C++ 面向对象程序设计
编程题整理（开卷考试用）

实验一 / 二 / 三 / 四 + MOOC编程题 · 按章节整理

2025/26 第二学期 · 总评 = 平时50% + 期末50%

题型分布

一、程序阅读：4 × 5' = 20'

二、程序填空：15 × 2' = 30'（4小题）

三、编程题1：15'

四、编程题2：15'

五、编程题3：20'

目录

第 3-4 章 · 类与对象 + 进阶

【实验一·题目1】 BookCard 借书证类

私有：string id; string stuName; int number。

构造函数带默认参数；display()显示3个成员；borrow()若number<10则+1并返回true，否则返回false。

```
#include <iostream>
#include <string>
using namespace std;
class BookCard {
private:
    string id;
    string stuName;
    int number;
public:
    BookCard(string i="                ", string name="雪峰", int num=3)
        : id(i), stuName(name), number(num) {}
    void display() {
        cout << id << " " << stuName << " " << number << endl;
    }
    bool borrow() {
        if (number >= 10) return false;
        number++;
        return true;
    }
};
void f(BookCard &bk) {
    if (!bk.borrow()) {
        bk.display();
        cout << "you have borrowed 10 books,can not borrow any more!" << endl;
    } else
        bk.display();
}
int main() {
    BookCard bk1("                ", "东平", 10), bk2;
    f(bk1);
    f(bk2);
    return 0;
}
```

输出：　　东平 10 / you have borrowed... /　　雪峰 4

【实验一·题目2】 Time 时间类 · 拷贝构造 · 参数传递分析

私有：int Hour, Minute, Second。构造函数、拷贝构造、析构函数、SetTime、Get系列、PrintTime。

重点：IncreaseOneSecond() 实现进位；分析f(Time)、f(Time&)、f(Time*)三种参数传递的构造/析构调用次数。

```
#include <iostream>
using namespace std;
class Time {
private:
    int Hour, Minute, Second;
public:
    Time(int h=0, int m=0, int s=0) : Hour(h), Minute(m), Second(s) {
        cout << "Constructor called." << endl;
    }
    Time(const Time &t) : Hour(t.Hour), Minute(t.Minute), Second(t.Second) {
        cout << "Copy constructor called." << endl;
    }
    ~Time() { cout << "Destructor called." << endl; }
    void SetTime(int h, int m, int s) { Hour=h; Minute=m; Second=s; }
    int GetHour()   { return Hour; }
    int GetMinute() { return Minute; }
    int GetSecond() { return Second; }
    void PrintTime() {
        cout << Hour << ":" << Minute << ":" << Second << endl;
    }
    void IncreaseOneSecond() {
        Second++;
        if (Second >= 60) { Second = 0; Minute++; }
        if (Minute >= 60) { Minute = 0; Hour++; }
        if (Hour   >= 24) { Hour   = 0; }
    }
};
void f(Time t)  { t.PrintTime(); }   // 值传递
// void f(Time &t) { t.PrintTime(); }  // 引用传递
// void f(Time *t) { t->PrintTime(); } // 指针传递
int main() {
    Time t1(10, 59, 59);
    t1.PrintTime();
    t1.IncreaseOneSecond();
    t1.PrintTime();
    f(t1);
    return 0;
}
```

