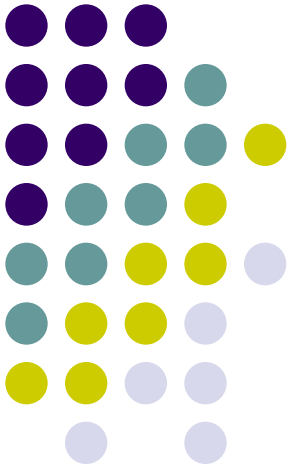
第七章
再談函數

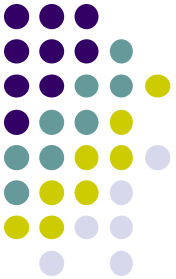
認識變數的等級

認識參照與函數

學習函數的多載

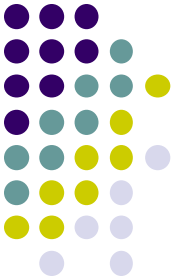
認識引數的預設值





變數的等級

- C++提供數種變數等級
  - 區域變數: auto
  - 外部變數: extern (全域變數)
  - 靜態區域變數: static auto
  - 靜態外部變數: static extern
  - 暫存變數: register
  - 常數變數: const



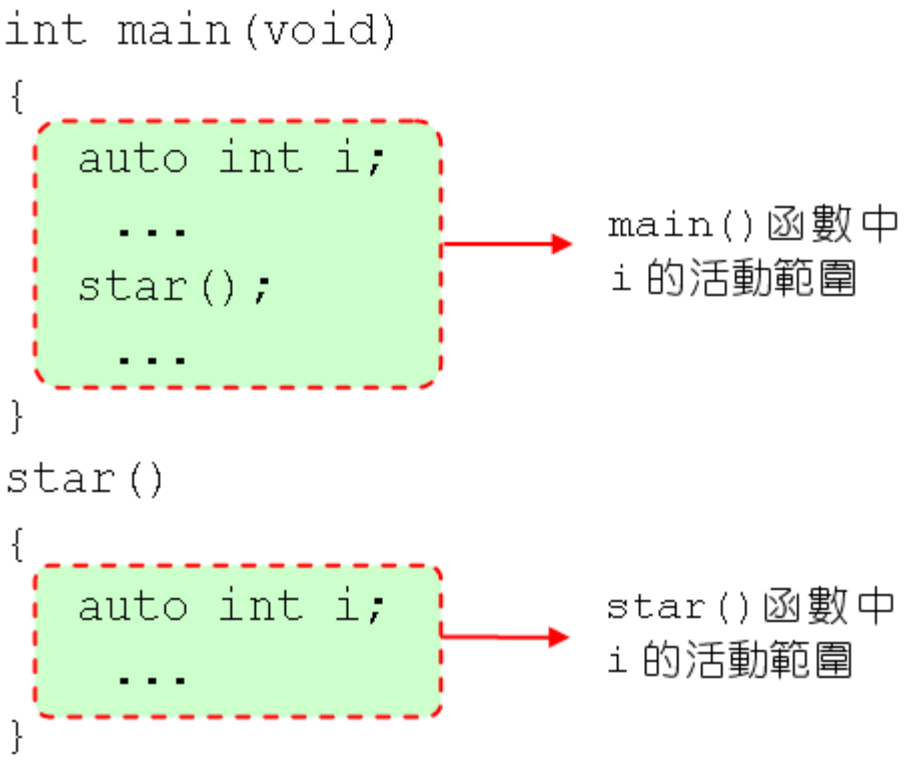
區域變數 (1/3)

