C++程式設計：類別

**目錄**

[C++程式設計：類別 1](#_Toc205662991)

[1. 抽象資料型別 (ADT) 2](#_Toc205662992)

[2. class 基本語法 2](#_Toc205662993)

[2.1 建立 class 2](#_Toc205662994)

[2.2 利用 class 建立 object 2](#_Toc205662995)

[2.3 使用 object 2](#_Toc205662996)

[3. 建構子（Constructor） 2](#_Toc205662997)

[3.1 範例：建立沒有參數的 Constructor 3](#_Toc205662998)

[3.2 範例：建立有參數的 Constructor 3](#_Toc205662999)

[4. 解構子（Destructor） 4](#_Toc205663000)

[4.1 語法： 4](#_Toc205663001)

[4.2 範例：使用 Destructor 4](#_Toc205663002)

[5. 指標物件 5](#_Toc205663003)

[5.1 範例：使用指標物件動態建立 6](#_Toc205663004)

[6. 物件陣列 7](#_Toc205663005)

[6.1 範例：使用物件陣列 7](#_Toc205663006)

[7. 朋友函數(friend) 8](#_Toc205663007)

[7.1 範例：未使用 friend 而失敗 8](#_Toc205663008)

[7.2 範例：使用 friend function 修正 9](#_Toc205663009)

[8. 朋友類別(friend class) 10](#_Toc205663010)

[8.1 範例：未使用 friend class 而失敗 10](#_Toc205663011)

[8.2 範例：使用 friend class 修正 11](#_Toc205663012)

[9. this 指標 11](#_Toc205663013)

[9.1 範例：this 指標 12](#_Toc205663014)

[10. 靜態資料成員（static data member） 12](#_Toc205663015)

[10.1 範例：使用 static 成員計算建立幾個物件 13](#_Toc205663016)

[11. static 成員函式 14](#_Toc205663017)

[11.1 範例：使用 static 函數計算建立幾個物件 14](#_Toc205663018)

[12. 物件傳參數方式 15](#_Toc205663019)

[12.1 物件傳值呼叫 15](#_Toc205663020)

[12.2 範例：物件傳值呼叫：比較兩個 Box 的體積 15](#_Toc205663021)

[12.3 物件傳址呼叫（使用指標） 16](#_Toc205663022)

[12.4 範例：物件傳址呼叫：使用指標將第二個 Box 的尺寸複製到第一個 Box 16](#_Toc205663023)

[12.5 物件傳參考呼叫（推薦） 18](#_Toc205663024)

[12.6 範例：物件傳參考呼叫：把 boxB 的尺寸設定到 boxA 18](#_Toc205663025)

# 抽象資料型別 (ADT)

ADT 是指一種由資料與操作構成的資料型別，它隱藏實作細節，只提供介面讓使用者操作。而class 就是一種實作 ADT 的方式，結合了「資料成員」與「成員函式」。

# class 基本語法

## 建立 class

class Student {

private: // 表示這個資料只有類別裡的函數可以用

    int StuID;  // 學號(變數)

public: //  表示這個函數可以在類別外呼叫

    void input\_data();  // 輸入資料(函數)

    void show\_data();   // 顯示資料(函數)

};

## 利用 class 建立 object

int main(void) {

    Student stud1;  // 建立一個 Student 物件叫 stud1

    return 0;

}

## 使用 object

int main() {

    Student stud1;  // 建立一個 Student 物件叫 stud1

    // 使用

    stud1.input\_data();    // 呼叫輸入資料的函數

    stud1.show\_data();     // 呼叫顯示資料的函數

    return 0;

}

# 建構子（Constructor）

建構子是「在你建立物件時自動執行的特殊函數」，用途是用來初始化物件裡的成員變數。其特點：

* 名字與 class 名稱一樣
* 不需要回傳型態
* 可以有多個建構子 => 支援「多載」（overloading）
* 沒有寫建構子，系統會自動幫你生一個「預設建構子」

## 範例：建立沒有參數的 Constructor

#include <iostream>

using namespace std;

class Student {

private:

    int StuID;

    float English, Math, Total, Average;

public:

