C++程式數計：運算子多載

**目錄**

[C++程式數計：運算子多載 1](#_Toc205821054)

[1. 二元運算子多載（最常見） 1](#_Toc205821055)

[1.1 範例：使用成員函式多載 +（沒用 friend） 2](#_Toc205821056)

[1.2 範例：使用 friend 外部函式多載 + 2](#_Toc205821057)

[2. 一元運算子多載 3](#_Toc205821058)

[2.1 範例：多載 ! 判斷是否為 0 3](#_Toc205821059)

[3. 複製機制 Shallow Copy & Deep Copy 3](#_Toc205821060)

[3.1 淺層複製（Shallow Copy）(預設) 4](#_Toc205821061)

[3.2 深層複製（Deep Copy） 4](#_Toc205821062)

[4. 特殊運算子多載 5](#_Toc205821063)

[4.1 範例：>> 與 << 運算子多載 5](#_Toc205821064)

[4.2 範例：= 運算子多載（指定／複製） 6](#_Toc205821065)

[4.3 範例：==運算子多載 8](#_Toc205821066)

[5. 型態轉換運算子多載 10](#_Toc205821067)

[5.1 範例：把「公分」自動轉成「公尺」 10](#_Toc205821068)

[5.2 範例：把「公分」自動轉成「公尺」，並配合多載 >> 10](#_Toc205821069)

[5.3 範例：類別 → 基本型別 11](#_Toc205821070)

[5.4 範例：基本型別 → 類別 13](#_Toc205821071)

[5.5 範例：類別 A → 類別 B 14](#_Toc205821072)

在 C++ 裡面，運算子（+ - \\* / > < 等）也可以被「重新定義」成你自己的函式，讓它支援你設計的類別（物件）操作，讓你可以這樣進行。

Student s1**,** s2**,** s3**;**

s3 **=** s1 **+** s2**;** // ✅ 自訂類別也能用「+」做相加

# 二元運算子多載（最常見）

例如：+、-、\\*、>、==

可以分為兩種：

* 相加的兩個，一個是單純是 class，另一個是物件
* 相加的兩個，兩個都是物件 => 要使用 friend

## 範例：使用成員函式多載 +（沒用 friend）

#include <iostream>

using namespace std;

class Student {

private:

    int Score;

public:

    Student(int s){

        Score = s;

    }

    // 二元運算子多載：左邊class本身 + 物件

    int operator+(const Student& b) {

        return this->Score + b.Score;

    }

};

int main() {

    Student x(90), y(75);

    cout << "總分: " << x + y << endl;  // 輸出：165

}

## 範例：使用 friend 外部函式多載 +

#include <iostream>

using namespace std;

class Student {

private:

    int Score;

public:

    Student(int s){

        Score = s;

    }

    // 二元運算子多載：左邊class本身 + 物件

    friend int operator+(Student& a, Student& b){

        return a.Score + b.Score;

    }

};

int main() {

    Student x(90), y(75);

    cout << "總分: " << x + y << endl;  // 輸出：165

}

# 一元運算子多載

例如：!、++、--、~

你可以多載 !a 表示「a 是否為 0」之類的用途。

## 範例：多載 ! 判斷是否為 0

#include <iostream>

using namespace std;

class IsZero {

private:

    int Num;

public:

    IsZero(int n) {

        Num = n;

    }

    bool operator!(){

        return Num <= 0;

    } // 回傳 true/false

};

int main() {

    IsZero a(3);

    if (!a) cout << "Num 小於等於 0";

    else cout << "Num 大於 0";

}

# 複製機制 Shallow Copy & Deep Copy

當你寫 `obj2 = obj1;`時， C++ 會建立一個「物件的副本」。但到底是真的一模一樣（值複製），還是只是共用同一塊記憶體（指標複製），就看你有沒有自己寫「複製邏輯」了。

## 淺層複製（Shallow Copy）(預設)

複製「指標位址」，不複製記憶體內容

多個物件共用同一塊記憶體 → 容易產生 bug

#include <iostream>

using namespace std;

/\*\*

 \* 下述程式碼會無法執行：

 \* 1. a.score 和 b.score 其實指向同一塊記憶體

 \* 2. 當 main() 結束後，a 和 b 都會執行解構子，兩次都 delete score，導致 雙重刪除（double delete）→ 程式崩潰

 \*/

class Student{

public:

    int\* score;

    Student(int s) {

        score = new int(s);

    }

};

int main() {

    Student a(90);

    Student b = a;      // Shallow Copy =>  b.score 和 a.score 是同一塊記憶體

    b->score = 50;

    cout << a->score << endl;  // 輸出：50（a 被影響）

    return 0;

