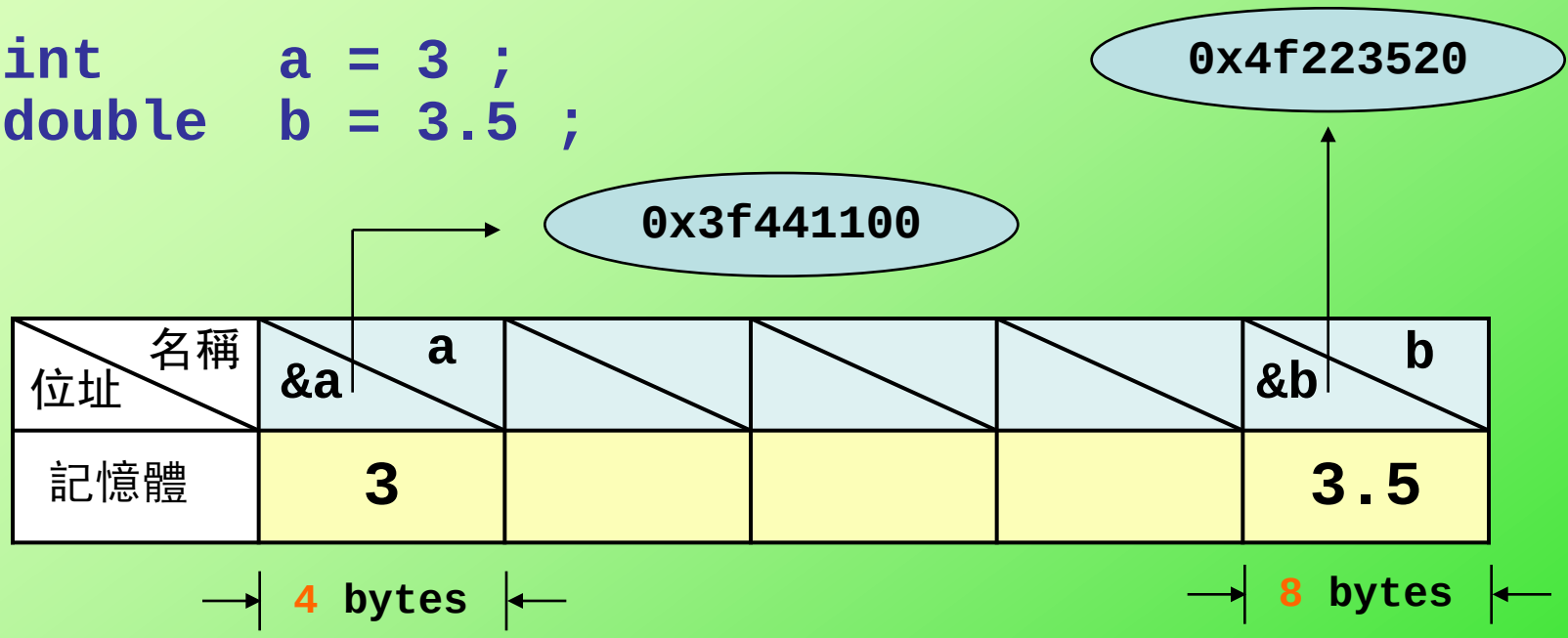
Chapter 5

指標

變數儲存位置

■ 敘述

```
int      a = 3 ;
double   b = 3.5 ;
```



a 的位址 = &a = 0x3f441100
b 的位址 = &b = 0x4f223520

位址運算子

- 變數位址　：變數資料所佔用記憶空間的起始位址
- 位址運算子　：變數名稱前使用 **&** 可取得變數位址

```
int   foo = 3 ;
cout << &foo ;          // 印出 foo 所佔用的起始位址。
                        // 位址是以十六進位表示的數字，
                        // 例如：0xbffff6b

double bar = 5.2 ;
cout << &bar << endl ;
```

address-of operator

指標資料型別

■ 指標資料型別：

用來儲存某同型別資料所在記憶空間的位址

```
int foo = 99 ;
int *p ;
p = &foo ;
```

```
// 定義指標 p，可以指向一整數位址
// 將整數 foo 的位址存放到指標
// 變數 p 中
```

上式程式碼二、三行可合併成

```
int *p = &foo ;
```

❖ 指標僅能指向同型別的變數位址

```
double  bar = 2.2 ;
// 錯誤
int  *p = &bar ;
```

pointer

指標（一）

- 指標定義：

```
int*  p ;
int * q ;
int  *r ;
```

// p q 與 r 皆是整數指標
// 皆可儲存整數位址

```
int *a , *b ;
int* c ,  d ;
```

// a 與 b 皆是整數指標
// c 是整數指標，但 d 只是整數

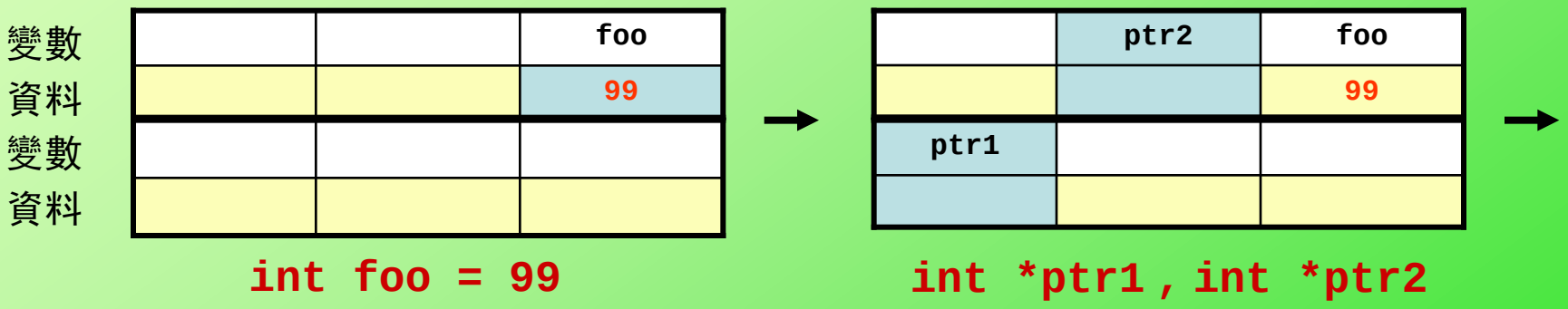
- 指標初值：

```
int *p = 0 ;
const int NULL = 0 ;
int *p = NULL ;
```

