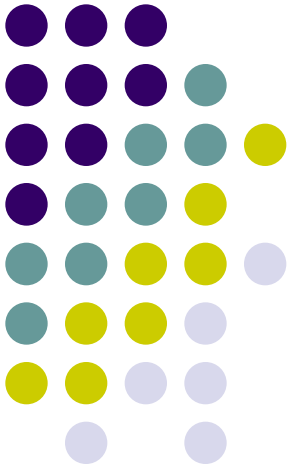
第六章
函 數

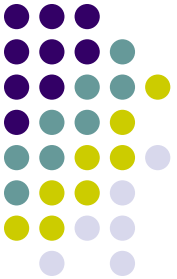
認識函數的基本架構

學習inline函數

認識變數的等級

同時使用數個函數





簡單的函數 (1/2)

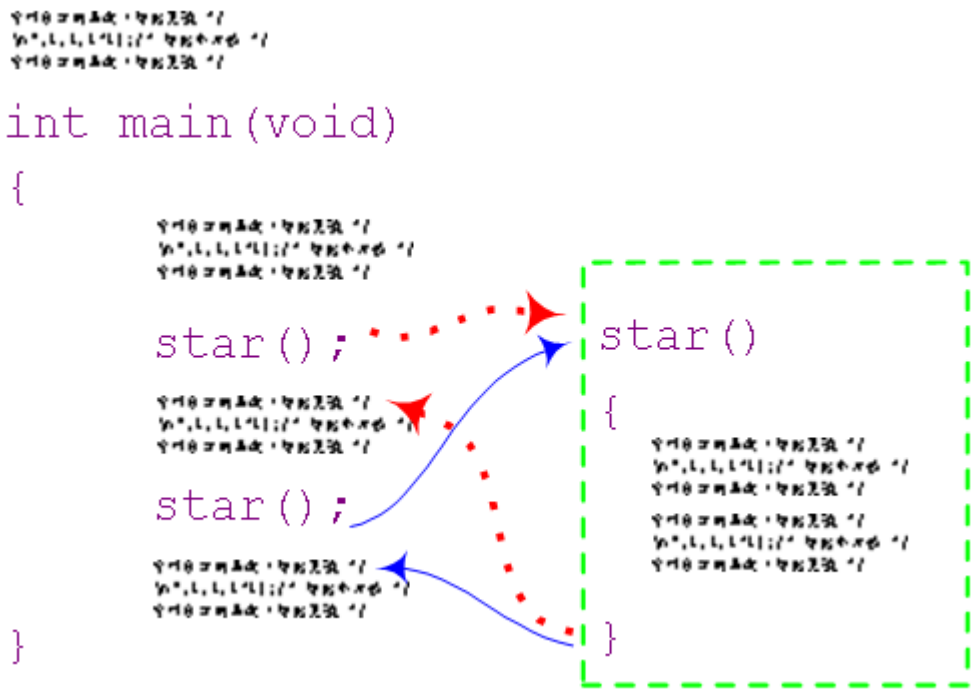
- 下面的範例計算6的平方值，並在運算結果前後列印星號

```
01  // prog6_1, 簡單的函數
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void star(void);                          // 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 << "*";                        // 印出*星號
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：

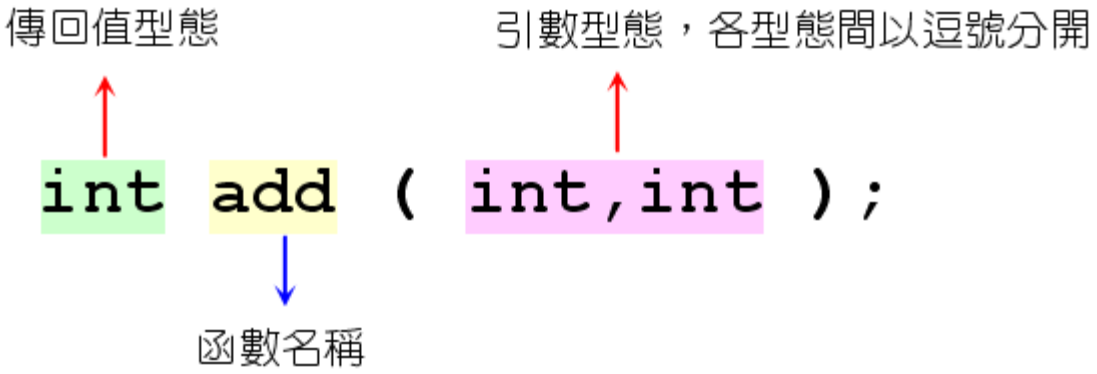


函數原型的宣告、撰寫與呼叫(1/3)

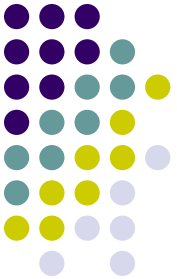
- 下面為「函數原型」（prototype）的宣告格式

