

# 計算機程式設計

## C語言 Recursive

郭忠義

[jykuo@ntut.edu.tw](mailto:jykuo@ntut.edu.tw)

臺北科技大學資訊工程系

# 遞迴概念

## ❑ 遞迴(Recursive)是程式設計重要觀念

- 遞迴函式是使用遞迴觀念建立的函式，可讓程式碼變簡潔。
- 設計遞迴函式需小心，否則易掉入無窮迴圈陷阱。

## ❑ 遞迴設計

- 問題要可拆解成規模較小但性質和原問題完全一樣的問題
- 範圍逐漸縮小到一個終止條件。

## ❑ 遞迴函式特性

- 每次呼叫時，都可使問題範圍逐漸縮小。
- 有一個終止條件，以便結束遞迴執行，否則會有無窮迴圈。

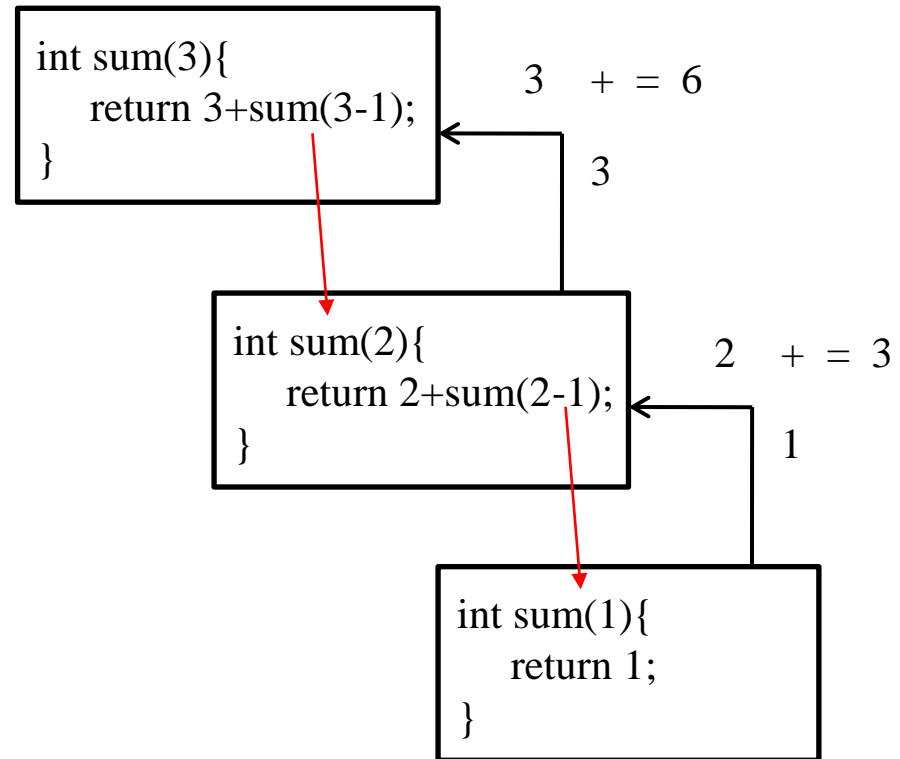
	優點	缺點
遞迴	程式簡潔，節省記憶體空間	參數的堆疊存取較費時
非遞迴	節省執行時間	程式較長，浪費記憶體空間

# 疊代(iterative)與遞迴(recursive)

□ 由 1 加到 3

```
01 int sum(int n) {  
02     int i;  
03     int tmp=0;  
04     for (i=0; i<n; i++)  
05         tmp = tmp + (i+1);  
06     return tmp;  
07 }
```

```
01 int sum(int n) {  
02     if (n==1)  
03         return 1;  
04     else  
05         return n+sum(n-1);  
06 }
```