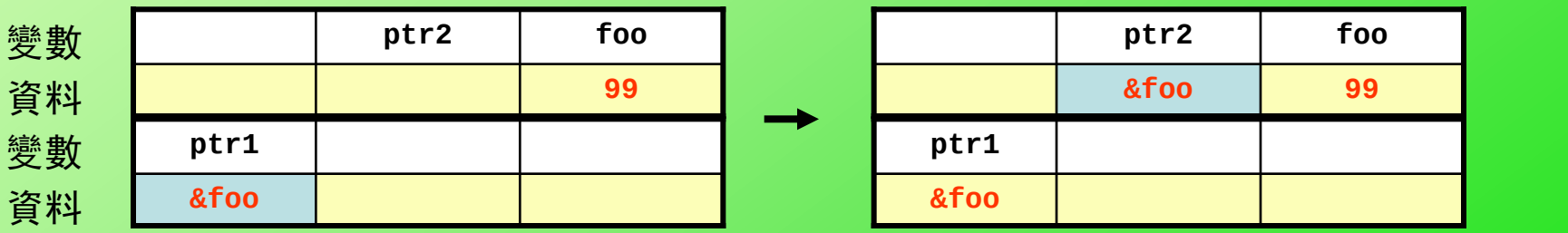
// 指標 p 尚未指向任何位址

指標（二）

```
int   foo = 99 ;
int*  ptr1  ;        // 定義一整數指標變數 ptr1
int * ptr2  ;        // 定義一整數指標變數 ptr2
ptr1 = &foo ;        // 將整數 foo 的位址存入指標變數 ptr1 內
ptr2 = ptr1 ;        // 將指標變數 ptr1 內所存放的位址複製
                     // 一份給 ptr2
```



int foo = 99　　int *ptr1 , int *ptr2

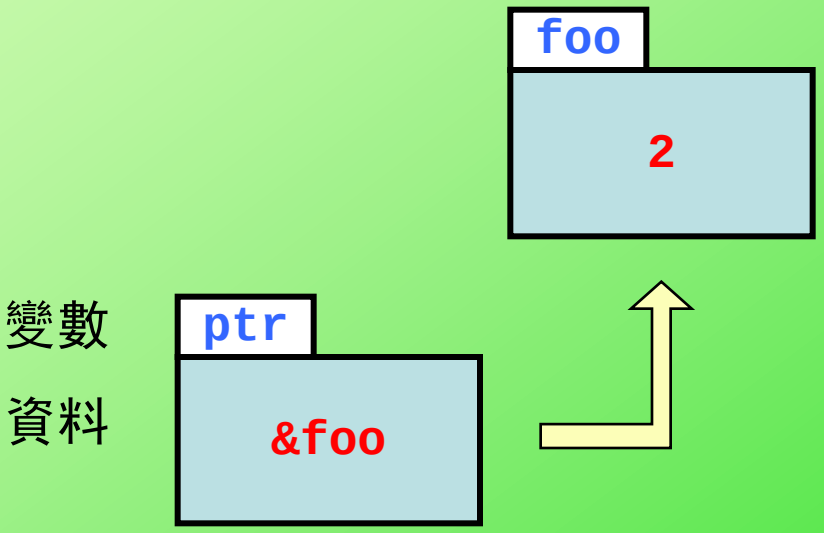


ptr1 = &foo　　ptr2 = ptr1

指標與參照運算子

- 空間儲存方式：

```
int  foo  = 2 ;
int  *ptr = &foo ;
```



- **ptr** 指標內所儲存的資料為指標所指向變數的位址
- **ptr** 指標可用參照運算子(*)取得指標所指向的資料
- 列印位址與資料

```
cout <<  ptr << endl ;  // 列印 foo 的位址
cout << *ptr << endl ;  // 列印 foo 的資料
```

參照運算子

- 指標之前使用參照運算子即可取得指標所指向位址之資料值

```
int  foo  = 2 ;
int  *ptr = &foo ;                          // 指標 ptr 指向 foo

cout << "*ptr = " << *ptr << endl ; // 列印：*ptr = 2

*ptr += 1 ;                                 // 將 ptr 所指向位址
                                            // 的資料加上 1
cout << "foo = " << foo << endl ;   // 列印：foo = 3

cout << "square = " << *ptr * *ptr // 列印：square = 9
     << endl ;
```

