

# MIS132 - C Lecture 2

程式設計實習課

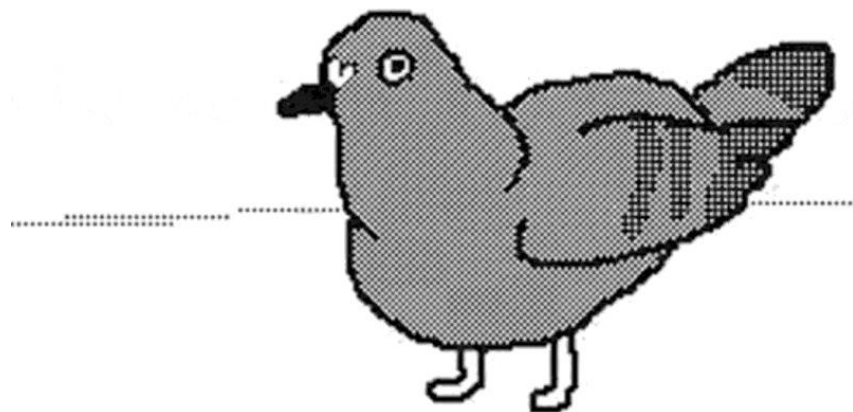
# Outline

- 何謂設計
- 迭代/遞迴
- 陣列/鏈結串列
- Function設計
- 應用演示
- 作業規範
- 練習題

# 何謂設計

程式「設計」包含設計程式的，  
結構、功能、流程、效率、可讀性，  
最終目的是能夠解決特定的問題或滿足特定的需求。

FLY



# 迭代設計

設計的種類for, switch, while, do-while

# 迭代設計

設計的種類 **for**, switch, while, do-while

for ( 初始值 ; 判斷式 ; 遞增、遞減 )

{

程式碼;

}

InClassPractice > C L02\_Show.c > ...

```
1  #include<stdio.h>
2
3  int main (){
4      int count = 1;
5
6      for(int count ; count < 10 ; count++){
7          printf("count to %d\n", count);
8      }
9
10     return 0;
11 }
```

# 迭代設計

設計的種類for, **switch**, while, do-while

switch(變數名稱或運算式) {

case 符合數字或字元:

陳述句一;

break;

case 符合數字或字元:

陳述句二;

break;

default:

陳述三;

break;

}

InClassPractice > C L02\_switch.c > ...

```
1  #include<stdio.h>
2
3  int main (){
4      int i;
5      scanf("%d", &i);
6
7      switch(i) {
8          case 1:
9              // Do something
10             break;
11          case 2:
12              // Do something
13             break;
14          case 100:
15          case 200:
16              // Do something
17             break;
18          default:
19              // Do something
20             break;
21      }
22 }
```

# 迭代設計

設計的種類for, switch, **while**, do-wile

```
while (expression) {  
    statements;  
}
```

```
InClassPractice > C L02_while.c > ...  
1  #include<stdio.h>  
2  
3  int main (){  
4      int count = 1;  
5  
6      while(count < 6){  
7          printf("count to %d\n", count);  
8          count++;  
9      }  
10     return 0;  
11 }
```

# 迭代設計

設計的種類for, switch, while, do-while

```
do {  
    statements;  
} while (expression);
```

InClassPractice > C L02\_dowhile.c > ...

```
1  #include<stdio.h>  
2  
3  int main (){  
4      int count = 1;  
5  
6      do{  
7          printf("count to %d\n", count);  
8          count++;  
9      } while(count < 1);  
10  
11     return 0;  
12 }
```



# 遞迴設計

```
InClassPractice > L02 > C L02_Recurrive.c > ...  
18  
19  int main(){  
20      int n  
21      printf("How many !:");  
22      scanf("%d",&n);  
23      for(int x = 1 ; x < n ; x++){  
24          printf("!");  
25      }  
26      return;  
27  }  
28
```

```
Z:\我的雲端硬碟\中山大學資管  ×  +  ∨  
How many !:10  
!!!!!!!!!!!!  
-----  
Process exited after 3.47 seconds with return value 0  
請按任意鍵繼續 . . . |
```

# 遞迴設計

```
#include <stdio.h>
```

```
void woo(int n) {  
    if (n == 0)  
        return;  
    woo(n - 1);  
    printf("!");  
}
```

```
int main() {  
    int n;  
    printf("How many !: ");  
    scanf("%d", &n);  
    woo(n);  
    return 0;  
}
```

Woo (4) call Woo(3)

# 遞迴設計

```
#include <stdio.h>

void woo(int n) {
    if (n == 0)
        return;
    woo(n - 1);
    printf("!");
}

int main() {
    int n;
    printf("How many !: ");
    scanf("%d", &n);
    woo(n);
    return 0;
}
```

Input : 4

Woo (4) call Woo(3)  
Woo (3) call Woo(2)

# 遞迴設計

```
#include <stdio.h>

void woo(int n) {
    if (n == 0)
        return;
    woo(n - 1);
    printf("!");
}

int main() {
    int n;
    printf("How many !: ");
    scanf("%d", &n);
    woo(n);
    return 0;
}
```

Woo (4) call Woo(3)  
Woo (3) call Woo(2)  
Woo (2) call Woo(1)

# 遞迴設計

```
#include <stdio.h>
```

```
void woo(int n) {  
    if (n == 0)  
        return;  
    woo(n - 1);  
    printf("!");  
}
```

```
int main() {  
    int n;  
    printf("How many !: ");  
    scanf("%d", &n);  
    woo(n);  
    return 0;  
}
```

Woo (4) call Woo(3)  
Woo (3) call Woo(2)  
Woo (2) call Woo(1)  
Woo (1) call Woo(0)

# 遞迴設計

```
#include <stdio.h>
```

```
void woo(int n) {  
    if (n == 0)  
        return;  
    woo(n - 1);  
    printf("!");  
}
```

```
int main() {  
    int n;  
    printf("How many !: ");  
    scanf("%d", &n);  
    woo(n);  
    return 0;  
}
```

```
Woo (4) call Woo(3)  
Woo (3) call Woo(2)  
Woo (2) call Woo(1)  
Woo (1) call Woo(0)  
Woo(0) return;
```

# 遞迴設計

```
#include <stdio.h>

void woo(int n) {
    if (n == 0)
        return;
    woo(n - 1);
    printf("!");
}

int main() {
    int n;
    printf("How many !: ");
    scanf("%d", &n);
    woo(n);
    return 0;
}
```

Woo (4) call Woo(3)  
Woo (3) call Woo(2)  
Woo (2) call Woo(1)  
Woo (1) call Woo(0)  
Woo(0) return;

**Stack**

globals

# 遞迴設計

```
#include <stdio.h>

void woo(int n) {
    if (n == 0)
        return;
    woo(n - 1);
    printf("!");
}

int main() {
    int n;
    printf("How many !: ");
    scanf("%d", &n);
    woo(n);
    return 0;
}
```

Woo (4) call Woo(3)  
Woo (3) call Woo(2)  
Woo (2) call Woo(1)  
Woo (1) call Woo(0)  
Woo(0) return;

## Stack

Woo(4) => "!" + Woo(3)

globals



# 遞迴設計

```
#include <stdio.h>

void woo(int n) {
    if (n == 0)
        return;
    woo(n - 1);
    printf("!");
}

int main() {
    int n;
    printf("How many !: ");
    scanf("%d", &n);
    woo(n);
    return 0;
}
```

Woo (4) call Woo(3)  
Woo (3) call Woo(2)  
Woo (2) call Woo(1)  
Woo (1) call Woo(0)  
Woo(0) return;

## Stack

Woo(3) => "!" + Woo(2)

Woo(4) => "!" + Woo(3)

globals

# 遞迴設計

```
#include <stdio.h>

void woo(int n) {
    if (n == 0)
        return;
    woo(n - 1);
    printf("!");
}

int main() {
    int n;
    printf("How many !: ");
    scanf("%d", &n);
    woo(n);
    return 0;
}
```

Woo (4) call Woo(3)  
Woo (3) call Woo(2)  
Woo (2) call Woo(1)  
Woo (1) call Woo(0)  
Woo(0) return;

## Stack

Woo(2) => "!" + Woo(1)

Woo(3) => "!" + Woo(2)

Woo(4) => "!" + Woo(3)

globals

# 遞迴設計

```
#include <stdio.h>

void woo(int n) {
    if (n == 0)
        return;
    woo(n - 1);
    printf("!");
}

int main() {
    int n;
    printf("How many !: ");
    scanf("%d", &n);
    woo(n);
    return 0;
}
```

Woo (4) call Woo(3)  
Woo (3) call Woo(2)  
Woo (2) call Woo(1)  
Woo (1) call Woo(0)  
Woo(0) return;

## Stack

Woo(1) => "!" + Woo(0)

Woo(2) => "!" + Woo(1)

Woo(3) => "!" + Woo(2)

Woo(4) => "!" + Woo(3)

globals

# 遞迴設計

```
#include <stdio.h>

void woo(int n) {
    if (n == 0)
        return;
    woo(n - 1);
    printf("!");
}

int main() {
    int n;
    printf("How many !: ");
    scanf("%d", &n);
    woo(n);
    return 0;
}
```

Woo (4) call Woo(3)  
Woo (3) call Woo(2)  
Woo (2) call Woo(1)  
Woo (1) call Woo(0)  
Woo(0) return;

Return

## Stack

Woo(0) => done

Woo(1) => "!" + Woo(0)

Woo(2) => "!" + Woo(1)

Woo(3) => "!" + Woo(2)

Woo(4) => "!" + Woo(3)

globals

!!!!

# 遞迴設計

```
#include <stdio.h>

void woo(int n) {
    if (n == 0)
        return;
    woo(n - 1);
    printf("!");
}

int main() {
    int n;
    printf("How many !: ");
    scanf("%d", &n);
    woo(n);
    return 0;
}
```

Woo (4) call Woo(3)  
Woo (3) call Woo(2)  
Woo (2) call Woo(1)  
Woo (1) call Woo(0)  
Woo(0) return;

Return

## Stack

Woo(0) => done

Woo(1) => "!" + Woo(0)

Woo(2) => "!" + Woo(1)

Woo(3) => "!" + Woo(2)

Woo(4) => "!" + Woo(3)

globals

!  
!  
!  
!