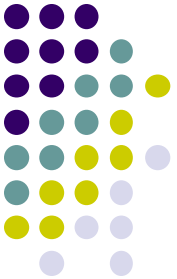
- 區域變數又稱為「自動變數」（automatic variable）
- 包含區域變數的程式碼區塊，區域變數的值自動消失
- 區域變數在程式執行時會以堆疊（stack）的方式存放，屬於動態的變數
- 下面的宣告皆是屬於區域變數的一種：

```
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

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

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

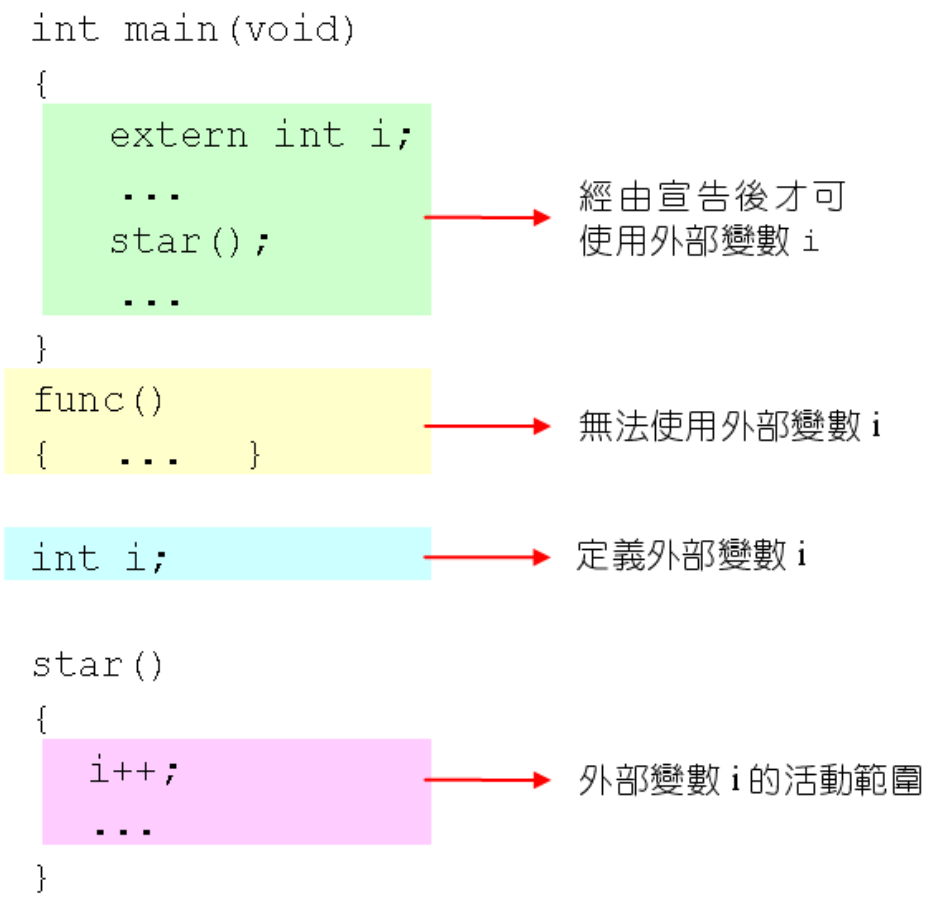
```
int main(void)
{
    extern int a;
    ...
}
 int a;
func()
{   ...   }
```

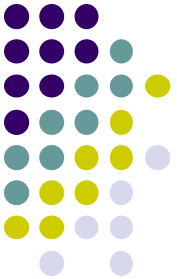
宣告 a 為外部整數變數，即可拓展外部變數 a 的活動範圍

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



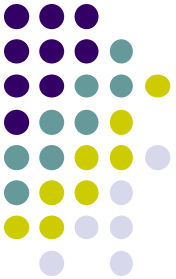


外部變數 (3/4)

- 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  }
```

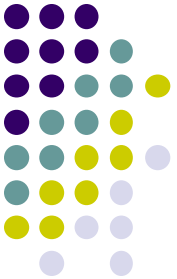
```
/* prog6_9 OUTPUT-----
pi=3.14
radius=1
peripheral length=6.28
area=3.14
----------------------*/
```



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  }
```

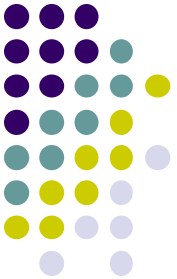
```
/* prog6_9 OUTPUT-----

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

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

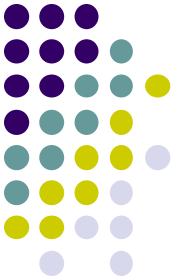
靜態外部變數 (1/2)

- 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)

