



04 Overloading (多載)

Fundamental Computer Programming- C++ Lab(II)

元智大學 | C++ 程式設計實習 (二)

張維元

課程投影片：[請從元智個人 Portal 下載](#)

Outline

- 01 物件導向與開發環境
- 02 陣列、向量與結構
- 03 類別與物件
- 04 物件導向程式設計: 多載
- 05 物件導向程式設計: 繼承
- 06 樣板
- 07 C++ 進階用法

什麼是程式？

程式語言是用來命令電腦執行各種作業的工具，是人與電腦溝通的橋樑。當電腦藉由輸入設備把程式讀入後，會儲存在主記憶體內，然後指令會依序被控制單元提取並解碼或翻譯成電腦可以執行的信號，並把信號送到各個裝置上，以執行指令所指派的動作。也就是說，人類與電腦溝通的語言稱為程式語言。

程式 = 利用一系列的指令告訴電腦如何執行工作



物件導向程式設計

物件導向程式設計（Object-oriented programming，OOP）是種具有物件概念的程式設計典範，同時也是一種程式開發的抽象方針。它可能包含資料、屬性、程式碼與方法。物件則指的是類別（class）的實例。它將物件作為程式的基本單元，將程式和資料封裝其中，以提高軟體的重用性、靈活性和擴充性。

把物件作為程式最小的單位，模擬實體世界的運作



作業 #05

■#練習：請定義一個 Book 的類別，找出最貴的物件與書名。

■Requirements：

1. 定義一個 Book 類別包含私有變數：price, title, author 與公有變數：f(...)
2. 宣告物件時 title 為必填，若未指定 price 給予 100 – 999 的亂數
3. 利用 **static** 與 **private member function** 判斷物件中價錢最高的價錢與書名
4. 最後利用 f(...) 印出物件中最貴價錢與書名

■Sample Input：參考下頁

■Sample Output：參考下頁

參考程式碼與結果

```
作業 #06.cpp ×
1  #include <iostream>
2  using namespace std;
3
4  class Book {
5      string title, author;
6      int price;
7      public:
8          void f(){
9              cout << "Max Price Book is ";
10         }
11     };
12
13     int main( ){
14
15         Book b1("AAA");
16         Book *b2 = new Book("BBB", 123);
17         Book *b3 = new Book("CCC", 369, "Bob");
18
19         b3 -> f();
20         return 0;
21     }
```

```
Console
Shell
❯ clang++-7 -pthread -std=c++17 -o main example01.cpp example02.cpp example03.cpp example04.cpp example05-1.cpp example05.cpp example06.cpp example07.cpp example08.cpp example09.cpp example10.cpp example11.cpp main.cpp 作業 #05.cpp 作業 #06 - 解答.cpp 作業 #06.cpp
❯ ./main
Max Price Book is AAA ( 450 )
❯
```

→ 三種不同的建構子

```
Book(string t){
    title = t;
    price = 100 + rand() % 900;
};

Book(string t, int p){
    title = t;
    price = p;
};

Book(string t, int p, string a){
    title = t;
    price = p;
    author = a;
};
```

參考程式碼與結果

```
作業 #06.cpp ×
1  #include <iostream>
2  using namespace std;
3
4  class Book {
5      string title, author;
6      int price;
7      public:
8          void f(){
9              cout << "Max Price Book is ";
10         }
11     };
12
13     int main( ){
14
15         Book b1("AAA");
16         Book *b2 = new Book("BBB", 123);
17         Book *b3 = new Book("CCC", 369, "Bob");
18
19         b3 -> f();
20         return 0;
21     }
```

```
Console Shell
❯ clang++-7 -pthread -std=c++17 -o main example01.cpp example02.cpp example03.cpp example04.cpp example05-1.cpp example05.cpp example06.cpp example07.cpp example08.cpp example09.cpp example10.cpp example11.cpp main.cpp 作業 #05.cpp 作業 #06 - 解答.cpp 作業 #06.cpp
❯ ./main
Max Price Book is AAA ( 450 )
❯
```

→ 找出最大值

```
class Book {
    static string maxTitle;
    static int maxPrice;
    string title, author;
    int price;
    void compare(){
        if(price > maxPrice){
            maxPrice = price;
            maxTitle = title;
        }
    };
};

string Book::maxTitle = "";
int Book::maxPrice = 0;
```

