

# 第三章 細部檢視類別

3-1 指定物件

3-2 傳遞物件給函數

3-3 從函數中傳回物件

3-4 簡介夥伴函數

## 3-1 指定物件

### ◆ 物件之間的指定(assignment)運算

- `int x = 5, y ;`
- `y = x ;`
- `myclass ob1, ob2 ;`
- `ob1 = ob2 ; // what happen?`

# What happen?

bit-wise copy (逐位元的複製)

```
struct test { int a , b } ;
```

```
void main() {
```

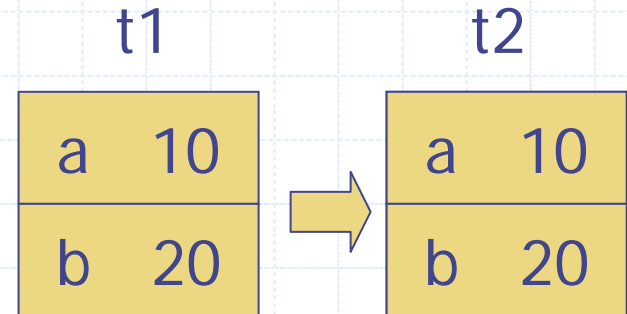
```
    test t1, t2 ;
```

```
    t1.a = 10; t1.b = 20 ;
```

```
    t2 = t1 ;
```

```
    cout << t2.a << " " << t2.b << endl ;
```

```
}
```



# 物件間的指定 (assignment)

```
#include <iostream>
using namespace std ;
class myclass {
    int a, b ;
public:
    myclass() {}
    myclass(int x, int y) {a = x; b = y ;}
    void show() { cout << a << " " << b ;}
};
void main() {
    myclass ob1(5, 3), ob2;
    ob2 = ob1; ob2.show() ;
}
```

Q1: Output?

Ans: bitwise copy for all data members

Q2: 這個程式有無bug?

# 範例一：指定敘述是否需同一 類型？

```
class myclass {int a, b ; public: void show();};
    void myclass::show(){cout<<a<<b<<endl;}
class yourclass{ int a, b; public: void show();} ;
    void yourclass ::show(){cout<<a<<b<<endl;}
int main() {
    myclass ob1 ;
    yourclass ob2;
    ob1 = ob2 ;
    return 0 ;
}
```

## 範例二：物件中有複合資料

```
class stack {
    int stk[10];
    int tos;
public:
    stack(){
        tos=0;
        for(int
i=0;i<10;i++)
            stk[i]=0;
    } void push(int num)
    {
        stk[tos]=num;
        tos++;
    }
};
```

```
int pop()
{
    return stk[--tos];
}
```

```
void print()
{
    for(int
i=0;i<tos;i++)
        cout<<stk[i]<<endl;
}
```

```
};
```

## 範例二：物件中有複合資料

```
int main() {  
    stack s1, s2 ;  
    s1.push(10); s1.push(20); s1.print() ;  
    s2 = s1 ; s2.print() ; return 0 ;  
}
```

# 範例三：如果物件中有指標

Q: 以下程式會發生哪些問題?

