其中  $k = [i(i-1)]/2 + j$

例如圖 2.4 之(b)的下三角形矩陣的  $a_{32}$  位於  $D(k)$ ，而

$$k = [3(3-1)]/2 + 2 = 5$$

## 2. 以行為主

上三角形矩陣的對應情形如下：

$$\begin{array}{cccccccccccc} a_{11} & a_{12} & a_{22} & a_{13} & a_{23} & a_{33} & \cdots & a_{ij} & \cdots & a_{nn} \\ \hline D(1) & D(2) & D(3) & D(4) & D(5) & D(6) & \cdots & D(k) & \cdots & D([n(n+1)]/2) \end{array}$$

$\therefore a_{ij} = D(k)$ ，其中  $k = [j(j-1)]/2 + i$

例如圖 2.4 之(a)的  $a_{34}$  位於  $D(k)$ ，其中

$$k = [4(4-1)]/2 + 3 = 6 + 3 = 9$$

而下三角形矩陣對應情形如下：

$$\begin{array}{cccccccccccc} a_{11} & A_{21} & A_{31} & A_{41} & \cdots & a_{22} & a_{32} & \cdots & a_{ij} & \cdots & a_{nn} \\ \hline D(1) & D(2) & D(3) & D(4) & \cdots & D(n+1) & D(n+2) & \cdots & D(k) & \cdots & D([n(n+1)]/2) \end{array}$$

$$\therefore a_{ij} = D(k), \text{ 其中 } k = n(j-1) - [j(j-1)]/2 + i$$

如圖 2.4 之(b)的  $a_{32}$  位於  $D(k)$ ，其中

$$k = 4(2-1) - [2(2-1)]/2 + 3 = 4 - 1 + 3 = 6$$

由此可知上三角形矩陣以列為主和下三角形以行為主的計算方式略同，而上三角形矩陣以行為主的計算方式與下三角形以列為主的計算方式相同。(請讀者注意，此處設定陣列的起始元素位置為 1)



### 練習題

1. 試撰寫一演算法將  $A_{n \times n}$  的上三角形儲存於一個  $B(1 : n(n+1)/2)$  的陣列中。
2. 之後，撰寫一個演算法從  $B$  陣列取出  $A(i, j)$ 。

## 2.6 魔術方陣

有一  $n \times n$  的方陣，其中  $n$  為奇數，請你在  $n \times n$  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。

做法很簡單，首先將 1 填入最上列的中間空格，然後往左上方走，規則如下：

- (1) 以 1 的級數增加其值，並將此值填入空格；
- (2) 假使空格已被填滿，則在原地的下一空格填上數字，並繼續往下做；
- (3) 若超出方陣，則往下到最底層或往右到最右方，視兩者那一個有空格，則將數目填上此空格；
- (4) 若兩者皆無空格則在原地的下一空格填上數字。

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

- 將 1 填入此方陣的最上列的中間空格，如下所示：

		j				
		0	1	2	3	4
i	0			1		
	1					
	2					
	3					
	4					

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

			1		
	2				

- 承 2，往左上方，依據規則(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 填在最右方的空格，如下所示：

		1			
					4
	3				
		2			

- 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				



5. 以此類推，依據上述的四個規則繼續填，填到 15 的結果如下：

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

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

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

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

奇數魔術方陣的片段程式如下：

### 📄 C# 片段程式

```
static void Magic()
{
    int i, j, p, q, key;

    /*初始化矩陣內容，矩陣全部清 0 */
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            Square[i, j] = 0;

    Square[0, (N - 1) / 2] = 1; /*將 1 放至最上列中間位置*/
    key = 2;
    i = 0;
```

```

j = (N - 1) / 2; /* i, j 記錄目前所在位置*/
while (key <= N * N)
{
    p = (i - 1) % N; /* p, q 為下一步位置, i, j 各減 1 表往西北角移動*/
    q = (j - 1) % N;
    /* p < 0 (超出方陣上方)*/
    if (p < 0) p = N - 1; /* 則將 p 移至 N - 1(最下列) */
    if (q < 0) q = N - 1; /* q < 0 (超出方陣左方) */
                          /* 則將 q 移至 N - 1(最右行) */
    if (Square[p, q] != 0) /*判斷下一步是否已有數字*/
        i = (i + 1) % N; /*已有則 i 往下 (填在原值下方)*/
    else
    {
        i = p; /*將下一步位置 assing 給目前位置 */
        j = q;
    }
    Square[i, j] = key;
    key++;
}
}