```
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



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

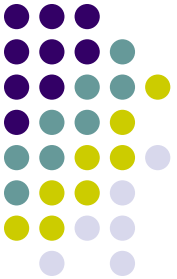
```
傳回值型態 函數名稱(型態 1 引數 1，…，型態 n 引數 n)
{
    變數宣告;
    敘述主體;
    return 運算式;
}
```

- 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)

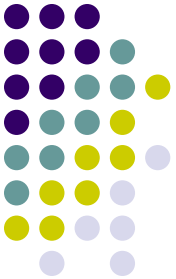


- 下面的敘述為常見的函數呼叫

```
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
- 宣告於函數內的變數稱為「區域變數」（local variable）

```
   ...
int square(int);        → 自訂函數的宣告
   ...
int main(void)          → 主函數
{
   ...
   j=square(i);         → 自訂函數的呼叫
   ...
}
   ...
int square(int i)
{
   int squ;
   squ=i*i;             → 自訂函數的內容
   return squ;
}
```



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

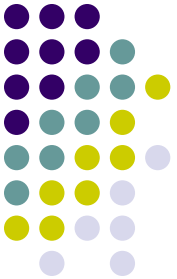
```
int square(int i)
{
    int squ;
    squ=i*i;
    return squ;
}
```

自訂函數的定義與宣告

```
    ...
int main(void)
{
    ...
    j=square(i);
    ...
}
    ...
```

主函數

自訂函數的呼叫

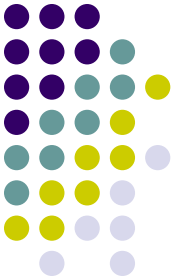


不使用函數原型的方式 (2/2)

- 下面的程式是不使用函數原型的方式所撰寫而成的

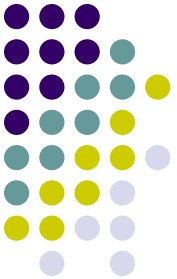
```
01  // prog6_2, 不使用函數原型的方式
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int square(int a)    // 自訂的函數 square(), 計算平方值
06  {
07     int squ;
08     squ=a*a;
09     return squ;
10  }
11
12  int main(void)       // 主程式
13  {
14     cout << "square(6)=" << square(6) <<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

```
/* prog6_2 OUTPUT---
square(6)=36
--------------------*/
```



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）。
- 傳址呼叫，call by address
- 是指呼叫函數時，所傳遞的資料是某個變數的位址。
- 傳值呼叫，call by value
- 將資料的值當做引數來傳遞給函數

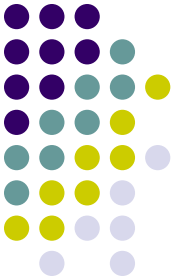


函數的引數與參數 (2/2)

- 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;
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  }
```

```
/* prog6_3 OUTPUT----

In main(),a=3,b=6
In func(),a=13,b=16
After func(),a=3,b=6

--------------------*/
```



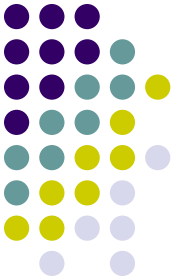
函數的傳回值 (1/3)

- 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;
```

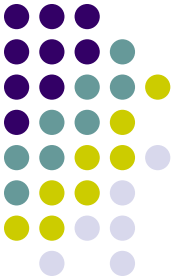


函數的傳回值 (2/3)

- 下面的程式可以利用函數傳回兩個整數的較大值

```
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;
19     else
20        return j;
21  }
```

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

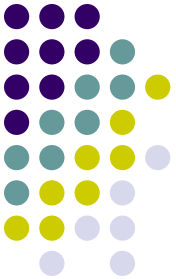


函數的傳回值 (3/3)

```
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;
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

```
/* prog6_5 OUTPUT---

%%%%%%
Printed!!

