

第六章

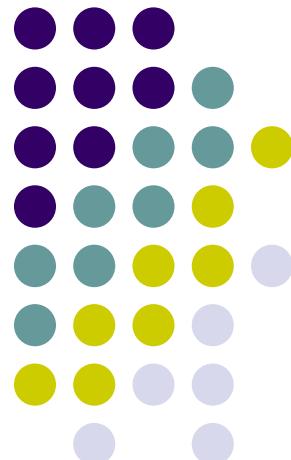
函 數

認識函數的基本架構

學習inline函數

認識變數的等級

同時使用數個函數





C/C++ 的函數

- Top-down design 的目的是要達到逐步單純化
- 將大問題細分成小問題
- 將解決這些小問題的方法，撰寫成較小的程式區塊
- 程式語言的函數
 - 如賦予程式區塊一個名字
 - 並且指定它的輸出與輸入
 - 則此程式區塊就是一個 **函數**



函數的功用

- 函數可以使程式的設計與維護更加容易
- 可提高程式的可讀性
- 函數可重複被呼叫，提高程式的再利用率
對程式的執行效率不好
- 每一個函數有自己的任務，可按任務來規劃函數
- 數學函數: $y = f(x)$
- f : 函數名稱、 x : 參數、 y : 回傳值
- $y = \sin(x)$ 、 $r = \text{random}()$ 、 $n = \text{power}(a, b)$



簡單的函數 (1/2)

- 下面的範例計算6的平方值，並在運算結果前後列印星號

```

01 // prog6_1, 簡單的函數
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 void star(void); 宣告在main之前 // 函數原型的宣告
06 int main(void)
07 {
08     star(); // 呼叫自訂的函數，印出星號
09     cout << "6*6=" << 6*6 << endl; // 印出 6 的平方值
10     star(); // 呼叫自訂的函數，印出星號
11     system("pause");
12     return 0;
13 }
14
15 void star(void) // 自訂的函數 star()
16 {
17     int j;
18     for(j=1;j<=8;j++)
19         cout << "*";
20     cout << endl;
21     return;
22 }
```

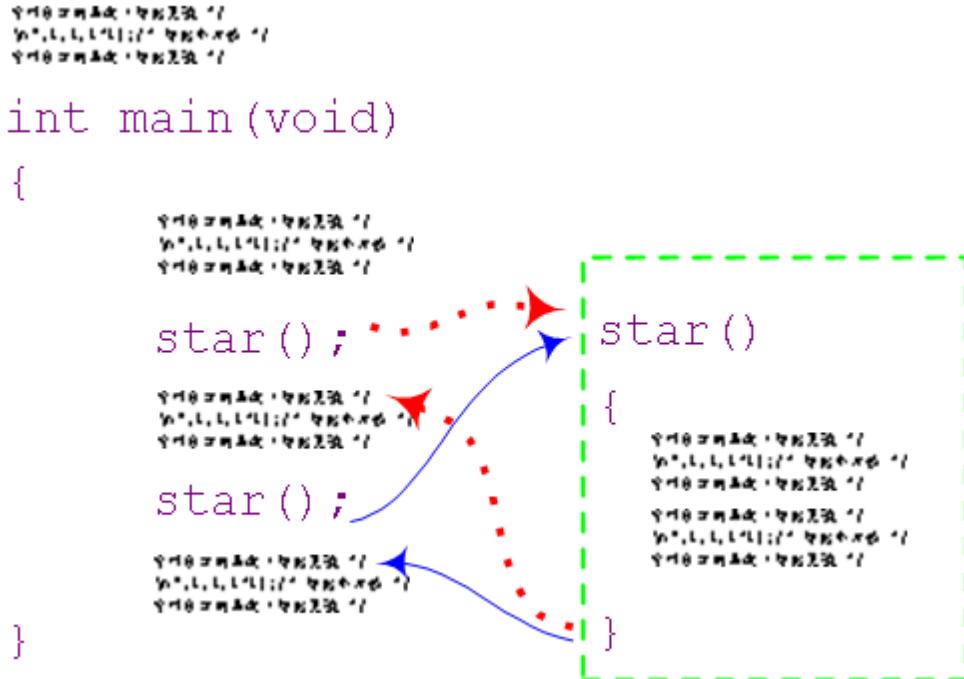
/* prog6_1 OUTPUT---

6*6=36
*****-----*/



簡單的函數 (2/2)

- 下圖說明函數呼叫與返回的方式：





函數原型的宣告、撰寫與呼叫(1/3)

- 下面為「函數原型」(prototype) 的宣告格式

傳回值型態 函數名稱(引數型態 1, 引數型態 2, ...);