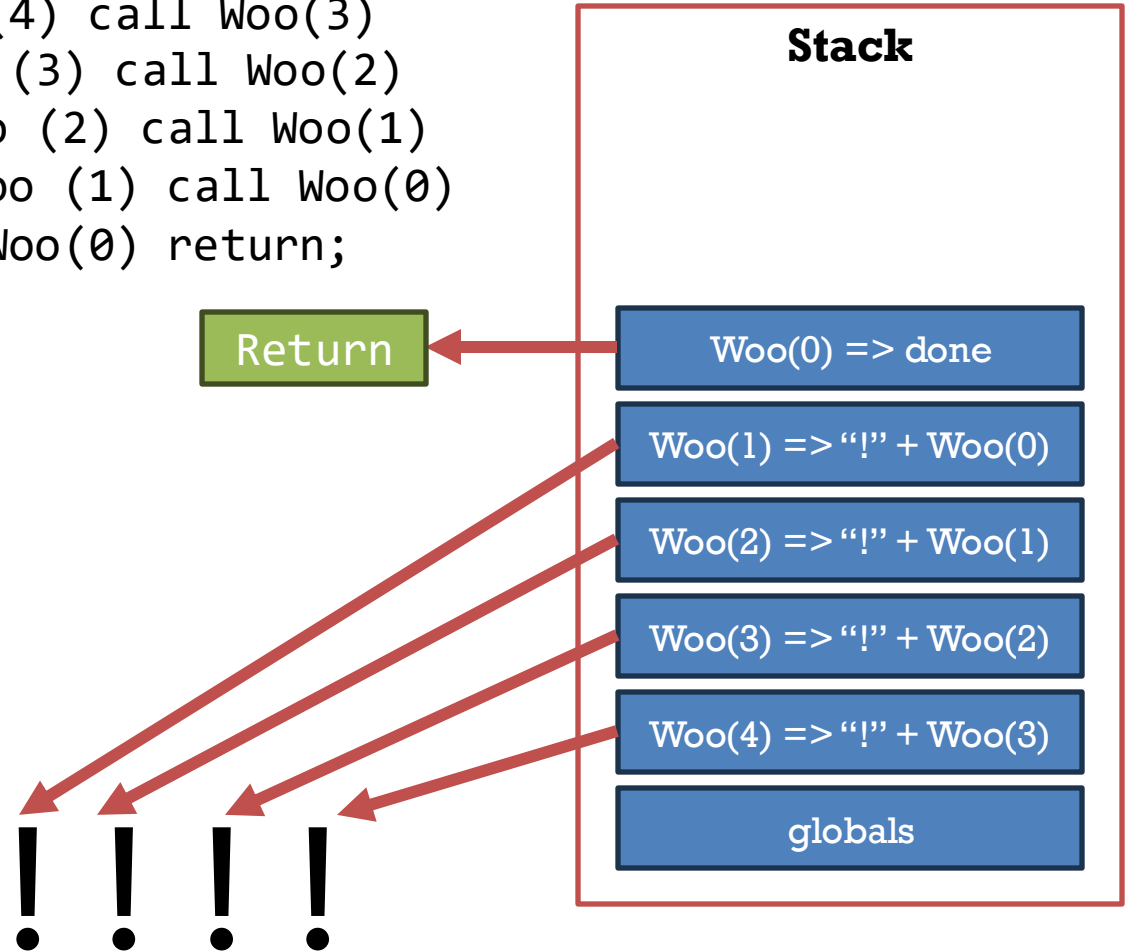
# 遞迴設計

```
#include <stdio.h>

void woo(int n) {
    if (n == 0)
        return;
    woo(n - 1);
    printf("!");
}

int main() {
    int n;
    printf("How many !: ");
    scanf("%d", &n);
    woo(n);
    return 0;
}
```

Woo (4) call Woo(3)  
Woo (3) call Woo(2)  
Woo (2) call Woo(1)  
Woo (1) call Woo(0)  
Woo(0) return;



# 遞迴設計

```
#include <stdio.h>

void woo(int n) {
    if (n == 0)
        return;
    woo(n - 1);
    printf("!");
}

int main() {
    int n;
    printf("How many !: ");
    scanf("%d", &n);
    woo(n);
    return 0;
}
```

What if...

```
Woo (999999999) call Woo(999999998)
Woo (999999998) call Woo(999999997)
Woo (999999997) call Woo(999999996)
Woo (999999996) call Woo(999999995)
Woo (999999995) call Woo(999999994)
Woo (999999994) call Woo(999999993)
Woo (999999993) call Woo(999999992)
Woo (999999992) call Woo(999999991)
Woo (999999991) call Woo(999999990)
Woo (999999990) call Woo(999999989)
Woo (999999989) call Woo(999999988)
Woo (999999988) call Woo(999999987)
Woo (999999987) call Woo(999999986)
Woo (999999986) call Woo(999999985)
.
.
.
```

# 遞迴設計

```
#include <stdio.h>

void woo(int n) {
    if (n == 0)
        return;
    woo(n - 1);
    printf("!");
}

int main() {
    int n;
    printf("How many !: ");
    scanf("%d", &n);
    woo(n);
    return 0;
}
```

What if...

Woo (999999999) call Woo(999999998)

Woo (999999998) call Woo(999999997)

Woo (999999997) call Woo(999999996)

Woo (999999996) call Woo(999999995)

Woo (999999995) call Woo(999999994)

Woo (999999994) call Woo(999999993)

Woo (999999993) call Woo(999999992)

Woo (999999992) call Woo(999999991)

Woo (999999991) call Woo(999999990)

Woo (999999990) call Woo(999999989)

Woo (999999989) call Woo(999999988)

Woo (999999988) call Woo(999999987)

Woo (999999987) call Woo(999999986)

Woo (999999986) call Woo(999999985)

...

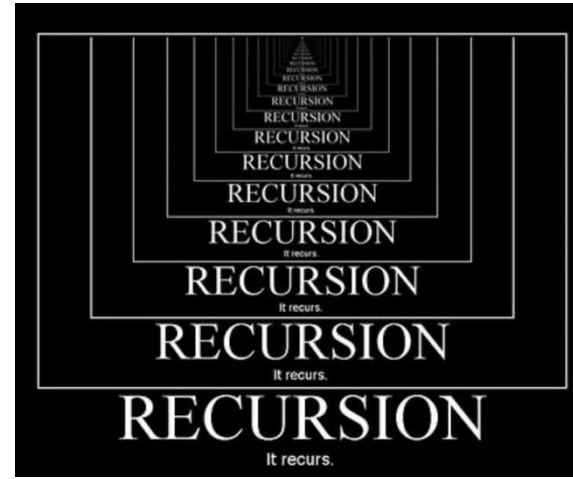


# 遞迴設計

```
#include <stdio.h>

void woo(int n) {
    if (n == 0)
        return;
    woo(n - 1);
    printf("!");
}
```

```
int main()
{
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    woo(n);
    return 0;
}
```



遞迴的概念：[影片連結](#)



# 迭代/遞迴設計

所以我到底該用迭代還是遞迴？

# 迭代/遞迴設計

所以我到底該用迭代還是遞迴？

(1)執行時間：

迭代通常時間較短，不像遞迴需要儲存結果並一直Call Function

(2)所需空間：

迭代不需要像遞迴一樣，一直將資料放在Stack

# 迭代/遞迴設計

所以我到底該用迭代還是遞迴？

(1)執行時間：

迭代**通常**時間較短，不像遞迴需要儲存結果並一直Call Function

(2)所需空間：

迭代不需要像遞迴一樣，一直將資料放在Stack

(3)程式簡潔程度：

遞迴總算扳回一成，沒錯，是遞迴我比較簡潔。

# 迭代/遞迴設計

使用Fibonacci舉例：

$$F_n \begin{cases} n = 0, & \text{if } n = 0 \\ n = 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{otherwise } (n > 1) \end{cases}$$

n	0	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	1	2	3	5	8	13	21	34	55

# 迭代/遞迴設計

使用Fibonacci舉例：

$$F_n = \begin{cases} n = 0, & \text{if } n = 0 \\ n = 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{otherwise } (n > 1) \end{cases}$$

n	0	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	1	2	3	5	8	13	21	34	55

# 迭代/遞迴設計

使用Fibonacci舉例：

$$F_n \begin{cases} n = 0, & \text{if } n = 0 \\ n = 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{otherwise } (n > 1) \end{cases} \longrightarrow f(n) == f(n-1) + f(n-2), \quad (n > 1) \quad [\text{遞迴關係}]$$

n	0	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	1	2	3	5	8	13	21	34	55

# 迭代/遞迴設計

使用Fibonacci舉例：

$$F_n = \begin{cases} n = 0, & \text{if } n = 0 \\ n = 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{otherwise } (n > 1) \end{cases}$$


n	0	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	1	2	3	5	8	13	21	34	55



# 迭代/遞迴設計

使用Fibonacci舉例：

$$F_n = \begin{cases} n = 0, & \text{if } n = 0 \\ n = 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{otherwise } (n > 1) \end{cases}$$

