**面向对象程序设计实践（C++）**

**课程实验报告**

|  |  |
| --- | --- |
| 题目 | 面向对象程序设计实践（C++）课程实验 |
| 姓名 | 计子毅 |
| 学号 | 2023212872 |
| 班级 | 2023211303 |

目录

[1 基础实验 2](#_Toc200891615)

[1.1 C++基础知识实验​ 2](#_Toc200891616)

[1.1.1 实践笔记 2](#_Toc200891617)

[1.1.2 源代码 2](#_Toc200891618)

[1.1.3 测试 4](#_Toc200891619)

[1.1.4 实验心得 4](#_Toc200891620)

[1.2 类与对象实验1​ 5](#_Toc200891621)

[1.2.1 实践笔记 5](#_Toc200891622)

[1.2.2 源代码 5](#_Toc200891623)

[1.2.3 测试 7](#_Toc200891624)

[1.2.4 实验心得 8](#_Toc200891625)

[1.3 类与对象实验2​ 9](#_Toc200891626)

[1.3.1 实践笔记 9](#_Toc200891627)

[1.3.2 源代码 9](#_Toc200891628)

[1.3.3 测试 12](#_Toc200891629)

[1.3.4 实验心得 13](#_Toc200891630)

[1.4 继承与派生实验 13](#_Toc200891631)

[1.4.1 源代码 13](#_Toc200891632)

[1.4.2 测试 15](#_Toc200891633)

[1.4.3 实验心得 15](#_Toc200891634)

[1.5 I/O 流实验 16](#_Toc200891635)

[1.5.1 实践笔记 16](#_Toc200891636)

[1.5.2 源代码 16](#_Toc200891637)

[1.5.3 测试 17](#_Toc200891638)

[1.5.4 实验心得 17](#_Toc200891639)

[1.6 重载实验1 18](#_Toc200891640)

[1.6.1 源代码 18](#_Toc200891641)

[1.6.2 测试 20](#_Toc200891642)

[1.6.3 实验心得 20](#_Toc200891643)

[1.7 重载实验2 21](#_Toc200891644)

[1.7.1 源代码 21](#_Toc200891645)

[1.7.2 测试 22](#_Toc200891646)

[1.7.3 实验心得 22](#_Toc200891647)

[2 综合题目——电商交易平台 23](#_Toc200891648)

[2.1 题目简要介绍 23](#_Toc200891649)

[2.1.1 题目一：账户管理子系统和商品管理子系统（单机版） 23](#_Toc200891650)

[2.1.2 题目二：交易管理子系统（单机版） 24](#_Toc200891651)

[2.1.3 题目三：电商交易平台（网络版） 24](#_Toc200891652)

[2.2 总体设计 24](#_Toc200891653)

[2.3 关键设计说明 25](#_Toc200891654)

[2.3.1 继承体系设计 25](#_Toc200891655)

[2.3.2 状态流转设计​ 26](#_Toc200891656)

[2.3.3 异常处理 26](#_Toc200891657)

[2.4 详细设计 27](#_Toc200891658)

[2.4.1 账户管理子系统详细设计 27](#_Toc200891659)

[2.4.2 商品管理子系统详细设计 27](#_Toc200891660)

[2.4.3 交易管理子系统详细设计 28](#_Toc200891661)

[2.4.4 数据库说明 29](#_Toc200891662)

[2.4.5 接口协议说明（网络版） 30](#_Toc200891663)

[2.5 实现 30](#_Toc200891664)

[2.5.1 主要问题与解决方案 30](#_Toc200891665)

[2.5.2 核心收获 32](#_Toc200891666)

# 基础实验

## C++基础知识实验​

### 实践笔记

**难点**：矩阵输出问题​

测试时发现未考虑到矩阵对齐输出

​**解决方案**​：

1. 包含iomanip头文件
2. 利用std::setw()函数设置输出的宽度为12

**难点**：矩阵输出错误

测试时发现输入4\*5的矩阵输出5\*4的矩阵

​**解决方案**​：

1. 更改输出循环中的行列变量位置
2. 在输出矩阵函数中添加对矩阵行列的判断

### 源代码

#include<iostream>

using namespace std;

//动态分配内存

int\*\* NewMatrix(int row, int column) {

int\*\* num;

num = new int\* [row];

for (int i = 0; i < row; i++) {

num[i] = new int[column];

