第十一章 (上篇)
函數樣板(Function Template) 與類別樣板(Class Template)

- 建立通用函數(Generic Functions) & 通用類別(Generic Classes)
  - Code Reuse 的另一種發揮
  - 煩人的事　經歷一次就夠了

爲何需要通用函數?

```
int abs(int x) { return (x>0)?x:-x; }
```

```
int fabs(float x) { return (x>0)?x:-x; }
```

```
int cabs(complex x) { return (x>0)?x:-x; }
```

取名困難不好記

爲何需要通用函數?

```
int abs(int x) { return (x>0)?x:-x; }
```

[overloading]
同樣的東西
爲何要寫三次?

```
int abs(float x) { return (x>0)?x:-x; }
```

```
int abs(complex x) { return (x>0)?x:-x; }
```

利用函數樣板來實現通用函數的理想

當有一組函數:
(1) 內容一樣
(2) 參數資料型態不同

```
int abs(int x) { return (x>0)?x:-x; }
int abs(float x) { return (x>0)?x:-x; }
int abs(complex x) { return (x>0)?x:-x; }
```

↓

把資料型態當參數傳過去

↓

建立函數模板

函數樣板的定義方式

```
T abs( T x) {
        return (x>0)?x:-x;
}
```

保留字

函數模板

```
template <class T>
T abs( T x) {
        return (x>0)?x:-x;
}
```

函數樣板的使用

```
template <class T>
T abs( T  x) {
        return (x>0)?x:-x;
}
void main() {
    int a = 3; float b=-2.83;  complex c(-5, -2) ;
    cout << abs(a) << endl ;
    cout << abs(b) << endl ;
    cout << abs(c)  << endl ;
}
```

編譯器到底做了甚麼?

```
template <class T>
T abs( T  x) {
        return (x>0)?x:-x;
}
void main() {
    int a = 3; float b=-2.83;  complex c(-5, -2) ;
    cout << abs(a) << endl ;
    cout << abs(b) << endl ;
    cout << abs(c)  << endl ;
}
```

```
int abs(int x) { return (x>0)?x:-x; }
```

自動產生

```
float abs(float x) { return (x>0)?x:-x; }
```

自動產生

```
complex abs(complex x) { return (x>0)?x:-x; }
```

另一種函數樣板的使用

```
template <class T>
T abs( T  x) {
        return (x>0)?x:-x;
}
void main() {
    // int a = 3; float b=-2.83;  complex c(-5, -2) ;
    cout << abs(3) << endl ;
    cout << abs(2.83) << endl ; // T=??
    cout << abs(complex(-5, -2))  << endl ;
}
```

EX: 通用的swap()

```
void swap(int& x, int& y) { int temp=x; x=y; y=temp;}
void swap(double& x, double& y) { ... }
void swap(frac& x, frac& y) { ... }
void main( ) {
    int a = 5, b =3 ;
    double d1=3.4, d2=5.6 ;
    frac f1(5, 3), f2(6, 7) ;
    swap(a, b);  swap(d1, d2); swap(f1, f2) ;
}
```

通用的swap()

```
template <class T>
void swap1(T& x, T& y) { T temp=x; x=y; y=temp;}
void main( ) {
   int a = 5, b =3 ;
   double d1=3.4, d2=5.6 ;
   swap1(a,b);
   cout<<a<<b<<endl;
   swap1(d1, d2);
   cout<<d1<<d2<<endl;
   }
```

EX: 通用的print_arr()

```
void main( ) {
    int x[] = {1,2,3} ; float y[] = {1.1,2.2,3.3};
    complex z[3] ={{1,1,},{2,2},{3,3}};
    print_arr(x, 3) ; // 印出 1  2  3
    print_arr(y, 3);  // 印出 1.1  2.2  3.3
    print_arr(z, 3) ; // 印出 1+1i  2+2i  3+3i
}
```

Q: 編譯器到底做了甚麼?

EX: 通用的ROR()

void ROR(int a[], int size) {…}

EX: 通用的max(a,b)

[寫法一]

```
template <class T>
T max(T a, T b) { return (a>b)?a:b; }
void main() {
    cout << max(5, 3) << endl;
    cout << max(-3.14, 5.2) <<endl;
    cout << max(2.3, 5) <<endl ; //可乎?
}
```

EX: max(a,b)

[寫法二]

```
template <class T1, class T2>
T1 max(T1 a, T2 b) { return (a>b)?a:b; }
void main() {
    cout << max(5, 3) << endl;
    cout << max(-3.14, 5.2) <<endl;
    cout << max(5, 7.8) <<endl ; //印出啥?
}
```

觀察

- 使用Function Template只是少打字而已，可執行檔的大小並未減低。
- 爲甚麼不使用Macro就好了?

