第六章 簡介運算子超載 (Operator Overloading)

6-1 運算子超載的基礎

6-2 超載二元運算子

6-3 超載邏輯與關係運算子

6-4 超載一元運算子

6-5 使用夥伴函數

6-6 細部檢視指定運算子

6-7 超載註標運算子

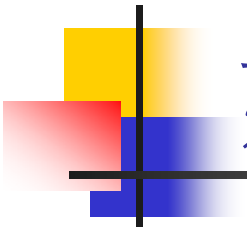
6-1 運算子超載的基礎

- 甚麼是運算子超載?
  - 讓運算子(符號)有不同的意義

EX: 運算子的預設意義(以 + 與 = 爲例)

```
class frac {……} ;
void main() {
    int x=5, y =3, z ;  z = x + y ; // 使用'=' '+'
    int a[10], b[10], c[10]; c = a + b ; // 可乎???
    frac f1(3,5), f2(2, 5), f3 ;
    f3 = f1 + f2 ; // 3/5 + 2/5 = 1, 可乎?
}
```

可以重新定義+, = 運算子應用在frac物件上的意義嗎?



如果沒有運算子超載能力

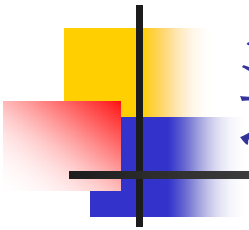
```
void main() {
    frac f1(3,5), f2(2, 5) , f3 ;
    f3.set(f1.add(f2)) ; // 模擬 f3 = f1 + f2 ;
    if (f1.great_equal(f2)) // f1 >= f2
          cout << " f1 >= f2" ;
    cout<<"f3="; f3.print() ; // cout << "f3="<<f3 ;
}
```

→ 也OK? 缺點? 你的看法

如何超載運算子

- 在類別中建立運算子函數(operator functions)
  - 語法

```
class classname {
   ……
   // overload 運算子 X
   return-type operatorX(arg-list) { ……}
}
```



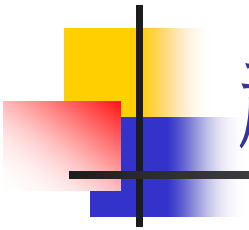
運算子的種類

- 二元運算子(Binary Operators)
  - EX: 5 + 3, 5>=3 …
  - 算術運算: +, -, *, /, =, +=, -=,….,
  - 關係運算: >, <, >=, <=…
  - 邏輯運算: &&, ||, &, |, ^
- 一元運算子
  - a++, --b, -c, +d
- 其他
  - [], (), new, delete, ->, ?:

6-2 超載二元運算子

- 範例一

```
class coord {
    int x, y ;
    ……
} ;
void main() {
    coord o1(10,10), o2(5,3), o3 ;
    o3 = o1 + o2;  // o3 ← (15, 13)
    o3.print() ; //later, you can use cout << o3 ;
}
```



超載運算子+，=

```
class coord {
    int x, y ;
    ……
} ;
void main() {
    coord o1(10,10), o2(5,3), o3 ;
    o3.set(o1.add(o2));  // o3 = o1 + o2 ;
    o3.print() ; //later, you can use cout << o3 ;
}
```

超載operator+, =

```
class coord {
    int x, int y;
public:
    coord(){}
    coord(int a, int b){x=a; y=b;}
coord add(coord c) {
            coord temp ;
            temp.x = this->x + c.x ;
            temp.y = this->y + c.y ;
            return temp ;
    }
    void set(coord c) {
            this->x = c.x; this->y=c.y;
    }
void print(){cout<<x<<y<<endl;}
} ;
```

o3.set(o1.add(o2)) ;

temp <= o1.add(o2) ;

o3 <= temp ;

超載op

```
void main() {
        coord o1(10,10), o2(5,3), o3 ;
        // 原 o3.set(o1.add(o2));
        o3.operator=(o1.operator+(o2));
}
```

```
class coord {
    ……
    coord operator+(coord c) { //原add()
        coord temp ;
        temp.x = this->x + c.x ;
        temp.y = this->y + c.y ;
        return temp ;
    }
    void operator=(coord c) { // 原 set()
        this->x = c.x; this->y=c.y;
    }
} ;
```

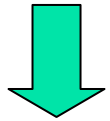
超載

```
void main() {
        coord o1(10,10), o2(5,3), o3 ;
        // 原 o3.set(o1.add(o2));
        // o3.operator=(o1.operator+(o2);
        o3 = o1 + o2 ;
}
```

```
class coor
    ……
    coord operator+(coord c) { //原add()
        coord temp ;
        temp.x = this->x + c.x ;
        temp.y = this->y + c.y ;
        return temp ;
    }
    void operator=(coord c) { // 原 set()
        this->x = c.x; this->y=c.y;
    }
} ;
```