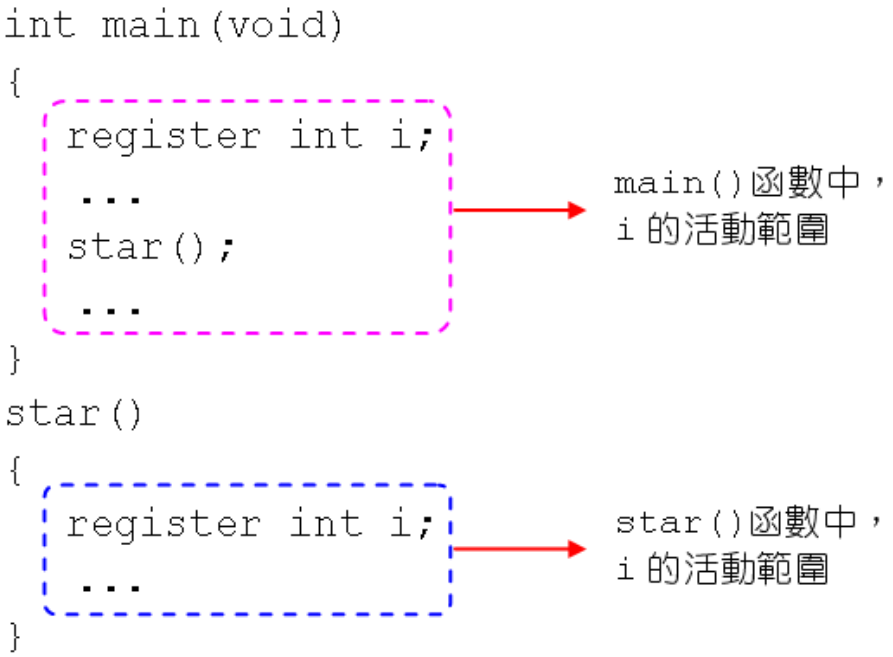
- 下面的程式可以認識靜態外部變數的生命週期與活動範圍

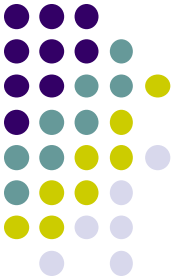
```
01  // prog6_10, 靜態外部變數
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  static int a;                    // 定義靜態外部整數變數 a
06  void odd(void);                  // 函數原型的宣告
07  int main(void)
08  {
09     odd();                        // 呼叫 odd()函數
10     cout << "after odd(), 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

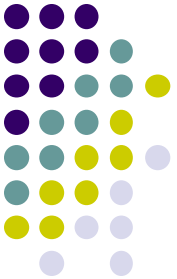




暫存器變數 (2/3)

- 下面的程式是使用暫存器變數的範例

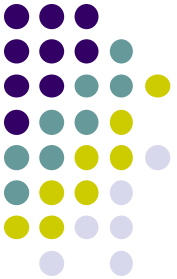
```
01  // prog6_11, 暫存器變數的使用範例
02  #include <iostream>
03  #include <cstdlib>
04  #include <ctime>
05  #include <iomanip>
06  using namespace std;
07  int main(void)
08  {
09     time t start,end;
10     register int i,j;                    // 定義暫存器整數變數 i 與 j
11     start=time(NULL);                    // 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
-----------------------------------------------------*/
```



const修飾子 (1/2)

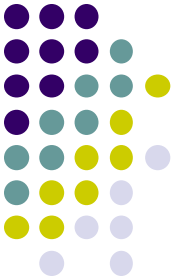
- 利用const來宣告變數，可避免變數值被修改
- const來宣告變數的範例

```
const short int max=32767;
```

```
01  // prog7_12, 使用 const
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07     const short int max=4;
08     int i;
09     for(i=1;i<=max;i++)          // 計算 i 的平方
10        cout << i << "*" << i << "=" << i*i << "\t";
11     system("pause");
12     return 0;
13  }
```

- 此程式是利用const宣告max為整數變數

```
/* prog7_12 OUTPUT----------
1*1=1   2*2=4   3*3=9   4*4=16
--------------------------*/
```



const修飾子 (2/2)

- 此程式是利用const宣告max為整數變數

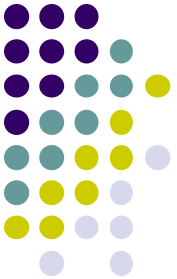
```
01  // prog7_12, 使用 const
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07     const short int max=4;
08     int i;
09     for(i=1;i<=max;i++)          // 計算 i 的平方
10        cout << i << "*" << i << "=" << i*i << "\t";
11     system("pause");
12     return 0;
13  }
```

```
/* prog7_12 OUTPUT----------
1*1=1   2*2=4   3*3=9   4*4=16
--------------------------*/
```

- 如果在第8行與第9行中間加入下列敘述