dereference operator

指標與常數（一）

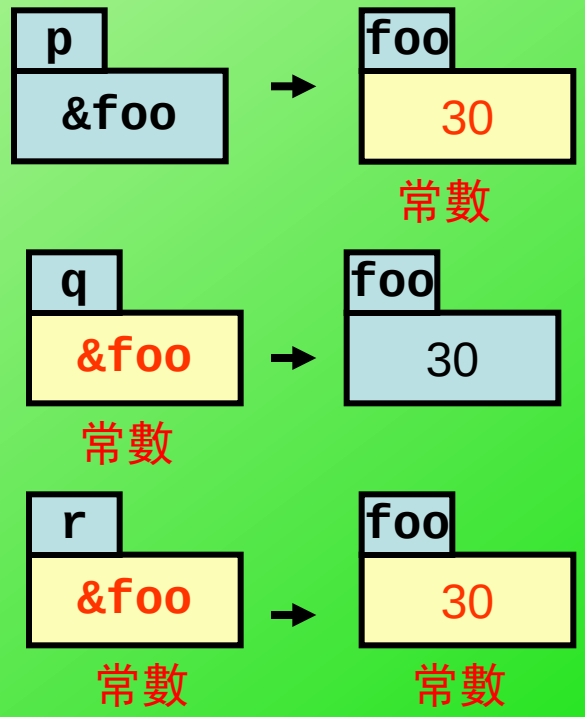
- 指標變數可取得兩筆資料的記憶空間，可以使用常數保留字(const)來設定哪一個記憶空間的資料為常數

```
int  foo = 30 ;
```

```
// 不能透過 p 更改 foo 的值
// 但 p 可以改指向其它位址
const int *       p = &foo ;
```

```
// q 永遠指向 foo 的位址
// 但 foo 的資料可透過 q 更動
int         * const q = &foo ;
```

```
// r 永遠指向 foo 的位址
// 且不能透過 r 改變 foo 的值
const int * const r = &foo ;
```



指標與常數（二）

- 三種指標常數：

int foo = 1	指標所在的記憶空間	指標指向的記憶空間
const int * ptr = &foo	O	X
int * const ptr = &foo	X	O
const int * const ptr = &foo	X	X

第一類型的常數指標是指不能透過指標來更改指向位址內所存放的資料，但不表示原擁有此記憶空間的變數本身不能更改其值

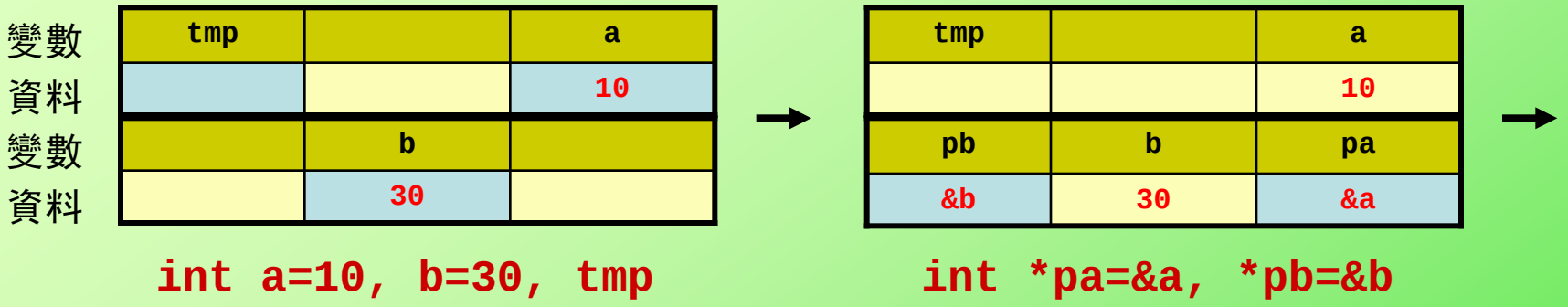
```
int       foo = 30   ;
const int * p = &foo ;
*p  = 50 ;                  // 錯誤
foo = 50 ;                  // 可以
```

指標基本運算（一）

- 使用指標交換兩變數 a 與 b 的數值

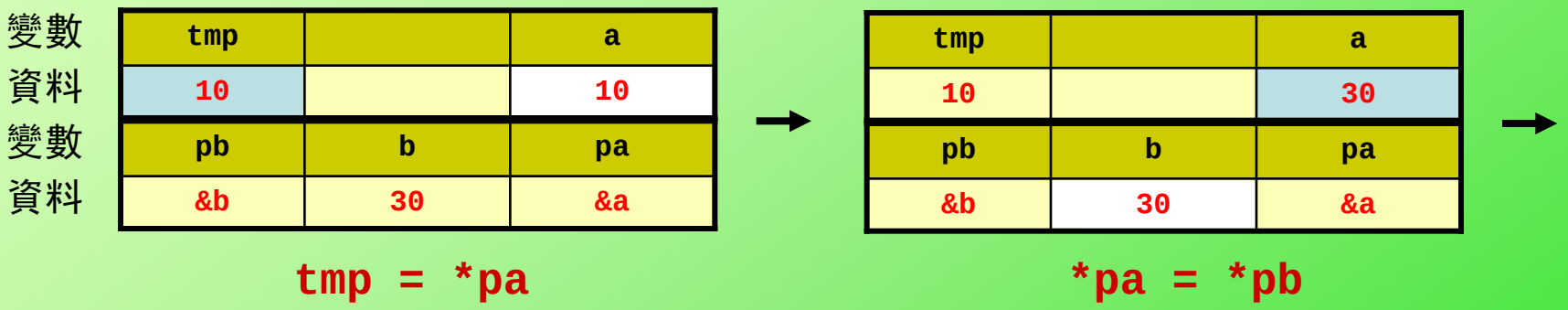
```
int   a = 10 , b = 30 , tmp ;
int *pa = &a , *pb = &b ;  // 指標 pa 與 pb 分別指向 a 與 b

tmp = *pa ;                // 將指標 pa 所指的資料複製給 tmp
*pa = *pb ;                // 將指標 pb 所指的資料複製到 pa
                           // 所指的位址
*pb = tmp ;                // 將 tmp 的資料複製到 pb 所指的
                           // 位址

// 列印 a = 30 ， b = 10
cout << "a = " << a << "\n"
     << "b = " << b << endl ;
```