- 下面的格式為合法的函數宣告格式

void : 沒有傳回值

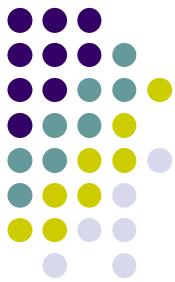
傳回值型態

引數型態，各型態間以逗號分開

int add (int,int);

函數名稱

不可重複



函數原型的宣告、撰寫與呼叫(2/3)

- 自訂函數撰寫的格式如下所示

```
傳回值型態 函數名稱(型態 1 引數 1, ..., 型態 n 引數 n)
{
    變數宣告;
    敘述主體;
    return 運算式;
}
```

- 呼叫函數的方式有兩種
 - 一種是將傳回值指定給某個變數接收
 - 另一種則是直接呼叫函數，不需要傳回值

變數 = 函數名稱(參數);

函數名稱(參數);



函數原型的宣告、撰寫與呼叫(3/3)

- 下面的敘述為常見的函數呼叫

```
i=func(); // 呼叫 func() 函數,並將傳回值給 i 存放  
star(); // 直接呼叫 star() 函數,沒有傳回值  
myfunc(4); // 呼叫 myfunc() 函數,並將引數 4 傳入函數中
```

右邊的格式為自訂函數
square() 的宣告與呼叫方式

二階段

一階段 `int square(int);` → 自訂函數的宣告

`int main(void){`

`j=square(i);` → 自訂函數的呼叫

}

```
int square(int i)
{
    int squ;
    squ=i*i;
    return squ;
}
```

→ 自訂函數的內容



不使用函數原型的方式 (1/2)

- 如果不使用函數原型，可採下面的寫法

```
int square(int i)
{
    int squ;
    squ=i*i;
    return squ;
}
```

→ 自訂函數的定義與宣告
定義代替宣告時（一、二階段合併的寫法）
定義在主程式之前

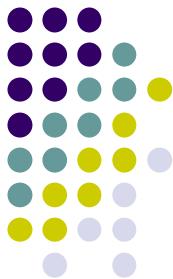
```
...
int main(void) → 主函數
{
    ...
    j=square(i); → 自訂函數的呼叫
    ...
}
```



不使用函數原型的方式 (2/2)

- 下面的程式是不使用函數原型的方式所撰寫而成的

```
01 // prog6_2, 不使用函數原型的方式          /* prog6_2 OUTPUT---  
02 #include <iostream>                         square(6)=36  
03 #include <cstdlib>                          -----*/  
04 using namespace std;  
05 int square(int a)    // 自訂的函數 square(), 計算平方值  
06 {  
07     int squ;  
08     squ=a*a;  
09     return squ;  
10 }  
11  
12 int main(void)      // 主程式  
13 {  
14     cout << "square(6)=" << square(6) << endl; // 印出 square(6)的值  
15     system("pause");  
16     return 0;  
17 }
```



函數的引數與參數 (1/2)

- 傳遞給函數的資料稱為函數的「引數」 (argument) 。
- 函數所收到的資料稱為「參數」 (parameter) 。



函數的引數與參數 (2/2)

• 下面是傳入兩個引數的例子

```

01 // prog6_3, 呼叫自訂函數
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 void func(int,int);           // 函數原型的宣告
06 int main(void)
07 {
08     int a=3,b=6;
09     cout << "In main()",a=" << a << ",b=" << b << endl; // 印出a,b 的值
10     func(a,b);
11     cout << "After func()",a=" << a << ",b=" << b << endl; /* prog6_3 OUTPUT---
12
13     system("pause");
14     return 0;
15 }
16
17 void func(int a,int b)           // 自訂的函數 func()，印出 a,b 的值
18 {
19     a+=10;
20     b+=10;
21     cout << "In func()",a=" << a << ",b=" << b << endl;
22     return;
23 }                                二個獨立存在的a,b
```

-----*/



函數的傳回值 (1/3)

- return敘述的格式如下所示

```
return 運算式;
```

- 函數的傳回值可以是變數、常數或是運算式
- 函數沒有傳回值時，可以在函數結束的地方加上分號

```
return;
```



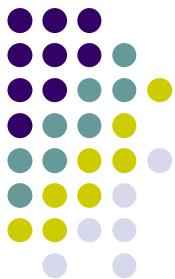
函數的傳回值 (2/3)

```

01 // prog6_4, 傳回較大值
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 int max(int,int);           // 函數原型的宣告
06 int main(void)
07 {
08     int a=12,b=35;
09     cout << "a=" << a << ", b=" << b << endl;      // 印出 a,b 的值
10     cout << "The larger number is " << max(a,b) << endl; // 印出較大值
11     system("pause");
12     return 0;
13 }
14
15 int max(int i,int j)      // 自訂的函數 max(), 傳回較大值
16 {
17     if (i>j)
18         return i; 一執行到 return,
19     else          後面都略過
20         return j;
21 }
```