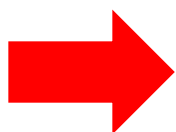
  $f(n) == n, \quad (n \leq 1) \quad [\text{邊界條件}]$

n	0	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	1	2	3	5	8	13	21	34	55

# 迭代/遞迴設計

使用Fibonacci舉例：

$$F_n \begin{cases} n = 0, & \text{if } n = 0 \\ n = 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{otherwise } (n > 1) \end{cases}$$



```
int f(int n) {  
    if (n <= 1)    return n;           // [邊界條件]  
    else return f(n-1) + f(n-2);       // [遞迴關係]  
}
```

n	0	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	1	2	3	5	8	13	21	34	55

# 迭代/遞迴設計

使用Fibonacci舉例：

$$F_n \begin{cases} n = 0, & \text{if } n = 0 \\ n = 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{otherwise } (n > 1) \end{cases} \longrightarrow f(n-1) + f(n-2) == f(n), \quad (n > 1) \quad [\text{迭代關係}]$$

n	0	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	1	2	3	5	8	13	21	34	55

# 迭代/遞迴設計

使用Fibonacci舉例：

$$F_n = \begin{cases} n = 0, & \text{if } n = 0 \\ n = 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{otherwise } (n > 1) \end{cases}$$

n	0	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	1	2	3	5	8	13	21	34	55

# 迭代/遞迴設計

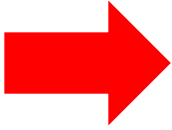
使用Fibonacci舉例：

$$F_n = \begin{cases} n = 0, & \text{if } n = 0 \\ n = 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{otherwise } (n > 1) \end{cases} \rightarrow n == f(n), \quad (n \leq 1) \quad [\text{初始條件}]$$

n	0	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	1	2	3	5	8	13	21	34	55

# 迭代/遞迴設計

使用Fibonacci舉例：

$$F_n \begin{cases} n = 0, & \text{if } n = 0 \\ n = 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{otherwise } (n > 1) \end{cases}$$


```
int Fib[0]=0, Fib[1]=1; //初始條件
if (n==0) return 0;
elseif (n==1) return 1;
else {
    for (i=2; i<=n; i++) //迭代關係
    {
        Fib[i]=Fib[i-1]+Fib[i-2];
    }
    return Fib[n]
}
```

n	0	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	1	2	3	5	8	13	21	34	55

# 迭代/遞迴設計

$$F_n = \begin{cases} n = 0, & \text{if } n = 0 \\ n = 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{otherwise } (n > 1) \end{cases}$$

n	0	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	1	2	3	5	8	13	21	34	55

迭代：

```
int Fib[0]=0, Fib[1]=1; //初始條件
if (n==0) return 0;
elseif (n==1) return 1;
else {
    for (i=2; i<=n; i++) //迭代關係
    {
        Fib[i]=Fib[i-1]+Fib[i-2];
    }
    return Fib[n]
}
```

遞迴：

```
int f(int n) {
    if (n <= 1) return n; // [邊界條件]
    else return f(n-1) + f(n-2); // [遞迴關係]
}
```

# 迭代/遞迴設計

$$F_n = \begin{cases} n = 0, & \text{if } n = 0 \\ n = 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{otherwise } (n > 1) \end{cases}$$

n	0	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	1	2	3	5	8	13	21	34	55

思考一下不同之處



迭代：

```
int Fib[0]=0, Fib[1]=1; //初始條件
if (n==0) return 0;
elseif (n==1) return 1;
else {
    for (i=2; i<=n; i++) //迭代關係
    {
        Fib[i]=Fib[i-1]+Fib[i-2];
    }
    return Fib[n]
}
```

遞迴：

```
int f(int n) {
    if (n <= 1) return n; // [邊界條件]
    else return f(n-1) + f(n-2); // [遞迴關係]
}
```



# 迭代/遞迴設計

$$F_n = \begin{cases} n = 0, & \text{if } n = 0 \\ n = 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{otherwise } (n > 1) \end{cases}$$

n	0	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	1	2	3	5	8	13	21	34	55

思考一下不同之處



迭代：

```
int Fib[0]=0, Fib[1]=1; //初始條件
if (n==0) return 0;
elseif (n==1) return 1;
else {
    for (i=2; i<=n; i++) //迭代關係
    {
        Fib[i]=Fib[i-1]+Fib[i-2];
    }
    return Fib[n]
}
```

遞迴：

```
int f(int n) {
    if (n <= 1) return n;           // [邊界條件]
    else return f(n-1) + f(n-2);    // [遞迴關係]
}
```

1. 執行時間：
2. 所需空間：
3. 程式簡潔程度：

# 迭代/遞迴設計

$$F_n = \begin{cases} n = 0, & \text{if } n = 0 \\ n = 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{otherwise } (n > 1) \end{cases}$$

n	0	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	1	2	3	5	8	13	21	34	55

思考一下不同之處



迭代：

```
int Fib[0]=0, Fib[1]=1; //初始條件
if (n==0) return 0;
elseif (n==1) return 1;
else {
    for (i=2; i<=n; i++) //迭代關係
    {
        Fib[i]=Fib[i-1]+Fib[i-2];
    }
    return Fib[n]
}
```

遞迴：

```
int f(int n) {
    if (n <= 1) return n;           // [邊界條件]
    else return f(n-1) + f(n-2);    // [遞迴關係]
}
```

1. 執行時間：迭代勝
2. 所需空間：
3. 程式簡潔程度：

# 迭代/遞迴設計

$$F_n = \begin{cases} n = 0, & \text{if } n = 0 \\ n = 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{otherwise } (n > 1) \end{cases}$$

n	0	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	1	2	3	5	8	13	21	34	55

思考一下不同之處



迭代：

```
int Fib[0]=0, Fib[1]=1; //初始條件
if (n==0) return 0;
elseif (n==1) return 1;
else {
    for (i=2; i<=n; i++) //迭代關係
    {
        Fib[i]=Fib[i-1]+Fib[i-2];
    }
    return Fib[n]
}
```

遞迴：

```
int f(int n) {
    if (n <= 1) return n;           // [邊界條件]
    else return f(n-1) + f(n-2);    // [遞迴關係]
}
```

1. 執行時間：**迭代勝**
2. 所需空間=>暫存/區域變數：\_\_\_\_\_；Stack儲存空間：\_\_\_\_\_
3. 程式簡潔程度：

# 迭代/遞迴設計

$$F_n = \begin{cases} n = 0, & \text{if } n = 0 \\ n = 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{otherwise } (n > 1) \end{cases}$$

n	0	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	1	2	3	5	8	13	21	34	55

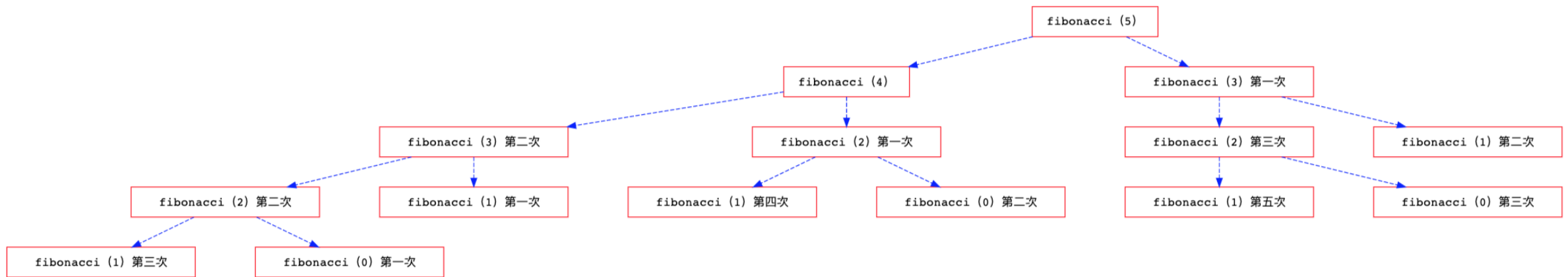
思考一下不同之處

迭代：

```
int Fib[0]=0, Fib[1]=1; //初始條件
```

遞迴：

```
int f(int n) {
```



# 迭代/遞迴設計

$$F_n \begin{cases} n = 0, & \text{if } n = 0 \\ n = 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{otherwise } (n > 1) \end{cases}$$

n	0	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	1	2	3	5	8	13	21	34	55

思考一下不同之處



迭代：

```
int Fib[0]=0, Fib[1]=1; //初始條件
if (n==0) return 0;
elseif (n==1) return 1;
else {
    for (i=2; i<=n; i++) //迭代關係
    {
        Fib[i]=Fib[i-1]+Fib[i-2];
    }
    return Fib[n]
}
```

遞迴：

```
int f(int n) {
    if (n <= 1) return n;           // [邊界條件]
    else return f(n-1) + f(n-2);    // [遞迴關係]
}
```

1. 執行時間：迭代勝
2. 所需空間：迭代省
3. 程式簡潔程度：

# 迭代/遞迴設計

$$F_n = \begin{cases} n = 0, & \text{if } n = 0 \\ n = 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{otherwise } (n > 1) \end{cases}$$

n	0	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	1	2	3	5	8	13	21	34	55

思考一下不同之處

迭代：

```
int Fib[0]=0, Fib[1]=1; //初始條件
if (n==0) return 0;
elseif (n==1) return 1;
else {
    for (i=2; i<=n; i++) //迭代關係
    {
        Fib[i]=Fib[i-1]+Fib[i-2];
        return Fib[n]
    }
}
```

遞迴：

```
int f(int n, int *F, bool *visited) {
    if (visited[n]) return F[n];
    visited[n] = true;
    if (n <= 1) {
        F[n] = n;
    } else {
        F[n] = f(n-1, F, visited) + f(n-2, F, visited);
    }
    return F[n];
}

int main() {
    int F[10+1];
    bool visited[10+1] = {};
    printf("%d\n", f(10, F, visited));
    return 0;
}
```

# 迭代/遞迴設計

$$F_n = \begin{cases} n = 0, & \text{if } n = 0 \\ n = 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{otherwise } (n > 1) \end{cases}$$

n	0	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	1	2	3	5	8	13	21	34	55

思考一下不同之處



迭代：

```
int Fib[0]=0, Fib[1]=1; //初始條件
if (n==0) return 0;
elseif (n==1) return 1;
else {
    for (i=2; i<=n; i++) //迭代關係
    {
        Fib[i]=Fib[i-1]+Fib[i-2];
    }
    return Fib[n]
}
```

遞迴：

```
int f(int n, int a, int b) {
    if (n == 0) return a;
    return f(n - 1, b, a + b);
}
```