指標基本運算（二）



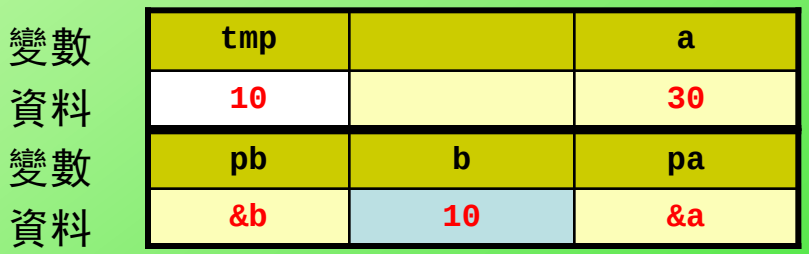

```
int a=10, b=30, tmp
```

```
int *pa=&a, *pb=&b
```




```
tmp = *pa
```

```
*pa = *pb
```




```
*pb = tmp
```

指標基本運算（三）

- 指標位址的指定

```
int foo = 3 ;
int *p , *q ;

p = &foo ;      // 指標 p 指到 foo 的位址

q = p ;         // 指標 p 的值存入指標 q，p 與
                // q 兩指標同指向 foo
```

指標基本運算（四）

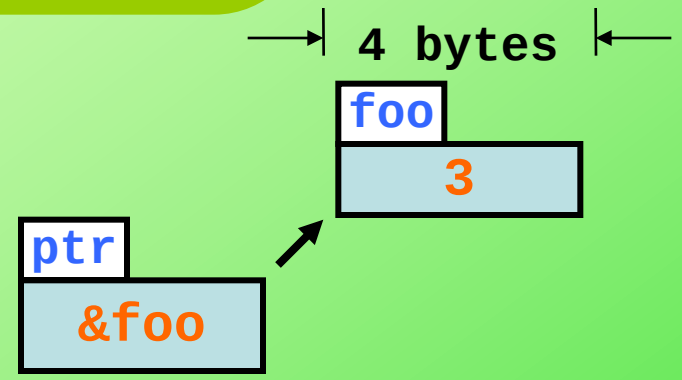
- 交換兩指標所指向的位址

```
int a = 3 , b = 10 ;
int *p = &a , *q = &b ;  // 指標 p 指向 a，指標 q 指向 b

// 輸出 p ， q 所指向的值，即 *p = 3 ，*q = 10
cout << "*p = " << *p << " ,"
     << "*q = " << *q << endl ;

int *r = p ;          // 指標 r 指向 p 所指的位址
p = q ;               // 指標 p 指向 q 所指的位址
q = r ;               // 指標 q 指向 r 所指的位址

// 輸出 p，q 所指向的值，即  *p = 10 ， *q = 3
cout << "*p = " << *p << " , "
     << "*q = " << *q << endl ;
```

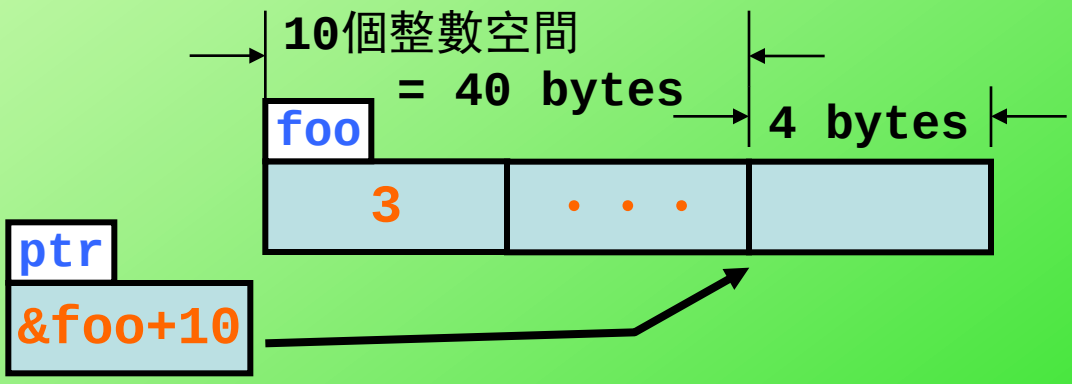
指標基本運算（五）

- 移動指標所指向的位址

```
int   foo = 3 ;
int  *ptr = &foo ;
```




```
ptr = ptr + 10 ;
```



- 移動後的指標所指向的位址可能不是程式所使用的空間，若將其資料更改，則可能引起程式在執行過程時中斷

run-time error

動態記憶空間配置

- 為了靈活使用計算機內有限的記憶空間，C++可在程式執行當中，臨時向作業系統要取適當的記憶空間來使用，當程式不須再使用此記憶空間時，也可以立即將之歸還給作業系統供其他程式使用

- new　　：動態配置記憶空間

- delete：動態歸還記憶空間

dynamical memory allocation

new 與 delete (一)

- 取得一個單位的記憶空間

```
int   *ptr ;

ptr = new int ;        // 向作業系統要一個整數空間，並
                       // 將此空間的位址存入 ptr 指標

*ptr = 10 ;            // 指標 ptr 所指向存入 10

*ptr += 1 ;            // 指標 ptr 所指向位址的資料加 1
 cout << *ptr ;        // 輸出指標 ptr 所指向位址的資料，即 11
```