# 階層

## □ 計算階層

○ 計算 $4!$ ， $n > 0$ ，使用第2條計算

$$n! = \begin{cases} 1 & n=0 \\ n*(n-1)*(n-2)*\dots*1 & n>0 \end{cases}$$

○  $4! = 4*3*2*1 = 24$

○ 將 $4!$ 的計算分解成子問題， $4! = 4*(4-1)! = 4*3!$

○ 將子問題 $3!$ 繼續分解，

➤  $3! = 3*(3-1)! = 3*2!$ ，

➤  $2! = 2*(2-1)! = 2*1!$ ，

➤  $1! = 1*(1-1)! = 1*0! = 1*1 = 1$

○ 最後知道 $1!$ 值，可計算出 $2! \sim 4!$ 的值

➤  $2! = 2*(2-1)! = 2*1! = 2$ ，

➤  $3! = 3*(3-1)! = 3*2! = 3*2 = 6$ ，

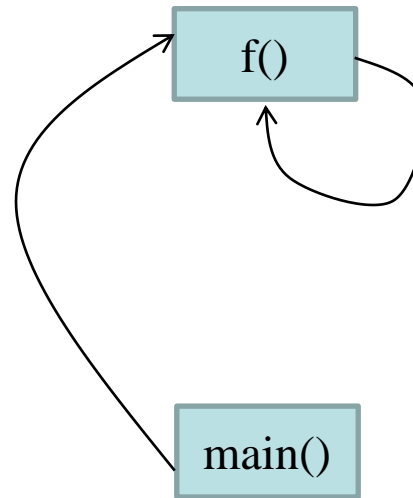
➤  $4! = 4*(4-1)! = 4*3! = 24$

# 遞迴種類

□ 依呼叫遞迴函式位置，分為兩種：

○ 直接遞迴(Direct Recursion)：自己呼叫自己。

```
01 #include <stdio.h>
02 int f(int x) {
03     if (x<0) return 1;
04     return f(x-1)+1;
05 }
06 int main() {
07     printf("%d",f(5));
08     return 0;
09 }
```



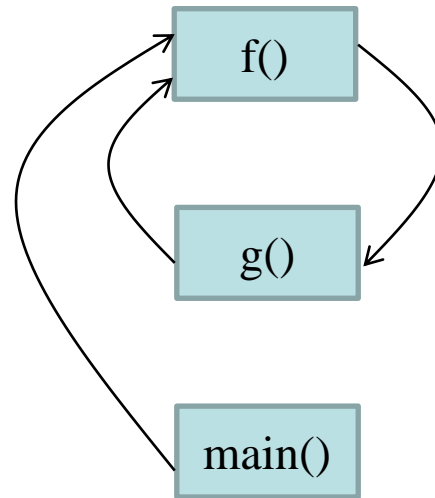
執行順序?執行結果?

# 遞迴種類

□ 依呼叫遞迴函式位置，分為兩種：

- 間接遞迴(Indirect Recursion)：至少需2個函式a()和b()，函式a()呼叫函式b()；函式b()呼叫函數a()。

```
01 #include <stdio.h>
02 int f(int x) {
03     if (x<0) return 1;
04     return g(x-1)+1;
05 }
06 int g(int y) {
07     if (y<0) return 2;
08     return f(y-1)+1;
09 }
10 int main() {
11     printf("%d",f(5));
12     return 0;
13 }
```