}

## 深層複製（Deep Copy）

複製資料內容，不共用記憶體。需要自己寫「複製建構子」與「賦值運算子」。

#include <iostream>

using namespace std;

class Student {

public:

    int\* score;

    Student(int s) {

        score = new int(s);

    }

    // ✅ 深層複製：拷貝建構子

    Student(const Student& other) {

        score = new int(\*(other.score));

    }

    // ✅ 深層複製：賦值運算子

    Student& operator=(const Student& other) {

        if (this != &other) {

            delete score;

            score = new int(\*(other.score));

        }

        return \*this;

    }

};

int main() {

    Student a(90);

    Student b = a;

    \*(b.score) = 50;

    cout << \*(a.score) << endl;  // 輸出：50（a 被影響）

    return 0;

}

# 特殊運算子多載

## 範例：>> 與 << 運算子多載

C++ iostream 類別庫中定義了 istream 及 ostream 這兩個子類別，並定義>>運算子及 << 運算子供我們直接使用。除了可作為位元位移運算子之外,也是 C++中執行輸出入的運算子。而 cin 及 cout 則分別為 istream 與 ostream 類別所定義的物件,主要作用是方便我們執行輸出入的操作。

當各位設計多載>>或 << 運算子時,在運算子左方必須有 istream&或 ostream&型態的運算元(例如 C++內建的 cin 與 cout 關鍵字),因此這兩個運算子必須被多載成非成員函數。

istream**&** **operator>>(**istream**&** 傳回參數**,**類別名稱**,**物件參數**)**

ostream**&** **operator** **<<** **(**ostream**&** 傳回參數**,**類別名稱**,**物件參數**)**

#include <iostream>

using namespace std;

class Score {

private:

    int var1;

public:

    friend istream& operator>>(istream& input, Score& s){

        cout << "請輸入一個整數:";

        input >> s.var1;

        return input;

    }

    friend ostream& operator<<(ostream& output, Score& s){

        output << "輸入的值為:" << s.var1 << endl;

        return output;

    }

};

int main() {

    Score st1;

    cin >> st1;    // 利用多載 >> 進行輸入

    cout << st1;   // 利用多載 << 進行輸出

    return 0;

}

## 範例：= 運算子多載（指定／複製）

= 是「指定」運算子，預設會做「淺層複製」（shallow copy），如果你的 class 有動態記憶體配置（new / delete），你就需要自己定義 operator= 做「深層複製」（deep copy）。

它是個 成員函式（非靜態）不能是 friend！

#include <iostream>

#include <cstring> // for strcpy, strlen

using namespace std;

class MyClass {

private:

    char\* m\_Name;      // 動態字串

    int m\_English;

    int m\_Math;

    int m\_Chinese;

public:

    // 建構子

    MyClass(const char\* cName = "", int Eng = 0, int Math = 0, int Ch = 0) {

        m\_Name = new char[strlen(cName) + 1]; // 配置記憶體

        strcpy(m\_Name, cName);

        m\_English = Eng;

        m\_Math = Math;

        m\_Chinese = Ch;

    }

    // 解構子

    // ~MyClass() {

    //     delete[] m\_Name; // 釋放記憶體

    // }

    // 賦值運算子多載（深複製）

    MyClass& operator=(const MyClass& other) {

        if (this != &other) {  // 避免自我賦值

            delete[] m\_Name;   // 清除舊記憶體

            m\_Name = new char[strlen(other.m\_Name) + 1];

            strcpy(m\_Name, other.m\_Name);

            m\_English = other.m\_English;

            m\_Math = other.m\_Math;

            m\_Chinese = other.m\_Chinese;

        }

        return \*this; // 回傳目前物件

    }

    // 輸出運算子多載

    friend ostream& operator<<(ostream& out, const MyClass& obj) {

        out << "學生姓名：" << obj.m\_Name << endl;

        out << "英文成績：" << obj.m\_English << endl;

        out << "數學成績：" << obj.m\_Math << endl;

        out << "國文成績：" << obj.m\_Chinese << endl;

        return out;

    }

};

int main() {

    char name[100];

    int eng, math, ch;

    MyClass student1("TA", 66, 83, 78);  // 使用建構子建立

    cout << "\n[student1 原始資料]\n" << student1;

    MyClass student2;

    student2 = student1;  // 呼叫 operator= 做深複製

    cout << "\n[student2 複製後資料]\n" << student2;

    return 0;

}

## 範例：==運算子多載

比較兩個學生的學號與分數

#include <iostream>

#include <cstring>

using namespace std;

class Student {

private:

    char Student\_Num[10];

    int Student\_Score;

public:

    // 預設建構子

    Student(const char\* num, int score) {

        strcpy(Student\_Num, num);

        Student\_Score = score;

    }

    // 比較運算子多載

    bool operator==(const Student& b) {

        return Student\_Score == b.Student\_Score;

    }

    bool operator>(const Student& b) {

        return Student\_Score > b.Student\_Score;

    }

    bool operator<(const Student& b) {

        return Student\_Score < b.Student\_Score;

    }

    // 友元函式讓 cin 可讀取 private 成員

    friend istream& operator>>(istream& input, Student& obj) {

        cout << "請輸入學生學號：";

        input >> obj.Student\_Num;

        cout << "請輸入學生分數：";

        input >> obj.Student\_Score;

        return input;

    }

    // 顯示用

    void show() {

        cout << "學號：" << Student\_Num << "，分數：" << Student\_Score << endl;

    }

    char\* getNum() {

        return Student\_Num;

    }

    int getScore() {

        return Student\_Score;

    }

};

int main() {

    Student s1("TA1", 78), s2("TA2", 99);

    if (s1 == s2) {

        cout << "兩位學生的分數相同。\n";

    } else if (s1 > s2) {

        cout << "學號 " << s1.getNum() << " 分數較高（" << s1.getScore() << " 分）\n";

        cout << "學號 " << s2.getNum() << " 分數較低（" << s2.getScore() << " 分）\n";

    } else {

        cout << "學號 " << s2.getNum() << " 分數較高（" << s2.getScore() << " 分）\n";

        cout << "學號 " << s1.getNum() << " 分數較低（" << s1.getScore() << " 分）\n";

    }

    return 0;

}

# 型態轉換運算子多載

C++ 允許將基本型別（如 int, float, double 等）使用 強制轉換 (cast) 的方式互相轉換

自訂類別之間的轉換，可以透過下方式：

1. 類別 → 基本型別：類別中多載型別轉換函數 `operator type()`

2. 基本型別 → 類別：目標類別中提供「帶參數的建構子

3. 類別 A → 類別 B：目標類別 B 中建構子接受 A 類別為參數 或 類別 A 中定義 operator B()

## 範例：把「公分」自動轉成「公尺」

#include <iostream>

using namespace std;

class Length {

private:

    int cm;  // 長度（公分）

public:

    Length(int x) {

        cm = x;

    }

    // 型別轉換運算子：Length → float

    operator float() {

        return cm / 100.0;  // 自動轉成公尺

    }

};

int main() {

    Length l(250);       // 250 公分

    float m = l;         // ⬅️ 自動呼叫 operator float()

    cout << "公尺長度為：" << m << " m" << endl;

    return 0;

}

## 範例：把「公分」自動轉成「公尺」，並配合多載 >>

#include <iostream>

using namespace std;

class Length {

private:

    int cm;

public:

    Length(int x) { cm = x; }

    // 輸出多載

    friend ostream& operator<<(ostream& out, Length l) {

        out << l.cm << " 公分";

        return out;

    }

    operator float() {

        return cm / 100.0;

    }

};

int main() {

    Length l(150);

    cout << "長度為：" << l << endl;  // 呼叫 << 多載

    float m = l;                      // 呼叫 operator float()

    cout << "轉成公尺為：" << m << " m" << endl;

    return 0;

}

## 範例：類別 → 基本型別

有時候你希望把你自己定義的類別（class）轉換成內建資料型態，像是 float、int。

因為匯率通常不會是整數，因此需要進行型態轉換

#include <iostream>

using namespace std;

class Dollar{

private:

public:

    int NT\_Dollar; // 表示擁有的新台幣

    float Exchange\_Rate; // 台幣對美金的匯率

    // 建構子多載

    Dollar() {

        NT\_Dollar = 1;

        Exchange\_Rate = 40;

    }

    Dollar(int Money, float Rate) {

        NT\_Dollar = Money;

        Exchange\_Rate = Rate;

    }

    // 輸入運算子多載：從 istream 輸入 NT\_Dollar 與 Exchange\_Rate

    friend istream& operator >> (istream& in, Dollar& Tmp\_Money){

        cout << "請輸入1美元可兌換的台幣金額：";

        in >> Tmp\_Money.Exchange\_Rate;

        cout << "請輸入欲兌換的台幣金額：";

        in >> Tmp\_Money.NT\_Dollar;

        return in;

    }

    // 輸出運算子多載：將 NT\_Dollar 輸出

    friend ostream& operator << (ostream& out, Dollar& Tmp\_Money){

        out << Tmp\_Money.NT\_Dollar << " 元";

        return out;