# 迭代/遞迴設計

$$F_n = \begin{cases} n = 0, & \text{if } n = 0 \\ n = 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{otherwise } (n > 1) \end{cases}$$

n	0	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	1	2	3	5	8	13	21	34	55

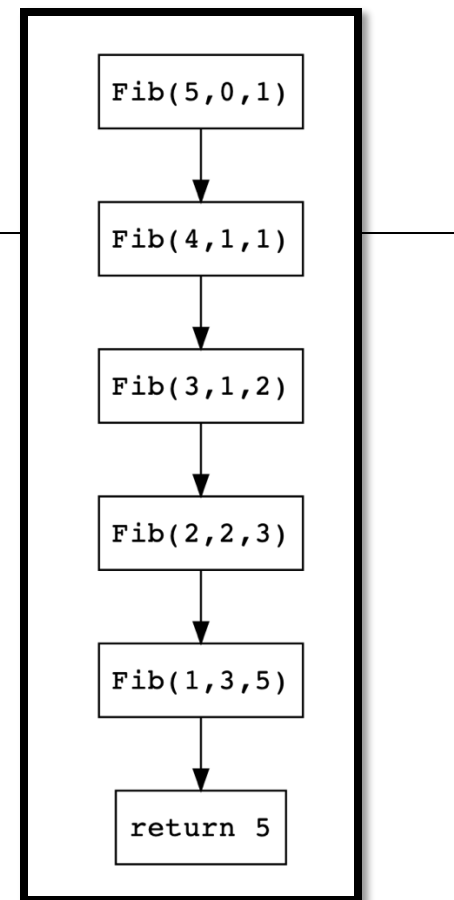
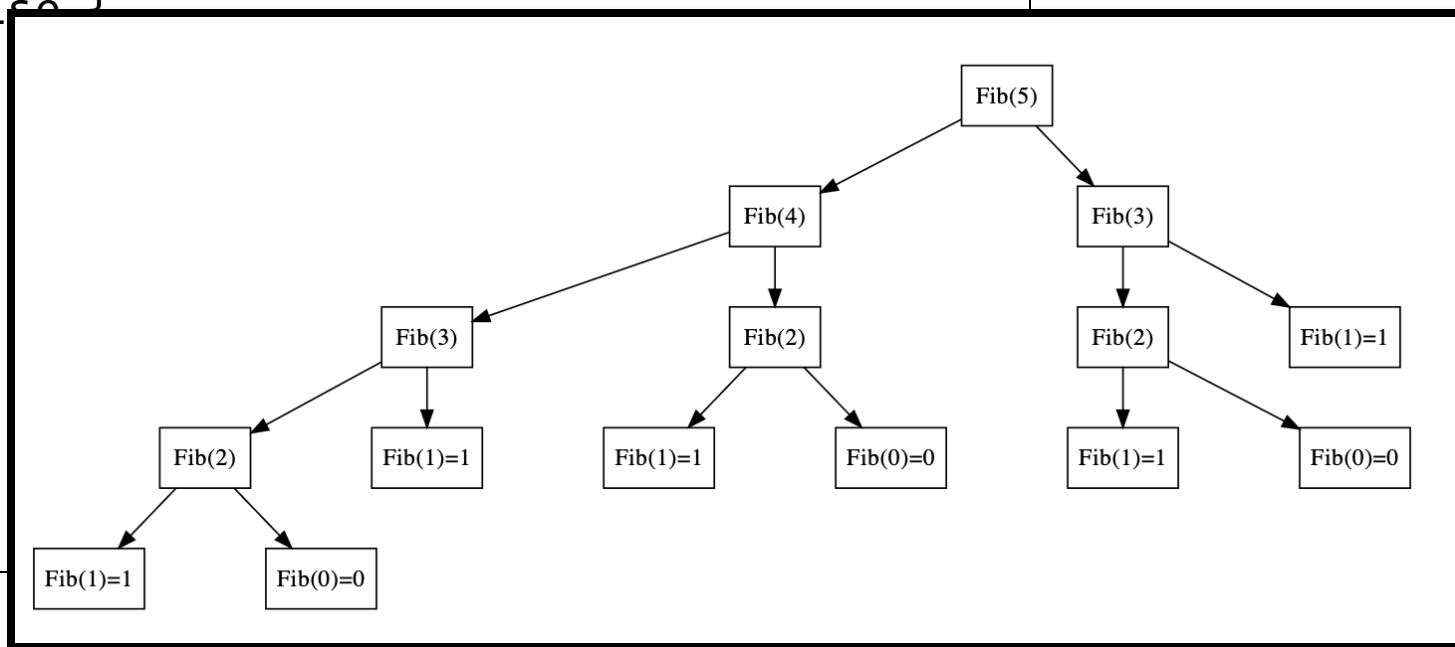
思考一下不同之處

迭代：

```
int Fib[0]=0, Fib[1]=1; //初始條件
if (n==0) return 0;
elseif (n==1) return 1;
else {
```

遞迴：

```
int f(int n, int a, int b) {
    if (n == 0) return a;
    return f(n - 1, b, a + b);
}
```





# 迭代/遞迴設計

$$F_n = \begin{cases} n = 0, & \text{if } n = 0 \\ n = 1, & \text{if } n = 1 \\ F_{n-1} + F_{n-2}, & \text{otherwise } (n > 1) \end{cases}$$

n	0	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	1	2	3	5	8	13	21	34	55

思考一下不同之處



迭代：

```
int Fib[0]=0, Fib[1]=1; //初始條件
if (n==0) return 0;
elseif (n==1) return 1;
else {
    for (i=2; i<=n; i++) //迭代關係
    {
        Fib[i]=Fib[i-1]+Fib[i-2];
    }
    return Fib[n]
}
```

遞迴：

```
int f(int n, int a, int b) {
    if (n == 0) return a;
    return f(n - 1, b, a + b);
}
```

1. 執行時間：遞迴勝
2. 所需空間：遞迴Stack空間大幅減少
3. 程式簡潔程度：遞迴省、扼要易懂、貼近數學原式

# 迭代/遞迴設計

所以我到底該用迭代還是遞迴？

# 迭代/遞迴設計

所以我到底該用迭代還是遞迴？

ANS：取決於問題的需求

# 迭代/遞迴設計

所以我到底該用迭代還是遞迴？

ANS：取決於問題的需求

若問題本身具有順序排列的特性(如1, 2, 3...)

迭代關係能夠得到較好地發揮與處理，

反之若問題本身複雜且無明確的順序特性

則通常遞迴能夠有比較好的發揮與處理(將大問題拆分成小問題)

# 迭代/遞迴設計

所以我到底該用迭代還是遞迴？

ANS：取決於問題的需求

若問題本身具有順序排列的特性(如1, 2, 3...)

迭代關係能夠得到較好地發揮與處理，

反之若問題本身複雜且無明確的順序特性

則通常遞迴能夠有比較好的發揮與處理(將大問題拆分成小問題)

n	0	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	1	2	3	5	8	13	21	34	55

# 陣列設計

# 陣列設計

一維陣列：`int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}`

二維陣列：`int b[2][3] = {{1, 2, 3}, {4, 5, 6}}`

多維陣列：`int c[2][2][2] = {1, 2, 3, 4, 5, 6, 7, 8}`

	0	1	2
0	9	8	7
1	6	5	4

# 鏈結串列設計

結構：

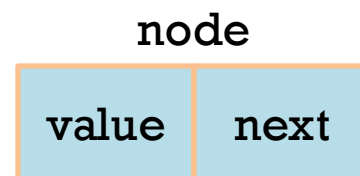
```
1 struct nodes{  
2     int value;  
3     struct nodes *next;  
4 };
```



# 鏈結串列設計

結構：

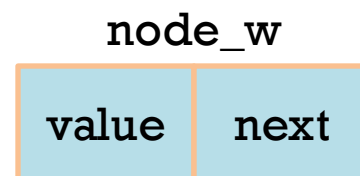
```
1 struct nodes{  
2     int value;  
3     struct nodes *next;  
4 };
```



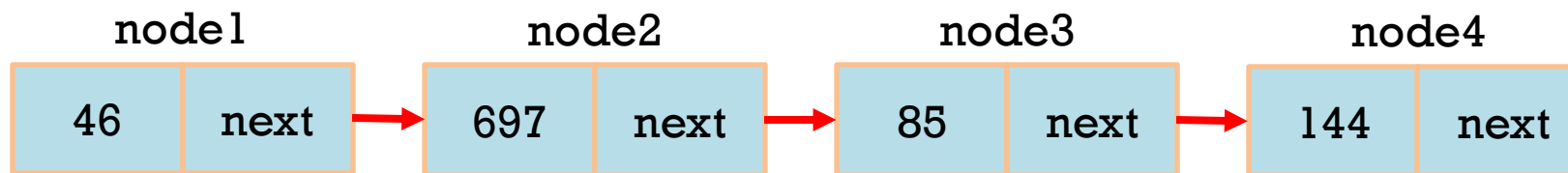
# 鏈結串列設計

結構：

```
1 struct nodes{  
2     int value;  
3     struct nodes *next;  
4 };
```