類別與物件

- 類別 (Class) 利用屬性與方法定義了物件的抽象樣板/結構
- 物件 (Object) 是類別的實體 (instance) ，可以使用類別方法

```
1  
2  
3 class Book {  
4     public:  
5         string title;  
6         int price;  
7 };  
8  
9
```

```
1 int main( ){  
2  
3     Book book1;  
4  
5     book1.title = "C++ How to Program";  
6     book1.price = 636;  
7  
8     cout << book1.title << endl;  
9     cout << book1.price << endl;  
10    return 0;  
11  
12 }
```

→ 定義了一個叫做「Book」的變數型態

成員函式

(member function)

```
1
2 class Book {
3     public:
4         string title;
5         int price; ← 成員變數
6         void f(void); ← 成員函式
7 };
8
9 void Book::f(){
10     cout << "I'm a book: " + title;
11 };
12     → 成員函式可存取成員變數
```

```
1
2 int main( ){
3
4     Book book1;
5     book1.title = "C++ How to Program";
6     book1.f();
7
8     return 0;
9 }
10
11
12
```

利用公開成員函式存取私有成員變數

```
1
2 class Book {
3     private: ← 私有成員
4         int price;
5     public: ← 公開函式
6         void set(int p);
7         double get(void);
8 };
9
10 void Book::set(int p){
11     price = p;
12 }; → 成員函式可存取成員變數
13
14 double Book::get(void){
15     return price * 0.9;
16 };
```

```
1 int main( ){
2
3     Book book1;
4     book1.set(600);
5     cout << "price: " << book1.get();
6
7     return 0;
8 }
9
10
```

static 修飾子 = 靜態成員變數

```
1
2 class Book {
3     static int i;
4     public:
5         int id;
6         void f(void);
7 };
8 int Book::i = 20210330; ← static
9                             要記得初始化
10 void Book::f(void){
11     cout << "I'm a book: " << id;
12     cout << "i = " << i++ << endl;
13 };
```

```
1
2 int main( ){
3
4     Book books[10];
5     for(int i = 0; i < 10; i++){
6         books[i].id = i;
7         books[i].f();
8     }
9     return 0;
10 }
```


→ static 是同一個 class 下，所有 object 共享的變數 11

建構子 (constructor)

建構子（constructor；ctor），是一個類別裡用於建立物件時初始化成員變數函式。在初始化一個新建的物件，能夠同時輸入參數用以設定實例變數。

```
1
2 class Book {
3     public:
4         int price;
5         string title;
6         Book(int p){
7             price = p;
8         };
9     };
10
```

```
1
2
3 int main( ){
4     Book book1(600);
5     cout << "price: " << book1.price;
6     return 0;
7 }
8
9
10
```



建構子多載

(Constructor Overloading)

```
1 class Book {  
2     public:  
3         int price;  
4         string title;  
5         Book(int p){ ← 一個輸入  
6             price = p;  
7         };  
8         Book(int p, string t){  
9             price = p;  
10            title = t;  
11        };  
12    };
```

```
1  
2  
3 int main( ){  
4     Book book1(600);  
5     Book book2(536, "C++ Prime");  
6  
7 }  
8  
9  
10
```