```

## 》 程式解說

先將方陣 `Square` 中的每一個元素皆設為 0，在最上列的中間方格 `Square[0, (N-1)/2]` 填 1。接下來的 `while (key <= N*N)` 內的敘述會不斷執行，直到方陣完全走完為止，其中 `(p, q)` 為下一步的位置，當 `p < 0` 表示超出方陣上方，依據規則調整 `p` 至最下層 `(N-1)`。同理，當 `q < 0` 表示超出方陣左方，調整 `q` 至最右方 `(N-1)` 的位置。

`if (Square[p, q] != 0)` 會判斷下一方格是否已有數字，若發現已有數字，則移動目前位置至原來的位置 `(i, j)` 下方；若下一方格沒有數字，則移動目前位置至下一步位置 `(p, q)`，將數字填入方格中。

以上述的 5\*5 方陣為例，來說明魔術方陣的演算法：

1. 首先將 1 放在 `Square[0, N-1]/2]` 的方格上，若 `N = 5`，則此方格為第 0 列、第 2 行。
2. 將目前的方格所代表的第 `N` 列和第 `N` 行存放在 `i` 與 `j` 中，此時 `i = 0`，`j = 2`。並將 2 指定給 `key`。
3. 當 `key ≤ 52` 時，將 `(i-1) % N` 即 `(0-1)%5 = -1`；`(j-1) % N` 即 `(2-1) % 5 = 1`，求目前方格左上方的座標，但因 `(-1, 1)` 已超出方陣最上列，故依規則將列座標調整至最下層 `N-1` 位置，即 `5-1 = 4`。由於 `(4, 1) = 0` 故將 4 指定給 `j`，然後將 `key` 的值放在 `(i, j) =(4, 1)` 方格上，`key++`。
4. 倘若 `key = 6`，此時的 `(i, j)` 為 `(1, 3)`。因為 `Square[0, 2]` 已有數字，即 `Square[p, q] != 0`，則將 `(1+1)%5 = 2`，將此數字指定給 `i`，即此方格為 `(2, 3)`，此表示在原來的方格往下移一格。

5. 利用同樣的方法即可完成魔術方陣。

有關奇數魔術方陣的程式實作，請參閱 2.8 之程式集錦。

### 練習題

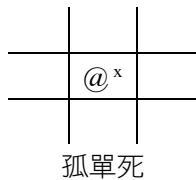
自行完成 9\*9 的魔術方陣。

## 2.7 生命細胞遊戲

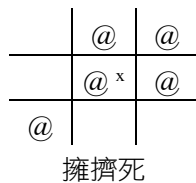
本章將以生命細胞遊戲(game of life)做為結束，此遊戲在 1970 年由英國數學家 J. H. CONWAY 所提出。生命細胞遊戲將陣列元素視為細胞，而某一細胞鄰居乃是指在其垂直、水平、對角線相鄰之細胞(cells)。

生命細胞遊戲的規則：

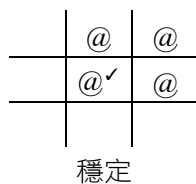
1. **孤單死**：若一活細胞只有一個或沒有鄰居細胞存活的，則在下一代，它將孤單而死。(下圖中以@表示一細胞，而 x 符號表示此細胞將死去)如：