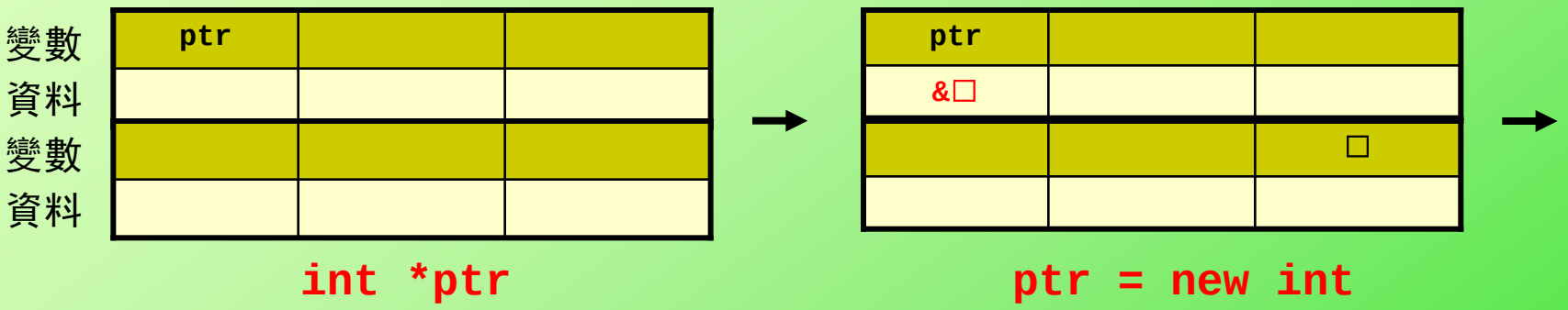
```
// 程式碼前三行可以使用
int *ptr = new int(10);
```

- 歸還記憶空間

```
delete ptr ;           // 歸還指標 ptr 所指向的動態記憶空間
                       // 回作業系統，但指標 ptr 變數仍存在
```

new 與 delete（二）

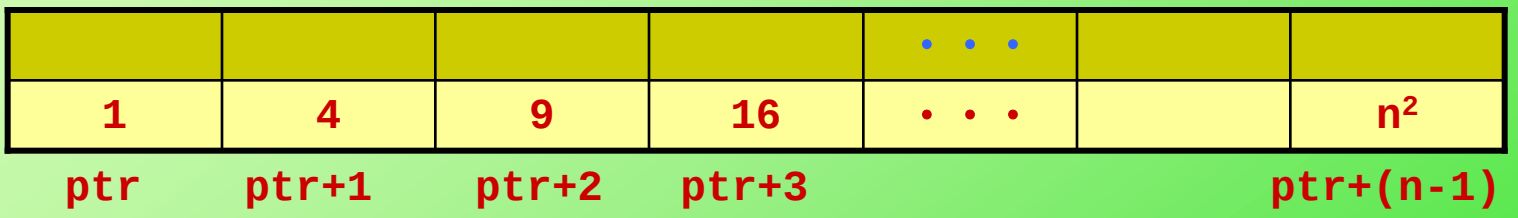
- 圖表表示



new 與 delete（三）

■ 取得 n 個單位的連續記憶空間

```
int  i , n ;
cin >> n ;
int *ptr = new int [n] ;  // ptr 指標指向 n 個連續整數空間的
                          // 首位整數位址
for( i = 0 ; i < n ; ++i )  *(ptr+i) = (i+1)*(i+1) ;
```



❖ 以上的 *(ptr+i) 可以使用 ptr[i] 取代，因此之前迴圈可簡化成

```
for( i = 0 ; i < n ; ++i )  ptr[i] = (i+1)*(i+1) ;
```

■ 歸還一整塊記憶空間

```
delete [] ptr ;   // 歸還指標 ptr 所指向的一整塊動態記憶空間，
                  // 但指標 ptr 仍會存在
```

new 與 delete（四）

- 由 new 取得的記憶空間會持續存在。當程式執行 delete 時，作業系統才會將指標所指向的記憶空間取回重新使用
- 由 new 取來的動態空間要隨時有指標記得這些動態空間的位址。若程式執行中，造成這些記憶空間的位址失去聯繫，這種現象被稱為記憶空間流失 (memory leak)
- 當程式執行結束後，所有使用的動態記憶空間(包含流失的記憶空間)都會全部退還給系統重新使用

```
int  foo = 3 ;
int *ptr = new int(5) ; // 指標 ptr 指向一動態空間，內存有整數 5
cout << *ptr << endl  ; // 列印指標 ptr 所指向的資料值 5

ptr = &foo ;            // 指標 ptr 改指向 foo 變數

cout << *ptr << endl  ; // 列印指標 ptr 所指向的資料值 3
```

記憶空間流失

指標與結構資料型別

- 指標可以使用 -> 來讀取結構資料型別的資料

```
struct Complex {
    double  re,  im ;
};

Complex foo ;                  // 定義一複數變數
Complex *ptr = &foo ;          // 讓一複數指標指到此複數變數

ptr -> re = 2 ;                // 利用指標來設定實數資料
ptr -> im = 5 ;                // 利用指標來設定虛數資料
```

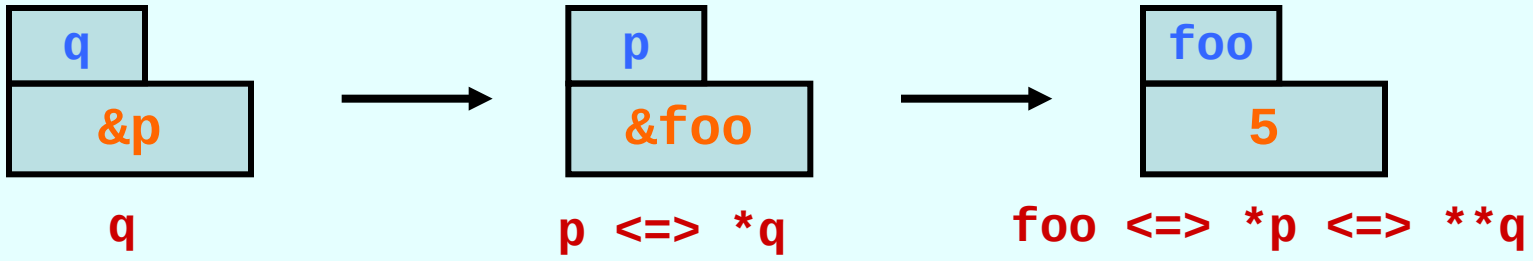
也可以使用以下方式

```
(*ptr).re = 2 ;
(*ptr).im = 5 ;
```