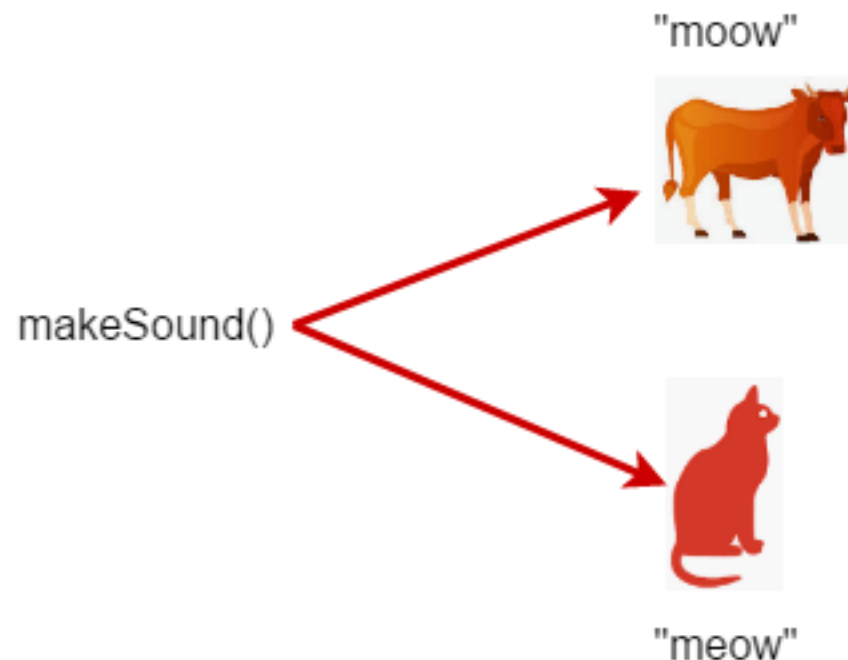
→ 多載 = 同一個函式能夠接受不同的參數輸入

多型

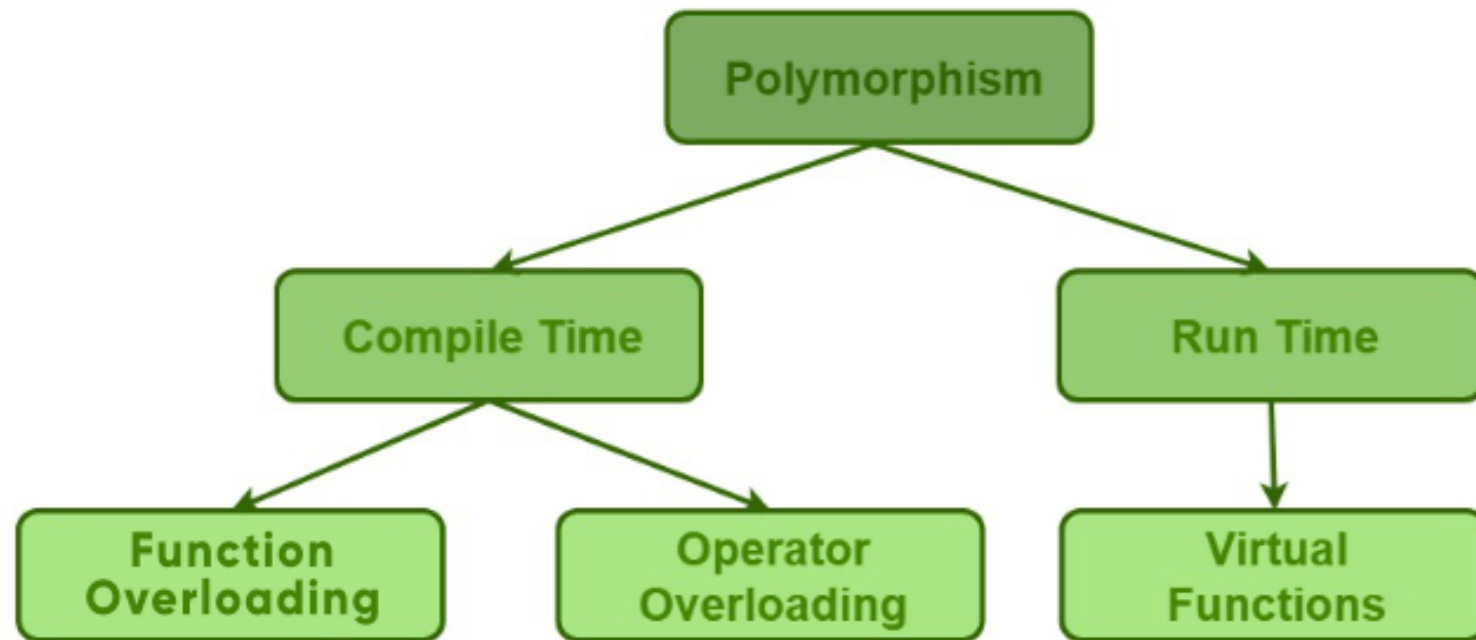
(polymorphism)