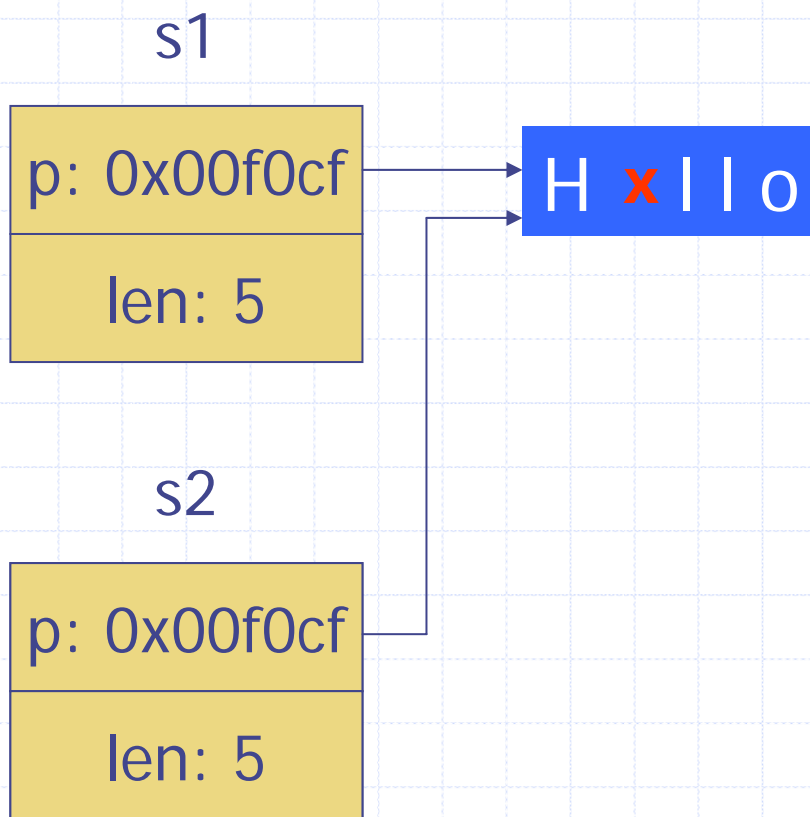
```
class strtype {
    char *p ; int len ;
public:
    strtype(){
        p=new char[10];
        len=1;
        p[0]='\0';
    }
    strtype(char *s) {
        p = new char[strlen(s)+1];
        len=strlen(s)+1;
        p=strcpy(p, s) ;
    }
    ~strtype() { delete[] p ; }
    void setChar(int pos, char c) { p[pos] = c ; }
    void show(){ cout<<len<<endl;}
};

void main() {
    strtype s1("Hello"), s2 ;
    s2 = s1 ;
    s2.setChar(1, 'x') ; s1.show() ;
}
```



# 範例三：Problem 1

```
strtype s1("Hello"), s2;
```

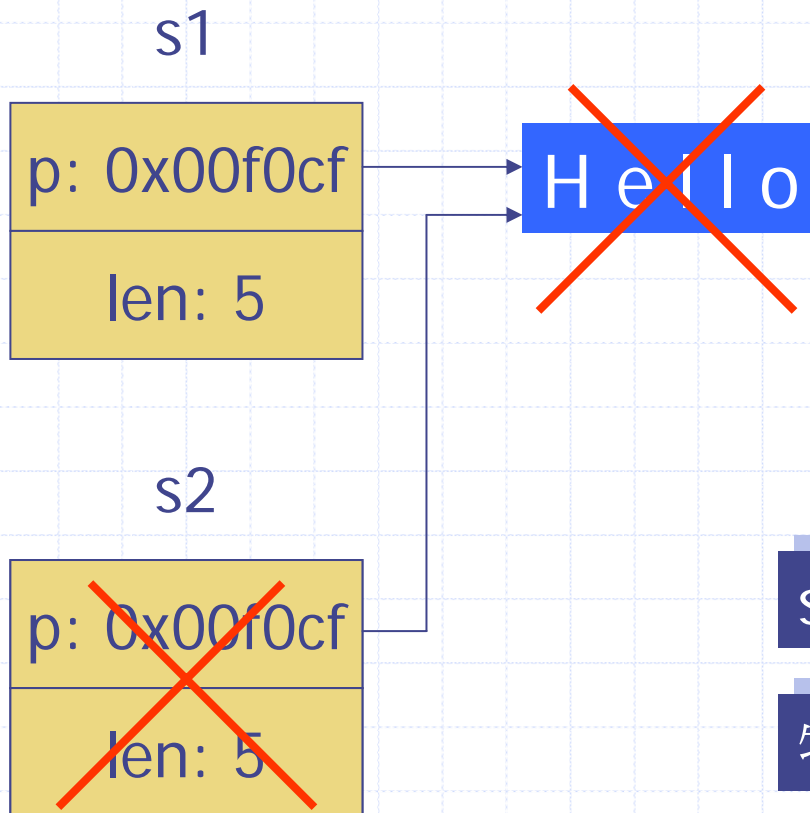


(1) 執行 `s2 = s1 ;`

(2) 執行 `s2.setChar(1, 'x')`

What's wrong?