【参数传递对比】调用一次f的次数统计

值传递 f(Time t)：构造1次、拷贝构造1次、析构2次

引用传递 f(Time &t;)：构造1次、拷贝构造0次、析构1次

指针传递 f(Time *t)：构造1次、拷贝构造0次、析构1次

结论：引用/指针不产生临时对象，避免拷贝开销，调用时无需创建新对象

【实验一·题目3】 Boy / Girl 类 · 友元函数 + 友元类

两个类都有 string name、int age。友元关系是单向的（A是B的友元，不代表B是A的友元）。

最终版：用顶层友元函数 VisitBoyGirl(Boy&, Girl&) 同时访问两个类的私有成员。

```
#include <iostream>
#include <string>
using namespace std;
class Girl;  // 前向声明
class Boy {
private:
    string name;
    int age;
public:
    Boy(string n, int a) : name(n), age(a) {}
    void Print() { cout << "Boy: " << name << ", Age: " << age << endl; }
    friend void VisitBoyGirl(Boy &b, Girl &g);
};
class Girl {
private:
    string name;
    int age;
public:
    Girl(string n, int a) : name(n), age(a) {}
    void Print() { cout << "Girl: " << name << ", Age: " << age << endl; }
    friend void VisitBoyGirl(Boy &b, Girl &g);
};
// 顶层友元函数
void VisitBoyGirl(Boy &b, Girl &g) {
    cout << "--- 通过顶层友元函数输出信息 ---" << endl;
    cout << "男孩 - 姓名: " << b.name << ", 年龄: " << b.age << endl;
    cout << "女孩 - 姓名: " << g.name << ", 年龄: " << g.age << endl;
}
int main() {
    Boy b1("张三", 20);
    Girl g1("李四", 19);
    VisitBoyGirl(b1, g1);
    return 0;
}
```

【备用版本】友元类（Girl访问Boy的私有数据）

```
class Boy {
private:
    string name; int age;
public:
    Boy(string n, int a) : name(n), age(a) {}
    friend class Girl;   // 声明 Girl 为友元类
};
class Girl {
public:
    void VisitBoy(Boy &b) {
        cout << "姓名:" << b.name << ", 年龄:" << b.age << endl;
    }
};
```

友元特性：单向（不对称）、不传递、破坏封装性需谨慎使用

【MOOC】 Student 类 · 对象数组 · 深拷贝

私有：int age; char *name。带参构造、无参构造（age=0, name="unnamed"）、析构（释放name）、SetMember、Getage、Getname。

main()中：Student stu[3]={Student(13,"wang")}; 第1个带参，其余无参；再 stu[2].SetMember(12,"zhang");

深拷贝核心：构造/SetMember 中用 new char[] 复制字符串，析构 delete[] name

```
#include<iostream>
#include<cstring>
using namespace std;
class Student {
private:
    int age;
    char *name;
public:
    Student(int m, char *n) {
        age = m;
        name = new char[strlen(n)+1];
        strcpy(name, n);
    }
    Student() {
        age = 0;
        name = new char[strlen("unnamed")+1];
        strcpy(name, "unnamed");
    }
    ~Student() { delete[] name; }
    void SetMember(int m, char *n) {
        age = m;
        delete[] name;
        name = new char[strlen(n)+1];
        strcpy(name, n);
    }
    int Getage() { return age; }
    char* Getname() { return name; }
};
int main() {
    Student stu[3] = { Student(13,"wang") };
    stu[2].SetMember(12,"zhang");
    for(int i=0; i<3; i++)
        cout << stu[i].Getage() << "," << stu[i].Getname() << "\n";
    return 0;
}
```

输出：13,wang / 0,unnamed / 12,zhang

【MOOC】 Car 类 · 构造函数默认参数

私有：string brand, type; int year; double price。

构造函数默认参数：brand=type="undefinition"，year=2000，price=0。

```
#include<iostream>
#include<string>
using namespace std;
class Car {
private:
    string brand, type;
    int year;
    double price;
public:
    Car(string b="undefinition", string t="undefinition",
        int y=2000, double p=0)
        : brand(b), type(t), year(y), price(p) {}
    string GetBrand() { return brand; }
    string GetType()  { return type; }
    int    GetYear()  { return year; }
    double GetPrice() { return price; }
};
int main() {
    Car car1("FIAT","Palio",2021,6.5);
    cout << car1.GetBrand() << " " << car1.GetType() << " "
         << car1.GetYear()  << " " << car1.GetPrice() << endl;
    Car car2;
    cout << car2.GetBrand() << " " << car2.GetType() << " "
         << car2.GetYear()  << " " << car2.GetPrice() << endl;
    return 0;
}
```