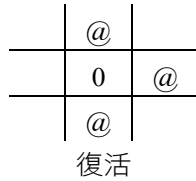
2. **擁擠死**：一活細胞有四個或四個以上鄰居亦是活的，則在下一代，它將因擁擠而死。(下圖中以 x 符號表示此細胞將死去)如：



3. **穩定**：一活細胞有二個或三個相鄰活細胞，則下一代它將繼續生存。(下圖中以✓符號表示此細胞將繼續存活之)如：



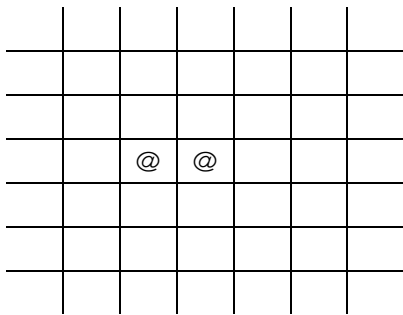
4. 復活：一死細胞正好有三個相鄰的活細胞，則下一代它將復活。(下圖中以 0 符號表示此位置會復活一細胞)如：



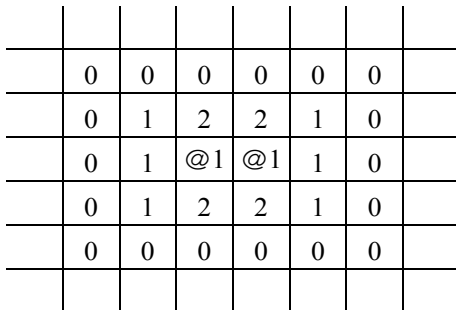
由上規則可得：

- 有 0, 1, 4, 5, 6, 7, 8 個相鄰細胞者在下一代將因孤單或擁擠而死。
- 有 2 個相鄰活細胞者，下一代會繼續其狀態不會改變。
- 有 3 個相鄰活細胞者不管其現在是生是死，下一代一定會是活的。

**範例一》**



我們寫出每一細胞鄰居的個數



根據規則(1)，圖中的細胞將會孤單而死。

### 範例二》

	0	0	0	0	0	0
	0	1	2	2	1	0
	0	2	@3	@3	2	0
	0	2	@3	@3	2	0
	0	1	2	2	1	0
	0	0	0	0	0	0

活細胞若鄰居為 2 或 3 的將會存活下來，但圖中的死細胞都是小於或等於 2，故不能再生，此圖已成為穩定狀態。

### 範例三》

	0	0	0	0	0	
	1	2	3	2	1	
	1	@1	@2	@1	1	
	1	2	3	2	1	
	0	0	0	0	0	

根據上述規則將成為

	0	1	1	1	0	
	0	2	@1	2	0	
	0	3	@2	3	0	
	0	2	@1	2	0	
	0	1	1	1	0	

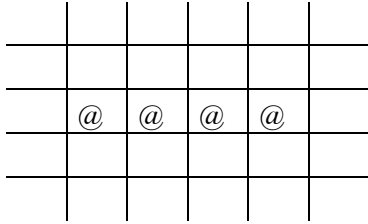
這二張圖將互相轉來轉去。

有關生命細胞遊戲的程式實作，請參閱 2.8 之程式集錦。

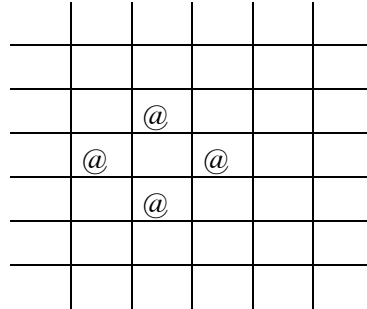
## 練習題

試完成下列生命細胞遊戲(其中@表示為一活細胞)。

(a)



(b)