--------------------*/
```



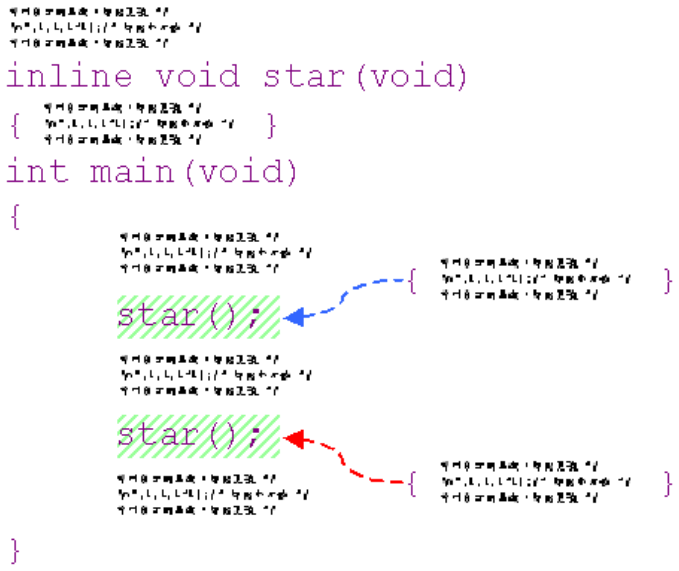
inline函數 (1/3)

- 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

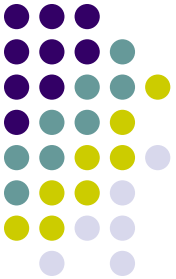
```
inline 傳回值型態 函數名稱(型態 1 引數 1,…,型態 n 引數 n)
{
    變數宣告;
    敘述主體;
    return 運算式;
}
    ...
int main(void)
{    ...    }
```

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



- 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

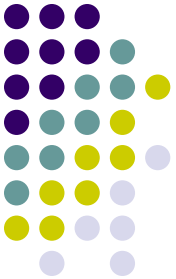


inline函數 (3/3)

- 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
11  {
12     star();
13     cout << "Hello, C++" << endl;
14     star();
15     system("pause");
16     return 0;
17  }
```

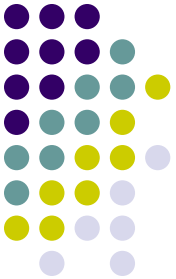
```
/* prog6_6 OUTPUT---
************
Hello, C++
************
--------------------*/
```



變數的等級

- C++提供auto、static auto、extern、static extern及register等五種變數等級
- 宣告變數時，可以一起將變數名稱及其等級同時宣告，如下面的敘述：

```
auto int i;                 // 宣告區域整數變數 i
extern char ch;             // 宣告外部字元變數 ch
static float f;             // 宣告靜態浮點數變數 f
```



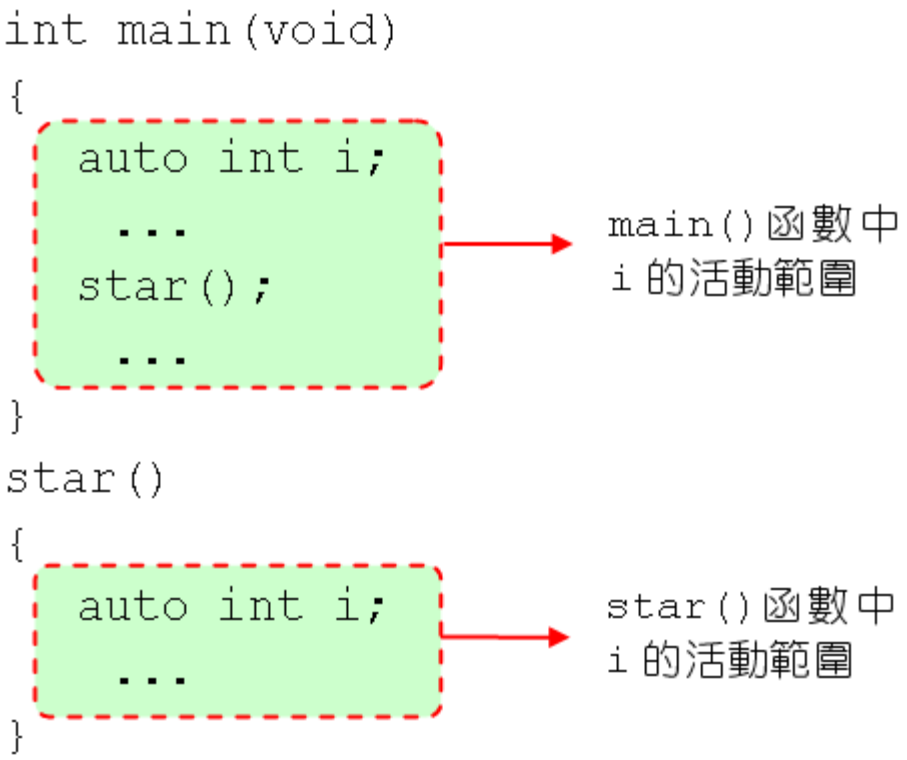
區域變數 (1/3)