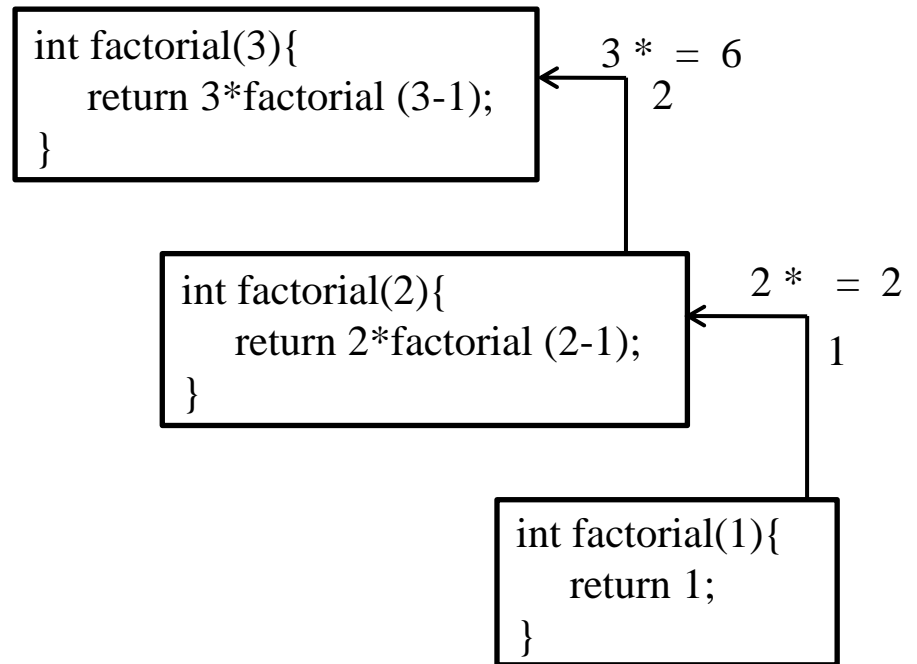


執行順序?執行結果?

# 階層

## □ 計算階層

```
01 int factorial(int n) {  
02   if ( n == 1)  
03     return 1;  
04   else  
05     return n * factorial(n-1);  
06 }
```



# 最大公因數

## ❑ 最大公因數(Greatest Common Divisor, GCD)

- 某幾個整數共有因數中最大的一個。
- 求兩個整數最大公因數：各自列出因數，找出最大公因數。

## ❑ 39, 27的最大公因數 $\Rightarrow 3$

- 39: 1, 3, 13, 39
- 27: 1, 3, 9, 27

## ❑ 輾轉相除法

- $39 \% 27 \Rightarrow 12$
- $27 \% 12 \Rightarrow 3$
- $12 \% 3 \Rightarrow 0$

$h$	$m$
39	27
-27	24
<hr/>	
12	3
12	
-12	
0	



# 最大公因數

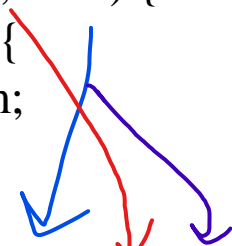
## ❑ 輾轉相除法

○  $39 \% 27 \Rightarrow 12$

○  $27 \% 12 \Rightarrow 3$

○  $12 \% 3 \Rightarrow 0$

```
01 int gcd(int m, int n) {  
02     if(n == 0) {  
03         return m;  
04     }  
05     else {  
06         return gcd(n, m % n);  
07     }  
08 }
```



```
int gcd(39, 27){  
    return gcd(27, 39%27);  
}
```

3

```
int gcd(27, 12){  
    return gcd(12, 27%12);  
}
```

3

```
int gcd(12, 3){  
    return gcd(3, 12%3);  
}
```

```
int gcd(3, 0){  
    return 3;  
}
```

# Exercise

□  $f(x)$

$$f(x) = \begin{cases} 1, & x = 1 \\ f(3 * x + 1), & x \text{ is odd} \\ f\left(\frac{x}{2}\right), & x \text{ is even} \end{cases}$$

□ 輸入K，算出

- 若不是整數，輸出must be integer
- 若不在範圍，輸出out of range
- 若在合法範圍，輸出結果數值X、X的所有數字總和

# Exercise 登山

- 小英要為登玉山準備，練習爬 101大樓的樓梯。
  - 可以一步踏一階、二階、最長跨三階。因此有很多種爬階方法。
  - 要爬到第一階，有一種爬法:  $f(1)=1$ 
    - (1)一次一階。
  - 要爬到第二階，有二種爬法:  $f(2)=2$ 
    - (1)一次一階，爬二次。
    - (2)一次爬二階
  - 要爬到第三階，有四種爬法:  $f(3)=4$ 
    - (1)一次一階，爬三次。
    - (2)先爬一階，再爬二階
    - (3)先爬二階，再爬一階
    - (4)一次三階。