输出：FIAT Palio 2021 6.5 / undefinition undefinition 2000 0

【MOOC】 Point + Line 类 · 对象成员 · 计算线段长度

Point类：private double X, Y。Point(a,b)、Point(Point &p;)拷贝构造、GetX/GetY。

Line类：private Point A, B; double length。构造函数计算并存入length。需要 #include。

```
#include<iostream>
#include<iomanip>
#include<cmath>
using namespace std;
class Point {
private:
    double X, Y;
public:
    Point(double a, double b) : X(a), Y(b) {}
    Point(Point &p) : X(p.X), Y(p.Y) {}
    double GetX() { return X; }
    double GetY() { return Y; }
};
class Line {
private:
    Point A, B;
    double length;
public:
    Line(Point p1, Point p2) : A(p1), B(p2) {
        double dx = A.GetX() - B.GetX();
        double dy = A.GetY() - B.GetY();
        length = sqrt(dx*dx + dy*dy);
    }
    double GetLength() { return length; }
};
int main() {
    double a, b, c, d;
    cin >> a >> b >> c >> d;
    Point p1(a, b), p2(c, d);
    Line L(p1, p2);
    cout << setprecision(3) << L.GetLength() << endl;
    return 0;
}
```

构造函数初始化列表中 A(p1), B(p2) 会调用 Point 的拷贝构造

【MOOC】 Student 类 · 静态成员 · 对象计数

私有：int age; string name。公有静态：static int count。

构造函数 count++，析构 count--。Print() 先输出 count 再输出成员。

静态成员必须在类外初始化：int Student::count = 0;

```
#include<iostream>
#include<string>
using namespace std;
class Student {
    int age;
    string name;
public:
    static int count;
    Student(int m, string n) : age(m), name(n) { count++; }
    Student() : age(0), name("NoName") { count++; }
    ~Student() { count--; }
    void Print() const {
        cout << count << endl;
        cout << "Name=" << name << " , age=" << age << endl;
    }
};
int Student::count = 0;
int main() {
    cout << "count=" << Student::count << endl;
    Student s1, *p = new Student(23, "ZhangHong");
    s1.Print();
    p->Print();
    delete p;
    s1.Print();
    Student Stu[4];
    cout << "count=" << Student::count << endl;
    return 0;
}
```

输出：count=0 / 2 / Name=NoName,age=0 / 2 / Name=ZhangHong,age=23 / 1 / Name=NoName,age=0 / count=5

第 5 章 · 继承性

【实验二·题目1】 Vehicle / Bicycle / Car / MotorCycle · 菱形继承 + 虚基类

基类 Vehicle（MaxSpeed, Weight），第二层 Bicycle 加 Height、Car 加 SeatNum，第三层 MotorCycle 同时继承 Bicycle 和 Car（菱形）。

Vehicle 必须声明为虚基类才能消除二义性；MotorCycle 构造函数初始化列表中必须显式调用 Vehicle 构造函数。

```
#include <iostream>
using namespace std;
class Vehicle {
protected:
    int MaxSpeed, Weight;
public:
    Vehicle(int m, int w) { MaxSpeed = m; Weight = w; }
    void Run()  { cout << "Vehicle is running." << endl; }
    void Stop() { cout << "Vehicle has stopped." << endl; }
    void Show() {
        cout << "MaxSpeed=" << MaxSpeed
             << ", Weight=" << Weight << endl;
    }
};
// 虚继承（关键！） —— 否则 MotorCycle 会有两份 Vehicle
class Bicycle : virtual public Vehicle {
protected:
    int Height;
public:
    Bicycle(int n, int w, int h) : Vehicle(n,w) { Height = h; }
    void Show() {
        Vehicle::Show();
        cout << "It's height is:" << Height << endl;
    }
};
class Car : virtual public Vehicle {
protected:
    int SeatNum;
public:
    Car(int m, int w, int s) : Vehicle(m,w) { SeatNum = s; }
    void Show() {
        Vehicle::Show();
        cout << "It's seatnum is:" << SeatNum << endl;
    }
};
// 第三层：同时继承 Bicycle 和 Car —— 必须显式调用虚基类 Vehicle
class MotorCycle : public Bicycle, public Car {
public:
    MotorCycle(int m, int w, int h, int s)
        : Vehicle(m,w), Bicycle(m,w,h), Car(m,w,s) {}
};
int main() {
    Bicycle b(100, 50, 170);
    Car c(120, 80, 5);
    MotorCycle mc(120, 80, 170, 5);
    b.Show();
    c.Show();
    // mc.Show();  // 二义性，需限定：mc.Bicycle::Show();
    return 0;
}
```