• 下面的程式可以利用函數
傳回兩個整數的較大值

/* prog6_4 OUTPUT-----
a=12, b=35
The larger number is 35
-----*/



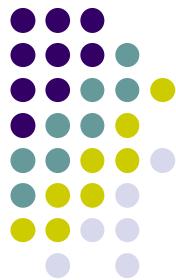
函數的傳回值 (3/3)

```

01 // prog6_5, 沒有傳回值的函數
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 void myprint(int,char);           // 函數原型的宣告
06 int main(void)
07 {
08     int a=6;
09     char ch='%';
10     myprint(a,ch);                // 呼叫自訂的函數，印出 a 個字元
11     cout << "Printed!!" << endl;      /* prog6_5 OUTPUT---
12     system("pause");
13     return 0;
14 }
15
16 void myprint(int n,char c)      // 自訂的函數 myprint()
17 {
18     int i;
19     for(i=1;i<=n;i++)
20         cout << c;                  // 印出字元
21     cout << endl;
22     return;
23 }
```

- prog6_5是沒有傳回值的函數之範例

哈哈哈哈哈哈哈哈
Printed!!
-----*/



取代

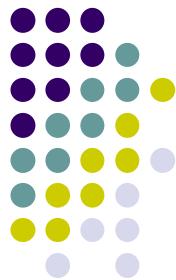
#define前置處理器 (1/4)

- 前置處理器的指令 #define 格式如下

```
#define 識別名稱 代換標記
```

- 下面的範例皆為合法的 #define 定義

```
#define MAX 65535          // 定義 MAX 為常數 65535  
#define IOC "I love C++!"  // 定義 IOC 為字串 I love C++!
```



#define前置處理器 (2/4)

- prog7_10是使用 #define 的範例

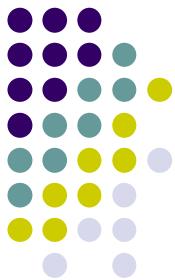
```

01 // prog7_10, 使用#define
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 #define PI 3.14
06 void peri(double), area(double); 若回傳值不同時，要分2行宣告
07 int main(void)
08 {
09     double r=1.0;
10     cout << "pi=" << PI << endl;
11     cout << "radius=" << r << endl;
12     peri(r); // 呼叫自訂的函數
13     area(r);
14     system("pause");
15     return 0;
16 }
17

```

/* prog7_10 OUTPUT-----

pi=3.14
radius=1
peripheral length=6.28
area=3.14
-----*/



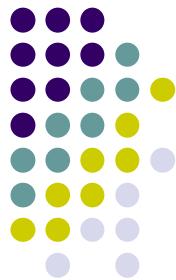
#define前置處理器 (3/4)

```
18 void peri(double r)           // 自訂的函數 peri()，印出圓周
19 {
20     cout << "peripheral length=" << 2*PI*r << endl;
21     return;
22 }
23
24 void area(double r)          // 自訂的函數 area()，印出圓面積
25 {
26     cout << "area=" << PI*r*r << endl;
27     return;
28 }
```

/* prog7_10 OUTPUT-----

pi=3.14
radius=1
peripheral length=6.28
area=3.14

-----*/



#define前置處理器 (4/4)

- #define定義的內容可以利用反斜線 (\) 將定義分行
- 下面的程式是使用 #define定義一段較長的字串之範

```
01 // prog7_11, 使用#define
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 #define WORD "Absence diminishes little passions \
06 and increases great ones."
07 int main(void)
08 {
09     cout << WORD << endl;
10     system("pause");
11     return 0;
12 }
```

換行繼續

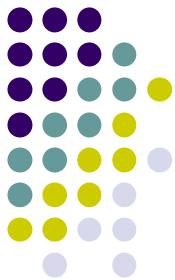
/* prog7_11 OUTPUT-----

Absence diminishes little passions and increases great ones.