```
max=10;      // 修改常數 max 的值
```

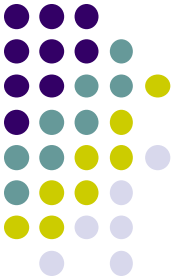
編譯器會出現assignment of read-only variable `max' 的訊息，說明這是不能被更改的常數



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）。

- 傳值呼叫（call by value）
  - 將資料的「值」當做引數來傳遞給函數。
- 傳參照呼叫（call by reference）
  - 將資料所在位置，以另一個「別名」作為識別。
- 傳址呼叫（call by address）
  - 呼叫函數時傳遞資料的「位址」至另一個參數。



函數的傳值 (1/2)

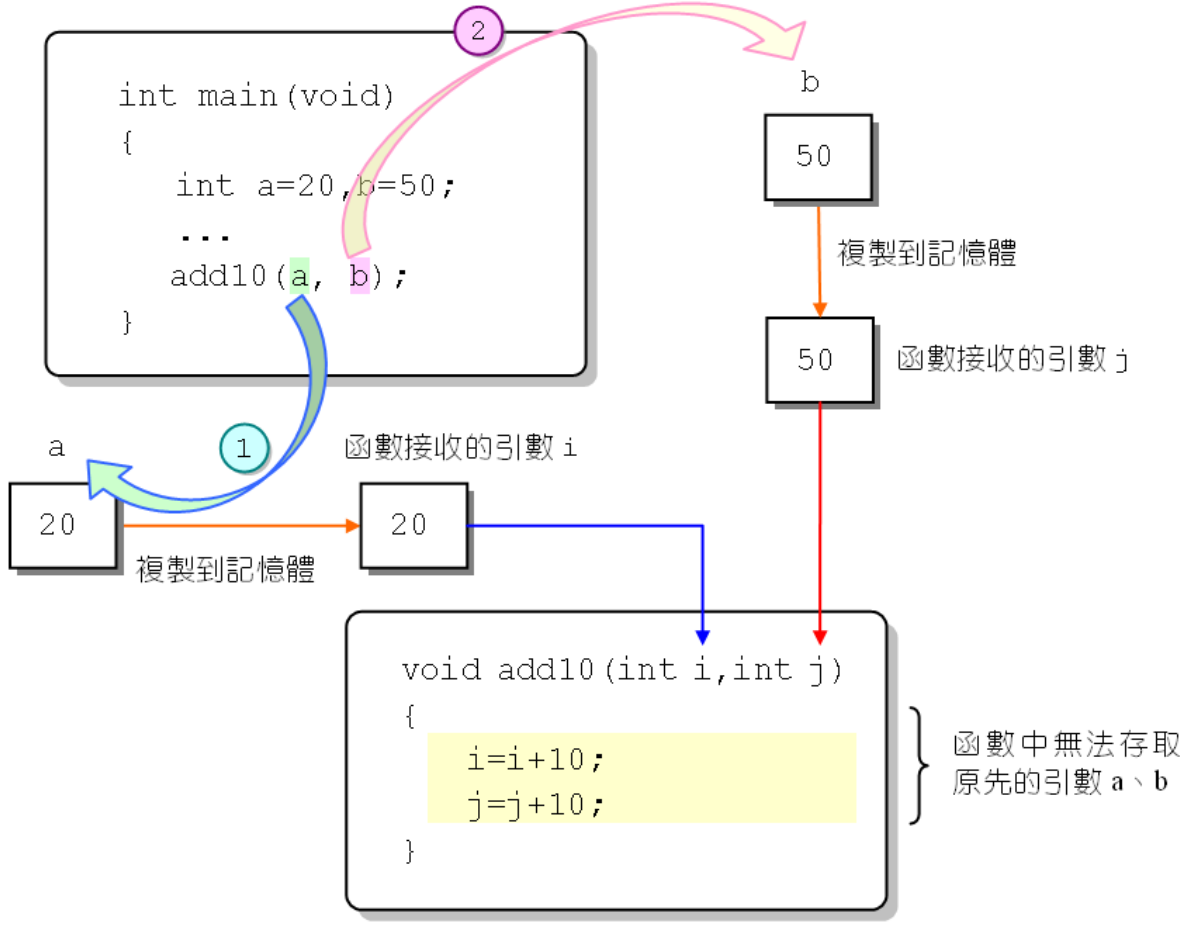
- 下面的程式可用來觀察函數裡，變數值的變化情形

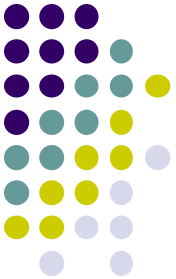
```
01  // prog7_1, 函數的傳值
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void add10(int,int);
06  int main(void)
07  {
08     int a=20,b=50;
09     cout << "before calling add10(): ";
10     cout << "a=" << a << ", b=" << b << endl;  // 印出 a、b 的值
11     add10(a,b);
12     cout << "after called add10(): ";
13     cout << "a=" << a << ", b=" << b << endl;  // 印出 a、b 的值
14     system("pause");
15     return 0;
16  }
17
18  void add10(int i,int j)
19  {
20     i=i+10;
21     j=j+10;
22  }
```

```
/* prog7_1 OUTPUT----------------
before calling add10(): a=20, b=50
after called add10(): a=20, b=50
--------------------------------*/
```

函數的傳值 (2/2)

- 以prog7_1為例，將函數傳值呼叫的方式繪製成圖



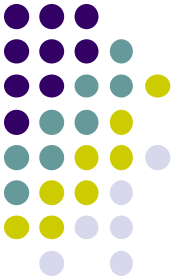


參照的基本認識 (1/3)

- C++提供參照（reference）來做為資料的別名
- 參照的宣告格式如下