    Student();  // 預設建構子（沒參數）

    Student(int id, float E, float M); // 自訂建構子（有三個參數）

    void show\_data(); // 顯示資料

};

// 作用域解析運算子中去給member值

Student::Student() {

    StuID = 920101;

    English = 60;

    Math = 80;

}

void Student::show\_data() {

    cout << Math << "\n";

}

int main() {

    Student stud1; // 建立object的時候，會自動去跑object的construct

    stud1.show\_data();

    return 0;

}

## 範例：建立有參數的 Constructor

#include <iostream>

using namespace std;

class Student {

private:

    int StuID;

    float English, Math, Total, Average;

public:

    Student();  // 預設建構子（沒參數）

    Student(int id, float E, float M); // 自訂建構子（有三個參數）

    void show\_data(); // 顯示資料

};

Student::Student(int id, float E, float M) {

    StuID = id;

    English = E;

    Math = M;

}

void Student::show\_data() {

    Total = English + Math;

    Average = Total / 2;

    cout << "學號是: " << StuID << endl;

    cout << "總分是: " << Total << "，平均是: " << Average << endl;

}

int main() {

    Student stud2(920102, 30, 40); // 呼叫三參數建構子

    stud2.show\_data();          // 顯示 stud2 的資料

}

# 解構子（Destructor）

解構子是一種特殊函數，當一個物件「生命週期結束」時，會自動執行來釋放資源或做收尾工作。其功能：

* 自動執行「清理工作」 => 程式結束、物件離開作用域、delete 物件時就會被呼叫。
* 幫你釋放記憶體或結束前的動作（像是關檔案、釋放陣列等）。

## 語法：

~類別名稱(); // 注意前面有一個 ~ 符號，不能有任何參數，也不寫回傳型態

## 範例：使用 Destructor

#include <iostream>

using namespace std;

class testN {

private:

    int no[10];     // 陣列

    int i;

public:

    testN();        // 建構子

    ~testN();       // 解構子

};

// 定義constructor

testN::testN() {

    int i;

    for(i = 0; i < 10; i++) {

        no[i] = i;

    }

    cout << "建構子執行完成。" << endl;

}

// 定義Destructor

testN::~testN() {

    cout << "解構子發動中。\n陣列內的內容：";

    for(i = 0; i < 10; i++) {

        cout << no[i] << " ";

    }

    cout << "\n物件已執行完畢。" << endl;

}

// 使用解構子的函式

void show\_result() {

    testN test1;  // 宣告物件（建構子會在這裡執行）

    return;       // 函數結束時 test1 的解構子會自動執行

}

int main() {

    show\_result();  // 呼叫函式，裡面會建立 test1

    return 0;

}

# 指標物件

上述的物件建立方式屬於「靜態建立物件」=> 當程式跑到這行時，stud1 就會自動建立，也會自動釋放記憶體。如果希望「動態建立物件」就要用「指標物件」的方式。

Student\* stud1 = new Student;

這時會：

* 在記憶體堆區（heap）分配空間。
* 回傳該物件的指標（Student\\*）。
* 你必須手動用 delete 去刪除物件。

## 範例：使用指標物件動態建立

#include <iostream>

using namespace std;

class Student {

private:

    char StuID[8];         // 學號（用字元陣列表示）

    float Score\_E, Score\_M, Score\_T, Score\_A;  // 分數資料

public:

    // 建構子：物件建立時自動執行

    Student() {

        cout << "\*\*\* 執行建構子 $$$ \*\*\*" << endl;

    }

    // 解構子：物件刪除時自動執行

    ~Student() {

        cout << "### 執行解構子 ###" << endl;

    }

    // 函數

    void input\_data() {

        cout << "請輸入學號及科系標籤：" << endl;

        cout << "學號：";

        cin >> StuID;

        cout << "請輸入英文成績：";

        cin >> Score\_E;

        cout << "請輸入數學成績：";

        cin >> Score\_M;

    }

    void show\_data(); // 這邊我用作用域解析運算子去做

};

void Student::show\_data(){

    Score\_T = Score\_E + Score\_M;

    Score\_A = Score\_T / 2;

    cout << "==============================" << endl;

    cout << "學號是：" << StuID << endl;

    cout << "總分是 " << Score\_T << " 分，平均是 " << Score\_A << " 分" << endl;

    cout << "==============================" << endl;