}

return num;

}

//矩阵初始化函数

void MatrixInit(int\*\*& num, int row, int column) {

for (int i = 0; i < row; i++) {

for (int j = 0; j < column; j++) {

cin >> num[i][j];

}

}

}

//矩阵输出函数

void PrintMatrix(int\*\* num, int row, int column) {

for (int i = 0; i < row; i++) {

for (int j = 0; j < column; j++) {

cout << num[i][j] << " ";

}

cout << endl;

}

}

//矩阵相加函数

void AddMatrix(int\*\* a, int\*\* b, int \*\*c,int row, int column) {

for (int i = 0; i < row; i++) {

for (int j = 0; j < column; j++) {

c[i][j] = a[i][j] + b[i][j];

}

}

}

//矩阵相减

void SubMatrix(int\*\* a, int\*\* b, int \*\*c,int row, int column) {

for (int i = 0; i < row; i++) {

for (int j = 0; j < column; j++) {

c[i][j] = a[i][j] - b[i][j];

}

}

}

//释放矩阵

void DeleteMatrix(int\*\* a, int row, int column) {

for (int i = 0; i < row; i++) {

delete[] a[i];

}

delete[] a;

}

int main() {

int \*\*A, \*\*B, \*\*C;

A = NewMatrix(4, 5);

B = NewMatrix(4, 5);

C = NewMatrix(4, 5);

MatrixInit(A, 4, 5);

MatrixInit(B, 4, 5);

AddMatrix(A, B, C,4, 5);

cout << "矩阵相加：" << endl;

PrintMatrix(C, 4, 5);

cout << "矩阵相减：" << endl;

SubMatrix(A, B,C, 4, 5);

PrintMatrix(C, 4, 5);

DeleteMatrix(A,4,5);

DeleteMatrix(B,4,5);

DeleteMatrix(C,4,5);

return 0;

}

### 测试

测试如下

电脑萤幕

AI 生成的内容可能不正确。

未符合预期

### 实验心得

本次实验通过实现矩阵的动态内存分配、初始化、运算及释放，深入理解了C++的内存管理机制。在调试过程中，遇到了两个关键问题：

​​矩阵输出对齐问题​​：最初直接使用cout输出矩阵，导致不同位数的数字无法对齐，影响可读性。通过引入<iomanip>库的setw()函数，设置固定输出宽度（如setw(12)），使矩阵整齐排列。这让我认识到，良好的输出格式不仅能提升用户体验，还能帮助调试时快速定位数据异常。

​​矩阵行列混淆问题​​：在测试时发现，输入4×5的矩阵却输出5×4的矩阵。检查后发现是循环变量i和j在输出函数中写反了。这个错误让我意识到，​​变量命名应当清晰​​（如row和col而非i和j），并在关键逻辑处添加注释，避免混淆。

​​分析​​：

​​动态内存管理​​：使用new和delete手动分配和释放内存，虽然灵活，但容易导致内存泄漏（如忘记释放）。后续实验可以考虑使用智能指针（如std::unique\_ptr）来避免此类问题。

​​代码健壮性​​：矩阵运算函数未检查输入矩阵的行列是否匹配，可能导致运行时错误。​​防御性编程​​（如添加assert(row1 == row2 && col1 == col2)）能提高程序的稳定性。

## 类与对象实验1​

### 实践笔记

**难点**：允许用户不输入默认圆的构造 ​

只传输了默认参数但输入时未允许用户不输入

​**解决方案**​：

1. 规定一个特定的输入，当用户输入时即以默认圆构造

### 源代码

#include<iostream>

using namespace std;

//动态分配内存

int\*\* NewMatrix(int row, int column) {

int\*\* num;

num = new int\* [row];

for (int i = 0; i < row; i++) {

num[i] = new int[column];

}

return num;

}

//矩阵初始化函数

void MatrixInit(int\*\*& num, int row, int column) {

for (int i = 0; i < row; i++) {

for (int j = 0; j < column; j++) {

cin >> num[i][j];

}

}

}

//矩阵输出函数

void PrintMatrix(int\*\* num, int row, int column) {

for (int i = 0; i < column; i++) {

for (int j = 0; j < row; j++) {

cout << num[i][j] << " ";

}

cout << endl;