```
資料型態 變數名稱;
資料型態 &參照名稱=變數名稱;
```

- 參照與指標
  - 效果一樣，都會更動到原本的資料
  - 差別在於參照使用起來與一般變數一樣，較為直覺，且在宣告的時候就要給定初值



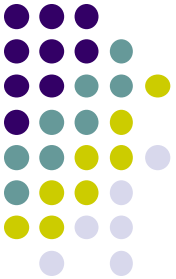
參照的基本認識 (2/3)

- 想為整數變數a使用參照ref，可以做出如下的宣告：

```
int a;              // 宣告整數變數 a
int &ref=a;         // 宣告變數 a 的參照 ref，並使 ref 指向變數 a
```

- 如果想將ref的值設成10，可以寫出下面的敘述：

```
int& ref=a;         // 宣告 ref 為變數 a 的參照
```

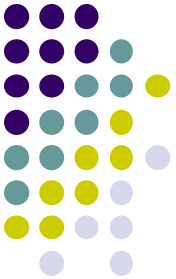


參照的基本認識 (3/3)

- 下面的程式碼是參照的使用範例常用的流程圖符號

```
01  // prog7_2, 參照的認識
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07     int num=5;
08     int &rm=num;                    // 宣告 rm 為 num 的參照
09
10     rm=rm+10;                       // 參照 rm 加 10
11     cout << "num=" << num << endl;  // 印出 num 的值
12     cout << "rm=" << rm << endl;    // 印出 rm 的值
13     system("pause");
14     return 0;
15  }
```

```
/* prog7_2 OUTPUT---

num=15
rm=15

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



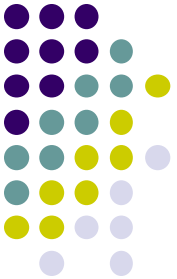
傳遞參照到函數 (1/4)

- 下面是將參照當成引數傳入函數的原型宣告

```
int func(int &,char &);          // 將參照當成引數傳入函數的函數原型之宣告
```

- 在定義函數時，於變數名稱前加上參照運算子&即可

```
int func(int &ref1,char &ref2)    // 將參照當成引數傳入函數的函數之定義
{
   ...
}
```



傳遞參照到函數 (2/4)

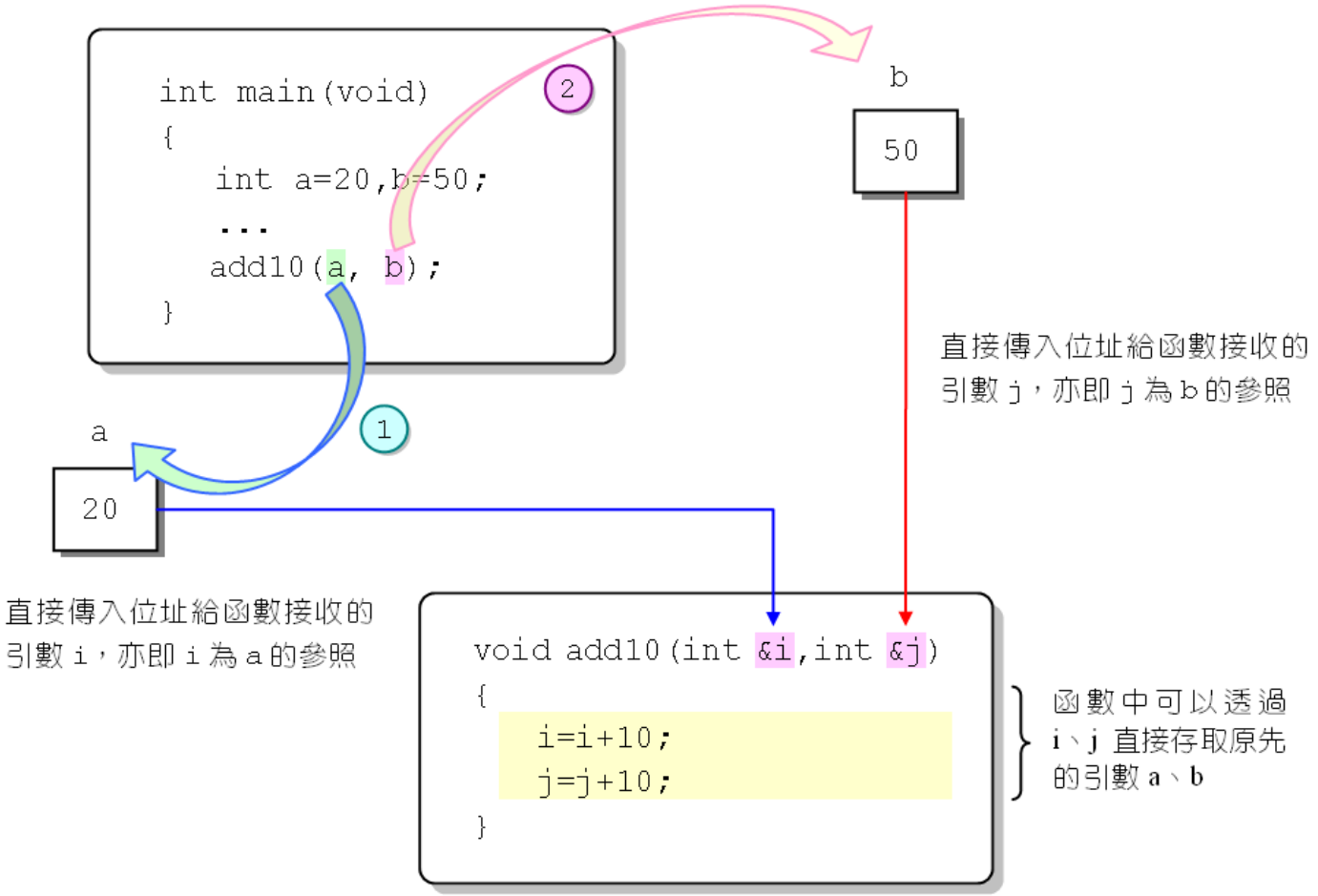
- prog7_3是以參照的方式傳遞到函數

```
01  // prog7_3, 傳參照到函數
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void add10(int &,int &);
06  int main(void)
07  {
08     int a=20,b=50;
09     cout << "before calling add10(): ";
10     cout << "a=" << a << ", b=" << b << endl;  // 印出 a、b 的值
11     add10(a,b);
12     cout << "after called add10(): ";
13     cout << "a=" << a << ", b=" << b << endl;  // 印出 a、b 的值
14     system("pause");
15     return 0;
16  }
17
18  void add10(int &i,int &j)
19  {
20     i=i+10;
21     j=j+10;
22     return;
23  }
```