# Exercise 登山

- 小英要為登玉山準備，練習爬 101大樓的樓梯。
  - 要爬到第四階，有七種爬法。
    - (1) 爬到第三階後，再一次爬一階， $f(3) = 4$
    - (2) 爬到第二階後，再一次爬二階， $f(2) = 2$
    - (3) 爬到第一階後，再一次爬三階， $f(1) = 1$
    - 以上三種皆不重複
  - 一次可跨 1~n 階時( $n \geq 1$ )，從第  $n+1$  階開始，其全部方法為前  $n$  階方法數總和。
    - 若  $n \geq 4$ ,  $f(n) = f(n-1) + f(n-2) + f(n-3)$
  - 請考慮使用遞迴與非遞迴方法，或配合指標使用。
  - 請使用 `unsigned long long int x; printf("%lld", x);`

# Exercise 登山

- 最大要爬 70 層。

輸入說明

-----

整數  $K$ ， $K < 70$ ，代表共爬  $K$  階。

輸出說明

-----

整數  $M$ ，代表共有幾種爬法。

Sample Input

-----

4

Sample Output

-----

7

Sample Input

-----

12

Sample Output

-----

927

Sample Input

-----

67

Sample Output

-----

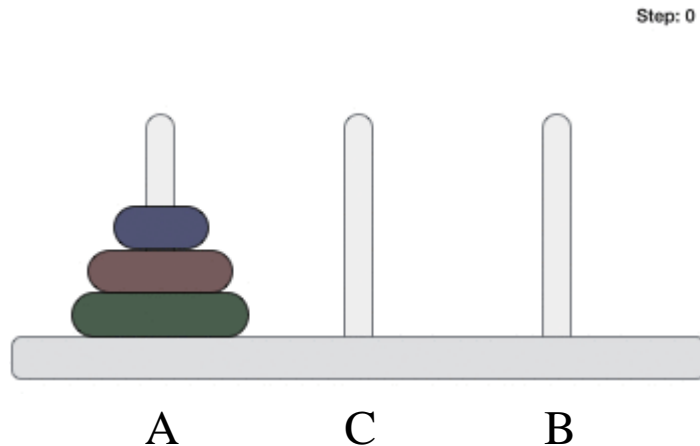
33326997224634006

8