指標的指標（一）

- 指標內所儲存的資料也可以是另一個指標的位址

```
int  foo = 5     ;
int   *p = &foo  ;        // 整數指標 p 指向整數 foo

int  **q = &p ;           // 指標的指標 q 指向整數指標 p
```




```
cout << "foo  = "   << foo   << endl
     << "  *p = "   << *p    << endl
     << " **q = "   << **q   << endl ;
```

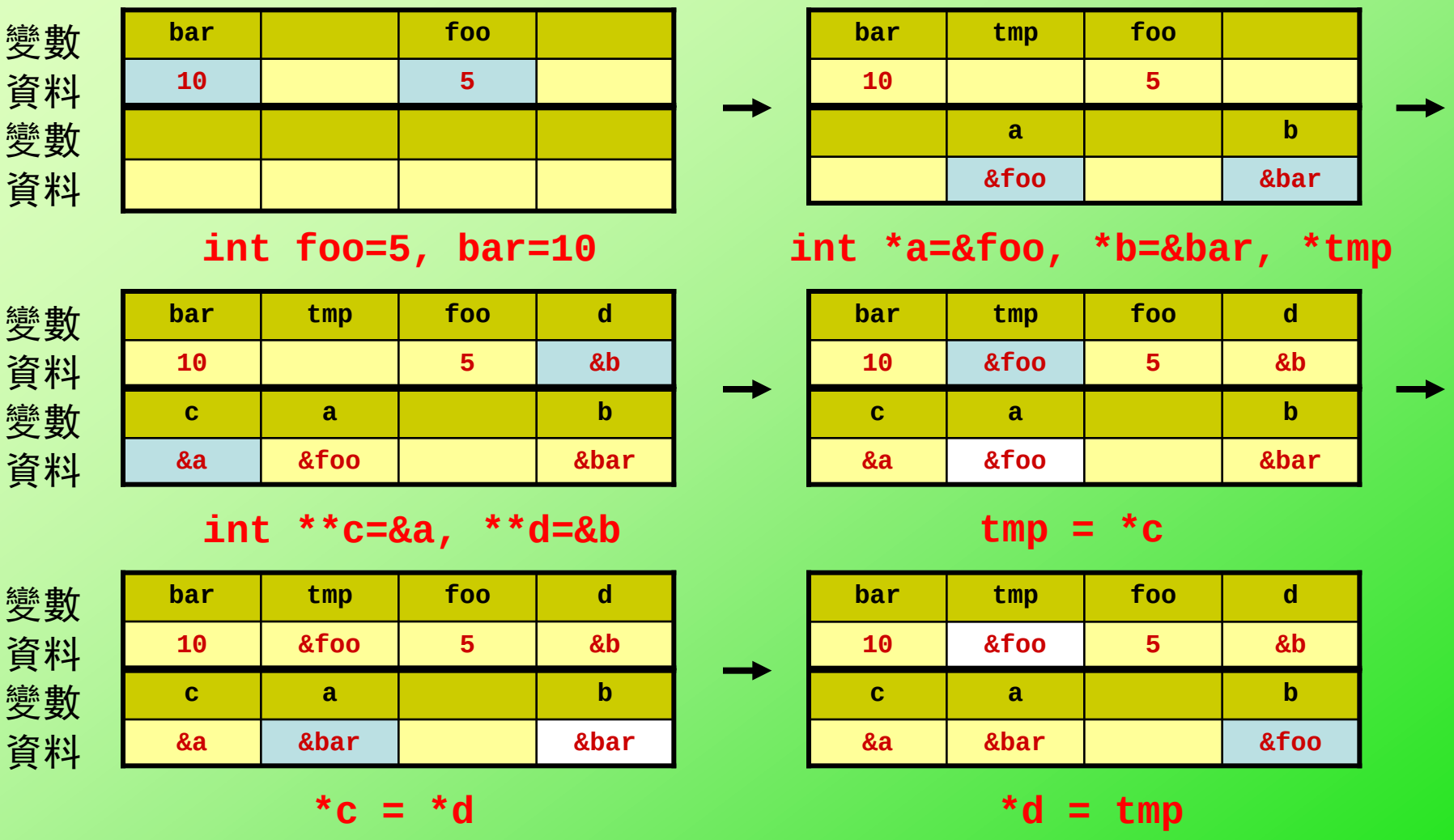
// 都輸出 5

指標的指標（二）

- 利用雙層指標將指標所指到的位址對調

```
int foo  = 5    , bar = 10   ;
int *a   = &foo , *b  = &bar ;    // a 指向 foo ， b 指向 bar

int *tmp ;
int **c = &a , **d = &b ;         // c 指向 a ， d 指向 b
cout << " **c " << **c << '\n'    // 印出 c，d在雙層指標運作後所
     << " **d " << **d << "\n\n"; // 指向的資料值，分別為 5 和 10

tmp = *c  ;                       // tmp 指向 c 所指向的位址
*c  = *d  ;                       // c 指向 d 所指向的位址
*d  = tmp ;                       // d 指向 tmp 所指向的位址
cout << " **c " << **c << '\n'
     << " **d " << **d << endl ;  // 印出 10 和 5
```