}

}

//矩阵相加函数

void AddMatrix(int\*\* a, int\*\* b, int \*\*c,int row, int column) {

for (int i = 0; i < row; i++) {

for (int j = 0; j < column; j++) {

c[i][j] = a[i][j] + b[i][j];

}

}

}

//矩阵相减

void SubMatrix(int\*\* a, int\*\* b, int \*\*c,int row, int column) {

for (int i = 0; i < row; i++) {

for (int j = 0; j < column; j++) {

c[i][j] = a[i][j] - b[i][j];

}

}

}

//释放矩阵

void DeleteMatrix(int\*\* a, int row, int column) {

for (int i = 0; i < row; i++) {

delete[] a[i];

}

delete[] a;

}

int main() {

int \*\*A, \*\*B, \*\*C;

A = NewMatrix(4, 5);

B = NewMatrix(4, 5);

C = NewMatrix(4, 5);

MatrixInit(A, 4, 5);

MatrixInit(B, 4, 5);

AddMatrix(A, B, C,4, 5);

cout << "矩阵相加：" << endl;

PrintMatrix(C, 4, 5);

cout << "矩阵相减：" << endl;

SubMatrix(A, B,C, 4, 5);

PrintMatrix(C, 4, 5);

DeleteMatrix(A,4,5);

DeleteMatrix(B,4,5);

DeleteMatrix(C,4,5);

return 0;

}

### 测试

测试如下

图片包含 日历

AI 生成的内容可能不正确。

符合预期

### 实验心得

本实验设计了一个圆类（Circle），并允许用户通过输入圆心坐标和半径构造圆对象。主要难点在于​​默认构造的实现​​和​​对象生命周期的观察​​。

​​默认构造问题​​：最初的设计要求用户必须输入所有参数，但实际应用中可能需要默认值（如圆心在原点、半径为1）。通过​​默认参数​​（Circle(int x=0, int y=0, int r=1)）解决了这一问题，同时允许用户跳过部分输入（如输入-1时采用默认值）。

​​对象生命周期调试​​：在构造函数和析构函数中添加cout输出语句，观察到对象的创建和销毁顺序。例如，当Circle对象被销毁时，会先调用Circle的析构函数，再调用基类（如果有）的析构函数。

​​分析​​：

​​默认参数的权衡​​：虽然默认参数提高了灵活性，但过度使用可能导致代码可读性下降（如Circle(int x=0, int y=0, int r=1, int color=0)）。更好的做法是​​重载构造函数​​，提供多种初始化方式。

​​调试技巧​​：通过输出日志观察对象生命周期，是理解C++ RAII（资源获取即初始化）机制的有效手段。在复杂项目中，可以结合调试器（如GDB）更高效地跟踪对象状态。

## 类与对象实验2​

### 实践笔记

**难点**：赋值运算符重载​

需要深拷贝，且需要考虑自赋值值情况

​**解决方案**​：

1. 检查传入对象是否为自身，若是则直接返回
2. 否则释放原有空间，开辟新空间后赋值

**难点**：矩阵输出问题​

测试时发现未考虑到矩阵对齐输出

​**解决方案**​：

1. 包含iomanip头文件
2. 利用std::setw()函数设置输出的宽度为12

### 源代码

#include<iostream>

using namespace std;

class Matrix {

private:

int line, row;

int\*\* num;

public:

Matrix(int l = 0, int r = 0) :line(l), row(r) {

num = new int\* [l];

for (int i = 0; i < l; i++) {

num[i] = new int[r];

}

}

~Matrix() {

for (int i = 0; i < line; i++) {

delete[] num[i];

}

delete[] num;

}

Matrix(const Matrix& a) {

line = a.line;

row = a.row;

num = new int\* [line];

for (int i = 0; i < line; i++) {

num[i] = new int[row];

}

for (int i = 0; i < line; i++) {

for (int j = 0; j < row; j++) {

num[i][j] = a.num[i][j];

}

}

}

void getMatrix() {

for (int i = 0; i < line; i++) {

for (int j = 0; j < row; j++) {

cin >> num[i][j];

}

}

}

void outMatrix() {

for (int i = 0; i < line; i++) {

for (int j = 0; j < row; j++) {

cout << num[i][j] << " ";

}

cout << endl;