為什麼要用 #define

- 可以增加程式的易讀性，即看到識別名稱通常就能夠明白所代表的意義
- 需要修改所定義的內容時，只要在相關的 #define 指令中更改即可
- 在某些場合可增加程式執行的速度



利用#define定義簡單的函數 (1/2)

- 函數是程式裡的模組
- 巨集是在前置處理器中的模組
- 適當的使用巨集可以取代簡單的函數



利用#define定義簡單的函數 (2/2)

- 下面是利用巨集定義函數的範例

```
01 // prog7_13, 使用巨集
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 #define POWER i*i*i
06 int main(void)
07 {
08     int i; 若不是i, POWER compile error
09     cout << "Input an integer:";
10     cin >> i;
11
12     // 計算並印出i的3次方
13     cout << i << "*" << i << "*" << i << "=" << POWER << endl;
14     system("pause");
15     return 0;
16 }
```

/ prog7_13 OUTPUT---*

Input an integer:**3**

3*3*3=27

-----*/



使用有引數的巨集

- 巨集也可以使用引數，如下面的程式所示

```
01 // prog7_14, 使用巨集
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 #define POWER(X) X*X*X
06 int main(void)
07 {
08     int i;
09     cout << "Input an integer:";
10     cin >> i;
11
12     // 計算並印出 i 的 3 次方
13     cout << i << "*" << i << "*" << i << "=" << POWER(i) << endl;
14     system("pause");
15     return 0;
16 }
```

/* prog7_14 OUTPUT---

Input an integer:**2**

2*2*2=8

-----*/



巨集括號的使用 (1/2)

- 將前例的POWER(i) 改成POWER(i+1) <錯誤的範例>

```
01 // prog7_15, 使用巨集
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 #define POWER(X) X*X*X
06 int main(void)
07 {
08     int i;
09     cout << "Input an integer:";
10     cin >> i;
11
12     // 計算並印出 i+1 的 3 次方
13     cout << i+1 << "*" << i+1 << "*" << i+1 << "=" << POWER (i+1) << endl;
14     system("pause");           /* prog7_15 OUTPUT---*/
15     return 0;
16 }
```

Input an integer:**2**
3*3*3=7



巨集括號的使用 (2/2)

- 經過前置處理器置換後的第13行，應是下面的敘述

```
cout << i+1 << "*" << i+1 << "*" << i+1 << "=" << i+1*i+1*i+1 << endl;
```

```

01 // prog7_16, 修改 prog7_15
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 #define POWER(X) (X)*(X)*(X)
06 int main(void)
07 {
08     int i;
09     cout << "Input an integer:";
10     cin >> i;
11
12     // 計算並印出 i+1 的 3 次方
13     cout << i+1 << "*" << i+1 << "*" << i+1 << "=" << POWER(i+1) << endl;
14     system("pause");
15     return 0;
16 }
```

- 正確的程式碼的修改如下所示

/ prog7_16 OUTPUT--*

Input an integer:**2**
3*3*3=27

-----/*

函數形式：int POWER(int x)

```
{
    return x*x*x;
}
```



標準的標頭檔

- 下圖為iostream的一隅

The screenshot shows the Dev-C++ 4.9.9.2 IDE interface. The title bar reads "Dev-C++ 4.9.9.2". The menu bar includes "檔案(F)", "編輯(E)", "搜尋(S)", "檢視(V)", "專案(P)", "執行(Z)", "除錯(D)", "工具(T)", "CVS", "視窗(W)", and "求助(H)". The toolbar contains various icons for file operations like Open, Save, and Build. The main editor window has a tab labeled "iostream" and displays the following C++ code:

```
39 #ifndef _GLIBCXX_IOSTREAM
40 #define _GLIBCXX_IOSTREAM 1
41
42 #pragma GCC system_header
43
44 #include <bits/c++config.h>
45 #include <iostream>
46 #include <iostream>
47
48 namespace std
49 {
50     /**
51      * @name Standard Stream Objects
52      *
53      * The <iostream> header declares the eight <em>standard
```

The status bar at the bottom shows "17:1", "插入模式", and "檔案裡總共有81行".