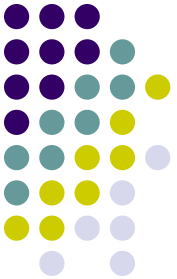
- 區域變數又稱為「自動變數」（automatic variable）
- 包含區域變數的程式碼區塊，區域變數的值自動消失
- 區域變數在程式執行時會以堆疊（stack）的方式存放，屬於動態的變數
- 下面的宣告皆是屬於區域變數的一種：

```
auto int i;     // 宣告區域整數變數 i
char ch;        // 宣告區域字元變數 ch（省略關鍵字 auto）
```

區域變數 (2/3)

- 下圖是區域變數i在所屬區段中的活動範圍之示意圖



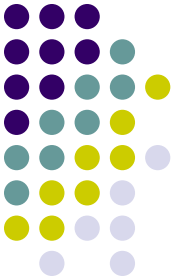


區域變數 (3/3)

- 由下面的程式裡可以看到區域變數的使用

```
01  // prog6_7, 區域變數
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void func(void);                                  // 函數原型的宣告
06  int main(void)
07  {
08     auto int a=10;
09     cout << "In Main(),a=" << a << endl;          // 印出 main() 中 a 的值
10     func();                                       // 呼叫自訂的函數
11     cout << "In Main(),a=" << a << endl;          // 印出 a 的值
12     system("pause");
13     return 0;
14  }
15
16  void func(void)                                  // 自訂的函數 func()
17  {
18     int a=30;
19     cout << "In func(),a=" << a << endl;          // 印出 func() 中 a 的值
20     return;
21  }
```

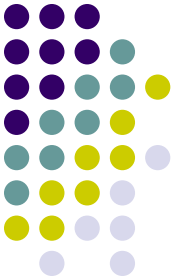
```
/* prog6_7 OUTPUT---

In Main(),a=10
In func(),a=30
In Main(),a=10

-------------------*/
```



靜態區域變數 (1/2)

- 靜態區域變數是在編譯時就已配置固定的記憶體空間
- 包含靜態區域變數的程式碼區塊執行完後，靜態區域變數的值不會自動消失
- 下面的敘述為靜態區域變數的範例

```
static float f;     // 定義靜態區域浮點數變數 f
```

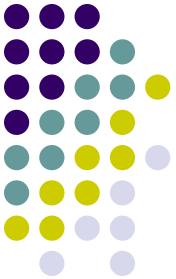


靜態區域變數 (2/2)

- 由下面的程式裡可以看到靜態區域變數a的變化

```
01  // prog6_8, 靜態區域變數
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void func(void);      // 函數原型的宣告
06  int main(void)
07  {
08     func();            // 呼叫自訂的函數
09     func();
10     func();
11     system("pause");
12     return 0;
13  }
14
15  void func(void)       // 自訂的函數 func()
16  {
17     static int a=10;
18     cout << "In func(),a=" << a << endl;   // 印出 func()中 a 的值
19     a+=20;
20     return;
21  }
```

```
/* prog6_8 OUTPUT---

In func(),a=10
In func(),a=30
In func(),a=50

--------------------*/
```



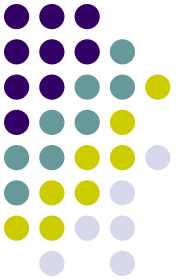
外部變數 (1/4)

- 外部變數（external variable）是在函數外面所宣告的變數
- 外部變數又稱為「總體變數」或「全域變數」
- 下面的程式片段是外部變數的宣告範例

```
int main(void)
{
    extern int a;
    ...
}
 int a;
func()
{   ...   }
```