指標的指標（三）


變數
資料
變數
資料
bar
foo
10
5
int foo=5, bar=10
bar
tmp
foo
10
5
a
b
&foo
&bar
int *a=&foo, *b=&bar, *tmp
變數
資料
變數
資料
bar
tmp
foo
d
10
5
&b
c
a
b
&a
&foo
&bar
int **c=&a, **d=&b
bar
tmp
foo
d
10
&foo
5
&b
c
a
b
&a
&foo
&bar
tmp = *c
變數
資料
變數
資料
bar
tmp
foo
d
10
&foo
5
&b
c
a
b
&a
&bar
&bar
*c = *d
bar
tmp
foo
d
10
&foo
5
&b
c
a
b
&a
&bar
&foo
*d = tmp

雙層指標與動態矩陣（一）

- 雙層指標可以指向動態產生的矩陣

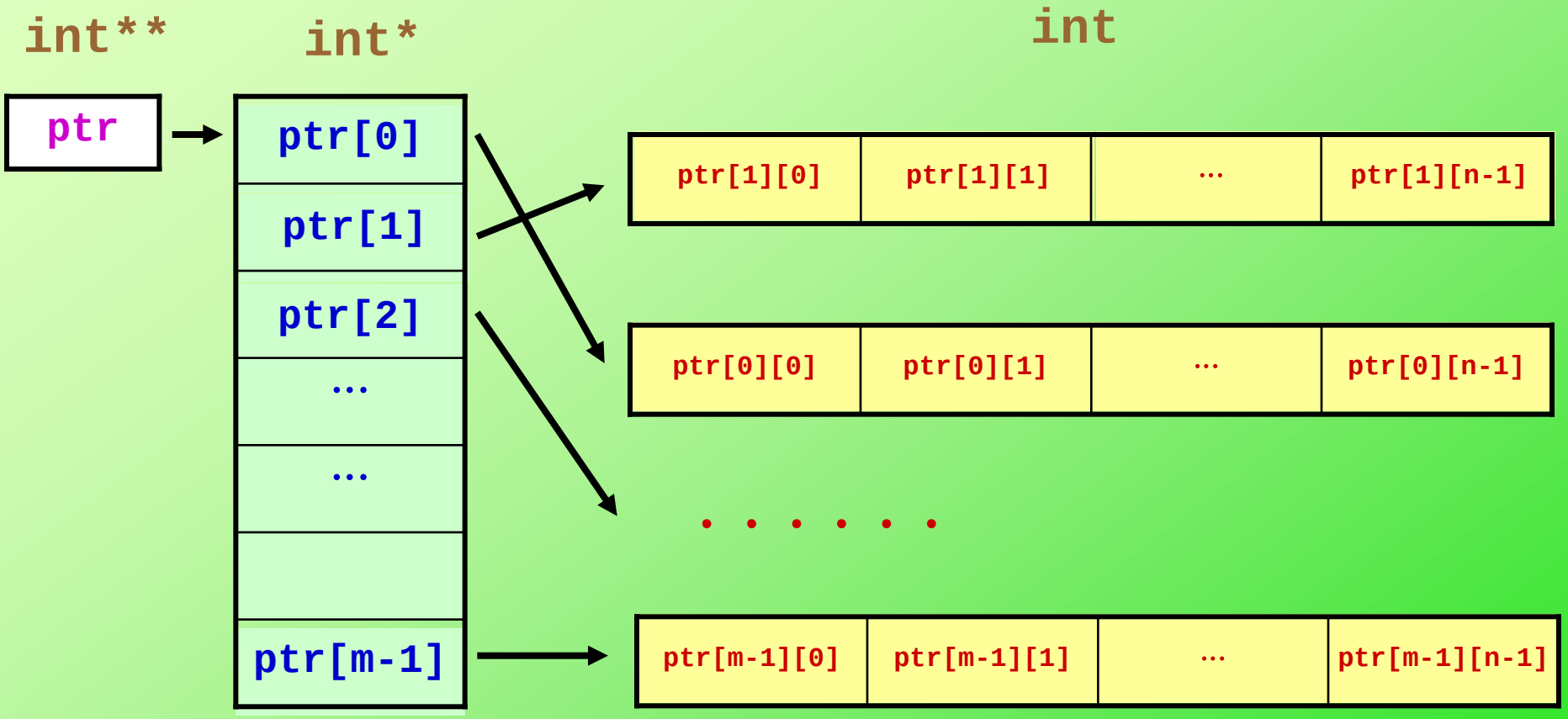
```
int  n  =  10 ;
```

```
// 指標 ptr1 ptr2 … ptrm 分別指向 n 個動態整數記憶空間的起始位址
int  *ptr1 = new int[n] ;
int  *ptr2 = new int[n] ;
 . . .
int  *ptrm = new int[n] ;
```

- 將 m 個指標合併使用

```
// 指標的指標 ptr 指向 m 個動態指標記憶空間的起始位址
int ** ptr = new int*[m] ;

// 讓每個 ptr[i] 指標指向 n 個動態整數記憶空間的起始位址
for( int i = 0 ; i < m ; ++i )   ptr[i] = new int[n] ;
```

雙層指標與動態矩陣（二）




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

```
for ( i = 0 ; i < m ; ++i )
    ptr[i] = new int[n] ;
```

雙層指標與動態矩陣（三）

- 使用雙層指標設定矩陣元素

```
// 將矩陣每一個元素歸零
int r , c ;
for ( r = 0 ; r < m ; ++r ) {
      for ( c = 0 ; c < n ; ++c ) ptr[r][c] = 0 ;
}
```

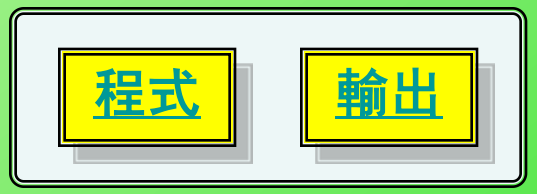
- 歸還動態矩陣記憶空間

```
for ( int r = 0 ; r < m ; ++r ) delete [] ptr[r] ;
delete [] ptr ;
```