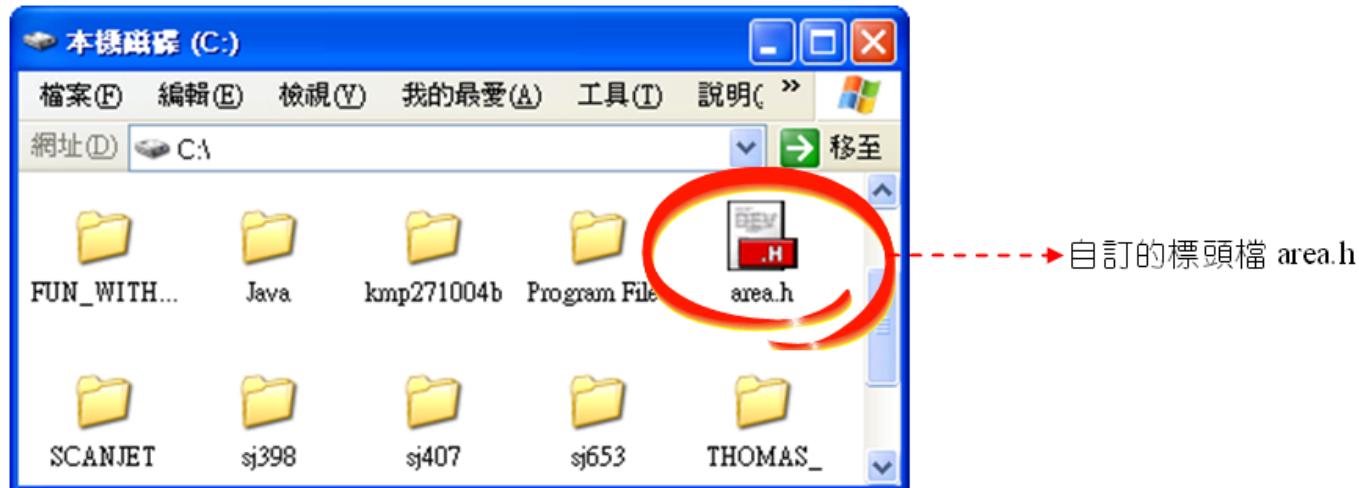


使用自訂的標頭檔 (1/3)

- 圓、長方形及三角形面積公式的巨集如下

```
#define PI 3.14  
#define CIRCLE(r) ((PI)*(r)*(r))  
#define RECTANGLE(length,height) ((length)*(height))  
#define TRIANGLE(base,height) ((base)*(height)/2)
```

- 將巨集於硬碟C的根目錄C:\中儲存成area.h





使用自訂的標頭檔 (2/3)

- 使用

```
#include <area.h>
```

時，#include會到系統所設定的目錄找尋被含括的檔案

- 使用

```
#include "area.h"
```

時，前置處理器則會依指定的目錄尋找該標頭檔案



使用自訂的標頭檔 (3/3)

- 以標頭檔area.h為例，計算三角形的面積

```
01 // prog7_17, 使用自訂的標頭檔 area.h
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 #include "C:\area.h"
06 int main(void)
07 {
08     float base,height;
09     cout << "Input the base of triangle:";
10     cin >> base;
11     cout << "Input the height of triangle:";
12     cin >> height;
13     // 計算三角形面積
14     cout << "The area of triangle is " << TRIANGLE(base,height) << endl;
15     system("pause");
16     return 0;
17 }
```

/* prog7_17 OUTPUT-----
Input the base of triangle:3
Input the height of triangle:5
The area of triangle is 7.5-----*/

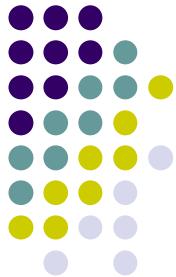


inline函數 (1/3)

- inline函數是在函數定義前多加一個inline關鍵字
- inline的功能像巨集，編譯器會在呼叫inline處，直接把程式碼嵌入到原始程式中，所以執行時效率較快
- inline函數的定義格式如下

```
inline 傳回值型態 函數名稱(型態 1 引數 1, …, 型態 n 引數 n)
{
    變數宣告;
    敘述主體;
    return 運算式;
}

...
int main(void)
{    ... }
```



inline函數 (2/3)

- 下圖說明編譯器如何將inline函數嵌入到原始程式

```

    可以使用此處，因為是inline函數
    fn", L, L, L-L1; /* 這段會被忽略 */
    可以使用此處，因為是inline函數
    int main(void)
    {
        可以使用此處，因為是inline函數
        fn", L, L, L-L1; /* 這段會被忽略 */
        可以使用此處，因為是inline函數
        star(); /* 這段會被忽略 */
        可以使用此處，因為是inline函數
        fn", L, L, L-L1; /* 這段會被忽略 */
        可以使用此處，因為是inline函數
        star(); /* 這段會被忽略 */
        可以使用此處，因為是inline函數
        fn", L, L, L-L1; /* 這段會被忽略 */
    }

```

- 編譯器可能會忽略inline函數的時機

- inline函數內容過大
- inline函數使用遞迴函數的呼叫方式，呼叫自己本身
- 所使用的編譯器本身不支援inline函數的使用



inline函數 (3/3)

- 下面是inline函數的使用範例

```

01 // prog6_6, inline 函數
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 inline void star(void)    // 自訂的函數 star()，繪製星號
06 {
07     cout << "*****" << endl;
08 }

09
10 int main(void)           // 主程式      /* prog6_6 OUTPUT---
11 {
12     star();
13     cout << "Hello, C++" << endl;
14     star();
15     system("pause");
16     return 0;
17 }
```