# 河內塔

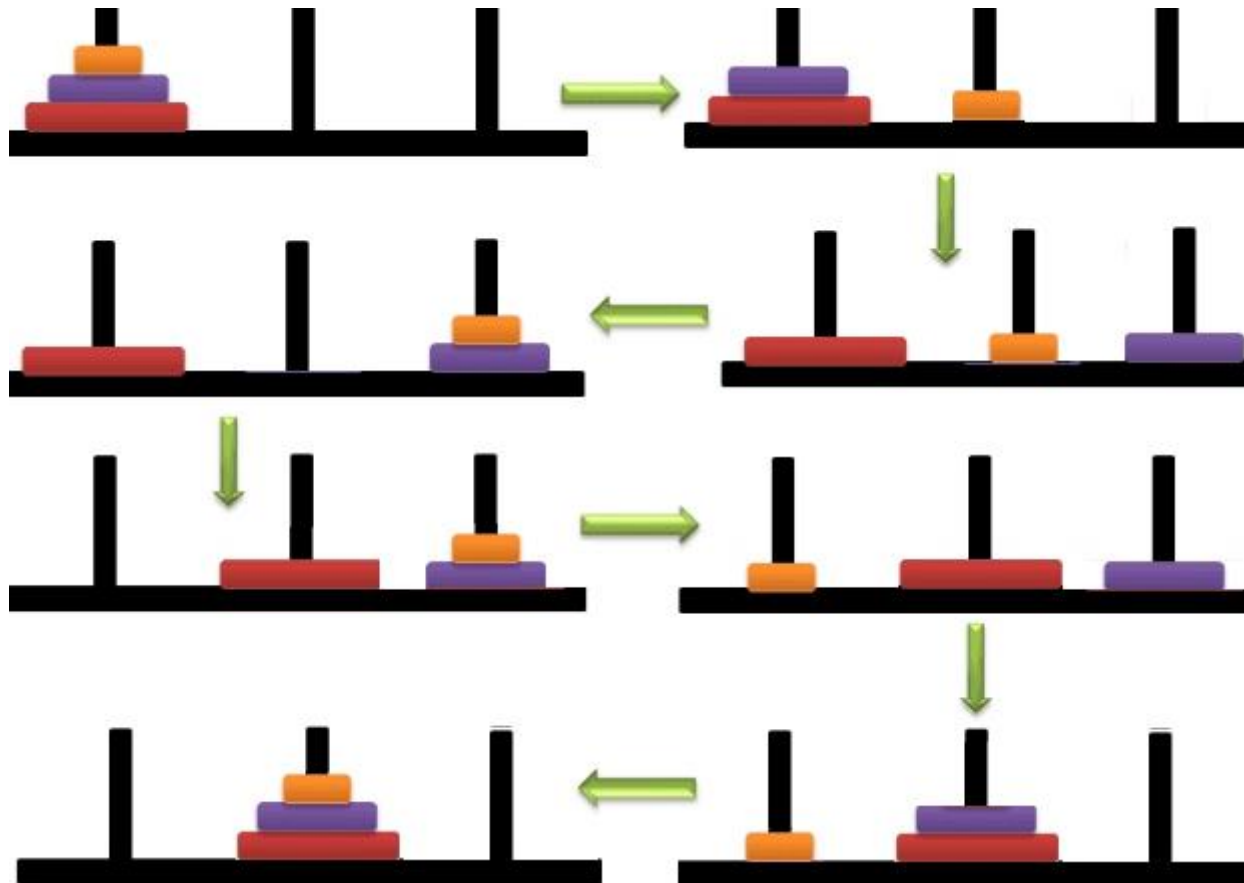
## 問題

- 有三根竿子A，B，C。A竿上有  $N$  個 ( $N > 1$ ) 穿孔圓盤，尺寸由下到上依次變小。按下列規則將所有圓盤移至 C 杆：
  - 每次只能移動一個圓盤。
  - 大盤不能疊在小盤上面。
  - 圓盤可以在任意一個竿子上。
- 可將圓盤臨時置於 B 竿，也可將從 A 竿移出的圓盤重新移回 A 竿，但都須遵循上述規則。



# 河內塔

## □ 問題



A

C

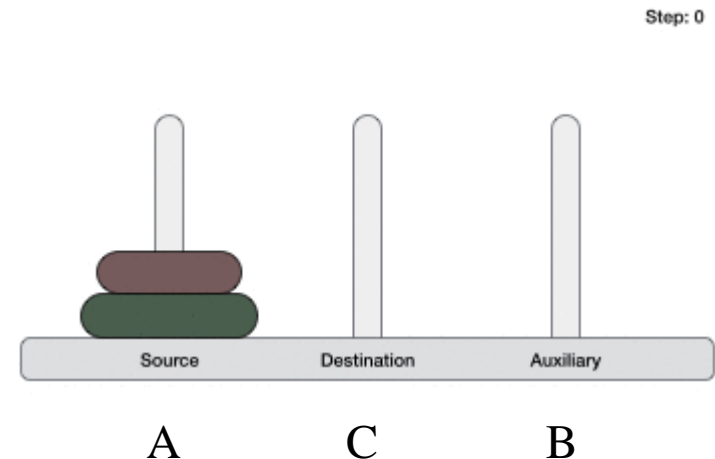
B

# 河內塔

## □ 解決

- 將較小的(頂部)圓盤移動到輔助竿子B。
- 將較大的(底部)圓盤移動到目標竿子C。
- 將較小的圓盤從輔助竿子B移動到目標竿子C。

```
void move(int i , char x , char y) {  
    static int c = 1;  
    printf("%d: %d from %c > %c\n", c++ , i , x , y);  
}  
void hanoi(int i , char A , char C , char B) {  
    if(i == 1) move(i , A , C);  
    else {  
        hanoi(i - 1 , A , B , C);  
        move(i , A , C);  
        hanoi(i - 1 , B , C , A);  
    }  
}  
// Hanoi(n, 'A', 'C', 'B');
```