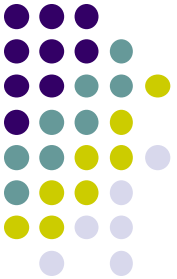
```
/* prog7_3 OUTPUT----------------
before calling add10(): a=20, b=50
after called add10(): a=30, b=60
---------------------------------*/
```

傳遞參照到函數 (3/4)

- 下圖是以prog7_3為例，說明參照呼叫的方式



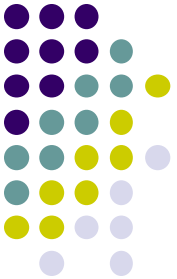
傳遞參照到函數 (4/4)



- 下面的程式是利用print() 函數，印出欲列印的字元

```
01  // prog7_4, 參照的傳遞
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void print(char,int &);
06  int main(void)
07  {
08     int i,count=0;
09     for(i=0;i<3;i++)
10        print('*',count);
11     cout << endl;
12     for(i=0;i<5;i++)
13        print('$',count);
14     cout << endl;
15     cout << "print() function is called " << count << " times.";
16     cout << endl;
17     system("pause");
18     return 0;
19  }
20
21  void print(char ch, int& cnt)          // 自訂函數print()
22  {
23     cout << ch;
24     cnt++;
25     return;
26  }
```

```
/* prog7_4 OUTPUT------------------
***
$$$$$
print() function is called 8 times.
-----------------------------------*/
```



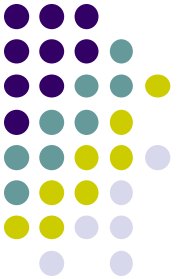
傳回值為參照的函數 (1/2)

- 函數的傳回值也可以是參照
- 舉例來說，於程式中宣告一名為max的函數，可傳回兩個整數中較大值之參照，函數原型為：

```
int &max(int &,int &);
```

- 想將傳回的參照值設為100，即可寫出下面的敘述：

```
max(i,j)=100;
```

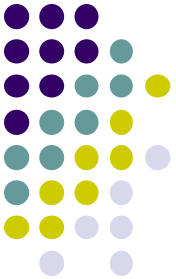


傳回值為參照的函數 (2/2)

- 下面是函數傳回參照的使用範例

```
01  // prog7_5, 傳回值為參照
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int &max(int &,int &);          // 宣告函數原型，其傳回值為參照
06  int main(void)
07  {
08     int i=10,j=20;
09     max(i,j)=100;                // 將 max()函數傳回的參照值重設為 100
10     cout << "i=" << i << ",j=" << j << endl;
11     system("pause");
12     return 0;
13  }
14
15  int &max(int &a,int &b)
16  {
17     if(a>b)
18        return a;
19     else
20        return b;
21  }
```