}

int main() {

    Student\* stud1 = new Student; // 使用 new 建立指標物件（在 heap 動態記憶體中建立）

    // 使用指標呼叫成員函數（用 ->）

    stud1->input\_data();

    stud1->show\_data();

    delete stud1; // 使用 delete 釋放指標物件（呼叫解構子）

    return 0;

}

# 物件陣列

陣列可以放 int、放 char、放 string，也可以放 object

## 範例：使用物件陣列

#include <iostream>

using namespace std;

class Baseball {

private:

    char player[20];

    int fires;

    int safes;

public:

    void inputplayer();   // 輸入資料

    void showplayer();    // 顯示資料

    float countsafe();    // 計算打擊率 = safes / fires

};

void Baseball::inputplayer() {

    cout << "請輸入球員名字：";

    cin >> player;

    cout << "請輸入打數：";

    cin >> fires;

    cout << "請輸入安打數：";

    cin >> safes;

}

void Baseball::showplayer() {

    cout << "球員：" << player << endl;

    cout << "打數：" << fires << "，安打數：" << safes << endl;

    cout << "打擊率：" << countsafe() << endl;

}

float Baseball::countsafe() {

    if (fires == 0) return 0.0;

    return (float) safes/fires;

}

int main(){

    Baseball b[3];        // 宣告陣列（3 位球員）

    for(int i=0; i<3; i++){

        b[i].inputplayer();  // 輸入每個球員資料

    }

    for(int i=0; i<3; i++){

        b[i].showplayer();   // 顯示球員資訊與打擊率

    }

    return 0;

}

# 朋友函數(friend)

C++ 類別通常會把成員設為 private，不能讓別人直接存取。但你可以指定某些「朋友（friend）」函數或類別來特別允許他們存取 private 成員！

## 範例：未使用 friend 而失敗

假設要宣告一個 class【Student】，裡面儲存期末總分(Score\_T)，然後要寫一個外部變數(add\_bonus)，作用是幫忙加分。

#include <iostream>

using namespace std;

class Student {

private:

    int Score\_T;  // 期末總分

public:

    Student(int s) {

        Score\_T = s;

    }

};

// 外部函數，想要幫忙加分

int add\_bonus(Student s) {

    return s.Score\_T + 10;  // ❌ 錯誤：Score\_T 是 private，外面看不到

}

int main() {

    Student s1(70);

    cout << "加分後的成績是：" << add\_bonus(s1) << endl;

    return 0;

}

// .\test.cpp:16:14: 錯誤：「int Student::Score\_T」 is private within this context

// Score\_T 是 private，外部函數 add\_bonus() 不是 Student 的成員 => 所以不能直接「偷看」成績 => 封裝

## 範例：使用 friend function 修正

#include <iostream>

using namespace std;

class Student {

private:

    int Score\_T;  // 期末總分

public:

    Student(int s) {

        Score\_T = s;

    }

    friend int add\_bonus(Student);  // ✅ 宣告這個外部函數是朋友！

};

// 外部函數，想要幫忙加分

int add\_bonus(Student s) {

    return s.Score\_T + 10;  // ❌ 錯誤：Score\_T 是 private，外面看不到

}

int main() {

    Student s1(70);

    cout << "加分後的成績是：" << add\_bonus(s1) << endl;

    return 0;

}

# 朋友類別(friend class)

friend 類別是一種權限授予機制，讓另一個類別可以存取本類別的 private 和 protected 成員。

## 範例：未使用 friend class 而失敗

假設要宣告一個 class【Student】，裡面儲存期末總分(Score\_T)。另一格 class【Teacher】負責批改與顯示成績，但它不是 Student 的成員類別

#include <iostream>

using namespace std;

class Student {

private:

    int Score\_T;

public:

    Student(int s) {

        Score\_T = s;

    }

};

class Teacher {

public:

    void show\_score(Student s) {

        cout << "學生分數是：" << s.Score\_T << endl;  // ❌ 錯誤！Score\_T 是 private

    }

};

int main() {

    Student s1(80);