插入：

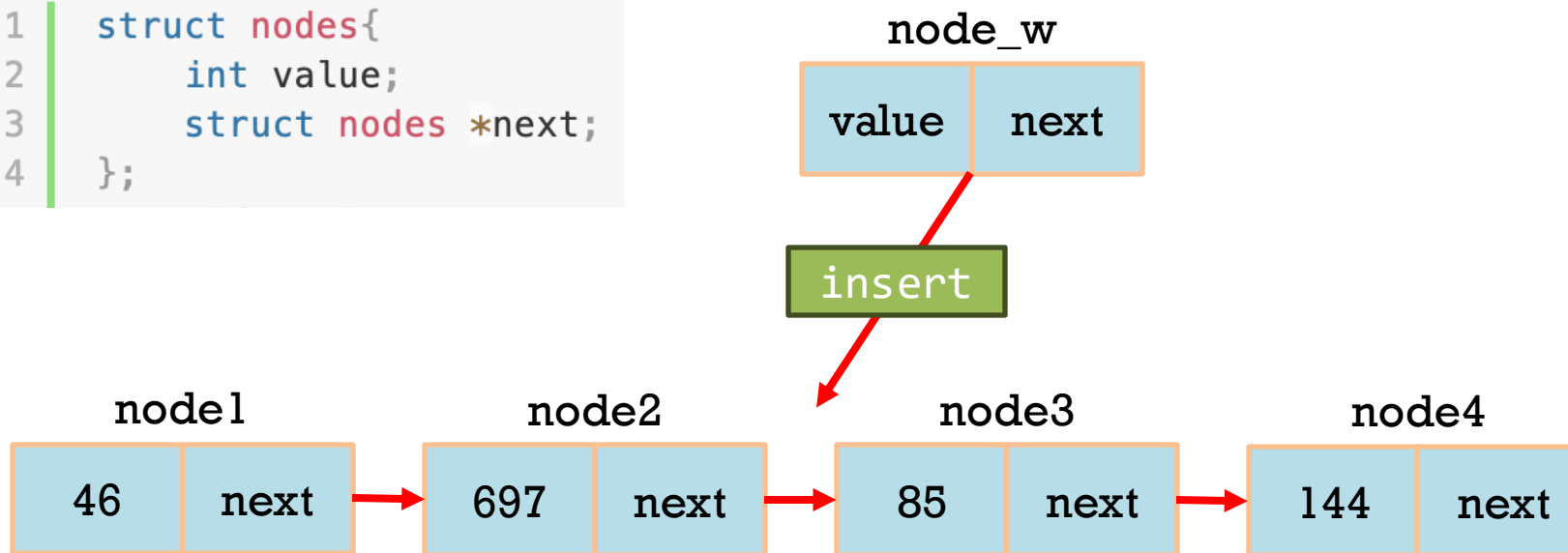


# 鏈結串列設計

結構：

```
1 struct nodes{  
2     int value;  
3     struct nodes *next;  
4 };
```

插入：

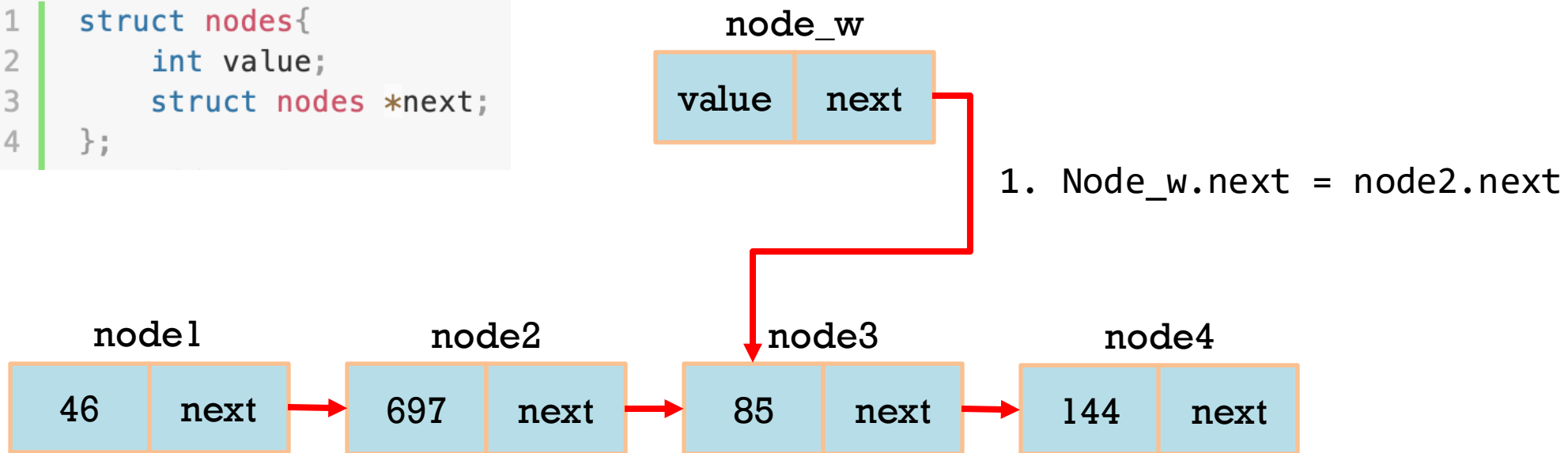


# 鏈結串列設計

結構：

```
1 struct nodes{  
2     int value;  
3     struct nodes *next;  
4 };
```

插入：

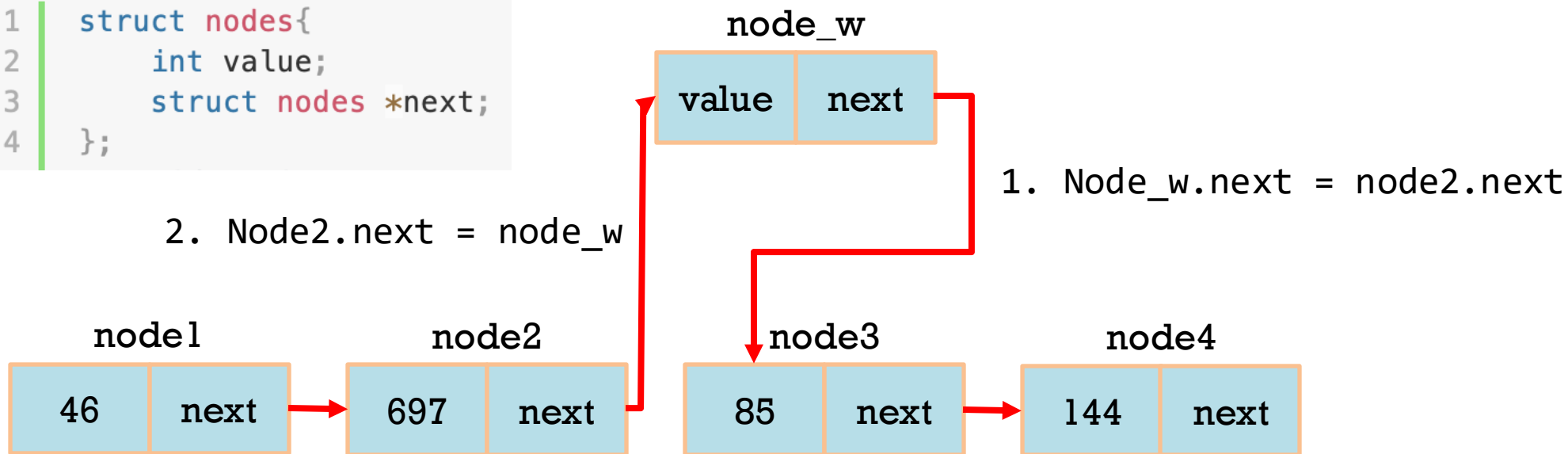


# 鏈結串列設計

結構：

```
1 struct nodes{  
2     int value;  
3     struct nodes *next;  
4 };
```

插入：

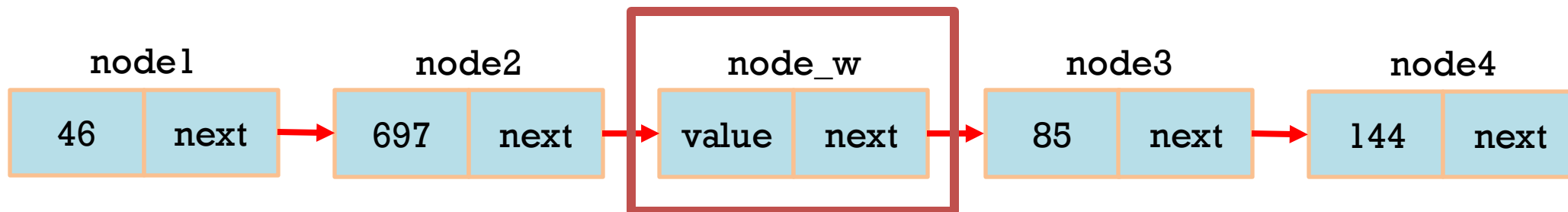


# 鏈結串列設計

結構：

```
1 struct nodes{  
2     int value;  
3     struct nodes *next;  
4 };
```

刪除：

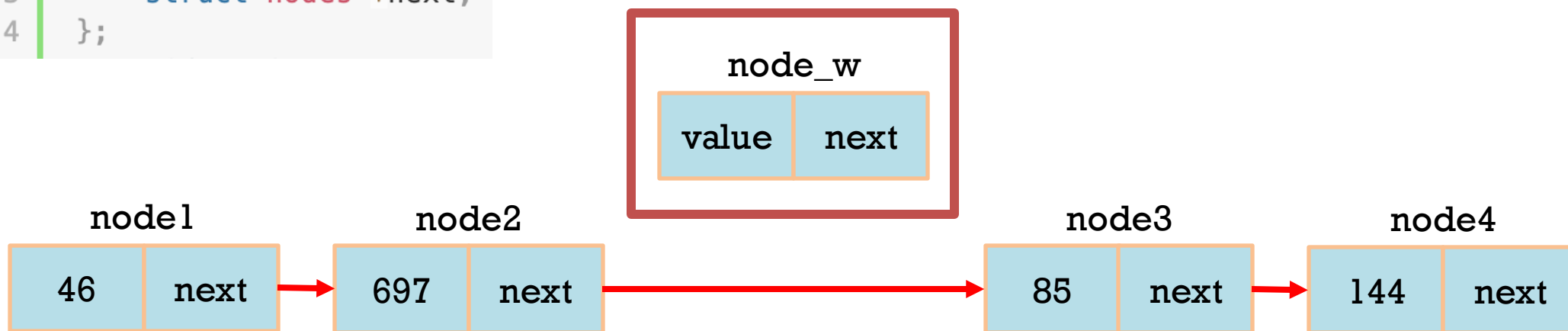


# 鏈結串列設計

結構：

```
1 struct nodes{  
2     int value;  
3     struct nodes *next;  
4 };
```