}

}

Matrix addMatrix(const Matrix& a) {

if (line != a.line || row != a.row) {

cerr << "矩阵大小不匹配，无法相加\n";

exit(114514);

}

Matrix temp(line,row);

for (int i = 0; i < line; i++) {

for (int j = 0; j < row; j++) {

temp.num[i][j] = num[i][j] + a.num[i][j];

}

}

return temp;

}

Matrix subMatrix(const Matrix& a) {

if (line != a.line || row != a.row) {

cerr << "矩阵大小不匹配，无法相减\n";

exit(114514);

}

Matrix temp(line, row);

for (int i = 0; i < line; i++) {

for (int j = 0; j < row; j++) {

temp.num[i][j] = num[i][j] - a.num[i][j];

}

}

return temp;

}

Matrix& operator=(const Matrix& a) {

if (this == &a) {

return \*this;

}

//释放旧元素

for (int i = 0; i < line; i++) {

delete[] this->num[i];

}

delete[] this->num;

line = a.line;

row = a.row;

num = new int\* [line];

for (int i = 0; i < line; i++) {

num[i] = new int[row];

for (int j = 0; j < row; j++) {

num[i][j] = a.num[i][j];

}

}

return \*this;

}

};

int main() {

Matrix A1(2,2), A2(2,2), A3(2,2);

A1.getMatrix();

A2.getMatrix();

A3 = A1.addMatrix(A2);

A3.outMatrix();

A3 = A1.subMatrix(A2);

A3.outMatrix();

Matrix \*pA1, \*pA2, \*pA3;

pA1 = new Matrix(2, 2);

pA2 = new Matrix(2, 2);

pA3 = new Matrix(2, 2);

pA1->getMatrix();

pA2->getMatrix();

\*pA3 = pA1->addMatrix(\*pA2);

pA3->outMatrix();

\*pA3 = pA1->subMatrix(\*pA2);

pA3->outMatrix();

delete pA1;

delete pA2;

delete pA3;

return 0;

}

### 测试

测试如下

图片包含 示意图

AI 生成的内容可能不正确。

符合预期

### 实验心得

本实验升级了矩阵运算，将其封装为Matrix类，并实现拷贝构造、赋值运算符重载和析构函数。关键问题包括：

​​深拷贝与浅拷贝​​：直接使用默认拷贝构造函数会导致多个对象共享同一块内存，引发双重释放（double-free）错误。通过手动实现深拷贝（new新内存并复制数据）解决了这一问题。

​​自赋值问题​​：在重载赋值运算符时，未检查a = a的情况，可能导致释放自身内存后访问非法数据。通过添加if(this == &a) return \*this;避免了这一风险。

​​分析​​：

​​三大件原则​​：当类涉及动态内存管理时，必须实现​​拷贝构造、赋值运算符和析构函数​​（即“三大件”），否则可能导致内存泄漏或未定义行为。

​​现代C++改进​​：使用std::vector替代原生数组可以避免手动内存管理，同时自动支持深拷贝，减少错误。

## 继承与派生实验

### 源代码

#include<iostream>

using namespace std;

#define PI 3.14

class Shape {

public:

Shape() {

cout << "Shape构造函数调用" << endl;

}

~Shape() {

cout << "Shape析构函数调用" << endl;

}

double Area() {

return 0;

}

};

class Rectangle :public Shape {

private:

int len, wid;

public:

Rectangle(int lenth=0,int width=0):len(lenth),wid(width) {

cout << "Rectangle构造函数调用" << endl;

}

~Rectangle() {

cout << "Rectangle析构函数调用" << endl;

}

double Area() {

return len \* wid;

}

};

class Circle :public Shape {

private:

int radius;

public:

Circle(int r=0) :radius(r) {

cout << "Circle构造函数调用" << endl;

}

~Circle() {

cout << "Circle析构函数调用" << endl;

}

double Area() {

return PI \* radius \* radius;

}

};

class Square :public Rectangle {

public:

Square(int l=0) :Rectangle(l, l) {

cout << "Square构造函数调用" << endl;

}

~Square() {

cout << "Square析构函数调用" << endl;

}

};

int main() {

Rectangle A(2,3);

cout << A.Area() << endl;

Circle B(1);

cout << B.Area() << endl;

Square C(2);

cout << C.Area() << endl;

return 0;