宣告 a 為外部整數變數，即可拓展外部變數 a 的活動範圍

定義 a 為外部整數變數



外部變數 (2/4)

- 從下圖的內容中可以看到外部變數i的活動範圍

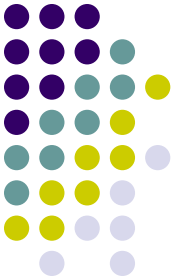
```
int main(void)
{
    extern int i;
    ...
    star();
    ...
}
func()
{   ...   }

int i;

star()
{
   i++;
   ...
}
```

經由宣告後才可使用外部變數 i

無法使用外部變數 i

定義外部變數 i

外部變數 i 的活動範圍

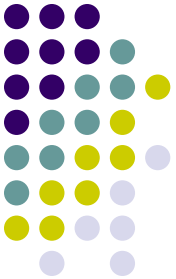


外部變數 (3/4)

- 下面的程式定義外部變數pi，利用它求取圓周及圓面積

```
01  // prog6_9, 外部變數
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void peri(double),area(double);     // 函數原型的宣告
06  int main(void)
07  {
08     extern double pi;                // 定義外部變數 pi
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  }
```

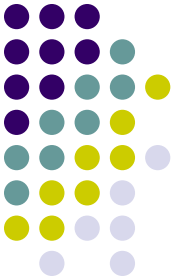
```
/* prog6_9 OUTPUT-----
pi=3.14
radius=1
peripheral length=6.28
area=3.14
----------------------*/
```



外部變數 (4/4)

```
17  double pi=3.14;                          // 外部變數 pi 設值為 3.14
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  }
```

```
/* prog6_9 OUTPUT-----

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

----------------------*/
```



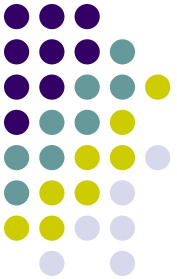
靜態外部變數 (1/2)

- 靜態外部變數只能在一個程式檔內使用
- 下圖為靜態外部變數i的活動範圍

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

static int i;

func()
{   ...   }
star()
{
   i++;
     ...
}
```

定義靜態外部變數 i

靜態外部變數 i 的活動範圍



靜態外部變數 (2/2)

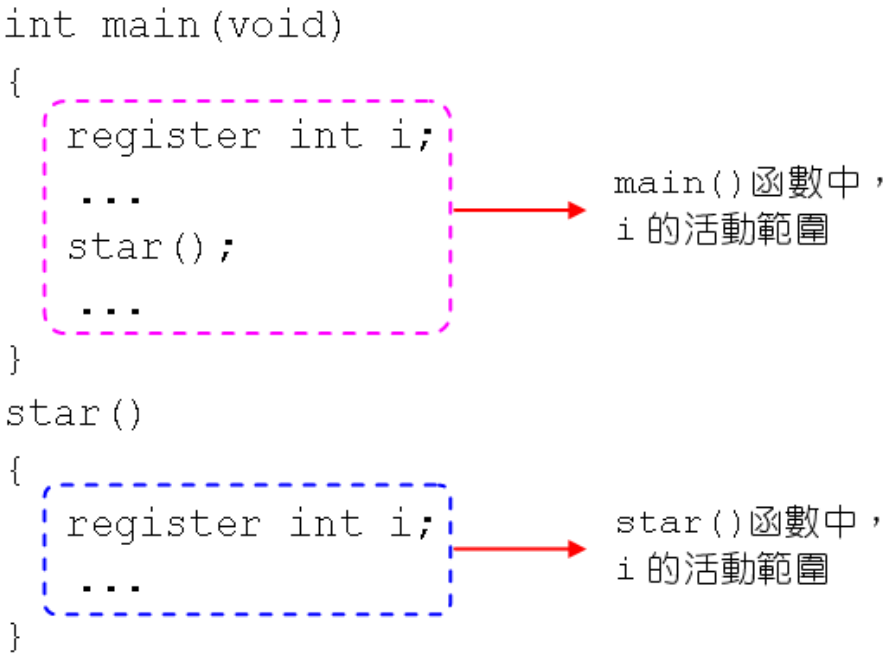
- 下面的程式可以認識靜態外部變數的生命週期與活動範圍

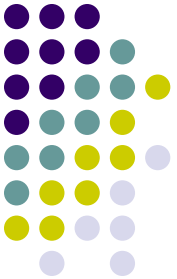
```
01  // prog6_10, 靜態外部變數
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  static int a;                    // 定義靜態外部整數變數 a
06  void odd(void);                  // 函數原型的宣告
07  int main(void)
08  {
09     odd();                        // 呼叫 odd()函數
10     cout << "after odd(), a=" << a << endl;
11     system("pause");
12     return 0;
13  }
14
15  void odd(void)                   // 自訂函數 odd()，判斷 a 為奇數或是偶數
16  {
17     a=10;
18     if(a%2==1)
19        cout << "a=" << a << ", a 是奇數" << endl;      // 印出 a 為奇數
20     else
21        cout << "a=" << a << ", a 是偶數" << endl;      // 印出 a 為偶數
22     return;
23  }
```