# 範例三：Problem 2



```
void main() {  
    strtype s1("Hello");  
    strtype s2 ;  
    s2 = s1 ;  
}
```

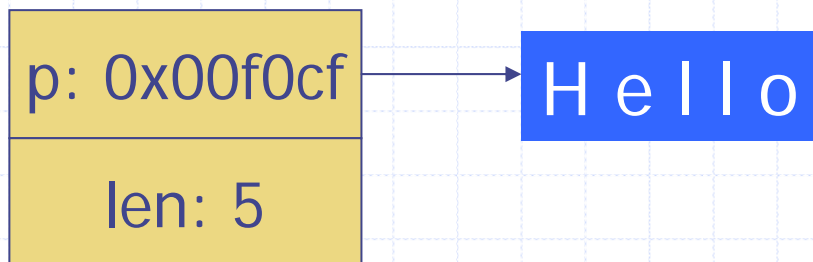
s1, s2何時被消滅?

物件消滅時會發生何事?

# 解決方式

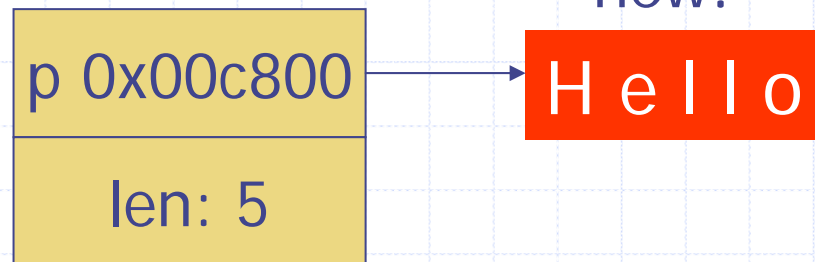
```
void main() {  
    strtype s1("Hello");  
    strtype s2;  
    s2 = s1;  
}
```

s1



```
void main() {  
    strtype s1("Hello");  
    strtype s2;  
    s2.set(s1);  
}
```

s2



# 解決方式

◆ 請寫operator=

```
class strtype{
```

```
.....
```

```
strtype& operator=(const strtype& s) {
```

```
    p = new char[s.len+1] ;
```

```
    strcpy(p, s.p) ; len = s.len ;
```

```
    return *this ;
```

```
}
```

```
};
```

# 解決方式

```
class strtype {
    char *p ; int len ;
public:
    strtype(){
        p=new char[10];
        len=1;
        p[0]='\0';
    }
    strtype(char *s) {
        p = new char[strlen(s)+1];
        len=strlen(s)+1;
        p=strcpy(p, s) ;
    }
}
```

Q: 以下程式會發生哪些問題?

```
~strtype() { delete[] p; }
strtype&
operator=(const strtype& s) {
    p = new char[s.len+1] ;
    strcpy(p, s.p) ;
    len = s.len ;
    return *this ;
}
void setChar(int pos, char c)
{ p[pos] = c ; }
void show(){ cout<<len<<endl;}
};
```

# 解決方式

```
void main() {  
    strtype s1("Hello"), s2 ;  
    s2 = s1 ;  
    s2.setChar(1, 'x') ;  
    s1.show() ;  
}
```

## 3-2 傳遞物件給函數

◆ 回想從前：參數傳遞的方式

```
void main() {  
    int x=5 ; fun1(x); fun2(&x); fun3(x) ;  
}  
  
void fun1(int a) {...} // call by ???  
void fun2(int *p) {...} // call by ???  
void fun3(int &r) {...} // call by ???
```

# 基礎:

## 物件的生成與constructors

```
class samp {  
    int i ;  
public:  
    samp() { i = 0 ; }  
    samp(int n) { i = n ; }  
    samp(const samp& s) { i = s.i ; }  
    show() { cout << i << endl ; }  
};  
void main() {  
    samp ob1, ob2(3), ob3(ob2) ;  
    .....  
}
```