}

### 测试

测试如下

文本

AI 生成的内容可能不正确。

符合预期

### 实验心得

通过构建Shape基类和Rectangle、Circle、Square子类，理解了继承与多态的核心概念。

​​纯虚函数的作用​​：将Shape::Area()声明为纯虚函数（virtual double Area() = 0;），强制子类必须实现该方法。这比普通虚函数更严格，确保所有子类都有正确的行为。

​​构造与析构顺序​​：通过输出日志观察到，构造时先调用基类构造函数，再调用子类构造函数；析构时顺序相反。这验证了C++对象初始化的底层机制。

​​分析​​：

​​接口设计​​：基类定义接口（如Area()），子类实现具体逻辑，符合​​开闭原则​​（对扩展开放，对修改封闭）。

​​虚析构函数​​：如果基类指针指向子类对象（Shape\* s = new Circle()），必须将基类析构函数声明为虚函数，否则可能导致资源泄漏。

## I/O 流实验

### 实践笔记

**难点**：未增加用户输入超过规定范围的数据的处理情况​

​**解决方案**​：

1. 增加上下界变量，随着更新范围而更新
2. 当用户输入超过上下界的数字时提醒用户

### 源代码

#include<iostream>

#include<cstdlib>

#include<ctime>

using namespace std;

void solve() {

srand(time(NULL));

int num = (rand() % 1000) + 1;

int l=1, r=1000;

int tnum=0;

while (tnum != num) {

cout << "请输入"<<l<<"-"<<r<<"的数字：" << endl;

cin >> tnum;

if (tnum > num) {

r = tnum;

cout << "猜大啦！" << endl;

}

else if (tnum < num) {

l = tnum;

cout << "猜小啦！" << endl;

}

}

if (tnum == num) cout << "猜对啦！" << endl;

}

int main() {

solve();

return 0;

}

### 测试

测试如下

文本

AI 生成的内容可能不正确。

符合预期

### 实验心得

猜数字游戏实验涉及用户输入验证和随机数生成。

​​输入验证​​：通过动态调整提示范围（如“请输入500-750的数字”），引导用户快速逼近目标值。同时，检查输入是否为数字（cin.fail()），避免程序崩溃。

​​随机数问题​​：使用rand() % 1000生成随机数，但未设置种子时每次运行结果相同。通过srand(time(NULL))以时间戳为种子，提高了随机性。

​​分析​​：

​​随机数质量​​：rand()是伪随机数，不适合安全敏感场景（如密码生成）。C++11的<random>库（如std::mt19937）提供了更高质量的随机数。

​​用户体验优化​​：可以增加尝试次数限制（如最多10次），并在游戏结束时显示统计信息（如“共猜了7次”）。

## 重载实验1

### 源代码

#include<iostream>

using namespace std;

#define PI 3.14

class Shape {

public:

Shape() {

cout << "Shape构造函数调用" << endl;

}

~Shape() {

cout << "Shape析构函数调用" << endl;

}

virtual double Area() = 0;

//virtual double Area() {};

};

class Rectangle :public Shape {

private:

int len, wid;

public:

Rectangle(int lenth = 0, int width = 0) :len(lenth), wid(width) {

cout << "Rectangle构造函数调用" << endl;

}

~Rectangle() {

cout << "Rectangle析构函数调用" << endl;

}

double Area() {

return len \* wid;

}

};

class Circle :public Shape {

private:

int radius;

public:

Circle(int r = 0) :radius(r) {

cout << "Circle构造函数调用" << endl;

}

~Circle() {

cout << "Circle析构函数调用" << endl;

}

double Area() {

return PI \* radius \* radius;

}

};

class Square :public Rectangle {

public:

Square(int l = 0) :Rectangle(l, l) {

cout << "Square构造函数调用" << endl;

}

~Square() {

cout << "Square析构函数调用" << endl;

}

};

int main() {

Rectangle A(2, 3);

cout << A.Area() << endl;

Circle B(1);

cout << B.Area() << endl;

Square C(2);

cout << C.Area() << endl;

//Shape S;

return 0;

}

### 测试

测试如下

文本

AI 生成的内容可能不正确。

符合预期

### 实验心得

通过虚函数实验，对比了普通虚函数与纯虚函数的区别。

​​纯虚函数的约束力​​：当Shape::Area()改为纯虚函数后，尝试实例化Shape对象会导致编译错误，确保抽象类不能被误用。

​​多态的应用​​：通过基类指针Shape\*统一管理不同子类对象（如Circle和Rectangle），调用Area()时会自动执行子类实现，体现了运行时多态的优势。