【三种继承方式对比】

public 继承：基类 public 成员 → 派生类 public，类外可访问

protected 继承：基类 public 成员 → 派生类 protected，类外不可访问

private 继承：基类所有成员 → 派生类 private，类外不可访问

报错信息：error: 'void Vehicle::Run()' is inaccessible within this context

【构造顺序】定义 MotorCycle mc 时

虚基类 Vehicle → Bicycle → Car → MotorCycle（析构相反）

【实验二·题目2】 Base / Derived · 赋值兼容原则 · 4种情况

赋值兼容的4种合法情况（基类 ← 派生类，单向）：

① 派生类对象 → 基类对象（切片，丢失派生类信息）

② 派生类对象 → 基类引用（初始化）

③ 派生类对象地址 → 基类指针

④ 派生类对象指针 → 基类指针

```
#include <iostream>
using namespace std;
class Base {
protected:
    int a;
public:
    Base(int x) : a(x) { cout << "Base constructor, a=" << a << endl; }
    void Show() { cout << "Base::Show, a=" << a << endl; }
};
class Derived : public Base {
private:
    int b;
public:
    Derived(int x) : Base(x), b(x*2) {
        cout << "Derived constructor, b=" << b << endl;
    }
    void Show() {
        cout << "Derived::Show, a=" << a << ", b=" << b << endl;
    }
};
int main() {
    Base b1(10);
    Derived d1(20);
    // ① 派生类对象 → 基类对象
    b1 = d1;          b1.Show();  // 调用 Base::Show（切片）
    // ② 派生类对象 → 基类引用
    Base &b2 = d1;    b2.Show();  // 调用 Base::Show（静态绑定）
    // ③ 派生类对象地址 → 基类指针
    Base *b3 = &d1;   b3->Show(); // 调用 Base::Show（除非虚函数）
    // ④ new 派生类 → 基类指针
    b3 = new Derived(30); b3->Show();
    return 0;
}
```

【不可逆性 — 编译报错示例】

以下4条语句全部编译报错，体现赋值兼容的不可逆：

```
Derived d5 = b1;    // 基类对象不能初始化派生类对象
Derived &d6 = b1;   // 基类对象不能绑定到派生类引用
Derived *d7 = &b1;  // 基类指针不能转为派生类指针
d7 = b3;            // Base* 不能赋给 Derived*
```

关键结论：基类指针/引用 调用同名函数时，除非声明为 virtual，否则总是调用基类版本（静态绑定）。这就是为什么多态需要虚函数！

【MOOC】 POINT → CIRCLE · 单继承

POINT类 protected: int x, y。公有：构造、change(a,b)、show()输出"(x,y)"。

CIRCLE 公有继承 POINT，增加私有 int r，重写 show() 输出圆心+半径。change() 直接继承。

```
#include<iostream>
using namespace std;
class POINT {
protected:
    int x, y;
public:
    POINT(int a, int b) : x(a), y(b) {}
    void change(int a, int b) { x = a; y = b; }
    void show() { cout << "(" << x << "," << y << ")" << endl; }
};
class CIRCLE : public POINT {
private:
    int r;
public:
    CIRCLE(int a, int b, int c) : POINT(a,b), r(c) {}
    void show() {
        cout << "the center of the circle is:" << endl;
        POINT::show();
        cout << "the radius is:" << r << endl;
    }
};
int main() {
    POINT p(2, 3);
    CIRCLE c(3, 4, 5);
    cout << "original p:" << endl; p.show();
    p.change(20, 30);
    cout << "changed p:" << endl; p.show();
    cout << "original c:" << endl; c.show();
    c.change(30, 40);
    cout << "changed c:" << endl; c.show();
    return 0;
}
```