# Exercise 葛雷碼 (Gray code)

- 反射二進位編碼-葛雷碼 (Gray code)，是編碼成兩個連續的不同位元。
  - 輸入  $n$ ，編碼範圍  $0 \leq i \leq 2^n - 1$ 。
  - $n = 3$ ，編碼 0~7 為 000, 001, 011, 010, 110, 111, 101, 100。
  - 其編碼規則

$$G_1 = \{0, 1\}$$

$$G_{1_r} = \{1, 0\}$$

$$G_n = \{0G_{(n-1)}, 1G_{(n-1)_r}\}$$

$$\text{if } G_n = \{g_1, g_2, g_3, \dots, g_n\}$$

$$G_{n_r} = \{g_n, g_{(n-1)}, g_{(n-2)}, \dots, g_1\}$$

[ $G_{n_r}$  是  $G_n$  的逆向順序]

$$G_{(n+1)} = \{0G_n, 1G_{n_r}\}$$

# 葛雷碼 (Gray code)

## □ 反射二進位編碼-葛雷碼 (Gray code)

例如

$$G_2 = \{0G_1, 1G_{1_r}\} = \{00, 01, 11, 10\}$$

$$G_{2_r} = \{10, 11, 01, 00\}$$

$$G_3 = \{0G_2, 1G_{2_r}\} = \{000, 001, 011, 010, 110, 111, 101, 100\}$$

$$G_{3_r} = \{100, 101, 111, 110, 010, 011, 001, 000\}$$

其遞迴公式為，

$$G(n, k) = k \quad \text{if } n=1$$

$$G(n, k) = 0G(n-1, k) \quad \text{if } k < 2^{(n-1)}$$

$$G(n, k) = 1G(n-1, 2^{n-1}-k) \quad \text{if } k \geq 2^{(n-1)}$$

$$\text{當 } G(4, 7) = 0G(4-1, 7) = 0G(3, 7) = 01G(3-1, 2^{3-1}-7) = 01G(2, 0) = 010G(2-1, 0) = 010G(1, 0) = 0100$$

依此撰寫遞迴程式，輸入n, k，輸出 Gray code。

# 葛雷碼 (Gray code)

## □ 反射二進位編碼-葛雷碼 (Gray code)

### 輸入說明:

第一行是一個測試案例資料，整數  $n$   $k$   
接著是一行 0 分隔測試資料  
第三行是第二個測試案例資料  
最後 -1 結束

### 輸出說明:

二進位 Gray code  
每一行是一個測試案例資料的結果

### Sample Input:

```
1 1
0
2 3
0
3 6
0
4 12
-1
```

### Sample Output:

```
1
10
101
1010
```

# Exercise 數位電路模擬

## □ 模擬一個數位電路。

- 輸入  $n$  是二進位 8 位元，輸出是二進位 4 位元。
- 輸入範圍從 00000000 到 11111111 (十進位 0~255).
- 此數位電路內具有回饋迴路，其功能如下：
  - $C(m) = m$  if  $m = 0$  or  $m = 1$
  - $C(m) = C(m/2)$  if  $m$  is even 偶數
  - $C(m) = C((m+1)/2)$  if  $m$  is odd 奇數
- 此電路有一個紀錄器，會記錄跑過幾次回饋迴路，最後輸出為回饋電路跑過的次數。
  - 例如  $m=00001010$ (十進位 10)，則電路內部運算回饋電路輸入依序為十進位 5, 3, 2, 1。
  - $C(10)=C(5)=C(3)=C(2)=C(1)=1$
  - 共跑過 4 次。則此電路輸出為 0100 (十進位 4)。

# Exercise 數位電路模擬I

□ 模擬一個數位IC，內有回饋電路與紀錄器電路。

○ 輸入m 是二進位 8 位元，輸出是二進位 4 位元。

○ 輸入範圍從 00000000 到 11111111 (十進位 0~255)。

○ 數位IC內有一個回饋電路，回饋方式：

➤  $C(m) = m$  if  $m = 0$  or  $m = 1$  (十進位)

➤  $C(m) = C(m/2)$  if  $m$  偶數(十進位)

➤  $C(m) = C((m+1)/2)$  if  $m$  奇數(十進位)