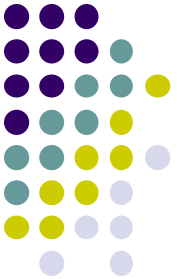
```
/* prog7_5 OUTPUT---
i=10,j=100
--------------------*/
```



多載 (1/5)

- 多載（overloading），是指相同的函數名稱，如果引數個數不同，或者是引數個數相同、型態不同的話，函數便具有不同的功能
- 以一個簡單的例子說明「函數的多載」之使用

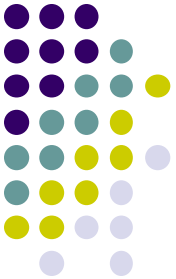
```
01  // prog7_6, 引數型態不同的函數多載
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int add(int,int);                    // 以多載的方式宣告函數原型
06  double add(double,double);

/* prog7_6 OUTPUT---

10+20=30
2.3+3.5=5.8

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



多載 (2/5)

```
07  int main(void)
08  {
09     int a=10,b=20;
10     double x=2.3,y=3.5;
11     cout << a << "+" << b << "=" << add(a,b) << endl;
12     cout << x << "+" << y << "=" << add(x,y) << endl;
13     system("pause");
14     return 0;
15  }
16
17  int add(int i,int j)                    // 自訂函數 add()
18  {
19     return i+j;                          // 傳回 i+j 的值
20  }
21
22  double add(double i,double j)           // 自訂函數 add()
23  {
24     return i+j;                          // 傳回 i+j 的值
25  }
```

```
/* prog7_6 OUTPUT---

10+20=30
2.3+3.5=5.8

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



多載 (3/5)

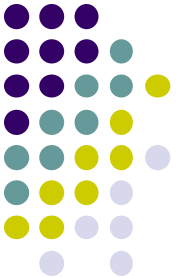
- 如果只有傳回值型態不同，則不能多載
  - 舉例來說，某個函數的原型如下

```
int func(int,int);    // 函數原型，傳回值型態為 int
```

  - 這個函數原型會與下面的原型相衝突而產生錯誤

```
long func(int,int);  // 函數原型，傳回值型態為 long
```

- 只有傳回值型態不同，則會讓編譯器難以分辨到底該使用哪一個函數

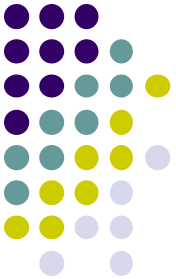


多載 (4/5)

- 接下來再看一個引數個數不同的函數多載

```
01  // prog7_7, 引數個數不同的函數多載
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void print(void);                  // 以多載的方式宣告函數原型
06  void print(int);
07  void print(char,int);
08  int main(void)
09  {
10     cout << "calling print(), ";
11     print();
12     cout << "calling print(8), ";
13     print(8);
14     cout << "calling print('+',3), ";
15     print('+',3);
16     system("pause");
17     return 0;
18  }
19
```

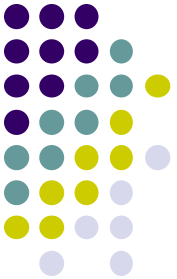
```
/* prog7_7 OUTPUT---------
calling print(), *****
calling print(8), ********
calling print('+',3), +++
-------------------------*/
```



多載 (5/5)

```
20  void print(void)                    // 沒有引數的print()，印出5個*
21  {
22     print(5);                        // 呼叫26~33行的print()，並傳入整數5
23     return;
24  }
25
26  void print(int a)                   // 有一個引數的print()，印出a個*
27  {
28     int i;
29     for(i=0;i<a;i++)
30        cout << "*";
31     cout << endl;
32     return;
33  }
34
35  void print(char ch,int a)           // 有二個引數的print()，印出a個ch
36  {
37     int i;
38     for(i=0;i<a;i++)
39        cout << ch;
40     cout << endl;
41     return;
42  }
```

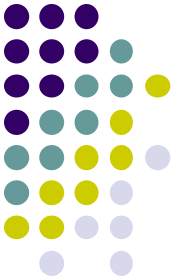
```
/* prog7_7 OUTPUT---------
calling print(), *****
calling print(8), ********
calling print('+',3), +++
-------------------------*/
```



預設引數 (1/4)

- 未傳入足夠的引數到函數時，預設的引數值就會被使用，這種方式稱為「預設引數」（default argument）
- 要設定預設，可在定義原型時，於引數後面設值給它

```
double circle(double, double pi=3.14);
```