被第1
行替代

Hello, C++

-----*/



使用函數還是巨集？

- 使用巨集時可以代替簡單的函數
- 程式裡使用到某巨集n次，在編譯時就會產生n段相同的程式碼，因此編譯後的程式碼會稍大
- 巨集佔用的記憶體較多，但是程式的控制權不用移轉，執行的速度較快
- 在複雜巢狀迴圈裡使用巨集，會比較容易感覺到執行效率的增加

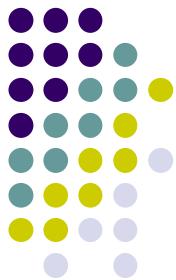


呼叫多個函數 (1/2)

- 下面的程式碼是在主程式裡呼叫多個函數的例子

```
01 // prog6_12, 呼叫多個函數
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 void sum(int), fact(int);
06 int main(void)
07 {
08     int a=5;                                /* prog6_12 OUTPUT---*/
09     fact(a);
10     sum(a);
11     system("pause");
12     return 0;
13 }
14
```

1*2*...*5=120
1+2+...+5=15
-----*/



呼叫多個函數 (2/2)

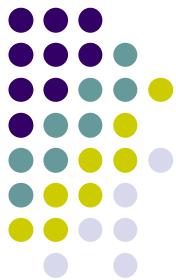
```
15 void fact(int a)          // 自訂函數 fact()，計算 a!
16 {
17     int i, total=1;
18     for(i=1;i<=a;i++)
19         total*=i;
20     cout << "1*2*...*" << a << "=" << total << endl; // 印出 a! 的結果
21     return;
22 }
23
24 void sum(int a)           // 自訂函數 sum()，計算 1+2+...+a 的結果
25 {
26     int i, sum=0;
27     for(i=1;i<=a;i++)
28         sum+=i;
29     cout << "1+2+...+" << a << "=" << sum << endl; // 印出計算結果
30     return;
31 }
```

/* prog6_12 OUTPUT---

1*2*...*5=120

1+2+...+5=15

-----*/



函數之間的相互呼叫 (1/2)

- 下面的程式碼是在函數間呼叫其它函數的例子

```
01 // prog6_13, 相互呼叫函數
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 void sum(int), fact(int);
06 int main(void)
07 {
08     int a=5;
09     fact(a);
10     sum(a+5);
11     system("pause");
12     return 0;
13 }
14
```

/* prog6_13 OUTPUT-----

1*2*...*5=120

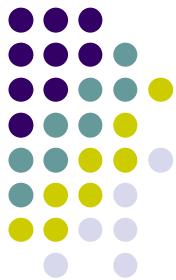
1+2+...+5=15

1+2+...+10=55

由 fact() 函數呼叫的 sum() 函數

由 main() 函數呼叫的 sum() 函數

-----*/



函數之間的相互呼叫 (2/2)

```
15 void fact(int a)          // 自訂函數 fact()，計算 a!
16 {
17     int i, total=1;
18     for(i=1;i<=a;i++)
19         total*=i;
20     cout << "1*2*...*" << a << "=" << total << endl; //印出 a!的結果
21     sum(a);
22     return;
23 }
24
25 void sum(int a)          // 自訂函數 sum()，計算 1+2+...+a 的結果
26 {
27     int i, sum=0;
28     for(i=1;i<=a;i++)
29         sum+=i;
30     cout << "1+2+...+" << a << "=" << sum << endl; //印出計算結果
31     return;
32 }
```



遞迴 執行效率 × 記憶體消耗量大

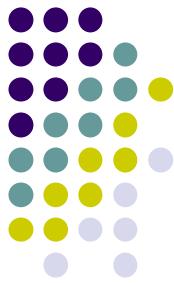
- 遞迴 - 函數自己呼叫自己
- 階乘函數 (factorial function , $n!$) 的遞迴

$$\text{fac}(n) = \begin{cases} 1 \times 2 \times \boxed{?} \times n ; & n \geq 1 \\ 1 ; & n = 0 \end{cases}$$