➤ 例如  $m=00001010$ (十進位 10)，則電路回饋依序為十進位 5, 3, 2, 1。

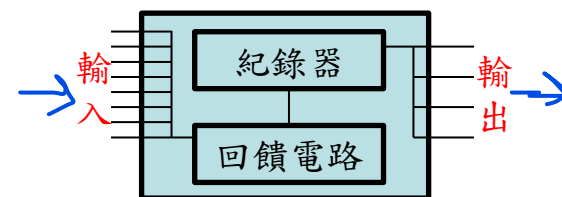
➤  $C(10) = C(5) = C(3) = C(2) = C(1) = 1$ ，共回饋 4 次。

○ 數位IC內有一個紀錄器，會記錄回饋電路的回饋次數。

➤  $R(m) = [C(m) \text{ 的回饋次數}]$ ，例如  $R(10) = 4$ 。

○ 數位IC的輸出為紀錄器所記錄之回饋電路的回饋次數。

➤ 若數位IC的輸入為  $m=00001010$ (十進位 10)，因回饋電路的回饋次數為4，則此數位IC輸出為 0100 (十進位 4)。



# Exercise 數位電路模擬I

- 模擬一個數位IC，內有回饋電路與紀錄器電路。

輸入說明:

二進位 8 bit 位元

第一行是第一個測試案例資料

接著是一行 0 分隔測試資料

第三行是第二個測試案例資料

....

最後 -1 結束

輸出說明:

二進位 4 bit 位元

每一行是一個測試案例資料的結果

Sample Input:

00000000

0

11111111

0

00000001

0

10000000

0

00111111

-1

Sample Output:

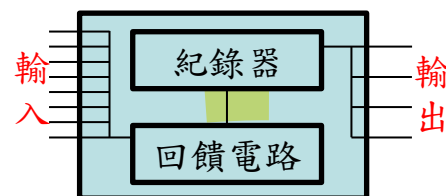
0000

1000

0000

0111

0110



# Exercise 數位電路模擬II

□ 模擬一個數位IC，內有回饋電路與紀錄器電路。

○ 輸入m 是二進位 8 位元，輸出是二進位 4 位元。

○ 輸入範圍從 00000000 到 11111111 (十進位 0~255).

○ 數位IC內有一個回饋電路，回饋方式:

➢  $C(m) = m$  if  $m = 0$  or  $m = 1$  (十進位)

➢  $C(m) = C(m/2)$  if m 偶數(十進位)

➢  $C(m) = C((m+1)/2)$  if m 奇數(十進位)

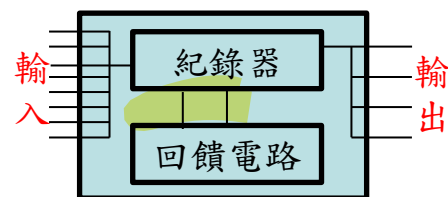
➢ 例如  $m=00001010$ (十進位 10)，則電路回饋依序為十進位 5, 3, 2, 1。

➢  $C(10) = C(5) = C(3) = C(2) = C(1) = 1$ ，共回饋 4 次。

○ 數位IC內有一個紀錄器，其功能

➢ 給予回饋電路輸入 0, 1, 2, ..., m，並記錄每次回饋次數， $R(0)$ ,  $R(1)$ , ...,  $R(m)$ 。例如  $R(10)=4$  [ $C(10)$ 的回饋次數]。

➢ 會累加所有回饋電路的回饋次數。 $Out(10) = R(0)+R(1)+...+R(10)$ 。



# Exercise 數位電路模擬II

- ❑ 模擬一個數位IC，內有回饋電路與紀錄器電路。
  - 數位IC輸出為記錄器累加回饋次數。
    - $m = 3$ ， $R(0)+R(1)+R(2)+R(3) = 0+0+1+2=3$  (二進位 0011)。

輸入說明:

二進位 8 bit 位元

第一行是第一個測試案例資料

接著是一行 0 分隔測試資料

第三行是第二個測試案例資料

....

最後 -1 結束

輸出說明:

二進位 11 bit 位元

每一行是一個測試案例資料的結果

Sample Input:

00000000

0

11111111

0

10101010

-1

Sample Output:

00000000000

11011111001

10001010001