刪除：

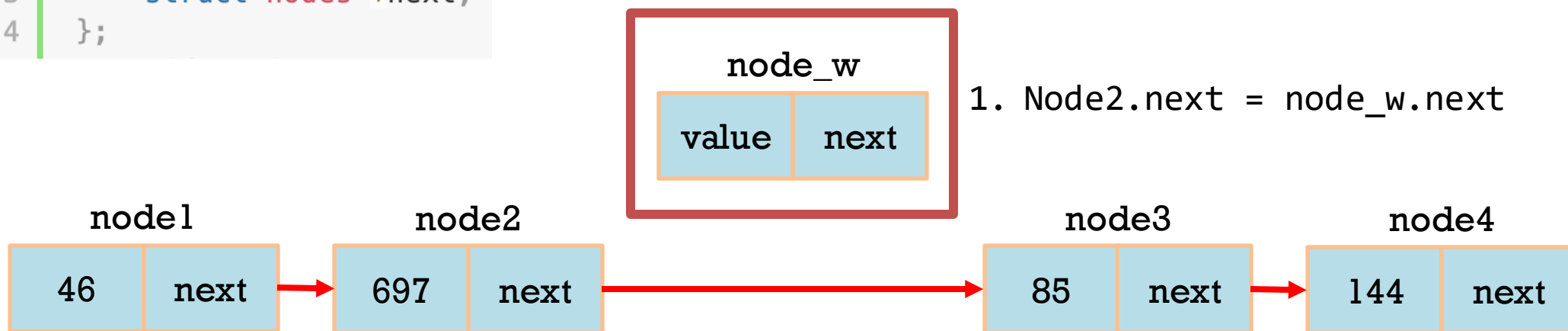


# 鏈結串列設計

結構：

```
1 struct nodes{  
2     int value;  
3     struct nodes *next;  
4 };
```

刪除：



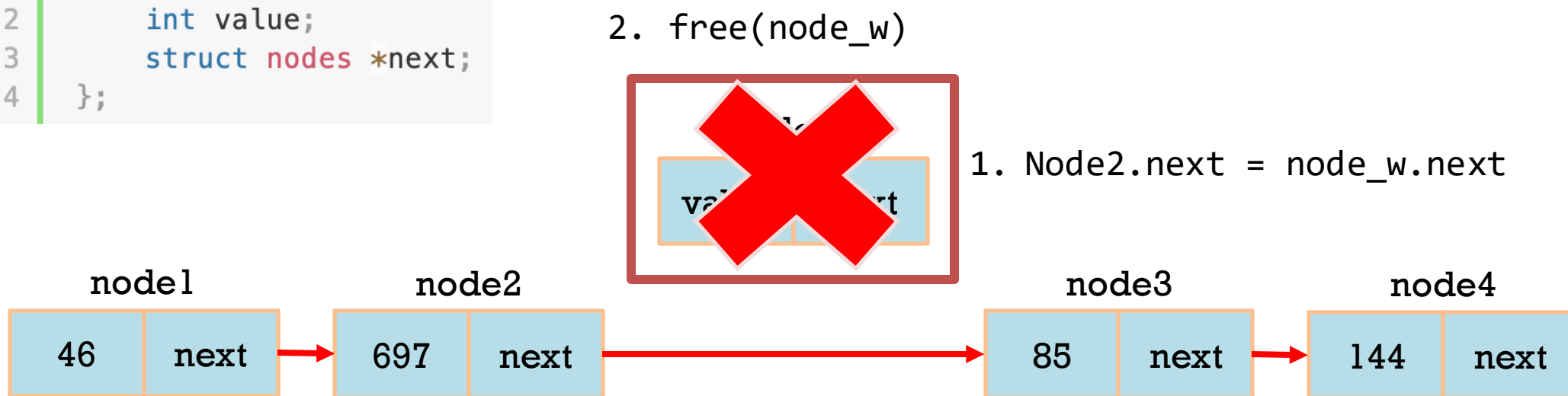


# 鏈結串列設計

結構：

```
1 struct nodes{  
2     int value;  
3     struct nodes *next;  
4 };
```

刪除：



# 陣列/鏈結串列設計

比較表

# 陣列/鏈結串列設計

## 比較表

項目	陣列	鏈結串列
空間佔用方式	佔用Memory連續空間	Node不一定佔用連續空間

# 陣列/鏈結串列設計

## 比較表

項目	陣列	鏈結串列
空間佔用方式	佔用Memory連續空間	Node不一定佔用連續空間
資料型態	相同(int, char, float)	不一定相同

# 陣列/鏈結串列設計

## 比較表

項目	陣列	鏈結串列
空間佔用方式	佔用Memory連續空間	Node不一定佔用連續空間
資料型態	相同(int, char, float)	不一定相同
支援讀取方式	支援Random Access	只支援Sequential Access

# 陣列/鏈結串列設計

## 比較表

項目	陣列	鏈結串列
空間佔用方式	佔用Memory連續空間	Node不一定佔用連續空間
資料型態	相同(int, char, float)	不一定相同
支援讀取方式	支援Random Access	只支援Sequential Access
可靠度	高	低(若遇斷鏈)

# 陣列/鏈結串列設計

## 比較表

項目	陣列	鏈結串列
空間佔用方式	佔用Memory連續空間	Node不一定佔用連續空間
資料型態	相同(int, char, float)	不一定相同
支援讀取方式	支援Random Access	只支援Sequential Access
可靠度	高	低(若遇斷鏈)
插入, 刪除操作	不便, 需挪動其他元素	方便

# 陣列/鏈結串列設計

## 比較表

項目	陣列	鏈結串列
空間佔用方式	佔用Memory連續空間	Node不一定佔用連續空間
資料型態	相同(int, char, float)	不一定相同
支援讀取方式	支援Random Access	只支援Sequential Access
可靠度	高	低(若遇斷鏈)
插入, 刪除操作	不便, 需挪動其他元素	方便
空間是否可動態擴充	不可	可



# 陣列/鏈結串列設計

所以何時該用陣列？何時該用鏈結串列？

項目	陣列	鏈結串列
空間佔用方式	佔用Memory連續空間	Node不一定佔用連續空間
資料型態	相同(int, char, float)	不一定相同
支援讀取方式	支援Random Access	只支援Sequential Access
可靠度	高	低(若遇斷鏈)
插入, 刪除操作	不便，需挪動其他元素	方便
空間是否可動態擴充	不可	可

# Function設計-基礎，但不簡單

什麼是function？

# Function設計-基礎，但不簡單

什麼是function？

將一個區塊的程式碼命名，以達到重用、模組化等功能。

class、object、變數在大部分的情況下都被視為名詞，  
function 就是讓這些名詞做動作的動詞。

# Function設計-基礎，但不簡單

什麼是function？

將一個區塊的程式碼命名，以達到重用、模塊化。

class、object、變數在大部分的情況下都是名詞。

function 就是讓這些名詞做動作的動詞。

```
InClassPractice > L02 > C L02_triangle.c > ...
1  #include<stdio.h>
2  #include<math.h>
3  void Triangle( int );
4
5  int main(){
6      int a;
7      scanf("%d", &a);
8      Triangle(a);
9      return 0;
10 }
11
12 void Triangle(int a) {
13     float area = 0;
14     area = a * a / 2 ; // 公式
15     printf("%.3f\n", area);
16 }
17
18
```