牢記運算式的真面目

```
o3 = o1 + o2 ;
```



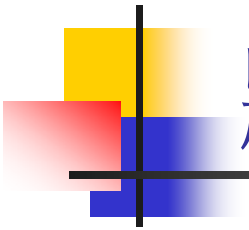
```
o3 = o1.operator+(o2) ;
```

```
o3.operator=(o1.operator+(o2)) ;
```

回憶如何超載運算子

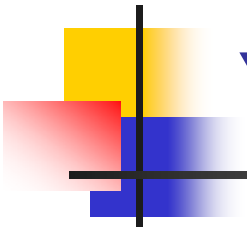
- 在類別中建立運算子函數(operator functions)
  - 語法

```
class classname {
   ……
   // overload 運算子 X
   return-type operatorX(arg-list) { ……}
}
```



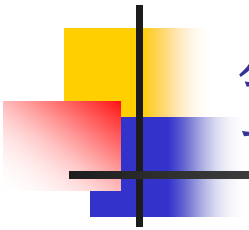
將超載運算子定義在class外

```
class coord {
    ……
    coord operator+(coord c) ;
    void operator=(coord c) ;
} ;
coord coord::operator+(coord c) {……}
void coord::operator=(coord c) {……}
```



Yet Another Operator+

```
class coord {
   …
   coord operator+(const coord& ob2) {
       return coord(x+ob2.x, y+ob2.y) ;
   }
   …
};
```



範例二:

- 承範例一
  - 新增 減號‘-’
  - 改善 operator=的功能(later)

範例三:

- 需求

```
void main() {
    coord o1(10,10), o2(5, 3), o3 ;
    o3 = o1 + 2 ;
}
```

Q: o3=o1 +2 的真面目?

範例三(續)

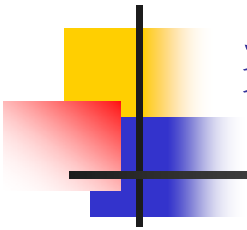
```
class coord {
    int x, y ;
public:
    coord() { x=y=0 ;}
    coord(int a) { x = a; y = 0 ;}
    coord(int a, int b) {x=a; y=b;}
    coord operator+(coord c) { ……}
    ……
}
```

```
void main() {
        coord o1(10,10), o2(3,5), o3 ;
        o3 = o1 + 2 ;
        // o3 = o1.operator+(2) ;
}
```

範例四: 請使用call by reference

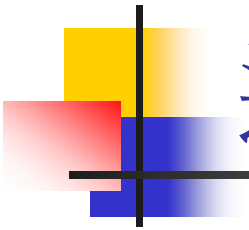
優點?

```
class coord {
    ……
    coord operator+(const coord& c) ;
    void operator=(const coord& c) ;
} ;
coord coord::operator+(const coord& c) {…}
void coord::operator=(const coord& c) {…}
```



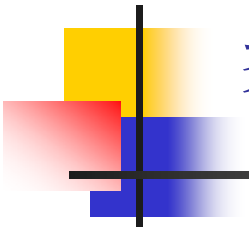
習題

```
class frac {
   int u, d ;
public:
   // 定義+, -, *, /, % = 運算子
} ;
```



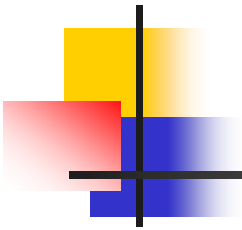
連加與連等

- 連加, OK?
  - o4 = o1+o2+o3 ;
- 連等, OK? 參考範例二
  - o1=o2=o3;



習題

- 爲frac加入 +=, -=, *=, /=



6-3 超載邏輯與關係運算子

- 二元運算子(Binary Operators)
  - 算術運算: +, -, *, /, =, +=, -=,...,
  - 關係運算: >, <, >=, <=, ==...
  - 邏輯運算: &&, ||, &, |, ^

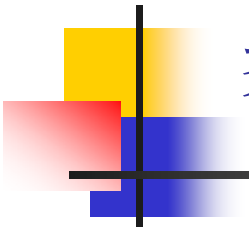
範例一:

Q: o1==o2的真面目?

```
class coord {
    int x, y ;
public:
    coord() {x = 0 ; y=0 ; }
    coord(int i, int j) { x=i; y=j;}
    bool operator==(const coord& ob2) ;
    bool operator&&(const coord& ob2) ;
};
void main() { coord o1(10,10); o2(5,3) ;
    if (o1 == o2) cout << “o1 == o2” <<endl ;
    if (o1 && o2) cout << “as you wish….” <<endl ;
}
```