[Note1]: template <class T> 的scope

```
template <class T>
T abs(T x) {
        ....
}
```

樣板一

```
template <class T>
T max(T a, T b) {
        .....
}
```

樣板二

[Note2]: 換不換行沒關係

```
template <class T>
T abs(T x) {
        ....
}
```

```
template <class T> T abs(T x) {
        ....
}
```

[Note3]:樣板參數的宣告的變化

```
// type name 隨你取
template <class Atype>
Atype abs(Atype x) {
        ....
}
```

```
// class 可用typename取代
template <typename Atype>
Atype abs(Atype x) {
        ....
}
```

[Note4]: template <….>與函數定義間不可有任何指令

```
template <class Atype>
const int x=18 ; // error
Atype abs(Atype x) {
        ....
}
```

[Note5]: 函數樣板與樣板函數

```
template <class T>
int abs( T  x) {
        return (x>0)?x:-x;
}
```

函數樣板
(Function Template)

```
int abs(int x) {
   return (x>0)?x:-x;
}
```

```
float abs(float x) {
   return (x>0)?x:-x;
}
```

```
frac abs(frac x) {
   return (x>0)?x:-x;
}
```

樣板函數 (Template Function)
產生函數(Generated Function)

函數呼叫規則 (Rules of Function Invocation) (一)

```
template <class T>
T add(T x, T y) { cout << “F1”; return x+y ; }
int add(int x, int y) { cout << “F2 “; return x+y ;}
void main() {
        cout << add(3,8) << endl ;
}
```

Result:
Rules:

函數呼叫規則(二)

```
template <class T1, class T2>
T1 add(T1 x, T2 y) { cout << "F1"; return x+y ; }
int add(int x, int y) { cout << "F2 "; return x+y ;}
void main() {
        cout <<  add(3.5, 8) << endl ;
}
```

Result:
Rules:

函數呼叫規則(三)

```
template <class T>
T add(T x, T y) { cout << “F1”; return x+y ; }
template <class T1, class T2>
T1 add(T1 x, T2 y) { cout << “F2 “; return x+y ;}
void main() {
        cout << add(3,8) << endl ;
}
```

Result:
Rules:

[作業] 通用的find與sort

```
int find(int a[], int size, int x) {
    ......
}
void sort( int a[], int size) {
}
void main() {
    int a[50]; complex c[50]; frac f[50] ;
    // 應用sort()與find()在int[], complex[]與frac[]
}
```

通用的find

```
template <class T>
int find(T a[], int
  size, T x) {
   int i=0;
   int result;
   for(i=0;i<size;i++)
   {
     if(x==a[i])
     {
         result=i;
         break;      }
   }
   return result;}
```

```
void main()
 {  int a[5]={1,2,3,4,5};
    float b[5]={1.1,2.2,3.3,4.4,5.5};
    cout<<find(a,5,3)<<endl;

           cout<<find(b,5,(float)4.4)<
<endl;
 }
```

T 的責任

```
template <class T>
T add(T x, T y) {
        T z = x+y ;
        return z ;
}
void main() {
        complex c1, c2 ;
        .....
        cout << add(c1, c2) ;
}
```

→

```
class complex {
        copy constructor
        operator=
        operator+
        operator<<
} ;
```

```
class list {
        copy constructor
        operator=
        operator+
        operator<<
} ;
```

爲何需要通用類別 (Generic Class)

// 你厭倦了爲不同的type寫class嗎?

class char_stack{ <u>char</u> data[10] ;….} ;

class int_stack {<u>int</u> data[10]; …} ;

class complex_stack{<u>complex</u> data[10]; ….};

……

→ <u>我需要通用的stack類別</u>

類別樣板(Class Template)

```
template <class T>
class stack {
   private:
            T data[10] ;
            int top, size ;
   public:
            stack():top(-1),size(10) {}
            stack(const stack& s) {
            for (int i=0 ; i<10; i++) data[i] = s.data[i] ;
            top = s.top ;
}
            T pop( ) { return data[top--] ; }
            void push(T x) { data[++top] = x ; }
        void print() {          for(int i=0;i<=top;i++)
                        {cout<<data[i]<<endl;}
            }
} ;
```

類別樣板的使用

```
void main( ) {
        stack<int> s1;
        s1.push(5) ; ........
        stack<float> s2 ;
        s2.push(3.14) ;
        statck<complex> s3 ;
        .....
        stack<int> s2(s1) ;
}
```

Q1: s1, s2 and s3 的資料型態為何?

Q2: 編譯時會發生甚麼事?

定義在類別樣板外的成員函數

回想

```
class stack {
   private:
          int data[10] ;
          int top, size ;
   public:
          stack():top(-1),size(10) {}
          int pop( ) { return data[top--] ; }
          void push( int x) ;
} ;
void stack::push( int x) { data[++top] = x ; }
```

定義在類別樣板外的成員函數

```
template <class T>
class stack {
        .......
        void push(T x) ; //如何定義push()
} ;
```

```
void stack::push(T x) { data[++top] = x; }
```

```
void stack<T>::push(T x) {data[++top] = x ; }
```

正確版本

```
template <class T>
void stack<T>::push(T x) {data[++top] = x ; }
```

EX: 重新定義 stack template

```
template <class T>
class stack {
   private:
         T data[10] ;
         int top, size ;
   public:
         stack();
         T pop( );
         void push(T x);
} ;
```