    Teacher t;

    t.show\_score(s1);

    return 0;

}

// .\test.cpp: 在 「void Teacher::show\_score(Student)」 成員函式中:

// .\test.cpp:16:37: 錯誤：「int Student::Score\_T」 is private within this contex

## 範例：使用 friend class 修正

#include <iostream>

using namespace std;

class Student {

private:

    int Score\_T;

public:

    Student(int s) {

        Score\_T = s;

    }

    friend class Teacher;  // ✅ 宣告整個 Teacher 類別是朋友

};

class Teacher {

public:

    void show\_score(Student s) {

        cout << "學生分數是：" << s.Score\_T << endl;  // ❌ 錯誤！Score\_T 是 private

    }

};

int main() {

    Student s1(80);

    Teacher t;

    t.show\_score(s1);

    return 0;

}

# this 指標

每個物件建立時，C++ 會自動產生一個隱藏的指標：this

this 是一個指向「目前這個物件」的指標，用來：

* 訪問這個物件的資料成員
* 回傳目前物件本身（回傳 \\*this）

## 範例：this 指標

#include <iostream>

using namespace std;

class Square{

private:

public:

    int a;

    // 建構子

    Square(int n){

        a = n \* n;

    }

    // 成員函數

    Square squ\_sum(Square b){

        this->a = this->a + b.a;

        return \*this;

    }

    int show\_data() {

        cout << this->a << endl;

        return 0;

    }

};

int main() {

    int n1 = 6, n2 = 8;

    Square first(n1), second(n2), third(0);      // 物件初始化

    third = first.squ\_sum(second);               // first 加 second，回傳新物件

    third.show\_data();                           // 印出結果

    return 0;

}

# 靜態資料成員（static data member）

是被 static 宣告的「類別成員變數」。它不屬於某個物件，而是整個類別共用。所有物件共享這個變數的「同一份記憶體」。

// 宣告靜態成員：只能宣告，不能初始化

class MyClass {

    static int counter;  // 宣告為靜態資料成員

};

int MyClass::counter = 0;  // 類別外部初始化靜態成員（這是必要的！）

## 範例：使用 static 成員計算建立幾個物件

#include <iostream>

#include <string.h>

using namespace std;

class Dog {

private:

    char\* pName;

    char\* pColor;

    static int counter;  // 靜態變數宣告：統計產生幾個 Dog 物件

public:

    // 建構子

    Dog(const char\* pN, const char\* pC) {

        pName = new char[strlen(pN)+1];

        strcpy(pName, pN);

        pColor = new char[strlen(pC)+1];

        strcpy(pColor, pC);

        counter++;  // 每建立一隻狗，就 +1

    }

    int getCounter() {

        return counter;

    }

};

// 類別外部初始化

int Dog::counter = 0;

int main() {

    Dog d1("小白", "白色");

    Dog d2("小黃", "黃色");

    Dog d3("小紅", "紅色");

    cout << d1.getCounter() << endl;  // 印出 3

    return 0;

}

# static 成員函式

被 static 關鍵字修飾的成員函式。它是「整個類別」共用，而不是「某個物件」的行為 => 所以它 不能存取非靜態成員（非 static 的變數或函式）

// 宣告靜態成員：只能宣告，不能初始化

class MyClass **{**

static int counter**;** // 宣告為靜態資料成員

**};**

int MyClass**::**counter **=** 0**;** // 類別外部初始化靜態成員（這是必要的！）

## 範例：使用 static 函數計算建立幾個物件

#include <iostream>

#include <string.h>

using namespace std;

class Dog {

private:

    char\* pName;

    char\* pColor;

    static int counter;  // 靜態變數宣告：統計產生幾個 Dog 物件

public:

    // 建構子

    Dog(const char\* pN, const char\* pC) {

        pName = new char[strlen(pN)+1];

        strcpy(pName, pN);

        pColor = new char[strlen(pC)+1];

        strcpy(pColor, pC);

        counter++;  // 每建立一隻狗，就 +1

    }

    static int getCounter() {

        return counter;

    }

};

// 類別外部初始化

int Dog::counter = 0;

int main() {

    Dog d1("小白", "白色");