```
/* prog6_10 OUTPUT---
a=10, a 是偶數
after odd(), a=10
----------------------*/
```

暫存器變數 (1/3)

- 暫存器變數利用CPU的暫存器（register）來存放資料
- 暫存器變數以register 來宣告
- 暫存器變數i的活動範圍

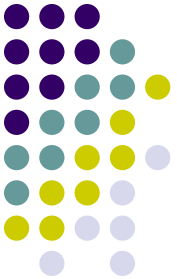




暫存器變數 (2/3)

- 下面的程式是使用暫存器變數的範例

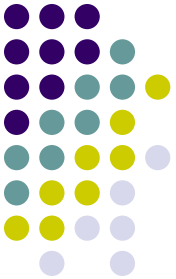
```
01  // prog6_11, 暫存器變數的使用範例
02  #include <iostream>
03  #include <cstdlib>
04  #include <ctime>
05  #include <iomanip>
06  using namespace std;
07  int main(void)
08  {
09     time t start,end;
10     register int i,j;                    // 定義暫存器整數變數 i 與 j
11     start=time(NULL);                    // 記錄開始時間
12     for(i=1;i<=50;i++)
13     {
14        for(j=1;j<=50;j++)
15        {
16           cout << setw(2) << i << "*" << setw(2) << j;
17           cout << "=" << setw(4) << i*j << "\t";
18        }
19        cout << endl;
20     }
```



暫存器變數 (3/3)

```
21      end=time(NULL);                          // 記錄結束時間
22      cout << "It spends " << difftime(end,start) << " seconds";
23      system("pause");
24      return 0;
25  }
```

```
/* prog6_11 OUTPUT-------------------------------------
 1* 1=   1    1* 2=   2    1* 3=   3    1* 4=   4    1* 5=   5
 1* 6=   6    1* 7=   7    1* 8=   8    1* 9=   9    1*10=  10
 1*11=  11    1*12=  12    1*13=  13    1*14=  14    1*15=  15
                                 ⋮
50*36=1800   50*37=1850   50*38=1900   50*39=1950   50*40=2000
50*41=2050   50*42=2100   50*43=2150   50*44=2200   50*45=2250
50*46=2300   50*47=2350   50*48=2400   50*49=2450   50*50=2500

It spends 1 seconds
----------------------------------------------------------*/
```

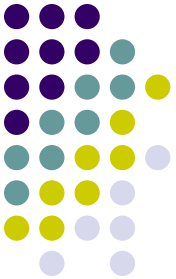


呼叫多個函數 (1/2)

- 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;
09     fact(a);
10     sum(a);
11     system("pause");
12     return 0;
13  }
14
```

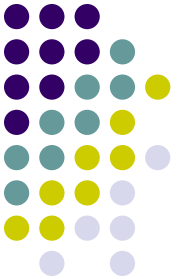
```
/* prog6_12 OUTPUT---

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
----------------------*/
```

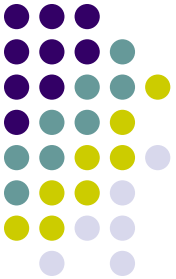


函數之間的相互呼叫 (1/2)

- 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    → 由 fact()函數呼叫的 sum()函數
1+2+...+10=55   → 由 main()函數呼叫的 sum()函數