0

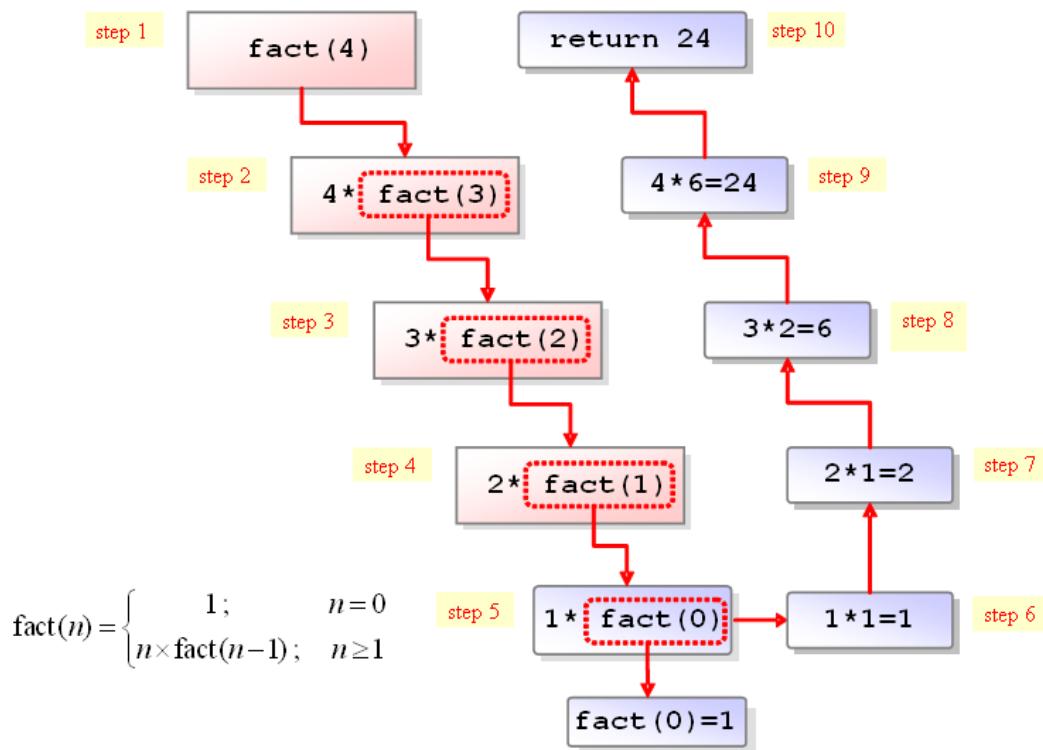
$$\boxed{? \times ? \times ? \times ? \times ? \times ? \times ?} = n \times \text{fac}(n-1)$$

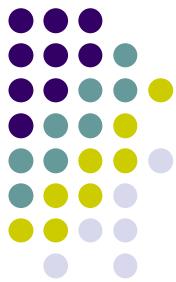
$$\text{fac}(n) = \begin{cases} n \times \text{fac}(n-1) ; & n \geq 1 \\ 1 ; & n = 0 \end{cases}$$



遞迴函數 (1/6)

- 遞迴函數就是函數呼叫自己本身
- 以階乘的遞迴為例，從下圖中可看到函數遞迴的情形





遞迴函數 (2/6)

- 下面的程式是利用遞迴計算階乘fact(a) 的運算結果

```
01 // prog6_14, 遞迴函數, 計算階乘
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 int fact(int);
06 int main(void)
07 {
08     int a;
09     do
10     {
11         cout << "Input an integer:";
12         cin >> a;
13     } while (a<=0);           // 確定輸入的 a 為大於 0 的數
14     cout << "1*2*...*" << a << "=" << fact(a) << endl;
15     system("pause");
16     return 0;
17 }
18
```

/* prog6_14 OUTPUT---

Input an integer:**-6**

Input an integer:**4**

1*2*...*4=24

-----*/



遞迴函數 (3/6)

```

19 int fact(int a)           // 自訂函數 fact()，計算 a!
20 {
21     if(a>0)
22         return (a*fact(a-1));
23     else
24         return 1;
25 }
```

$$y = 5$$

$$\text{value}(5) = \text{value}(4) \times \text{value}(3)$$

$$\text{value}(4) = \text{value}(3) \times \text{value}(2)$$

$$\text{value}(3) = \text{value}(2) \times \text{value}(1)$$

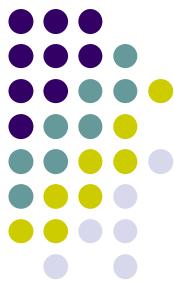
/ prog6_14 OUTPUT---*

Input an integer:**-6**

Input an integer:**4**

1*2*...*4=24

-----/*



遞迴函數 (4/6)

- 下表列出遞迴函數fact(a)的執行過程及所傳回的結果

執行順序	a 的值	fact(a) 的值	傳回值
1	4	fact(4)，未知	$4 * \text{fact}(3)$
2	3	fact(3)，未知	$3 * \text{fact}(2)$
3	2	fact(2)，未知	$2 * \text{fact}(1)$
4	1	fact(1)，未知	$1 * \text{fact}(0)$
5	0	$\text{fact}(0)=1$	1
6	1	$\text{fact}(1)=1$	1
7	2	$\text{fact}(2)=2$	2
8	3	$\text{fact}(3)=6$	6
9	4	$\text{fact}(4)=24$	24