# Function設計-基礎，但不簡單

什麼是function？

將一個區塊的程式碼命名，以達到重用、模組化等功能。

class、object、變數在大部分的情況下都被視為名詞，  
function 就是讓這些名詞做動作的動詞。

使用Function的好處？

# Function設計-基礎，但不簡單

什麼是function？

將一個區塊的程式碼命名，以達到重用、模組化等功能。

class、object、變數在大部分的情況下都被視為名詞，  
function 就是讓這些名詞做動作的動詞。

使用Function的好處？

1. 可讀性
2. 可維護性
3. 抽象化
4. 可重用

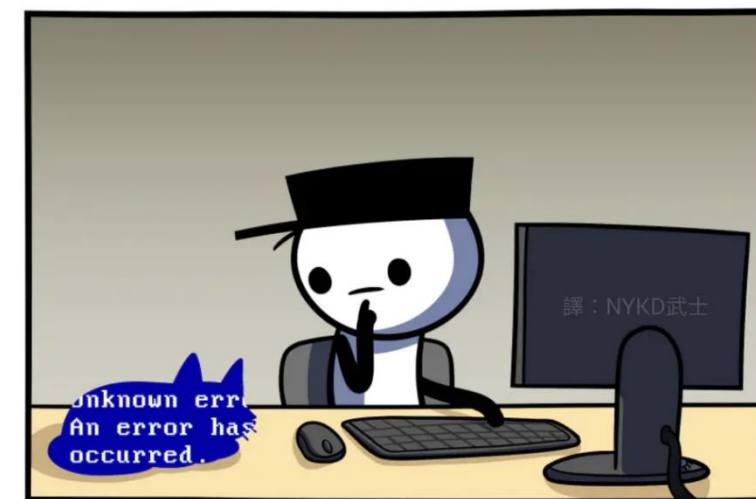
# Function設計-基礎，但不簡單

什麼是function？

將一個區塊的程式碼命名，以達到重用、模組化。class、object、變數在大部分的情況下都被視為名詞，function就是讓這些名詞做動作的動詞。

使用Function的好處？

1. 可讀性
2. 可維護性
3. 抽象化
4. 可重用



# Function設計-基礎，但不簡單

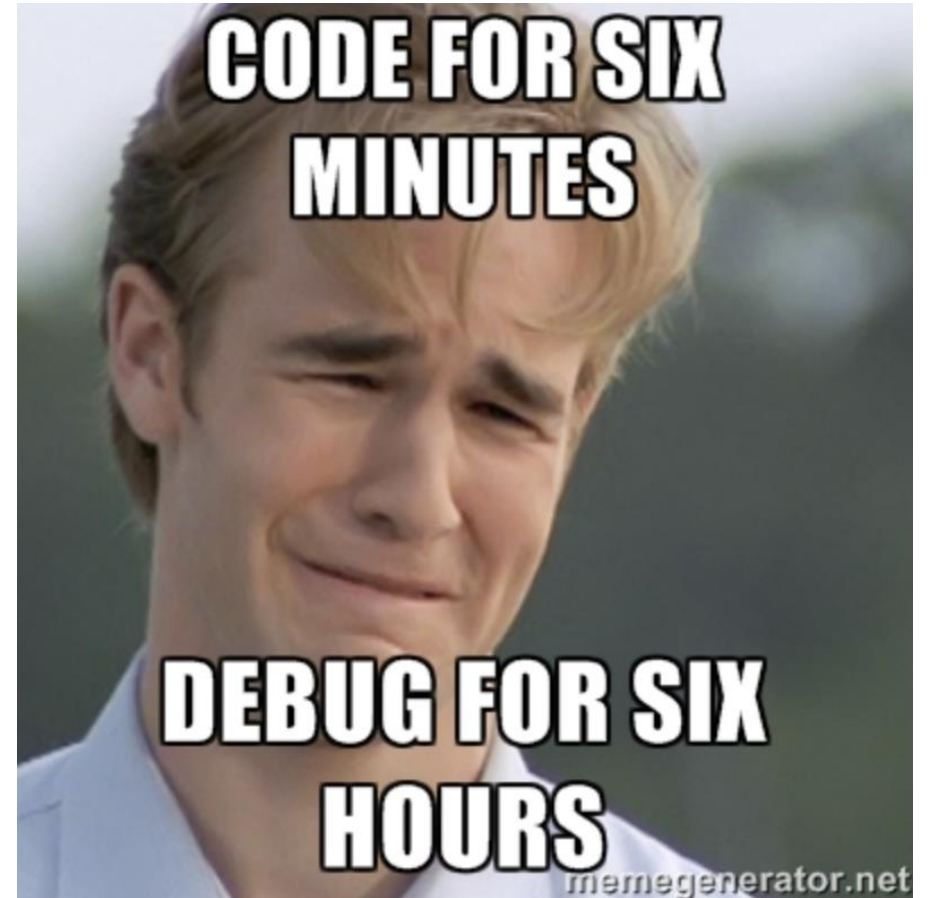
那要如何設計好的function？



# Function設計-基礎，但不簡單

那要如何設計好的function？

1. 行數
  - a. 長度盡量不超過50行
  - b. 但也避免只有1行的function



# Function設計-基礎，但不簡單

那要如何設計好的function？

1. 行數
2. 參數數量

# Function設計-基礎，但不簡單

那要如何設計好的function？

1. 行數
2. 參數數量 – 太多

```
function studentInfo(sID, birth, class, course, email, gender, survey){}
```



# Function設計-基礎，但不簡單

那要如何設計好的function？

1. 行數
2. 參數數量
3. 命名 – 計算學生的年齡

```
sDIbirthInfo(sID, birth){}  
getStdAge(sID, birth){}
```

# Function設計-基礎，但不簡單

那要如何設計好的function？

1. 行數
2. 參數數量
3. 命名
4. 抽象化 – 只做好一件事情

```
getUsersByNameAndAddressAndPhone(){}  

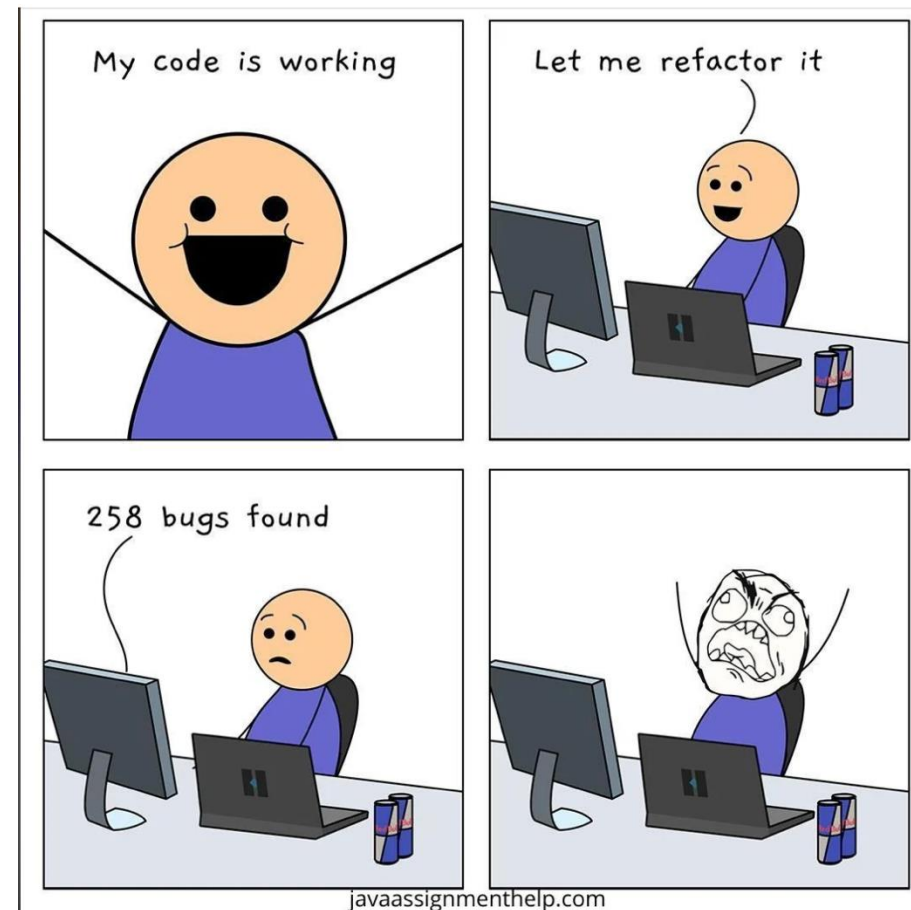
```

# Function設計-基礎，但不簡單

## 那要如何設計好的function？

1. 行數
2. 參數數量
3. 命名
4. 抽象化 – 只做好一件事情

```
getUsersByName(){}  
getAddress(){}  
getPhone(){}  
}
```



# Function設計-基礎，但不簡單

那要如何設計好的function？

1. 行數
2. 參數數量
3. 命名
4. 抽象化
5. 保持純函數 ( pure function )

相同的輸入，永遠會得到相同的輸出，沒有顯著副作用

Me: Makes a small CSS change

My Site:



# 應用演示



# 應用演示

程式「設計」包含設計程式的結構、功能、流程、效率、可讀性。  
目的是解決特定的問題或滿足特定的需求。

# 應用演示

假設問題：矩陣轉換

給定一個 $M \times N$ 的矩陣，但凡有一個元素為0

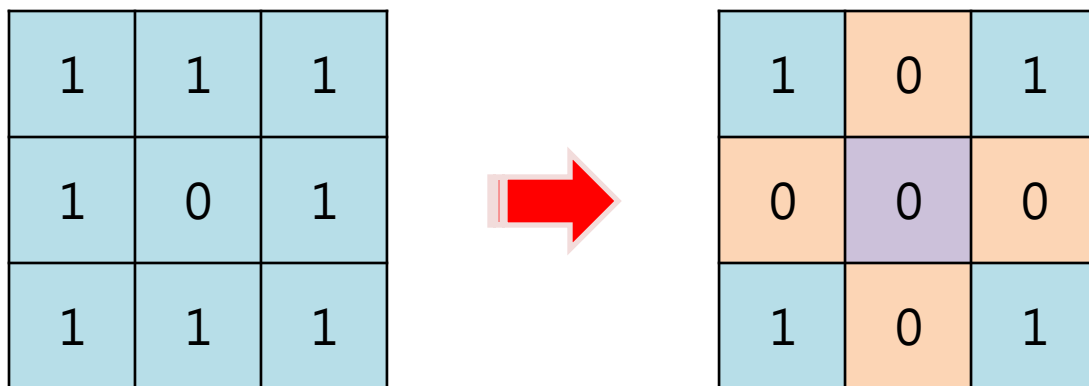
其所在的行與列皆轉成0

# 應用演示

假設問題：矩陣轉換

給定一個 $M \times N$ 的矩陣，但凡有一個元素為0  
其所在的行與列皆轉成0

範例1：



# 應用演示


假設問題：矩陣轉換

給定一個 $M \times N$ 的矩陣，但凡有一個元素為0

其所在的行與列皆轉成0

範例2：

1	4	7	8
0	3	5	0
2	6	2	4



0	4	7	0
0	0	0	0
0	6	2	0

# 應用演示

假設問題：矩陣轉換

給定一個 $M \times N$ 的矩陣，但凡有一個元素為0

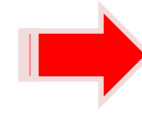
其所在的行與列皆轉成0

額外條件：假設今天電腦的儲存空間很少，我們必須要設計出最少空間使用方法，又該如何進行呢？

# 應用演示

解決方法1：

1	4	7	8
0	3	5	0
2	6	2	4



0	4	7	0
0	0	0	0
0	6	2	0

# 應用演示

解決方法1：

直觀的解決辦法是產生一個M×N額外的陣列，進行確認與改寫。

1	4	7	8
0	3	5	0
2	6	2	4



0	4	7	0
0	0	0	0
0	6	2	0

# 應用演示

解決方法1：

直觀的解決辦法是產生一個M×N額外的陣列，進行確認與改寫。

1	4	7	8
0	3	5	0
2	6	2	4



0	4	7	0
0	0	0	0
0	6	2	0

1. 複製一個一模一樣的矩陣

```
int newM[M][N];
for (int i = 0; i < M; ++i) {
    for (int j = 0; j < N; ++j) {
        newM[i][j] = matrix[i][j];
    }
}
```