## 2.8 程式集錦

### (一) 矩陣相乘

#### 《C# 程式語言實作》

```

/*
File name: Matrix.cs
February, 2018
Description: 矩陣相乘實作
將兩矩陣行列相乘之和放入第三個矩陣
*/

using System;

namespace Matrix
{
    class Matrix
    {
        const int N = 5;
        static void Main(string[] args)
        {
            int[,] A = {
                { 1, 2, 3, 4, 5 },
                { 1, 2, 3, 4, 5 },
                { 1, 2, 3, 4, 5 },
                { 1, 2, 3, 4, 5 },
                { 1, 2, 3, 4, 5 }
            };
            int[,] B = {
                { 5, 4, 3, 2, 1 },
                { 5, 4, 3, 2, 1 },
            };
        }
    }
}

```

```

        { 5, 4, 3, 2, 1 },
        { 5, 4, 3, 2, 1 },
        { 5, 4, 3, 2, 1 }
    };
    int[,] C = new int[N, N];
    int i, j, k, sum;

    /* 將 A 矩陣每一列元素與 B 矩陣每一列元素
    相乘之和放入 C 矩陣之中 */
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
        {
            sum = 0;
            for (k = 0; k < N; k++)
                sum = sum + A[i, k] * B[k, j];
            C[i, j] = sum;
        }

    /*列出三矩陣內容*/
    Console.WriteLine("\nContent of Matrix A :\n\n");
    outputMatrix(A);
    Console.WriteLine("\nContent of Matrix B :\n\n");
    outputMatrix(B);
    Console.WriteLine("\nContent of Matrix C :\n\n");
    outputMatrix(C);
    Console.WriteLine("\n");
    Console.ReadKey();
}

static void outputMatrix(int[,] m)
{
    int i, j;

    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
            Console.WriteLine("{0, 5}", m[i, j]);
        Console.WriteLine();
    }
}
}
}

```

### 輸出結果

Content of Matrix A :

```

1 2 3 4 5
1 2 3 4 5
1 2 3 4 5

```

```

1  2  3  4  5
1  2  3  4  5

Content of Matrix B :

5  4  3  2  1
5  4  3  2  1
5  4  3  2  1
5  4  3  2  1
5  4  3  2  1

Content of Matrix C :

75  60  45  30  15
75  60  45  30  15
75  60  45  30  15
75  60  45  30  15
75  60  45  30  15

```

## (二) 多項式相加

### 📄 C# 程式語言實作

```

/*
File name: AryPadd.cs
February, 2018
Description: 多項式相加實作
利用陣列表示法做多項式相加
*/

using System;

namespace AryPadd
{
    class AryPadd
    {
        const int DUMMY = -1;
        static void Main(string[] args)
        {
            /* 多項式的表示方式利用只儲存非零項法
            分別儲存每一個非零項的指數及係數，
            陣列第一元素放多項式非零項個數。
            ex: 下列 A 多項式有 3 個非零項，其多項式為：
            5x 四次方 + 3x 二次方 + 2 */

            int[] A = { DUMMY, 3, 4, 5, 2, 3, 0, 2 };
            int[] B = { DUMMY, 3, 3, 6, 2, 2, 0, 1 };
            int[] C = new int[13];

            Padd(A, B, C); /*將 A 加 B 放至 C */
                          /*顯示各多項式結果*/