在物件導向的程式語言和特性中，多型指為不同資料類型的實體提供統一的介面，或使用一種符號能夠表示多個不同的目的。

→ 同一個東西，可以有不同的呈現方式



有兩種實現多型的做法



有兩種實現多型的做法

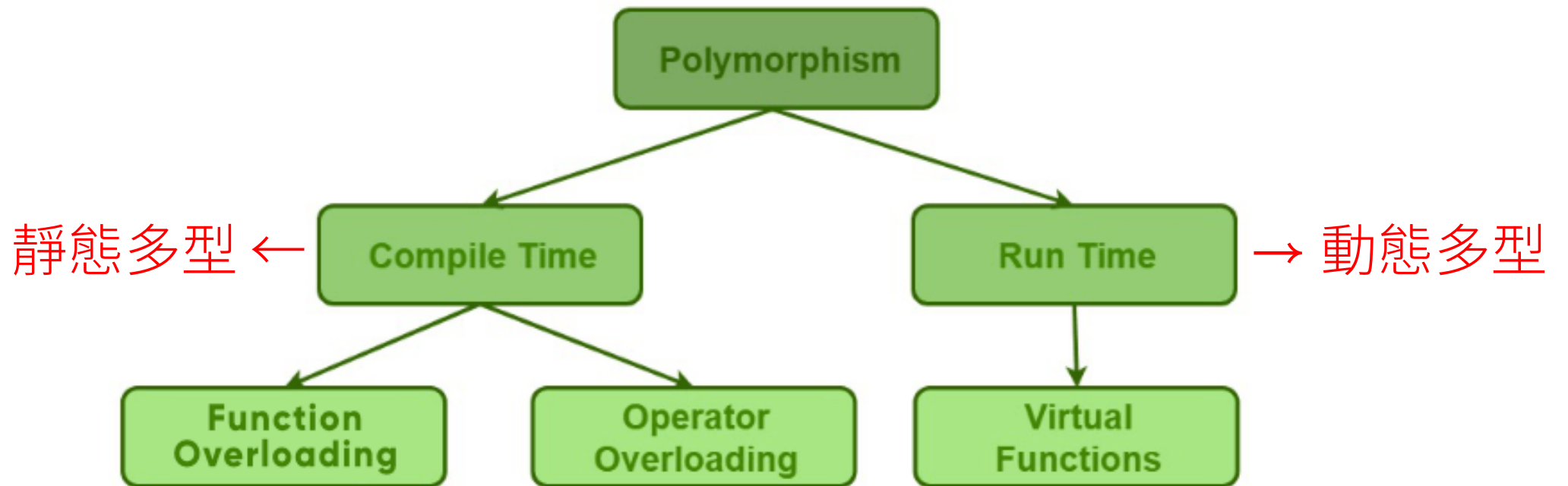
■ 多載 (Overloading)

相同函數名稱，不同參數，依據傳入參數呼叫對應的函數/運算

■ 覆寫 (Overriding)

相同函數名稱，不同類別，子類別可以覆寫父類別的函數/運算

有兩種實現多型的做法



多載 (overloading)

函式多載 (Overloading) 是物件導向的程式語言中具有的特性，允許建立數項名稱相同但輸入類型或個數不同的子程式，它可以簡單地稱為一個單獨功能可以執行多項任務的能力。

- 函式多載 (Function Overloading)
- 運算子多載 (Operator Overloading)

多載 (overloading)

函式多載 (Overloading) 是物件導向的程式語言中具有的特性，允許建立數項名稱相同但輸入類型或個數不同的子程式，它可以簡單地稱為一個單獨功能可以執行多項任務的能力。

- 函式多載 (Function Overloading)
- 運算子多載 (Operator Overloading)

→ 同一個函式名稱，可以接受不同的參數輸入

函式多載

→ 同一個函式名稱，可以接受不同的參數輸入

```
1 class F {  
2     public:  
3         void print(int i) {  
4             cout << "整數：" << i << endl;  
5         }  
6         void print(double f) {  
7             cout << "小數：" << f << endl;  
8         }  
9         void print(int i, double f) {  
10            cout << "整數和小數：" << i;  
11            cout << ", " << f << endl;  
12        }  
13 };  
14
```

```
1  
2  
3 int main(void){  
4  
5     F f;  
6  
7     f.print(1);  
8     f.print(1.5);  
9     f.print(1, 1.5);  
10  
11     return 0;  
12 }  
13  
14
```