    Dog d2("小黃", "黃色");

    Dog d3("小紅", "紅色");

    cout << Dog::getCounter() << endl;  // 印出 3

    return 0;

}

# 物件傳參數方式

## 物件傳值呼叫

將「物件的副本」傳進函數中。

函數內操作的是複製品，不會影響原始物件。

其特點：

* 安全，不會改變原物件
* 效率低（大型物件複製成本高）

## 範例：物件傳值呼叫：比較兩個 Box 的體積

#include <iostream>

using namespace std;

class Box {

private:

    double length, width, height;

public:

    // 建構子

    Box(double l = 1, double w = 1, double h = 1) {

        length = l;

        width = w;

        height = h;

    }

    // 計算體積

    double volume() {

        return length \* width \* height;

    }

    // 顯示尺寸

    void show() {

        cout << "尺寸：" << length << " x " << width << " x " << height << endl;

    }

};

// 傳值呼叫函數

void compare(Box b1, Box b2) {

    b1.show();

    b2.show();

    if (b1.volume() > b2.volume())

        cout << "📦 第 1 個盒子比較大！" << endl;

    else if (b1.volume() < b2.volume())

        cout << "📦 第 2 個盒子比較大！" << endl;

    else

        cout << "📦 兩個盒子一樣大！" << endl;

}

int main() {

    Box boxA(3, 4, 5);   // 體積 60

    Box boxB(2, 6, 5);   // 體積 60

    compare(boxA, boxB); // 傳值呼叫

    return 0;

}

## 物件傳址呼叫（使用指標）

傳遞的是物件的「記憶體位址」 => 函數內可以修改原物件的值

函數內可以透過 -> 操作原始物件。

## 範例：物件傳址呼叫：使用指標將第二個 Box 的尺寸複製到第一個 Box

#include <iostream>

using namespace std;

class Box {

private:

    double length, width, height;

public:

    Box(double l = 1, double w = 1, double h = 1) {

        length = l;

        width = w;

        height = h;

    }

    void set(double l, double w, double h) {

        length = l;

        width = w;

        height = h;

    }

    void show() {

        cout << "尺寸：" << length << " x " << width << " x " << height << endl;

    }

};

// 傳址呼叫：將 b2 的尺寸複製給 b1

void update(Box\* b1, Box\* b2) {

    cout << "🛠️ 正在複製 boxB 的尺寸到 boxA...\n";

    \*b1 = \*b2;  // 或使用 b1->set(...) 也可以

}

int main() {

    Box boxA(2, 3, 4);

    Box boxB(5, 5, 5);

    cout << "🔸 原始資料：" << endl;

    boxA.show();

    boxB.show();

    update(&boxA, &boxB);  // 傳址呼叫

    cout << "\n✅ 更新後 boxA：" << endl;

    boxA.show();

    return 0;

}

## 物件傳參考呼叫（推薦）

傳遞的是「原物件的別名（alias），使用 & 修飾參數來實現

優點：

* 不會產生物件複製，節省記憶體與時間
* 看起來就像傳值，但其實能改到原本的內容
* 可以直接用`.`，比指標直覺

## 範例：物件傳參考呼叫：把 boxB 的尺寸設定到 boxA

#include <iostream>

using namespace std;

class Box {

private:

    double length, width, height;

public:

    // 建構子

    Box(double l = 1, double w = 1, double h = 1) {

        length = l;

        width = w;

        height = h;

    }

    void set(double l, double w, double h) {

        length = l;

        width = w;

        height = h;

    }

    void show() {

        cout << "尺寸：" << length << " x " << width << " x " << height << endl;

    }

};

// 傳參考呼叫函式

void update(Box& b1, Box& b2) {

    cout << "🔧 將 boxB 的尺寸設定給 boxA...\n";

    b1 = b2;  // 或使用 b1.set(...) 也可以

}

int main() {

    Box boxA(3, 3, 3);

    Box boxB(5, 5, 5);

    cout << "🚩 更新前：" << endl;

    boxA.show();

    boxB.show();

    update(boxA, boxB);  // 傳參考呼叫

    cout << "\n✅ 更新後 boxA：" << endl;

    boxA.show();

    return 0;

}