```



```

Console.WriteLine("A = ");
outputP(A, A[1] * 2 + 1); /*A[1]*2 + 1 為陣列 A 的大小*/
Console.WriteLine("\nB = ");
outputP(B, B[1] * 2 + 1);
Console.WriteLine("\nC = ");
outputP(C, C[1] * 2 + 1);
Console.WriteLine("\n");
Console.ReadKey();
}

/*此為之前所解說第二種方法的多項式相加函數*/
static void Padd(int[] a , int[] b, int[] c)
{
    int p, q, r, m, n;
    char result;

    m = a[1]; n = b[1];
    p = q = r = 2;
    while ((p <= 2 * m) && (q <= 2 * n))
    {
        /*比較 a 與 b 的指數*/
        result = compare(a[p], b[q]);
        switch (result)
        {
            case '=':          /* a 的指數等於 b 的指數 */
                c[r + 1] = a[p + 1] + b[q + 1]; /*係數相加*/
                if (c[r + 1] != 0)
                {
                    c[r] = a[p]; /*指數 assign 給 c */
                    r += 2;
                }
                p += 2; q += 2; /*移至下一個指數位置*/
                break;
            case '>':          /* a 的指數大於 b 的指數 */
                c[r + 1] = a[p + 1];
                c[r] = a[p];
                p += 2; r += 2;
                break;
            case '<':          /* a 的指數小於 b 的指數 */
                c[r + 1] = b[q + 1];
                c[r] = b[q];
                q += 2; r += 2;
                break;
        }
    }
    while (p <= 2 * m)
    { /*將多項式 a 的餘項全部移至 c */
        c[r + 1] = a[p + 1];
        c[r] = a[p];
        p += 2; r += 2;
    }
}

```

```

while (q <= 2 * n)
{ /*將多項式 b 的餘項全部移至 c */
    c[r + 1] = b[q + 1];
    c[r] = b[q];
    q += 2; r += 2;
}
c[1] = r / 2 - 1; /*計算 c 總共有多少非零項*/
}

static char compare(int x, int y)
{
    if (x == y)
        return '=';
    else if (x > y)
        return '>';
    else
        return '<';
}

static void outputP(int[] p, int n)
{
    int i;

    Console.WriteLine("");
    for (i = 1; i <= n; i++)
        Console.WriteLine("{0, 3}", p[i]);
    Console.WriteLine(" ");
}
}
}

```

### 輸出結果

```

A = ( 3 4 5 2 3 0 2 )
B = ( 3 3 6 2 2 0 1 )
C = ( 4 4 5 3 6 2 5 0 3 )

```

## (三) 奇數魔術方陣

### C# 程式語言實作》

```

/*
File name: OddMagic.cs
February, 2018
Description: ODD Magic Matrix Implementation
奇數魔術方陣實作
*/
using System;

namespace OddMagic

```

```

{
class OddMagic
{
    const int MAX = 15; /*矩陣最大為 15 x 15 */
    static int[,] Square = new int[MAX, MAX]; /*定義整數矩陣*/
    static int N; /*矩陣行列大小變數*/
    static void Main(string[] args)
    {
        int i, j;
        /*讀取魔術矩陣的大小 N, N 為奇數且 0 <= N <= 15 */
        do
        {
            Console.WriteLine("\nEnter odd matrix size : ");
            N = Convert.ToInt16(Console.ReadLine());
            if (N % 2 == 0 || N <= 0 || N > 15)
                Console.WriteLine("Should be > 0 and < 15 odd number");
            else
                break;
        } while (true);

        Magic(); /*將 square 變為 N x N 的魔術矩陣*/

        /*顯示魔術矩陣結果*/
        Console.WriteLine("\nThe " + N + "*" + N + " Magic Matrix\n");
        Console.WriteLine("-----\n");
        for (i = 0; i < N; i++)
        {
            for (j = 0; j < N; j++)
                Console.WriteLine("{0, 4}", Square[i, j]);
            Console.WriteLine();
        }
        Console.ReadKey();
    }

    static void Magic()
    {
        int i, j, p, q, key;

        /*初始化矩陣內容, 矩陣全部清 0 */
        for (i = 0; i < N; i++)
            for (j = 0; j < N; j++)
                Square[i, j] = 0;

        Square[0, (N - 1) / 2] = 1; /*將 1 放至最上列中間位置*/
        key = 2;
        i = 0;
        j = (N - 1) / 2; /* i, j 記錄目前所在位置*/
        while (key <= N * N)
        {
            p = (i - 1) % N; /* p, q 為下一步位置, i, j 各減 1 表往西北角移動*/
            q = (j - 1) % N;