# 範例一：

```
1 class samp {
2     int i ;
3     public:
4     samp(int n) { i = n ;}
5     int get_i() { return i ;}
6 } ;
7 int sqr_it(samp o) { return o.get_i()*o.get_i() ; }
8 void main() {
9     samp a(10);
10    cout << sqr_it(a) << endl ;
11    cout << a.get_i() << endl;
}
```

呼叫 `sqr_it(a)`後

(1) 生成物件 `o`

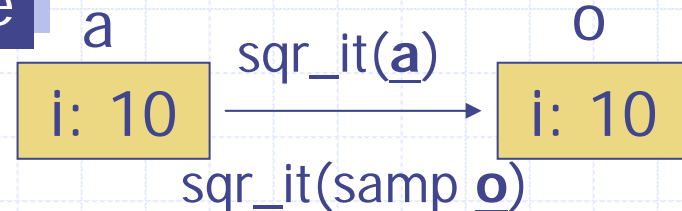
(2) `o`將會以`a`為初值

(3) 同 `samp o(a)` ;

→ 會呼叫那一種constructor?

→ 沒有對應的constructor該如何?

call-by-value



# Constructor vs. 函數呼叫

```
1 class samp {
2     int i ;
3     public:
4     samp(int n) { i = n ;}
5     samp(const samp& s) { cout << "copy\n"; i = s.i ;}
6     int get_i() { return i ;}
7 } ;
8 int sqr_it(samp o) { return o.get_i()*o.get_i() ; }
9 void main() {
10     samp a(10);
11     cout << sqr_it(a) << endl ;
12 }
```

呼叫時, 哪一個建構子會被呼叫?(5)  
Copy constructor

# EX: 哪一個constructor被呼叫?


```
1 class samp {
2     int i ;
3     public:
4     samp(int n) { i = n ;}
5     samp(const samp& s) { cout << "copy\n"; i = s.i ;}
6     int get_i() { return i ;}
7 } ;
8 int sqr_it(samp o) { return o.get_i()*o.get_i() ; }
9 void main() {
10     samp a(10);
11     cout << sqr_it(12) << endl ;
12 }
```

呼叫時, 哪一個建構子會被呼叫?(4)

## 範例二：程式的輸出為何？

```
class samp {
    int i ;
public:
    samp(int n) { i = n ;}    void set(int n) { i = n ;}
    int get_i() { return i; } void print() { cout << i; }
};

void sqrt_it(samp o) {o.set(o.get_i() * o.get_i()) ; }
void main() {
    samp a(10) ; sqrt_it(a) ; a.print() ;
}
```



# 範例三:程式的輸出為何?

```
class samp {  
    int i ;  
public:  
    samp(int n) { i = n ;}    void set(int n) { i = n ;}  
    int get_i() { return i; } void print() { cout << i; }  
};  
void sqrt_it(samp *o) {o->set(o->get_i() * o->get_i()); }  
void main() {  
    samp a(10) ; sqrt_it(&a) ; a.print() ;  
}
```

呼叫時, 哪一個建構子會被呼叫?100

指標之外, 有無其他選擇?

# EX: call-by-reference

呼叫時, 哪一個建構子會被呼叫? 100

```
class samp {  
    int i ;  
public:  
    samp(int n) { i = n ; }    void set(int n) { i = n ; }  
    int get_i() { return i; }  void print() { cout << i; }  
};  
void sqrt_it(samp& o) {o.set(o.get_i() * o.get_i()); }  
void main() {  
    samp a(10) ; sqrt_it(a) ; a.print() ;  
}
```

## 範例四:

# Copy Constructor 的重要性

```
class strtype {
    char *p ; int len ;
public:
    strtype(char *s) { p = new char[strlen(s)+1]; .....}
    ~strtype() { delete[] p ; }
    void setChar(int pos, char c) { p[pos] = c ; }
};
void showMsg(strtype s) { /* do something ...*/ }
void main() {
    strtype s1("NTU welcomes you!") ;
    showMsg(s1) ;
}
```

no copy constructor: bitwise copy

# 結論

- ◆ 如果class中的資料成員牽涉到動態記憶體配置,則以下成員函數不可省:
  - copy constructor
  - operator=() 或 set()
- ◆ 儘量使用call-by-reference, why?
  - void showMsg(const strtype& str) ;
  - void showMsg(strtype str) ;