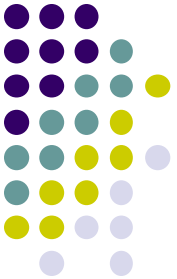


預設引數 (2/4)

- 下面的程式是函數引數預設值的使用範例

```
01  // prog7_8, 引數的預設值
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  double circle(double,double pi=3.14);  // 函數原型,第 2 個引數預設為 3.14
06  int main(void)
07  {
08     cout << "circle(2.0,3.14159)=" << circle(2.0,3.14159) << endl;
09     cout << "circle(2.0)=" << circle(2.0) << endl;
10     system("pause");
11     return 0;
12  }
13
14  double circle(double r, double pi)  // circle()函數的定義，計算圓面積
15  {
16     return (pi*r*r);
17  }
```

```
/* prog7_8 OUTPUT---------
circle(2.0,3.14159)=12.5664
circle(2.0)=12.56
-------------------------*/
```



預設引數 (3/4)

- 沒有使用預設值的引數，要放置在引數列的左邊
  - 舉例來說，函數原型如下

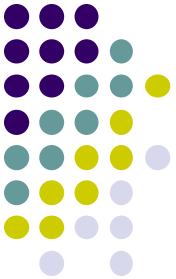
```
void func(int, double, int n=3, char ch='k');
```

  - 下面都是合法的func() 函數呼叫

```
func(5, 1.9);              // n 預設為 3，ch 預設為'k'
func(8, 6.3, 4);           // ch 預設為'k'
func(4, 3.7, 9, 'a');      // 均不使用預設值
```

  - 下列的函數呼叫，會造成編譯時期或是邏輯上的錯誤：

```
func();                    // 最少必須有兩個引數
func(6);                   // 最少必須有兩個引數
func(2, 1.9, 'b');         // 邏輯錯誤的函數呼叫
```



預設引數 (4/4)

- 下面的程式是有加入引數預設值的函數呼叫

```
01  // prog7_9, 引數的預設值
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int sum(int start=1,int end=10,int di=1);   // 函數原型
06  int main(void)
07  {
08     cout << "sum()=" << sum() << endl;
09     cout << "sum(2)=" << sum(2) << endl;
10     cout << "sum(2,8)=" << sum(2,8) << endl;
11     cout << "sum(1,15,3)=" << sum(1,15,3) << endl;
12     system("pause");
13     return 0;
14  }
15
16  int sum(int start,int end,int di)           // 計算數值的累加
17  {
18     int i,total=0;
19     for(i=start;i<=end;i+=di)
20        total+=i;
21     return total;
22  }
```

```
/* prog7_9 OUTPUT---

sum()=55
sum(2)=54
sum(2,8)=35
sum(1,15,3)=35

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

命令列引數的使用(1/3)

- 在MS-DOS模式下鍵入如下的指令：

```
type mytext.txt
dir c:\
```

其中mytext.txt與c:\ 均屬於命令列的引數

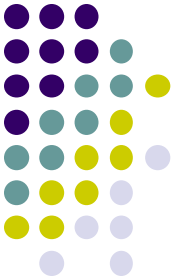
- 命令列引數的使用格式

```
int main(argc, argv)
int argc;
char *argv[];
{
   ...
}
```

```
int main(int argc, char *argv[])
{
   ...
}
```

argv代表引數值，就是使用者在命令列中輸入的資料

argc代表包括指令本身的引數個數

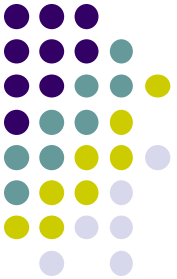


命令列引數的使用(2/3)

- 下面的程式是命令列引數的使用範例

```
/* prog7_18 OUTPUT----------

C:\>sayhello How do you do?
The value of argc is 5
argv[0]=sayhello
argv[1]=How
argv[2]=do
argv[3]=you
argv[4]=do?

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

```
01  // prog7_18, 命令列引數的使用
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(int argc, char *argv[])
06  {
07     int i ;
08     cout << "The value of argc is " << argc;  // 印出命令列引數的內容
09     cout << endl;
10     for(i=0;i<argc;i++)
11        cout << "argv[" << i << "]=" << argv[i] << endl;
12     system("pause");
13     return 0;
14  }
```



命令列引數的使用(3/3)

- 下面是另一個命令列引數的使用範例

```
01  // prog7_19, 命令列引數的使用
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(int argc, char *argv[])
06  {
07     int a=atoi(argv[1]);    // 將命令列引數轉換成數值
08     int b=atoi(argv[2]);
09     cout << a << "+" << b << "=" << a+b << endl;
10     system("pause");
11     return 0;
12  }
```

```
/* prog7_19 OUTPUT---

C:\>sample 2 5
2+5=7

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


-The End-