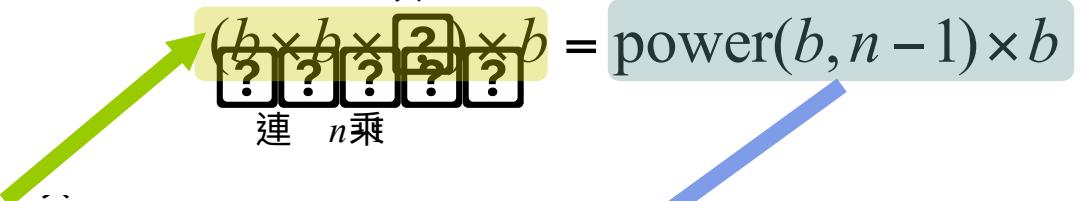


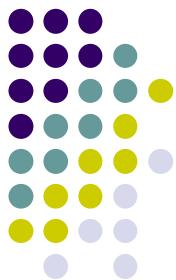
遞迴-次方函數

- 次方函數 b^n :

$$\text{power}(b, n) = \begin{cases} b \times \underset{\substack{\text{連} \\ \text{乘}}}{\boxed{b \times \dots \times b}} ; & n \geq 1 \\ 1 ; & n = 0 \end{cases}$$

$$\text{power}(b, n) = \begin{cases} b \times \text{power}(b, n - 1) ; & n \geq 1 \\ 1 ; & n = 0 \end{cases}$$



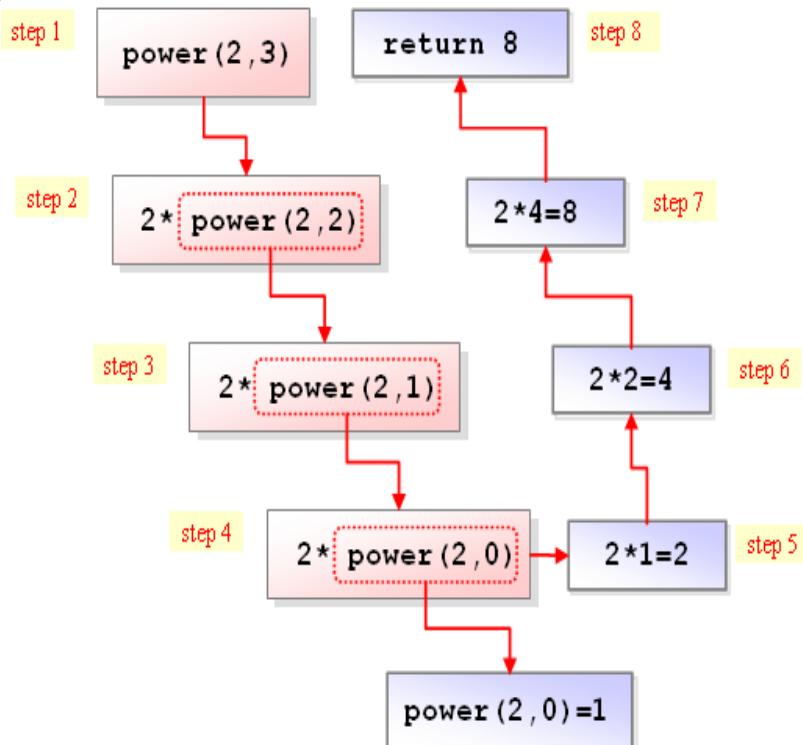


遞迴函數 (5/6)

```

01 // prog6_15, 遞迴函數, 計算次方
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 int power(int, int);
06 int main(void)
07 {
08     int a=2, b=3;
09     cout << a << "^" << b << "=";
10     cout << power(a, b) << endl;
11     system("pause");
12     return 0;
13 }
14
15 int power(int a, int b)
16 {
17     if(b==0)
18         return 1;
19     else return (a*power(a, b-1));
20 }
```

下面是遞迴函數求次方的程式



/* prog6_15 OUTPUT---

$2^3=8$

0

-----*/



遞迴函數 (6/6)

- 下表列出遞迴函數power() 的執行過程

執行順序	a	b	power(a, b)	傳回值
1	2	3	power(2, 3) , 未知	$2 * \text{power}(2, 2)$
2	2	2	power(2, 2) , 未知	$2 * \text{power}(2, 1)$
3	2	1	power(2, 1) , 未知	$2 * \text{power}(2, 0)$
4	2	0	power(2, 0)=1	1
5	2	1	power(2, 1)=2	2
6	2	2	power(2, 2)=4	4
7	2	3	power(2, 3)=8	8



-The End-