範例一(續)

```
bool coord::operator==(const coord& ob2) {
   return (this->x==ob2.x) && (this->y==ob2.y);
}
```



習題:

- 替frac加上==, >, <, >=, <= 運算子

6-4 超載一元運算子

- 一元運算子
  - ++, --, +, -
- 需求:

o1.operator++()

```
void main() {
   coord o1(10, 10), o2 ;
   o2 = ++o1 ;  o1.print() ; // 前置++, prefix
   o2 = o1++ ; o1.print() ;//後置++, postfix
}
```

範例一: 前置++

```
class coord {
    int x, y ;
public:
    ……
    coord operator++() {
        x++; y++ ; return *this;
    }
}
```

Q1:爲何不寫 void operator++()?

範例二: 如何分辨prefix與postfix ++

```
class coord {
int x, y;
public:
coord(){}
    coord(int a, int b){x=a; y=b;}
    coord& operator++() { // prefix ++
          x++; y++ ; return *this;
    }
    coord operator++(int) {//postfix ++
          coord temp = *this ;
          x++; y++ ;
          return temp ;
    }
}
```

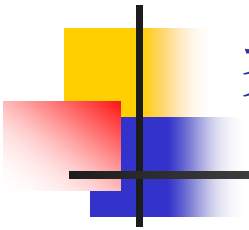
```
++o1 ;
```

```
o1++ ;
// 解釋函數內容
```

範例三: 超載負號

這樣的operator-對嗎?
try o1 = -o2 ; // o2(10,10)

```
class coord {
    coord& operator-() {
        x = -x; y =-y; return *this;
    }
    coord operator-(const coord& c) {
        ……
    }
} ;
```



習題:

- class frac
  - 加入++, --, -

6-5 使用夥伴運算子函數

- 源起
  - o3 = o1+o2 ; → o3 = o1.operator+(o2) ;
  - o3 = o1+5 ; → o3 = o1.operator+(5) ;
  - o3 = **5** + o1 ; → o3 = 5.operator+(o1);

?????

使用夥伴函數來定義運算子!

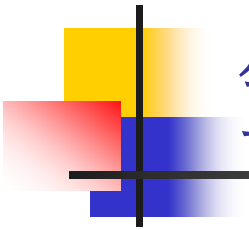
範例一：記起來

參考 operator+(coord, coord)
Q1:是否爲class coord的成員?
Q2:可否取用x, y ?

```
#include<iostream.h>
class coord {
    friend coord operator+(coord ob1, coord ob2) ;
    ……
} ;
coord operator+(coord ob1, coord ob2) {
    return coord(ob1.x+ob2.x, ob1.y+ob2.y) ;
}
void main() {
    coord o1(10,10), o2(5, 3), o3;
    o3 = o1 + o2 ;              // o3 = operator+(o1, o2) ;
}
```

範例二: 10+ob2的解決

```
class coord{
    int x, y ;
public:
    coord(){x=0;y=0;}
    coord(int a){x=a;y=a;}
    coord(int a, int b){x=a;y=b;}
    friend coord operator+(coord ob1, coord ob2);
    void print(){cout<<x<<y<<endl;}
} ;coord operator+(coord ob1, coord ob2) {
    return coord(ob1.x+ob2.x, ob1.y+ob2.y) ;
}
void main() {    coord o1(10,10), o2(5, 3), o3;
    o3 = 10 + o2 ;  // It's OK, why ?}
```



範例三: 自己看

- 一元運算符號的負載
  - 使用friend functions

很多個class的operator+

```
class coord { ……};
class frac {……} ;
coord operator+(coord ob1, coord ob2) ;
frac operator+(frac ob1, frac ob2) ;
void main() {
   coord o1(10,10), o2(5,3), o3;
   o3 = o1 + o2 ;  // 呼叫哪一個operator+
   frac f1(5,3), f2(2,7), f3;
   f3 = f1 + f2 ;
}
```

定義二元運算子函數的常態

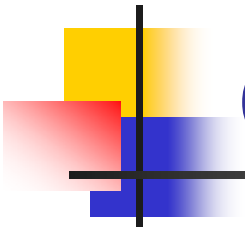
Q: operator=需要用此種方式嗎?

```
class frac {
    // 將以下之外部函數宣告成朋友函數
};
frac operaotr+(const frac& f1, const frac& f2) {…}
frac operaotr-(const frac& f1, const frac& f2) {…}
frac operaotr*(const frac& f1, const frac& f2) {…}
frac operaotr/(const frac& f1, const frac& f2) {…}
bool operaotr==(const frac& f1, const frac& f2) {…}
bool operaotr>(const frac& f1, const frac& f2) {…}
……
```