測試stack template

```
void main() {
    stack<int> s1 ;
    for (int i = 0 ; i<5; i++)
     s1.push(i) ;
      s1.print() ;
stack<char> s2 ;
    for (i = 0 ; i<5; i++)
    s2.push('A'+i) ;
    s2.print() ;
}
```

More on Class Template

實際測試一次

```
template <class T>
class stack {
   private:
          T data[10] ;     int top, size ;
   public:
          stack();
          stack(const stack& s) {...}
          ......
} ;
```

```
template <class T>
stack<T>::stack(const stack<T>& s) {
          for (int i=0 ; i<10; i++) data[i] = s.data[i] ;
          top = s.top ;
}
```

EX: 通用二維座標

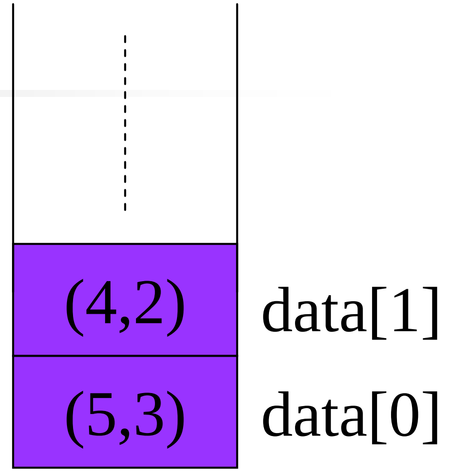
```
// 改寫爲class template 使main中的程式碼可以運作
template <class T>
class point  {
    T x, y ;
public:
    point(T a, T b) { x = a; y=b;}
    void print() { cout << x << “ ”<< y ; }
} ;
void main() {
    point<double> p1(3.5, 6.3) ;
    point<int> p2(3, 9) ;
     p1.print();
     p2.print();


}
```

EX: 再度測試stack template

```
void main() {
   stack<int> s1 ;
   for (int i = 0 ; i<5; i++) s1.push(i) ;
   s1.print() ;
   stack<complex> s2 ;
  for (int i = 0 ; i<5; i++) s2.push(complex(i, i)) ;
   s2.print() ;
   stack<stack<int> > ss ;  // 注意 > >之間要有空格
   ss.push(s1) ; ss.push(s1) ;  ss.print() ;
}
```

不過載operator<<

```
template <class T>
void stack {
    T data[10]; int top, size;
    ...
    void print() { for (int i = 0 ; i<=top; i++) {
            cout << data[i] << " " ; // 根本不work!
    }
}
void main() { stack<complex> s; ..... s.print() ; }
```



過載 operator<<()

```
class complex {
    double a, b ;
public:
    ......
    void print() {cout << a << “+” << b<<“i”; }
} ;
void main() {
    complex c(5,3);  c.print() ;
    cout << c ;           // 可以這樣嗎?
                          // 轉成 operator<<(cout, c);
}
```

過載 operator<<()

```
class complex {
    double a, b ;
public:
    ......
    friend ostream& operator<<(ostream& out, const complex& c) ;
} ;
ostream& operator<<(ostream& out, const complex& c) {
    out << c.a << "+" << c.b<<"i"; //做與print()相同的事
    return out ;
}
void main() {complex c(5,3);   cout << c <<
```

operator<<(cout, c)

結論

- 每個class都應該寫operator<<
  - 只要將原先的print()或show()改寫即可
- 自我練習
  - complex, list, stack, frac

EX: 測試stack template

```
void main() {
    stack<int> s1 ;
    for (int i = 0 ; i<5; i++) s1.push(i) ;
    cout << s1 << endl ;
    stack<complex> s2 ;
    for (int i = 0 ; i<5; i++) s2.push(complex(i, i)) ;
    cout << s2 << endl ;
    stack<stack<int>> ss ;
    ss.push(s1) ; ss.push(s1) ; cout << ss << endl ;
}
```

Note that

1. Template
2. Sort
3. Operator<<

Template的參數

```
template <class T, int n>
class stack {
        T data[n] ;
        ……
} ;
void main( ) {
        stack<int,50> s1 ;
        stack<int,30> s2 ;
        stack<float, 40> s3 ;
        stack<float, 70> s4
}
```

What's the data type of s1, s2, s3 and s4?

缺點:

練習

- 課本 11-11 ~ 11-16

作業(or 自我練習)

complex> input a

5 3

complex> input b

-1 2

complex> eval (a+b)*(a-b)/(2.5*a)

??????

自我挑戰: 完成以下SortedList

```
void main() {
    SortedList<int> L1;
    L1.insert(10);  L1.insert(25) ;   L1.insert(13); L1.insert( 20) ;
    cout << L1 << endl ; // 10     13   20   25
    SortedList<Frac> L2;
    L2.insert(Frac(3,5));  L2.insert( Frac(2,5)) ;
    L2.insert(Frac(1,13)); L2.insert(Frac(4,20)) ;
    cout << L2 << endl ; // 1/13  1/5   2/5   3/5
    SortedList<SortedList<int>> LL ;
    LL.insert(L1);  LL.insert(L1);  cout << LL <<endl ;
}
```