函式多載

→ 同一個函式名稱，可以接受不同的參數輸入

```
1 class F {  
2     public:  
3         void print(int i) {  
4             cout << "整數：" << i << endl;  
5         }  
6         void print(double f) {  
7             cout << "小數：" << f << endl;  
8         }  
9         void print(int i, double f) {  
10            cout << "整數和小數：" << i;  
11            cout << ", " << f << endl;  
12        }  
13 };  
14
```

```
1  
2  
3 int main(void){  
4  
5     F f;  
6  
7     f.print(1);  
8     f.print(1.5);  
9     f.print(1, 1.5);  
10  
11     return 0;  
12 }  
13  
14
```

函式多載

→ 同一個函式名稱，可以接受不同的參數輸入
不同個數、不同型態

```
1 class F {
2     public:
3         void print(int i) {
4             cout << "整數：" << i << endl;
5         }
6         void print(double f) {
7             cout << "小數：" << f << endl;
8         }
9         void print(int i, double f) {
10            cout << "整數和小數：" << i;
11            cout << ", " << f << endl;
12        }
13 };
14
```

```
1
2
3 int main(void){
4
5     F f;
6
7     f.print(1);
8     f.print(1.5);
9     f.print(1, 1.5);
10
11     return 0;
12 }
13
14
```

函式多載

```
1 class F {
2     public:
3         F(int i) {
4             cout << "整數：" << i << endl;
5         }
6         F(double f) {
7             cout << "小數：" << f << endl;
8         }
9         F(int i, double f) {
10            cout << "整數和小數：" << i;
11            cout << ", " << f << endl;
12        }
13 };
14
```

```
1
2
3
4 int main(void){
5
6     F f1(1);
7     F f2(1.5);
8     F f3(1, 1.5);
9
10    return 0;
11 }
12
13
14
```

→ 建構子也是一種函式，也可以實現多載

建構子多載

(Constructor Overloading)

```
1 class Book {  
2     public:  
3         int price;  
4         string title;  
5         Book(int p){ ← 一個輸入  
6             price = p;  
7         };  
8         Book(int p, string t){  
9             price = p;  
10            title = t;  
11        };  
12    };
```

```
1  
2  
3 int main( ){  
4     Book book1(600);  
5     Book book2(536, "C++ Prime");  
6  
7 }  
8  
9  
10
```

→ 多載 = 同一個函式能夠接受不同的參數輸入

建構子函數多載

= 當宣告物件時會自動呼叫的初始化函式

```
1 class F {  
2     public:  
3         F(int i) {  
4             cout << "整數：" << i << endl;  
5         }  
6         F(double f) {  
7             cout << "小數：" << f << endl;  
8         }  
9         F(int i, double f) {  
10            cout << "整數和小數：" << i;  
11            cout << ", " << f << endl;  
12        }  
13 };  
14
```

```
1  
2  
3  
4 int main(void){  
5  
6     F f1(1);  
7     F f2(1.5);  
8     F f3(1, 1.5);  
9  
10    return 0;  
11 }  
12  
13  
14
```

→ 建構子也是一種函式，也可以實現多載

#思考一下

請問該如何實現兩個物件的合併？



\$123



\$456

#思考一下

請問該如何實現兩個物件的合併？



利用函式合併物件

如果想要讓多個 Object 合併成一個 Object ，我們可利用函式回傳物件來實現。

```
1 class Book {  
2     public:  
3         int price;  
4         Book merge(Book b){  
5             Book tmp;  
6             tmp.price = price + b.price;  
7             return tmp;  
8         }  
9     };  
10
```

```
1 int main( ){  
2  
3     Book b1, b2;  
4     b1.price = 123;  
5     b2.price = 456;  
6     Book b3 = b1.merge(b2);  
7     cout << b3.price << endl;  
8     return 0;  
9 }  
10
```