6-6 細部檢查指定運算子

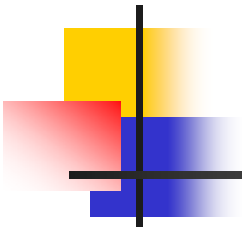
- operatr=要不要有回傳值?
  - void operaotr=(const coord& ob);
  - o1 = o2 = o3 ; // o1 = (o2=o3) ;
  - o1.operator=(o2.operator=(o3)) ;

那要回傳什麼?



operator=的回傳值

```
class coord {
   …
   coord operator=(const coord& ob) {
       x = ob.x ; y = ob.y ;
       return ob ;
   }
}
```

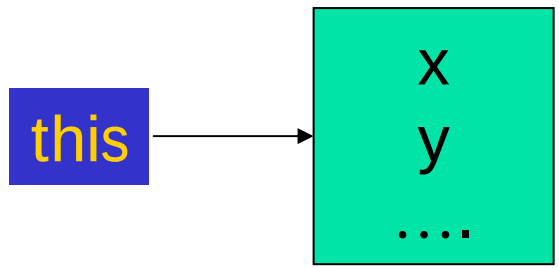


operator=的回傳值

```
class coord {
   …
   coord& operator=(const coord& ob) {
       x = ob.x ; y = ob.y ;
       return ob ;
   }
}
```

operator=的回傳值

```
class coord {
   …
   coord& operator=(const coord& ob) {
       x = ob.x ; y = ob.y ;
       return *this ;
   }
}
```



operator=的內容

```
class coord {
   coord& operator=(const coord& ob) {
        x = ob.x ; y = ob.y ;
        return *this;
   }
} ;
```

Q1: 不寫operator=會不會怎樣? 如 o1 = o2 ;

bit-wise copy

Q2: 那寫這個幹嘛?

當成員中有pointer時

範例一

與copy constructor幾乎相同!

```
class strtype {
    char *p; int len ;
    ……
} ;
strtype& strtype::operator=(const strtype&ob){
    if (&ob == this) return ; // what means?
    if (len < ob.len) { delete[] p; p = new
    char[ob.len+1] ;}
    len = ob.len ;  strcpy(p, ob.p) ;
    return *this ;
}
```

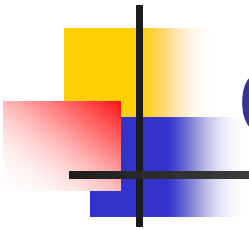
6-7 超載註標運算子

- 需求

```
class SafeArray {
    int a[50] ;
    ……
} ;
void main() {
    SafeArray s ;
    cout << s.get(10) ; // cout << s.a[10] ;
    s.insert(10, 77) ; // s.a[10] = 77 ;
    …
}
```

但我可否使用:

```
cout << s[10];
s[10] = 77 ;
```



operator[]

```
SafeArray s ;
cout << s[10] ;
// cout << s.operator[](10) ;
s[10] = 77 ;
// s.operator[](10) = 77 ;
```

奇怪, 怎麼會有寫在等號右邊的函數?

可寫在等號右邊的函數

```
int buffer ;
int& get_buffer() ;
void main() {
   get_buffer() = 10 ;
   cout << get_buffer() <<endl ;
}
int& get_buffer() { return buffer ;}
```

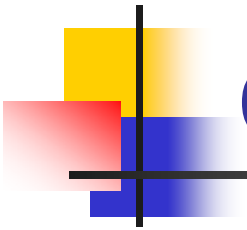
可寫在等號右邊的函數

```
const int SIZE = 20 ;
int a[SIZE] ;
int& getElement(int index) { return a[index] ; }
void main() {
   getElement(5) = 100 ; // a[5] = 100 ;
   cout << getElement(5) <<endl ;
}
```

安全的陣列

```
class SafeArray {
      int a[20] ;
public:
      int& operator[](int index) {
          if(index<0 || index >19)
                {cout<<“out of boundary”<<end
}
          else  {return a[index];}
       // DIY, 檢查index的範圍
      }
}
void main() {
      SafeArray sa;
      sa[5] = 10 ;  cout << sa[5] ;
}
```

```
EX:
void main() {
          SafeArray sa(20) ;
          ……
}
```



class Frac完成了沒?