    }

    float Get\_Rate() {

        return Exchange\_Rate;

    }

    // 類別型別轉換成 float 的 operator

    operator float() {

        float US\_Dollar;

        US\_Dollar = (float)NT\_Dollar / Exchange\_Rate;  // 匯率計算

        return US\_Dollar;

    }

};

int main() {

    Dollar TWD;

    float USD;

    cin >> TWD;                  // 使用 >> 輸入 TWD 的內容

    USD = (float)TWD;           // 呼叫型別轉換函數

    cout << endl;

    cout << "兌換比率 (美金 : 台幣) = 1 : " << TWD.Get\_Rate() << endl;

    cout << "要兌換的台幣金額 = " << TWD << endl;

    cout << "可兌換美金 = " << USD << " 元" << endl;

    return 0;

}

## 範例：基本型別 → 類別

#include <iostream>

using namespace std;

float Exchange\_Rate;  // 全域變數：台幣對美元匯率

class Dollar {

public:

    float NT\_Dollar;  // 新台幣金額

    Dollar() {

        NT\_Dollar = 1;

        Exchange\_Rate = 40;

    }

    Dollar(float Money, float Rate) {

        NT\_Dollar = Money;

        Exchange\_Rate = Rate;

    }

    // 型別轉換建構子：從 float 轉換為 Dollar 類別

    Dollar(float Money) {

        cout << "執行多載 - 型別轉換建構子函數。" << endl;

        NT\_Dollar = Money \* Exchange\_Rate;  // 將美元轉台幣

    }

    // 輸出多載：讓 cout << Dollar 可以印出內容

    friend ostream& operator << (ostream& out, Dollar Tmp\_Money) {

        out << Tmp\_Money.NT\_Dollar << " 元";

        return out;

    }

    // Getter：取得台幣金額（可加也可不加）

    float GetNT() {

        return NT\_Dollar;

    }

};

int main() {

    Dollar TWD;     // 台幣物件

    float USD, Rate;

    cout << "請輸入1美元可兌換的台幣金額：";

    cin >> Rate;

    cout << "請輸入要兌換的美金金額：";

    cin >> USD;

    Exchange\_Rate = Rate;

    TWD = USD;      // ❗ 此行會觸發「型別轉換建構子」

    cout << endl;

    cout << "兌換比率（美金 : 台幣）= 1 : " << Exchange\_Rate << endl;

    cout << "要兌換的美金金額 = " << USD << " 元" << endl;

    cout << "可兌換的台幣金額 = " << TWD << endl;

    return 0;

}

## 範例：類別 A → 類別 B

#include <iostream>

using namespace std;

// 宣告 USD 類別

class USD;

// 台幣類別 NTD

class NTD {

    float NT\_Exchange\_Rate;  // 表示 1 美金可兌換的台幣金額

public:

    // 建構子

    NTD() {

        NT\_Exchange\_Rate = 40;

    }

    NTD(float Rate) {

        NT\_Exchange\_Rate = Rate;

    }

    // 讓 cin >> NTD 成功

    friend istream& operator >> (istream& in, NTD& Tmp\_Money) {

        cout << "請輸入匯率比（台幣：美金）" << endl;

        cout << "1 美金等於：台幣 ";

        in >> Tmp\_Money.NT\_Exchange\_Rate;

        return in;

    }

    // operator：把 NTD 類別轉成 USD 類別

    operator USD();

};

// 美金類別 USD

class USD {

    float US\_Exchange\_Rate;

public:

    USD() {

        US\_Exchange\_Rate = 1;

    }

    USD(float Rate) {

        US\_Exchange\_Rate = Rate;

    }

    // 讓 cout << USD 成功

    friend ostream& operator << (ostream& out, USD Tmp\_Money) {

        out << Tmp\_Money.US\_Exchange\_Rate;

        return out;

    }

};

// operator USD()：實作 NTD → USD 類型轉換

NTD::operator USD() {

    float US\_Exchange\_Rate = 1 / NT\_Exchange\_Rate;  // 換算美金：1 / 台幣

    return USD(US\_Exchange\_Rate);

}

// main 主程式

int main() {

    NTD NT\_Dollar;

    USD US\_Dollar;

    cin >> NT\_Dollar;         // 輸入 NTD 的匯率內容

    US\_Dollar = NT\_Dollar;    // 呼叫 operator USD()，完成轉換

    cout << endl;

    cout << "等於匯率比（美金：台幣）" << endl;

    cout << "1 塊美金：" << US\_Dollar << endl;

    return 0;

}