利用函式合併物件

如果想要讓多個 Object 合併成一個 Object ，我們可利用函式回傳物件來實現。

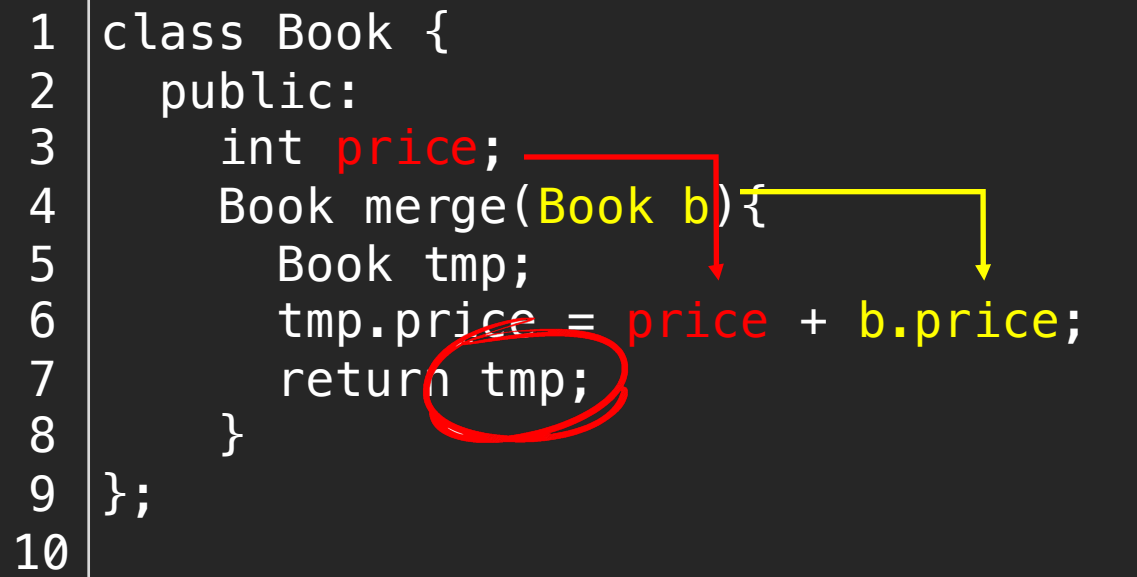
```
1 class Book {  
2     public:  
3         int price;  
4         Book merge(Book b){  
5             Book tmp;  
6             tmp.price = price + b.price;  
7             return tmp;  
8         }  
9     };  
10
```

```
1 int main( ){  
2  
3     Book b1, b2;  
4     b1.price = 123;  
5     b2.price = 456;  
6     Book b3 = b1.merge(b2);  
7     cout << b3.price << endl;  
8     return 0;  
9 }  
10
```

利用函式合併物件

如果想要讓多個 Object 合併成一個 Object，我們可利用函式回傳物件來實現。

```
1 class Book {
2     public:
3         int price;
4         Book merge(Book b){
5             Book tmp;
6             tmp.price = price + b.price;
7             return tmp;
8         }
9 };
10
```



```
1 int main( ){
2
3     Book b1, b2;
4     b1.price = 123;
5     b2.price = 456;
6     Book b3 = b1.merge(b2);
7     cout << b3.price << endl;
8     return 0;
9 }
10
```

#思考一下

請問該如何實現兩個物件的合併？



\$123

.merge(



\$456

)



\$579

#思考一下

請問該如何實現兩個物件的合併？



\$123

+



\$456



\$579

運算子多載

→ 運算子多載就是利用「運算子 (Operator)」來取代函式

```
1 class Book {  
2     public:  
3         int price;  
4         Book operator+(Book b){  
5             Book tmp;  
6             tmp.price = price + b.price;  
7             return tmp;  
8         }  
9     };  
10
```

```
1 int main( ){  
2  
3     Book b1, b2;  
4     b1.price = 123;  
5     b2.price = 456;  
6     Book b3 = b1 + b2;  
7     cout << b3.price << endl;  
8     return 0;  
9 }  
10
```

運算子多載

→ 運算子多載就是利用「運算子 (Operator)」來取代函式
operator+

```
1 class Book {  
2     public:  
3         int price;  
4         Book operator+(Book b){  
5             Book tmp;  
6             tmp.price = price + b.price;  
7             return tmp;  
8         }  
9     };  
10
```

```
1 int main( ){  
2  
3     Book b1, b2;  
4     b1.price = 123;  
5     b2.price = 456;  
6     Book b3 = b1 + b2;  
7     cout << b3.price << endl;  
8     return 0;  
9 }  
10
```