CIRCLE::show() 中用 POINT::show() 调用基类同名函数（避免无限递归）

【MOOC】 Data / Teacher / Student / Postgrad / Tpost · 虚基类菱形继承

Teacher 和 Student 都虚继承 Data；Postgrad 继承 Student；Tpost 多重继承 Teacher 和 Postgrad。

关键：Tpost 构造函数必须显式调用虚基类 Data(n)，否则 name 不会被正确初始化

```
#include<iostream>
#include<string>
using namespace std;
class Data {
protected:
    string name;
public:
    Data() {}
    Data(string n) : name(n) {}
};
class Teacher : virtual public Data {
protected:
    float sal;
public:
    Teacher() {}
    Teacher(string n, float s) : Data(n), sal(s) {}
};
class Student : virtual public Data {
protected:
    string id;
public:
    Student() {}
    Student(string n, string i) : Data(n), id(i) {}
};
class Postgrad : public Student {
protected:
    string dn;
public:
    Postgrad() {}
    Postgrad(string n, string i, string d)
        : Data(n), Student(n,i), dn(d) {}
};
class Tpost : public Teacher, public Postgrad {
public:
    Tpost() {}
    Tpost(string n, string i, string d, float s)
        : Data(n), Teacher(n,s), Postgrad(n,i,d) {}
    void print() {
        cout << "name=" << name << endl;
        cout << "id="   << id   << endl;
        cout << "dn="   << dn   << endl;
        cout << "sal="  << sal  << endl;
    }
};
int main() {
    string name, id, dn;
    float sal;
    cin >> name >> id >> dn >> sal;
    Tpost t(name, id, dn, sal);
    cout << "The teacher and postgraduate:" << endl;
    t.print();
    return 0;
}
```

构造调用顺序：虚基类Data → Teacher → Student → Postgrad → Tpost

第 6 章 · 多态性

【实验三·题目1】 Point 类 · 运算符重载（综合）

成员函数：前置++、双目-。友元函数：双目+（两版本：Point+Point, Point+double）、插入运算符<<。

为什么 Point+double 必须用友元？因为左操作数若是 double 则成员函数无法处理。

```
#include <iostream>
using namespace std;
class Point {
private:
    double x, y;
public:
    Point(double a = 0.0, double b = 0.0) : x(a), y(b) {}
    // 成员函数：前置++
    Point& operator++() { ++x; ++y; return *this; }
    // 成员函数：双目-
    Point operator-(const Point &p) const {
        return Point(x - p.x, y - p.y);
    }
    // 友元函数：版本1 Point + Point
    friend Point operator+(const Point &p1, const Point &p2) {
        return Point(p1.x + p2.x, p1.y + p2.y);
    }
    // 友元函数：版本2 Point + double
    friend Point operator+(const Point &p, double d) {
        return Point(p.x + d, p.y + d);
    }
    // 友元函数：插入运算符<<
    friend ostream& operator<<(ostream &os, const Point &p) {
        os << "(" << p.x << "," << p.y << ")" << endl;
        return os;
    }
};
int main() {
    Point pt1(10.5, 20.8), pt2(-5.3, 18.4), pt3;
    cout << "original pt1,pt2,pt3 are:\n";
    cout << pt1 << pt2 << pt3;
    pt3 = pt1 + 100.8;
    cout << "after pt3=pt1+100.8, pt3 is:" << pt3;
    pt3 = pt1 + pt2;
    cout << "after pt3=pt1+pt2, pt3 is:" << pt3;
    pt3 = ++pt1; ++pt2;
    cout << "after ++ pt1,pt2,pt3 are:\n";
    cout << pt1 << pt2 << pt3;
    pt3 = pt1 - pt2;
    cout << "after pt3=pt1-pt2, pt3 is:" << pt3;
    return 0;
}
```

输出关键：(10.5,20.8) (-5.3,18.4) (0,0) / (111.3,121.6) / (5.2,39.2) / (11.5,21.8) / (15.8,2.4)

【实验三·题目2】 Container 抽象类 · 派生 Cube / Sphere / Cylinder

基类 Container 含3个纯虚函数：SurfaceArea()、Volume()、Print()。protected: double radius; 常量 PI。