```

```

        /* p < 0 (超出方陣上方)*/
        if (p < 0) p = N - 1; /* 則將 p 移至 N - 1(最下列) */
        if (q < 0) q = N - 1; /* q < 0 (超出方陣左方) */
                           /* 則將 q 移至 N - 1(最右行) */
        if (Square[p, q] != 0) /*判斷下一步是否已有數字*/
            i = (i + 1) % N; /*已有則 i 往下 ( 填在原值下方*/
        else
        {
            i = p; /*將下一步位置 assing 給目前位置 */
            j = q;
        }
        Square[i, j] = key;
        key++;
    }
}
}
}
}

```

## 輸出結果

```

Enter odd matrix size : 5
The 5*5 Magic Matrix

```

```

-----
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

```

```

Enter odd matrix size : 7
The 7*7 Magic Matrix

```

```

-----
28 19 10 1  48 39 30
29 27 18 9  7  47 38
37 35 26 17 8  6  46
45 36 34 25 16 14  5
 4  44 42 33 24 15 13
12  3  43 41 32 23 21
20 11  2  49 40 31 22

```

## (四) 生命細胞遊戲

### 《C# 程式語言實作》

```

/*
File name: LifeGame.cs
February, 2018
Description: Game of Life Implementation
生命細胞遊戲實作
*/

```

```

using System;

namespace LifeGame
{
    class LifeGame
    {
        const int MAXROW = 10;
        const int MAXCOL = 25;
        const int DEAD = 0;
        const int ALIVE = 1;
        static int[,] map = new int[MAXROW, MAXCOL];
        static int[,] newmap = new int[MAXROW, MAXCOL];
        static int Generation;
        static void Main(string[] args)
        {
            int row, col;
            char ans;

            init(); /*起始 map */
            outputMap();
            do
            {
                /* 計算每一個(row, col)之 cell 的鄰居個數
                依此個數決定其下一代是生是死。
                將下一代的 map 暫存在 newmap 以防 overwrite map。*/
                for (row = 0; row < MAXROW; row++)
                    for (col = 0; col < MAXCOL; col++)
                        switch (Neighbors(row, col))
                        {
                            case 0:
                            case 1:
                            case 4:
                            case 5:
                            case 6:
                            case 7:
                            case 8:
                                newmap[row, col] = DEAD;
                                break;
                            case 2:
                                newmap[row, col] = map[row, col];
                                break;
                            case 3:
                                newmap[row, col] = ALIVE;
                                break;
                        }
                Copymap(); /*將 newmap copy to map */
            }
            do
            {
                Console.WriteLine("\nContinue next Generation (y/n)? ");
                ans = Char.ToUpper(Console.ReadLine().ToCharArray()[0]);
                if (ans == 'Y' || ans == 'N')

```

```

        break;
    } while (true);
    if (ans == 'Y') outputMap();
} while (ans == 'Y');
Console.WriteLine("\n");
Console.ReadKey();
}

static void init()
{
    int row, col;

    /*起始 map 狀態，一開始 cells 皆會 DEAD */
    for (row = 0; row < MAXROW; row++)
        for (col = 0; col < MAXCOL; col++)
            map[row, col] = DEAD;

    Console.WriteLine("Game of life Program \n");
    Console.WriteLine("Enter (x, y) where (x, y) is a living cell\n");
    Console.WriteLine(" 0 <= x <= " + (MAXROW - 1) + " , 0 <= y <= " + (MAXCOL - 1)
        + "\n");
    Console.WriteLine("Terminate with (x, y) = ( -1, -1)\n");
    /* 輸入活細胞之位置，以(-1, -1)結束輸入 */
    do
    {
        string ln = Console.ReadLine();
        row = Convert.ToInt32(ln.Split()[0]);
        col = Convert.ToInt32(ln.Split()[1]);
        if (0 <= row && row < MAXROW && 0 <= col && col < MAXCOL)
            map[row, col] = ALIVE;
        else
            Console.WriteLine("(x, y) exceeds map ranage!\n");
    } while (row != -1 || col != -1);
}

static int Neighbors(int row, int col)
{
    int count = 0, c, r;
    /* 計算每一個 cell 的鄰居個數
    因為 cell 本身亦被當做鄰居計算
    故最後還要調整 */
    for (r = row - 1; r <= row + 1; r++)
        for (c = col - 1; c <= col + 1; c++)
        {
            if (r < 0 || r >= MAXROW || c < 0 || c >= MAXCOL)
                continue;
            if (map[r, c] == ALIVE)
                count++;
        }
    /*調整鄰居個數*/
    if (map[row, col] == ALIVE)