九九乘法表：指標版

輸出：

```
1x1= 1 2x1= 2 3x1= 3 4x1= 4 5x1= 5 6x1= 6 7x1= 7 8x1= 8 9x1= 9
1x2= 2 2x2= 4 3x2= 6 4x2= 8 5x2=10 6x2=12 7x2=14 8x2=16 9x2=18
1x3= 3 2x3= 6 3x3= 9 4x3=12 5x3=15 6x3=18 7x3=21 8x3=24 9x3=27
1x4= 4 2x4= 8 3x4=12 4x4=16 5x4=20 6x4=24 7x4=28 8x4=32 9x4=36
1x5= 5 2x5=10 3x5=15 4x5=20 5x5=25 6x5=30 7x5=35 8x5=40 9x5=45
1x6= 6 2x6=12 3x6=18 4x6=24 5x6=30 6x6=36 7x6=42 8x6=48 9x6=54
1x7= 7 2x7=14 3x7=21 4x7=28 5x7=35 6x7=42 7x7=49 8x7=56 9x7=63
1x8= 8 2x8=16 3x8=24 4x8=32 5x8=40 6x8=48 7x8=56 8x8=64 9x8=72
1x9= 9 2x9=18 3x9=27 4x9=36 5x9=45 6x9=54 7x9=63 8x9=72 9x9=81
```

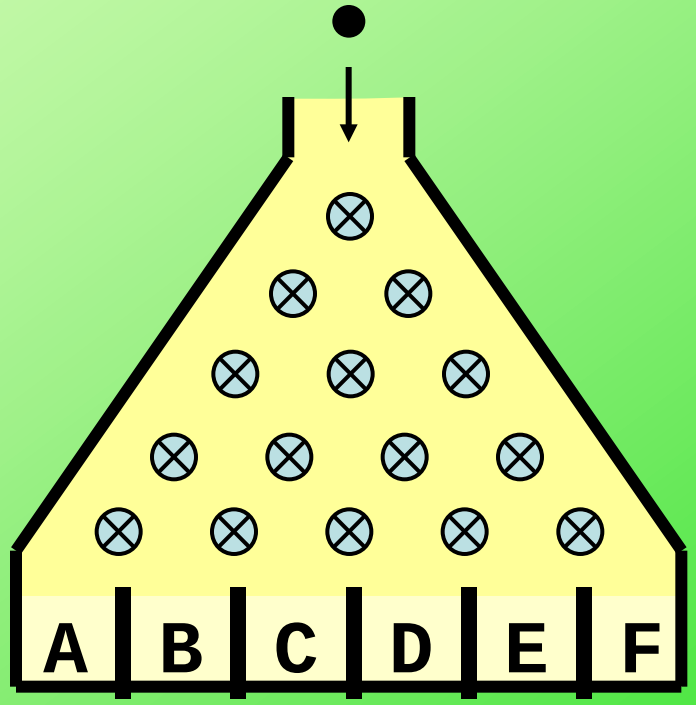


彈珠臺模擬

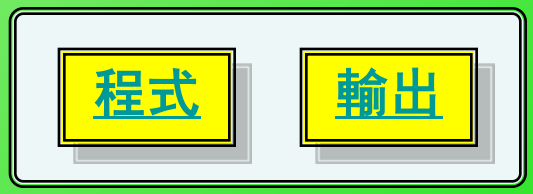
- 彈珠碰到滾輪後會等機會地往左右兩方滾下，求彈珠在各底層位置的機率？

- 作法：

  假設有 M 個彈珠，利用亂數函式決定彈珠在撞到滾輪後往左或往右的方向，統計各位置的彈珠數量，除以 M 即為機率



滾 輪 彈 珠 台 n=5



向量相乘（一）

$$ab = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} \begin{pmatrix} b_0, b_1, \cdots, b_{n-1} \end{pmatrix}$$

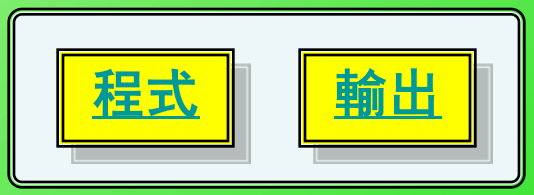
$$= \begin{pmatrix} a_0b_0 , a_0b_1 , \cdots , a_0b_{n-1} \\ a_1b_0 , a_1b_1 , \cdots , a_1b_{n-1} \\ \\ a_{n-1}b_0 , a_{n-1}b_1 , \cdots , a_{n-1}b_{n-1} \end{pmatrix}$$

$$= \begin{pmatrix} M_{0,0} , M_{0,1} , \cdots , M_{0,n-1} \\ M_{1,0} , M_{1,1} , \cdots , M_{1,n-1} \\ \\ M_{n-1,0} , M_{n-1,1} , \cdots , M_{n-1,n-1} \end{pmatrix} = M$$

向量相乘（二）

- 由使用者輸入長度，在程式中以 **new** 向系統取得 n^2 個同型別記憶空間，如下圖：

- 這 n^2 個動態記憶空間在計算機記憶體中是連續排列的



矩陣分解

$$S = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]$$

S ：nxn 方陣

A B C D：mxm 方陣

n = 2m

■ $S = \{s_{i,j}\}$, $A = \{a_{i,j}\}$, $B = \{b_{i,j}\}$ …

則

$a_{i,j} = s_{i,j}$ 0 <= i , j < m

$b_{i,j} = s_{i,j+m}$

$c_{i,j} = s_{i+m,j}$

$d_{i,j} = s_{i+m,j+m}$

程式 輸出