派生 Cube（边长用 radius 存）、Sphere（半径）、Cylinder（半径+高，需新增 height 成员）。

```
#include <iostream>
using namespace std;
const double PI = 3.14159;
class Container {
protected:
    double radius;
public:
    Container(double r) : radius(r) {}
    virtual double SurfaceArea() = 0;    // 纯虚函数
    virtual double Volume()      = 0;
    virtual void   Print()       = 0;
};
class Cube : public Container {
public:
    Cube(double side) : Container(side) {}
    double SurfaceArea() { return 6 * radius * radius; }
    double Volume()      { return radius * radius * radius; }
    void Print() {
        cout << "正方体：边长=" << radius
             << "，表面积=" << SurfaceArea()
             << "，体积="   << Volume() << endl;
    }
};
class Sphere : public Container {
public:
    Sphere(double r) : Container(r) {}
    double SurfaceArea() { return 4 * PI * radius * radius; }
    double Volume()      { return 4.0/3.0 * PI * radius * radius * radius; }
    void Print() {
        cout << "球体：半径=" << radius
             << "，表面积=" << SurfaceArea()
             << "，体积="   << Volume() << endl;
    }
};
class Cylinder : public Container {
private:
    double height;
public:
    Cylinder(double r, double h) : Container(r), height(h) {}
    double SurfaceArea() { return 2 * PI * radius * (radius + height); }
    double Volume()      { return PI * radius * radius * height; }
    void Print() {
        cout << "圆柱体：半径=" << radius << "，高=" << height
             << "，表面积=" << SurfaceArea()
             << "，体积="   << Volume() << endl;
    }
};
void TopPrint(Container &r) { r.Print(); }   // 顶层函数
int main() {
    Container *p;
    Cube cube(3);
    Sphere sphere(2);
    Cylinder cylinder(2, 5);
    p = &cube;     p->Print();
    p = &sphere;   p->Print();
    p = &cylinder; p->Print();
    TopPrint(cube);
    TopPrint(sphere);
    TopPrint(cylinder);
    return 0;
}
```

Container c; 会报 error C2259（不能实例化抽象类），因为有未实现的纯虚函数

【MOOC】 抽象类 shape · 圆柱+球体积

纯虚函数：virtual double volume()=0; PI=3.1415; 用基类指针调用实现动态多态。

```
#include<iostream>
using namespace std;
const double PI = 3.1415;
class shape {
public:
    virtual double volume() = 0;
};
class cylinder : public shape {
private:
    double r, h;
public:
    cylinder(double r, double h) : r(r), h(h) {}
    double volume() { return PI*r*r*h; }
};
class sphere : public shape {
private:
    double r;
public:
    sphere(double r) : r(r) {}
    double volume() { return (4.0/3) * PI * r * r * r; }
};
int main() {
    shape *p;
    double r, h;
    cin >> r >> h;
    cylinder cy(r, h);
    sphere sp(r);
    p = &cy; cout << p->volume() << endl;
    p = &sp; cout << p->volume() << endl;
    return 0;
}
```

【MOOC】 Matrix 矩阵类 · 运算符重载（+ 和 =）

成员：int row, col; int *m。重载+检查行列相等，不等则输出"program terminated!"并 exit(0)。

重载=深拷贝赋值，行列不等也退出。析构 delete[] m。默认构造 m=nullptr。

operator+ 中不能用 Matrix C(r,c) 这种带参构造（会触发cin读取），要用默认构造再手动设置

```
#include<iostream>
#include<stdlib.h>
using namespace std;
class Matrix {
    int row, col;
    int *m;
public:
    Matrix() : row(0), col(0), m(nullptr) {}
    Matrix(int r, int c) : row(r), col(c) {
        m = new int[r*c];
        for(int i=0; i<r*c; i++) cin >> m[i];
    }
    Matrix(const Matrix &M) : row(M.row), col(M.col) {
        m = new int[row*col];
        for(int i=0; i<row*col; i++) m[i] = M.m[i];
    }
    ~Matrix() { delete[] m; }

    Matrix operator+(const Matrix &B) {
        if(row != B.row || col != B.col) {
            cout << "program terminated!" << endl;
            exit(0);
        }
        Matrix C;
        C.row = row; C.col = col;
        C.m = new int[row*col];
        for(int i=0; i<row*col; i++) C.m[i] = m[i] + B.m[i];
        return C;
    }
    Matrix& operator=(const Matrix &B) {
        if(this == &B) return *this;
        if(row != B.row || col != B.col) {
            cout << "program terminated!" << endl;
            exit(0);
        }
        for(int i=0; i<row*col; i++) m[i] = B.m[i];
        return *this;
    }
    void disp() {
        for(int i=0; i<row; i++) {
            cout << "\t";
            for(int j=0; j<col; j++) cout << *(m + i*col + j) << "\t";
            cout << endl;
        }
    }
};
int main() {
    int row_a, col_a, row_b, col_b;
    cin >> row_a >> col_a;  Matrix A(row_a, col_a);
    cin >> row_b >> col_b;  Matrix B(row_b, col_b), C;
    C = A + B;  C.disp();
    A = B;      A.disp();
    return 0;
}
```