​​分析​​：

​​设计模式基础​​：纯虚函数是​​工厂模式​​、​​策略模式​​等设计模式的基础，通过抽象接口解耦具体实现。

​​性能开销​​：虚函数通过虚表（vtable）实现，会引入少量运行时开销，但在大多数场景下可忽略不计。

## 重载实验2

### 源代码

#include<iostream>

using namespace std;

class Point {

private:

int x, y;

public:

Point(int xx=0, int yy=0) :x(xx), y(yy) {};

Point& operator++() {

x++;

y++;

return \*this;

}

Point operator++(int) {

Point temp = \*this;

++(\*this);

return temp;

}

Point& operator--() {

x--;

y--;

return \*this;

}

Point operator--(int) {

Point temp = \*this;

--(\*this);

return temp;

}

void print() {

cout << x << " " << y << endl;

}

};

int main() {

Point P(5, 5);

Point a;

a=P++;

a.print();

a = ++P;

a.print();

a = P--;

a.print();

a = --P;

a.print();

return 0;

}

### 测试

测试如下

文本

AI 生成的内容可能不正确。

符合预期

### 实验心得

通过重载Point类的前后置自增运算符，深入理解了运算符重载的细节。

​​前后置运算符的区别​​：

前置++p返回引用（Point&），直接修改原对象。

后置p++返回临时对象（Point），需先保存原状态再自增。

​​哑元参数的作用​​：后置运算符通过operator++(int)中的int参数与前置版本区分，这是C++语法规定的特殊设计。

​​分析​​：

​​返回值优化​​：前置运算符应返回引用以避免拷贝，后置运算符必须返回临时对象以保证语义正确。

​​运算符重载的滥用​​：虽然可以重载几乎所有运算符，但应遵循直觉（如++应用于自增，而非实现其他功能）。

# 综合题目——电商交易平台

## 题目简要介绍

本次实验任务是使用 C++ 语言，基于面向对象程序设计方法，设计并实现一个逐步完善功能的电商交易平台。从单机版的账户管理和商品管理子系统，到增加交易管理功能，再到网络版的实现，逐步满足用户注册登录、商品管理、交易等多样化需求。

对题目的需求理解为：通过分层设计和面向对象编程，构建一个功能完整、具备错误处理能力的电商平台。在实现过程中，注重数据持久化存储，利用文件操作实现用户和商品信息的保存；通过继承体系设计，提高代码的复用性和扩展性；同时，针对不同版本的特点，分别解决单机和网络环境下的功能实现与交互问题。

### 题目一：账户管理子系统和商品管理子系统（单机版）

题目一程序要求支持以下功能：

1) 用户注册&登录：支持新用户注册平台账号，已注册用户用平台账号登录平台，要求已注册用户的信息长久保留。

2) 修改账户密码：支持登录后对用户账号的密码修改。

3) 余额管理：支持用户账号中余额的查询、充值、消费等。

4) 添加商品：支持商家添加新商品，要求已添加的商品信息长久保留。

5) 展示平台商品信息：支持针对不同类型用户、无论登录与否均展示平台商品信息。

6) 搜索平台商品信息：支持依据某种条件（比如：名称）对平台商品进行筛选，并展示筛选结果。

7) 商品信息管理：支持商家对其商品的信息进行管理，包括但不限于价格管理、剩余量管理等。

主要要求如下：

● 要求 1：账户至少需要账号名、密码、账户余额、账户类型（商家/消费者）等内容。

● 要求 2：所有的用户账户信息应写入文件，注册新账户的时候，要求注册的账户名不能已经存在于文件中，即账户名唯一。

● 要求 3：至少设计一层继承体系（用户基类-用户子类）。设计一个用户基类，然后让商家类和消费者类等用户子类继承它，具体的用户是用户子类的实例对象（<<张三 >> 是消费者类的实例对象）。用户基类为抽象类，不能实例化，至少具有一个纯虚函数 getUserType()用于指示用户类别。