## 3-3 從函數中傳回物件

### ◆ 回想從前

```
void fun1() { ... }
```

```
int fun2() { ... }
```

```
double fun3() { ... }
```

```
char *fun4() { ... }
```

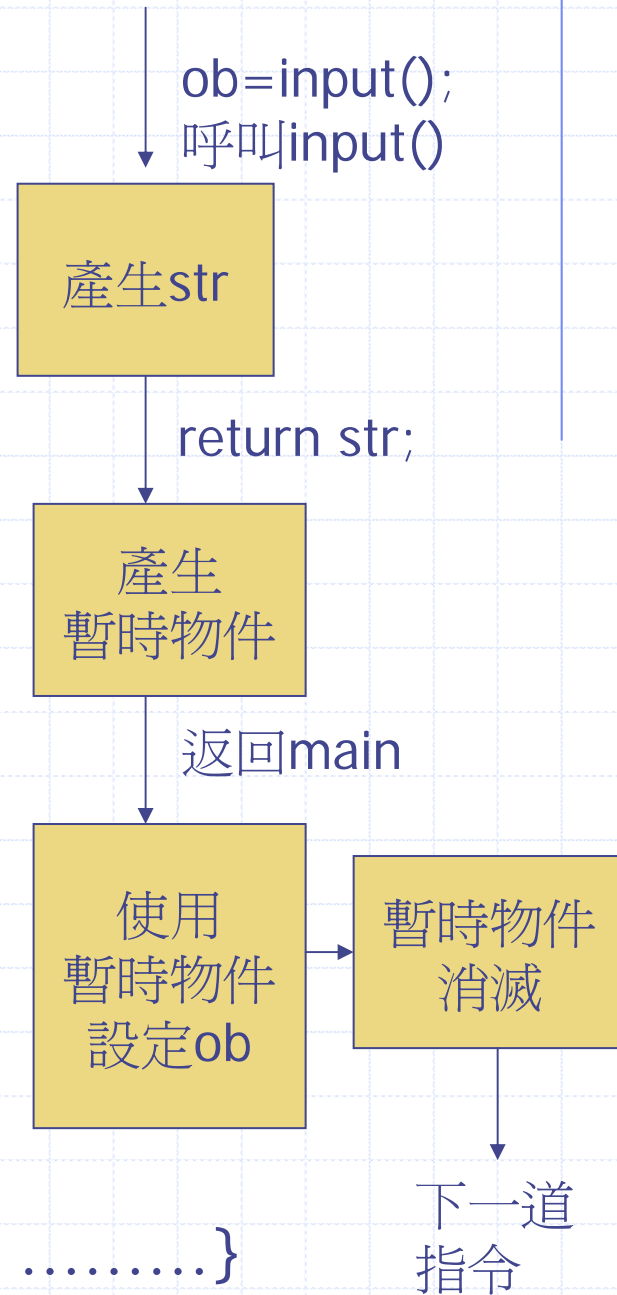
```
myclass fun5() { .....
```

# 範例一

```
class samp {  
    char s[80];  
public:  
    void show() {cout << s << endl ;}  
    void set(char *str) {strcpy(s, str) ;}  
};
```

```
samp input() {  
    char s[80]; cin >> s ;  
    samp str; str.set(s) ;  
    return str ;  
}
```

```
void main() { samp ob; ob = input(); ..... }
```



# 描述

- ◆ 當物件被函數傳回時會自動建立一個暫時物件(temporary object)，內容即為回傳值(return value)。

# 深入剖析物件的回傳

```
class samp {
    char s[80];
public:
    samp() {cout << "default\n" ; }
    samp(const samp& ob) {
        cout << "copy\n"; strcpy(s, ob.s) ; }
    ~samp() {cout << "destroy\n" ; }
    void show() {cout << s << endl ;}
    void set(char *str) {strcpy(s, str) ; }
};

samp input() { char s[80]; cin >> s ;
    samp str; str.set(s) ; return str; (copy constructor)
}

void main() { samp ob; ob = input(); ob.show();}
```