第 8 章 · I/O 流类库

【实验四·题目1】 Course 类 · 文件读取 · 重载 >> 和 <<

文件 d:\course.txt 存有若干门课程及选课人数。Course类含 string name; int num。重载>>从文件读，<<输出到屏幕。

ifstream派生自istream、ofstream派生自ostream，所以重载用 istream&/ostream& 形参即可同时支持cin和文件

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
class Course {
private:
    string name;
    int num;
public:
    Course() : name(""), num(0) {}
    friend istream& operator>>(istream &in, Course &c);
    friend ostream& operator<<(ostream &out, const Course &c);
};
istream& operator>>(istream &in, Course &c) {
    in >> c.name >> c.num;
    return in;
}
ostream& operator<<(ostream &out, const Course &c) {
    out << "课程名称：" << c.name
        << "  选课人数：" << c.num << endl;
    return out;
}
int main() {
    ifstream fin("d:\\course.txt");
    if (!fin) {
        cout << "文件打开失败！" << endl;
        return 1;
    }
    Course c;
    int count = 0;
    while (fin >> c) {
        cout << c;
        count++;
    }
    cout << "共有 " << count << " 条记录。" << endl;
    fin.close();
    return 0;
}
```

【实验四·题目2】 文本文件 · 小写转大写

ReadFile(char *s)：打开文件，逐字符读取并显示到屏幕。

Change(char *s1, char *s2)：读 s1 文件，将小写字母转大写，写入 s2 文件。

```
#include <iostream>
#include <fstream>
using namespace std;

void ReadFile(char *s) {
    ifstream fin(s);
    if (!fin) {
        cout << "文件 " << s << " 打开失败！" << endl;
        return;
    }
    char ch;
    while (fin.get(ch))
        cout << ch;
    cout << endl;
    fin.close();
}

void Change(char *s1, char *s2) {
    ifstream fin(s1);
    ofstream fout(s2);
    if (!fin || !fout) {
        cout << "文件打开失败！" << endl;
        return;
    }
    char ch;
    while (fin.get(ch)) {
        if (ch >= 'a' && ch <= 'z')
            ch = ch - 'a' + 'A';   // 转换为大写
        fout.put(ch);
    }
    fin.close();
    fout.close();
}

int main() {
    cout << "ff.txt 的内容如下：" << endl;
    ReadFile("ff.txt");
    Change("ff.txt", "ff2.txt");
    cout << "ff2.txt 的内容如下：" << endl;
    ReadFile("ff2.txt");
    return 0;
}
```

get(ch)逐字符读取；put(ch)逐字符写出；while循环读到文件结束自动退出

【实验四·题目3】 Student 类 · 二进制文件读写（选做但重要）

学号、姓名、性别、成绩。用 write() 写入对象数组到 stu.dat，再用 read() 读出显示。

打开二进制文件必须加 ios::binary；读取时 in.eof() + in.gcount()==0 防止多读一条空记录

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
const int num = 3;
class Student {
private:
    char nu[20];
    char na[20];
    char se[5];
    int  sc;
public:
    Student() {}
    Student(char *nu, char *na, char *se, int s);
    friend ostream& operator<<(ostream &out, const Student &s);
};

Student::Student(char *nu, char *na, char *se, int s) {
    strcpy(this->nu, nu);
    strcpy(this->na, na);
    strcpy(this->se, se);
    sc = s;
}

ostream& operator<<(ostream &out, const Student &s) {
    out << "学号：" << s.nu << "  姓名：" << s.na
        << "  性别：" << s.se << "  成绩：" << s.sc << endl;
    return out;
}

void CreateBiFile(char *filename) {
    ofstream out(filename, ios::binary);
    Student stu[3] = {
        Student("                ", "          ", "女", 92),
        Student("                ", "李明",   "男", 85),
        Student("                ", "王芳",   "女", 88)
    };
    out.write((char*)stu, sizeof(stu));     // 二进制写
    out.close();
}

void ReadBiFile(char *filename) {
    Student stu[num];
    int i = 0;
    ifstream in(filename, ios::binary);
    while (!in.eof()) {
        in.read((char*)&stu[i], sizeof(Student));    // 二进制读
        if (in.gcount() == 0) break;     // 防止多读一条空记录
        cout << stu[i];
        i++;
        if (i >= num) break;
    }
    in.close();
}

int main() {
    CreateBiFile("stu.dat");
    ReadBiFile("stu.dat");
    return 0;
}
```