● 要求 4：电商平台上至少有三类商品：如食物、服装、图书等，每类商品中至少有三个具体的商品（如图书中可以有《C++ Primer》、《Effertive C++》等），每个具体的商品请至少包含商品描述、商品原价、商品剩余量等数据。所有的商品信息需要存储在文件或数据库中，不能写在代码中。

● 要求 5：至少设计一层继承体系（商品基类-商品子类）。设计一个商品基类，然后让图书类、电子商品类和服装类等商品子类继承它，具体的商品是商品子类的实例对象（<<C++ Primer >> 是图书类的实例对象）。商品基类至少具有一个虚函数 getPrice()用于计算具体商品的价格。

● 要求 6：支持对同一品类下所有产品打折的活动。

● 要求 7：支持一定的错误场景处理能力，例如读文件错误，输入数据不合法等等。

### 题目二：交易管理子系统（单机版）

在题目一的基础上支持在电商平台上购物的功能，实现以下功能：

1) 购物车管理：支持消费者向购物车添加、删除指定数量的商品，也支持消费者修改当前购物车商品的拟购买数量。

2) 订单生产：选择购物车的商品生成订单，计算并显示订单总金额。

3) 网上支付：消费者使用余额支付订单，支付成功后，消费者被扣除的余额应转至商家余额中。

主要要求如下：

● 要求 1：当订单生成后，处于未支付状态时，应将对应数量的商品冻结，不能参与新订单的产生，避免商品被超额售卖。

● 要求 2：支持一定的错误场景处理能力，如生成订单失败，支付失败等。

### 题目三：电商交易平台（网络版）

在题目一、二的基础上，将单机版电商交易系统修改为网络版。网络版要求实现如下功能：

1) 用户登录：用户通过客户端以账号密码登录平台。

2) 展示平台商品信息：通过客户端展示平台商品信息。

3) 搜索平台商品信息：通过客户端依据商品名称对平台商品进行搜索筛选，并展示筛选结果。

4) 购物车管理：通过客户端支持消费者向购物车添加、删除指定数量的商品。

5) 订单生产： 系统生成订单，通过客户端展示用户的订单信息。

6) 订单支付：用户通过客户端向系统提交订单支付的申请，并展示系统完成支付的状态。

主要要求如下：

● 要求 1：网络版需要实现的功能的要求与单机版要求一致。

● 要求 2：要求采用传统 CS 结构而非 BS 结构，客户端与服务器系统之间使用 socket 进行通信，不能使用 rpc 框架。

● 要求 3：支持一定的错误场景处理能力。

## 总体设计

本电商交易平台可划分为账户管理子系统、商品管理子系统、交易管理子系统（网络版还涉及客户端和服务器端交互）。各子系统之间相互协作，共同完成电商平台的各项功能。

账户管理子系统负责用户信息的管理，包括注册、登录、密码修改和余额管理等；商品管理子系统实现商品的添加、展示、搜索和信息管理；交易管理子系统则处理购物车操作、订单生成和支付流程。在网络版中，客户端负责与用户交互，向服务器端发送请求并展示结果，服务器端接收请求，处理业务逻辑并返回数据。

图示

AI 生成的内容可能不正确。

## 关键设计说明

### 继承体系设计

图形用户界面, 图示

AI 生成的内容可能不正确。

### 状态流转设计​

电脑屏幕截图

AI 生成的内容可能不正确。

### 异常处理

|  |  |
| --- | --- |
| 异常场景 | 处理方式 |
| 文件读写失败 | 错误日志 |
| 账户名重复 | 提示 |
| 库存不足 | 提示 |
| 余额不足 | 提示 |
| 网络中断 | 退出+错误日志 |
| 密码错误 | 提示 |
| 用户不存在 | 提示注册 |
| 输入金额不符合0.00格式 | 提示正确格式 |
| 输入的名字或密码的格式不正确 | 提示正确格式 |
| 商品重复或不存在 | 提示 |

## 详细设计

### 账户管理子系统详细设计

设计用户基类User，包含账号名username、密码password、账户余额balance、账户类型userType等成员变量，以及纯虚函数getUserType()用于指示用户类别。商家类Merchant和消费者类Consumer继承自User类，重写getUserType()函数。

|  |
| --- |
|  |

用户注册时，将新用户信息写入文件，同时检查用户名是否已存在；用户登录时，从文件读取信息进行验证；余额管理功能通过对balance变量的操作实现查询、充值和消费。

### 商品管理子系统详细设计

设计商品基类Product，包含商品描述description、商品原价originalPrice、商品剩余量quantity等成员变量，以及虚函数getPrice()用于计算具体商品价格。图书类Book、服装类Clothing、电子商品类Electronic等商品子类继承自Product类，重写getPrice()函数以实现不同的价格计算逻辑。