```

```

        count--;
        return count;
    }

    /*顯示目前細胞狀態*/
    static void outputMap()
    {
        int row, col;
        Console.WriteLine("\n\n Game of life cell status\n");
        Console.WriteLine("{0, 20}-----Generation " + ++Generation + "-----\n", " ");
        for (row = 0; row < MAXROW; row++)
        {
            Console.WriteLine("{0, 20}", " ");
            for (col = 0; col < MAXCOL; col++)
                if (map[row, col] == ALIVE)
                    Console.Write('@');
                else
                    Console.Write('-');
            Console.WriteLine();
        }
    }

    /*將 newmap copy 至 map 中*/
    static void Copymap()
    {
        int row, col;
        for (row = 0; row < MAXROW; row++)
            for (col = 0; col < MAXCOL; col++)
                map[row, col] = newmap[row, col];
    }
}

```

### 輸出結果

(略)，請自行執行程式。(你可以用此程式來測試 2.7 節的範例是否正確)。

資料如下：

範例一	範例二	範例三
3 2	3 3	3 2
3 9	3 4	3 3
-1 -1	4 3	3 4
	4 4	-1 -1
	-1 -1	

同時也提供給您以下幾組測試資料：

(a) 3	8	(b) 4	5	(c) 3	8
3	9	4	6	3	9
3	10	4	7	3	10
3	11	5	5	2	10
3	12	5	6	4	10
-1	-1	5	7	5	10
		6	5	-1	-1
		6	6		
		-1	-1		

## 2.9 動動腦時間

1. 假設有一陣列 A，其 A(0, 0)與 A(2, 2)的位址分別在 $(1204)_8$ 與 $(1244)_8$ ，求 A(3, 3)的位址(以 8 進位表示)。[2.1]
2. 有一三維陣列 A(-3 : 2, -2 : 4, 0 : 3)，以列為主排列，陣列的起始位址是 318，試求 A(1, 3, 2)所在的位址。[2.1]
3. 有一二維陣列 A(0 : m-1, 0 : n-1)，假設 A(3, 2)在 1110，而 A(2, 3)在 1115，若每個元素佔一個空間，請問 A(1, 4)所在的位址。[2.1]
4. 若將一對稱矩陣(symmetric matrix)視為上三角形矩陣來儲存，亦即  $a_{11}$  儲存在 A(1)， $a_{12} = a_{21}$  儲存在 A(2)， $a_{22}$  在 A(3)， $a_{13} = a_{31}$  在 A(4)， $a_{23} = a_{32}$  在 A(5)，及  $a_{ij}$  在 A(k)地方。

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \quad \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{bmatrix}$$

試求 A(i, j)儲存的位址(可用 MAX 與 MIN 函數來表示，其中 MAX 函數表示取 i, j 的最大值，MIN 函數則是取 i, j 最小值)。[2.5]

5. 有一正方形矩陣，其存放在一維陣列的形式如下：

$$\begin{bmatrix} A(1) & A(2) & A(5) & A(10) & \dots \\ A(4) & A(3) & A(6) & A(11) & \dots \\ A(9) & A(8) & A(7) & A(12) & \dots \\ A(16) & A(15) & A(14) & A(13) & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

讓  $a_{ij}$  儲存在 A(k)，試求 A(i, j)所在的位址，可用 MAX 及 MIN 函數來表示。[2.5]