----------------------*/
```

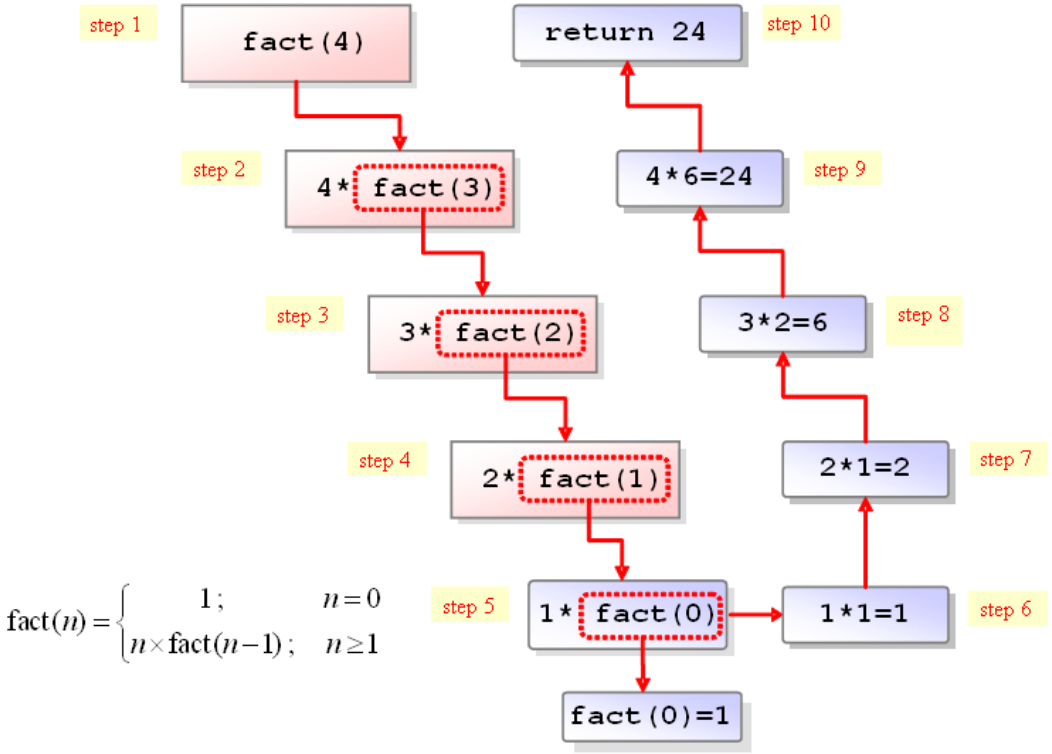


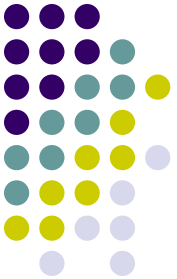
函數之間的相互呼叫 (2/2)

```
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



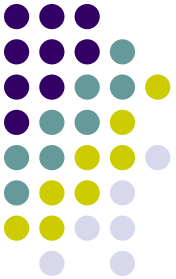


遞迴函數 (2/6)

- 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
```

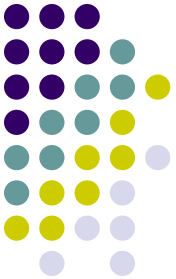
```
/* prog6_14 OUTPUT---

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  }
```

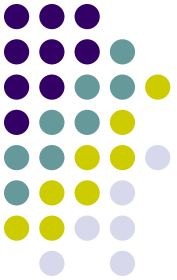
```
/* prog6_14 OUTPUT---

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*fact(3)
2	3	fact(3)，未知	3*fact(2)
3	2	fact(2)，未知	2*fact(1)
4	1	fact(1)，未知	1*fact(0)
5	0	fact(0)=1	1
6	1	fact(1)=1	1
7	2	fact(2)=2	2
8	3	fact(3)=6	6
9	4	fact(4)=24	24

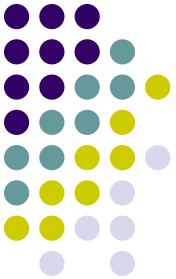


遞迴函數 (5/6)

- 下面是以遞迴函數求次方的程式

```
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;          // 印出 a^b 的結果
11     system("pause");
12     return 0;
13  }
14
15  int power(int a,int b)                  // 自訂函數 power(),計算 a^b
16  {
17     if(b==0)
18        return 1;
19     else return (a*power(a,b-1));
20  }
```

```
/* prog6_15 OUTPUT---
2^3=8
----------------------*/
```



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* power(2,2)
2	2	2	power(2,2)，未知	2* power(2,1)
3	2	1	power(2,1)，未知	2* 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-