Binary 運算子多載

→ operator+ 可以用來代表 + 號運算，參數就是右邊的變數

```
1
2 class Book {
3     public:
4         int price;
5         Book operator+(Book b){
6             Book tmp;
7             tmp.price = price + b.price;
8             return tmp;
9         }
10    };
```

```
1 int main( ){
2
3     Book b1, b2;
4     b1.price = 123;
5     b2.price = 456;
6     Book b3 = b1 + b2;
7     cout << b3.price << endl;
8     return 0;
9 }
10
```

Binary 運算子多載

→ operator+ 也不一定要有回傳值

```
1 class Book {  
2     public:  
3         int price;  
4         void operator+(Book b){  
5             cout << price + b.price;  
6         }  
7     };  
8  
9  
10
```

```
1 int main( ){  
2  
3     Book b1, b2;  
4     b1.price = 123;  
5     b2.price = 456;  
6     b1 + b2;  
7  
8     return 0;  
9 }  
10
```

class 中的 this 代表該物件

```
1 class Book {  
2     public:  
3         int price;  
4         void operator+(Book b){  
5             cout << this->price + b.price;  
6         }  
7     };  
8  
9  
10
```

```
1 int main( ){  
2  
3     Book b1, b2;  
4     b1.price = 123;  
5     b2.price = 456;  
6     b1 + b2;  
7  
8     return 0;  
9 }  
10
```

本週上機練習

作業 #06

■#練習：利用 Card 與 Cards 類別設計一個洗牌程式。

■Requirements：

1. 定義 Card 和 Cards 兩種類別代表一張與一組撲克牌
2. Card 類別包含 face 和 suit 私有整數變數代表花色和大小
3. Cards 類別包含 一個 vector 變數存放所有 Card
4. Cards 類別包含 show(…) 和 shuffle(…) 成員函式印出卡片和洗牌

■Sample Input：參考下頁

■Sample Output：參考下頁

■Note：僅限 03/09 或 03/16 上課繳交

參考程式碼與結果

- 你可以參考下列程式碼修改，也可以自己從頭開始寫。只要執行結果必須符合右邊的格式即可。

```
作業 #07.cpp
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 string suits[4] = { "黑桃", "梅花", "愛心", "方塊" };
6 string faces[13] = {
7     "A", "2", "3", "4", "5", "6", "7",
8     "8", "9", "10", "J", "Q", "K"
9 };
10
11 class Card { ...
20 };
21
22 class Cards { ...
34 };
35
36 int main(){
37     Cards cards;
38     cout << "==== 洗牌前 =====\n";
39     cards.show();
40     cout << "==== 洗牌後 =====\n";
41     cards.shuffle();
42     cards.show();
43 }
```

```
Console Shell
ple02.cpp example03.cpp example04.cpp example05-1.cpp x n
ple05.cpp example06.cpp example07.cpp example08.cpp exampl
e09.cpp example10.cpp example11.cpp main.cpp 作業 #05 - 解
答.cpp 作業 #06 - 解答.cpp 作業 #06.cpp
> ./main
==== 洗牌前 ====
黑桃 A
黑桃 2
黑桃 3
黑桃 4
黑桃 5
黑桃 6
==== 洗牌後 ====
方塊 6
黑桃 A
黑桃 2
愛心 4
愛心 5
梅花 5
> []
```

```
class Card {
public:
    Card(int s,int f);
private:
    int suit, face;
};
```

```
class Cards {
private:
    vector<Card> cards;
public:
    Cards();
    void shuffle();
    void show();
};
```


提示

① 洗牌怎麼做？

➡ 任意兩張牌交換，重複交換多次即可達到洗牌效果。

② 物件該如何交換？

➡ 物件等同於變數，變數怎麼換、物件就怎麼換。

```
1 Card a, b, temp;  
2  
3  
4 temp = a;  
5 a = b;  
6 b = temp;
```

Thanks for listening.

元智大學 | C++ 程式設計實習

Wei-Yuan Chang