附录 · 易错知识点速查表

一、构造函数调用情况（第3-4章重点）

① A a, b[3]; → 全部调用无参构造

② A *p = new A; → 调用无参构造

③ A *p = new A[3]; → 调用3次无参构造

④ A a, b = a; → a:无参构造，b:复制构造（初始化语法）

⑤ A a, b(a); → a:无参构造，b:复制构造

⑥ A a, b; b = a; → 两次无参构造 + 一次赋值运算（不是拷贝构造！）

⑦ void fun(A x); A a; fun(a); → 复制构造（对象作值参数）

调用复制构造函数的3种情况：① 用同类对象初始化 ② 作函数值参数 ③ 作函数返回值

二、运算符重载要点（第6章重点）

只能重载为成员函数：= () [] ->

只能重载为友元函数：>> <<（流运算符）

不能重载：. .* :: sizeof ?:

常用模板：

```
class X {
public:
    // 双目 +  成员函数（左操作数必须是本类对象）
    X operator+(const X &b) const;

    // 双目 +  友元函数（左操作数可以任意类型，对称）
    friend X operator+(const X &a, const X &b);

    // 单目 -  成员函数
    X operator-() const;

    // 前缀 ++ 成员函数
    X& operator++() { ++data; return *this; }

    // 后缀 ++ 成员函数（用 int 参数区分前后缀）
    X operator++(int) { X tmp = *this; ++data; return tmp; }

    // 赋值 = 成员函数（必须返回引用支持链式）
    X& operator=(const X &b);

    // 友元：输出 <<
    friend ostream& operator<<(ostream &out, const X &obj);
    // 友元：输入 >>
    friend istream& operator>>(istream &in, X &obj);
};
```

三、虚函数 / 抽象类（第6章）

纯虚函数：virtual 返回类型 函数名(参数) = 0;

抽象类：含至少一个纯虚函数的类。不能实例化对象，只能定义指针或引用。

动态多态三要素：① 继承关系 ② virtual函数 ③ 基类指针或引用调用

派生类必须实现所有纯虚函数，才能被实例化；否则它也是抽象类。

四、继承构造/析构顺序（第5章）

构造顺序：虚基类 → 非虚基类（按声明顺序）→ 成员对象（按声明顺序）→ 本类构造函数体

析构顺序：与构造完全相反

多重继承同名冲突：用 基类名::成员名 限定

菱形继承二义性：用 virtual 继承变成虚基类

赋值兼容原则：基类指针/引用可指向派生类对象；派生类对象可赋值给基类对象（切片）

五、深拷贝 vs 浅拷贝（第3章）

何时需要深拷贝：类中有指针成员，且构造函数用 new 申请了内存。

浅拷贝危害：默认拷贝构造只复制指针值（地址）→ 两个对象指向同一内存 → 析构时 double-free（指针悬挂）

解决方案：自定义拷贝构造函数 + 重载赋值运算符，重新申请内存并复制内容。

```
// 深拷贝构造函数（字符串版）
ClassName(const ClassName &obj) {
    name = new char[strlen(obj.name) + 1];
    strcpy(name, obj.name);
}
// 深拷贝赋值运算符（注意自赋值检查 + 先释放旧空间）
ClassName& operator=(const ClassName &obj) {
    if (this == &obj) return *this;     // 自赋值检查
    delete[] name;                      // 释放旧空间
    name = new char[strlen(obj.name) + 1];
    strcpy(name, obj.name);
    return *this;
}
```

六、文件操作四步（第8章）

① 定义文件流 → ② 打开文件 → ③ 读/写 → ④ 关闭

```
#include<fstream>
using namespace std;
// 文本文件
ofstream outf("data.txt");   ifstream inf("data.txt");
outf << x;                   inf >> x;
outf.put(ch);                inf.get(ch);
outf.close();                inf.close();

// 二进制文件（必须加 ios::binary）
ofstream outf("data.dat", ios::binary);
outf.write((char*)&obj, sizeof(obj));

ifstream inf("data.dat", ios::binary);
inf.read((char*)&obj, sizeof(obj));
```