2. 當newM = 0 則Matrix的行與列 = 0

```
for (int r = 0; r < M; ++r) {
    for (int c = 0; c < N; ++c) {
        if (newM[r][c] == 0) {
            for (int j = 0; j < N; ++j) {
                matrix[r][j] = 0;
            }
            for (int i = 0; i < M; ++i) {
                matrix[i][c] = 0;
            }
        }
    }
}
```



# 應用演示

解決方法2：

有沒有解決辦法是產生一個 $M+N$ 額外的陣列，進行確認與改寫。

1	4	7	8
0	3	5	0
2	6	2	4



0	4	7	0
0	0	0	0
0	6	2	0

# 應用演示

解決方法2：

有沒有解決辦法是產生一個M+N額外的陣列，進行確認與改寫。

1	4	7	8
0	3	5	0
2	6	2	4



0	4	7	0
0	0	0	0
0	6	2	0

1. 透過row[M], col[N]紀錄包含0的行列

```
void zeroMatrix(int matrix[M][N]) {  
    int row[M] = {0}; // 用於記錄含有0的列  
    int col[N] = {0}; // 用於記錄含有0的行  
  
    // 找出含有0的列與行  
    int i, j;  
    for (i = 0; i < M; i++) {  
        for (j = 0; j < N; j++) {  
            if (matrix[i][j] == 0) {  
                row[i] = 1;  
                col[j] = 1;  
            }  
        }  
    }  
}
```

2. 當row[m], col[n] = true  
則將matrix所在的列更新為0

```
// 將含有0的列，更新Matrix為0  
for (i = 0; i < M; i++) {  
    if (row[i] == 1) {  
        for (j = 0; j < N; j++) {  
            matrix[i][j] = 0;  
        }  
    }  
}  
  
// 將含有0的行，更新Matrix為0  
for (j = 0; j < N; j++) {  
    if (col[j] == 1) {  
        for (i = 0; i < M; i++) {  
            matrix[i][j] = 0;  
        }  
    }  
}
```

# 應用演示

解決方法3：

有沒有解決辦法是只使用 $O(1)$ 空間，進行確認與改寫。

1	4	7	8
0	3	5	0
2	6	2	4



0	4	7	0
0	0	0	0
0	6	2	0

# 應用演示

1	4	7	8
0	3	5	0
2	6	2	4



0	4	7	0
0	0	0	0
0	6	2	0

解決方法3：

有沒有解決辦法是只使用**0(1)**空間，進行確認與改寫。

第一步：檢查第一行與第一列是否含有0

```
//檢查第一行是否含有0
int j, i;
for (j = 0; j < N; j++) {
    if (matrix[0][j] == 0) {
        row_has_zero = 1;
        break;
    }
}

// 检查第一列是否含有0
for (i = 0; i < M; i++) {
    if (matrix[i][0] == 0) {
        col_has_zero = 1;
        break;
    }
}
```

row\_has\_zero = 0  
col\_has\_zero = 1

1	4	7	8
0	3	5	0
2	6	2	4

# 應用演示

解決方法3：

有沒有解決辦法是只使用**0(1)**空間，進行確認與改寫。

第二步：除了和第一行與第一列是否含有0

1	4	7	8
0	3	5	0
2	6	2	4



0	4	7	0
0	0	0	0
0	6	2	0

```
// 除了第一行和第一列以外的資料，若有0則將對應的行與列標記為0
for (i = 1; i < M; i++) {
    for (j = 1; j < N; j++) {
        if (matrix[i][j] == 0) {
            matrix[i][0] = 0;
            matrix[0][j] = 0;
        }
    }
}
```

1	4	7	8
0	3	5	0
2	6	2	4



1	4	7	0
0	3	5	0
2	6	2	4

# 應用演示

解決方法3：

有沒有解決辦法是只使用**0(1)**空間，進行確認與改寫。

第三步：將第一列和第一行有0的，整列或整行替換成0

1	4	7	8
0	3	5	0
2	6	2	4

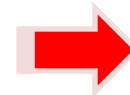


0	4	7	0
0	0	0	0
0	6	2	0

```
// 將含有0的行，全部改成0
for (i = 1; i < M; i++) {
    if (matrix[i][0] == 0) {
        for (j = 1; j < N; j++) {
            matrix[i][j] = 0;
        }
    }
}

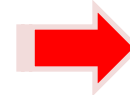
// 將含有0的列，全部改成0
for (j = 1; j < N; j++) {
    if (matrix[0][j] == 0) {
        for (i = 1; i < M; i++) {
            matrix[i][j] = 0;
        }
    }
}
```

1	4	7	0
0	3	5	0
2	6	2	4



1	4	7	0
0	0	0	0
2	6	2	4

1	4	7	0
0	0	0	0
2	6	2	4



1	4	7	0
0	0	0	0
2	6	2	0

# 應用演示

解決方法3：

有沒有解決辦法是只使用**0(1)**空間，進行確認與改寫。

第四步：確認，第一部檢查的col\_has\_zero, row\_has\_zero

1	4	7	8
0	3	5	0
2	6	2	4



0	4	7	0
0	0	0	0
0	6	2	0

```
// 如果第一行含有0，則第一行設為0
if (row_has_zero) {
    for (j = 0; j < N; j++) {
        matrix[0][j] = 0;
    }
}

// 如果第一列含有0，則第一列設為0
if (col_has_zero) {
    for (i = 0; i < M; i++) {
        matrix[i][0] = 0;
    }
}
}
```

row\_has\_zero = 0  
col\_has\_zero = 1

1	4	7	0
0	0	0	0
2	6	2	0



0	4	7	0
0	0	0	0
0	6	2	0

# 應用演示

1	4	7	8
0	3	5	0
2	6	2	4



0	4	7	0
0	0	0	0
0	6	2	0

程式「設計」包含設計程式的結構、功能、流程、效率、可讀性。  
目的是解決特定的問題或滿足特定的需求



# L02 作業繳交規範

1. 請繳交 .c 檔案，.c檔以外的格式不予批改
2. 檔案名稱請命名「姓名\_L#\_p#.c」
  - practice 1 -> 謝沛璇\_L02\_p01.c
3. 不接受延遲與補交，請同學務必準時繳交 (due date 3/26 23:59)
4. 最後請務必寫上註解，若無法理解會斟酌扣分

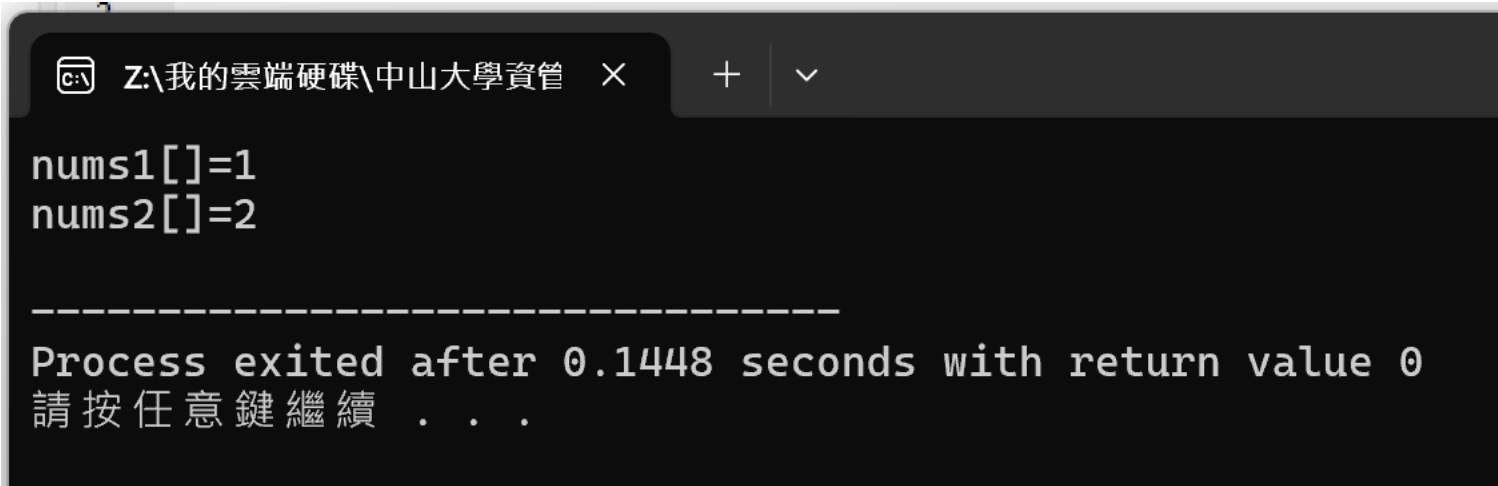
# L02 評分標準

1. 程式結果正確：60% (只要與結果相符正確)
2. 符合程式要求：20% (如限制使用for迴圈, 但若無限制則為送分)
3. 程式註解清楚：20%

# Practice 1.

- 給定一組**整數**數列：
- `int nums1[] = {1, 1, 2, 3, 4, 6, 1, 1, 1};`
- `int nums2[] = {2, 2, 1, 1, 1, 2, 2, 3, 8, 2, 2, 2, 7};`
- 回傳該陣列中，「出現次數」>「 $n/2$ 次」的數字

[**假設前提**：數列不為空，且必定存在該數字出現次數 $>n/2$ ]



```

Z:\我的雲端硬碟\中山大學資管  ×  +  ∨
nums1[]=1
nums2[]=2

-----
Process exited after 0.1448 seconds with return value 0
請按任意鍵繼續 . . .
```

# Practice 1.

- 給定一組整數數列：
- `int nums1[] = {1, 1, 2, 3, 4, 6, 1, 1, 1};`
- `int nums2[] = {2, 2, 1, 1, 1, 2, 2, 3, 8, 2, 2, 2, 7};`
- 回傳該陣列中，「出現次數」>「 $n/2$ 次」的數字

[**假設前提**：數列不為空，且必定存在該數字出現次數 $>n/2$ ]

[優化設計方向]

1. 嘗試運行的次數控制在 $O(n)$
2. 控制額外空間在 $O(1)$