## 範例二：

◆ 拜託你寫copy constructor好嗎？

# 3-4 簡介夥伴函數 (friend functions)

## ◆ 什麼是夥伴函數?

```
class myclass {  
    int n, d ;  
public:  
    myclass(int i, int j) { n = i; d = j ; }  
};  
bool isfactor(myclass ob) { return !(ob.n % ob.d) ;}  
void main() {  
    myclass ob(10, 2) ;  
    if (isfactor(ob)) cout << “ 2 是 10因數” ;  
}
```

# 甚麼是夥伴函數？

```
class myclass {
    int n, d ;
public:
    myclass(int i, int j) { n = i; d = j ; }
    friend bool isfactor(myclass ob) ;
};
bool isfactor(myclass ob) { return !(ob.n % ob.d) ;}
void main() {
    myclass ob(10, 2) ;
    if (isfactor(ob)) cout << “2 是 10因數” ;
}
```

# 夥伴函數的用途

## ◆ 運算子超載(operator overloading)

- operator + ()
- oprrator << (), operator >> ()

## ◆ 讓函數能存取二個多個不同類別的私用成員

- 破壞封裝.....



# NOTE:

```
class myclass {
    int n, d ;
public:
    myclass(int i, int j) { n = i; d = j ; }
    friend bool isfactor(myclass ob) ;
};
bool isfactor(myclass ob) { return !(ob.n % ob.d) ;}
void main() {
    myclass ob(10, 2) ;
    if (isfactor(ob)) cout << “ 2 是 10因數” ;
}
```

# NOTE:

- ◆ 夥伴函數不會被繼承
- ◆ 一個函數可以是多個類別的夥伴函數

# 範例一：多個類別的夥伴

```
class truck ; // why this?
```

```
class car {  
    int speed;  
public:  
    car(int sp){speed=sp;}
```

```
→ friend int sp_greater(car c, truck t) ;  
};
```

```
class truck {  
    int speed;  
public:  
    truck (int sp){speed=sp;}
```

```
→ friend int sp_greater(car c, truck t) ;  
};
```

```
→ int sp_greater(car c, truck t) { return c.speed-t.speed;}
```

# 範例一：討論

◆ 一定要如此嗎？破壞封裝精神

```
main()
{
    car c(60);
    truck t(50);
    cout << sp_greater(c,t) << endl;
}
```

# 範例二：將其他類別的成員函數做為夥伴函數

```
class car;
class truck {
    int speed;
public:
    truck (int sp){speed=sp;}
    friend class car;
};

class car {
    int speed;
public:
    car(int sp){speed=sp;}
    int sp_greater(truck t){
        return speed-t.speed; }
};
```

```
main()
{
    car c(60);
    truck t(50);
    cout << c.sp_greater(t)
    << endl;
}
```

# 範例三: friend 的有用之處

```
#include<iostream.h>
class frac {
    int q, p ;
public:
    frac(int n, int m){q=n;p=m;}
    friend ostream& operator<<(ostream& out,const frac& ob);
};
ostream& operator<<(ostream& out, const frac& ob){
    out<<ob.q<<"/"<<ob.p;
    return out;
}
void main() {
    frac f(3,5);
    cout<<f<<endl ; // f.print();
}
```

# NOTE:

```
#include<iostream.h>
// using namespace std;
class coord{
int x,y;
public:
    coord(){x=0;y=0;}
    coord(int a,int b){x=a;y=b;}
friend ostream &operator<<(ostream&stream,coord&
    ob);
friend istream &operator>>(istream &stream,coord
    &ob);
};
```

# NOTE:

```
ostream &operator<<(ostream &stream, coord& ob)
{ stream<<ob.x<<" , "<<ob.y<<'\n';
  return stream;}

```

```
istream &operator>>(istream &stream, coord &ob)
{ cout<<"Enter coordinates\n";
  stream>>ob.x>>ob.y;
  return stream;}

```

```
int main()
{
  coord a(1,1),b(2,3);
  cout<<a<<b;
  cin>>a;
  cout<<a;
  return 0;
}

```



# 課後練習 & 綜合學習