图示

AI 生成的内容可能不正确。商品添加时，将新商品信息写入文件；展示商品信息时，从文件读取并按要求展示；搜索商品通过遍历商品列表，依据条件筛选；商品信息管理则对商品的价格、剩余量等进行修改。

### 交易管理子系统详细设计

购物车管理通过ShoppingCart类实现，包含商品列表和商品数量信息，提供添加、删除和修改商品数量的方法。订单生成由Order类负责，根据购物车商品信息计算订单总金额，并将订单信息保存。支付功能在Payment类中实现，处理消费者余额扣除和商家余额增加操作，同时在订单生成后冻结对应商品数量，避免超额售卖。

|  |
| --- |
|  |

### 数据库说明

本系统使用csv文件存储用户账户信息和商品信息。用户账户信息文件以每行存储一个用户信息，格式为 “账号名 密码 余额 账户类型”；商品信息文件每行存储一个商品信息，格式为 “商品名称 商品描述 原价 折后价 剩余量 商品类型”。通过文件流操作实现数据的读写，保证数据的持久化。

### 接口协议说明（网络版）

在网络版中，客户端与服务器端使用 socket 进行通信。自定义协议数据单元格式为：[消息类型][数据长度][具体数据]。消息类型用于区分不同的请求（如登录请求、商品展示请求等），数据长度表示具体数据的字节数，以便服务器端准确接收数据。底层承载协议为 TCP 协议，保证数据传输的可靠性。

图示

AI 生成的内容可能不正确。

## 实现

### 主要问题与解决方案

#### 浮点数精度与输入验证​

**问题描述​**​：  
在电商交易系统中，涉及金额计算（如商品价格、余额充值、消费等）时，浮点数精度可能导致计算误差。此外，用户输入金额时可能输入非法字符或超出合理范围的值，影响系统稳定性。

​**​解决方案​**​：

​**​浮点数精度控制​**​：

使用 setprecision(2) 和 fixed 确保金额始终显示两位小数。

在关键计算（如订单总价、余额扣除）时，采用 double 类型存储，并避免多次累积计算以减少误差。

​**​输入验证​**​：

检查输入是否为数字（isdigit），并处理可能的负值或非数值输入。

限制小数位数（最多两位），防止用户输入过长小数导致溢出。

示例代码（UserBehavior::Recharge 方法）：  
bool isValid = true;

bool hasDecimal = false;

int decimalDigits = 0;

for (char c : input) {

if (c == '.') {

if (hasDecimal) isValid = false; *// 多个小数点*

hasDecimal = true;

} else if (c == '-' || !isdigit(c)) {

isValid = false; *// 非法字符*

} else if (hasDecimal && ++decimalDigits > 2) {

isValid = false; *// 超过两位小数*

}

}

if (!isValid) cout << "输入无效！请输入非负数且最多两位小数（如：100 或 50.25）\n";

#### 名字或密码字符串输入验证​

**问题描述​**​：  
用户名和密码可能包含空格或为空，导致文件存储格式错误或登录验证失败。

​**​解决方案​**​：

​**​输入过滤​**​：

使用 getline(cin, str) 读取整行输入，检查是否包含空格（str.find(' ') != string::npos）。

禁止空输入（str.length() == 0）。

示例代码（UserBehavior::Sign\_Up 方法）：

if (name.length() == 0 || name.find(' ') != string::npos) {

cout << "用户名不能为空或包含空格！\n";

return;

}

**唯一性检查​​：**

注册时通过 Usersheet::findUsername 检查用户名是否已存在。

#### ​单机版搜索商品输出冗余信息

**问题描述​**​：  
搜索商品时，若未找到结果，系统仍会输出表格头部信息，用户体验不佳。

​**​解决方案​**​：

​**​结果预检查​**​：

在 Products::chooseSearchMode 中，先检查 pname\_key 或 type\_key 是否包含目标键值，若无则直接提示“无搜索结果”。

if (pname\_key.find(spname) == pname\_key.end()) {

cout << "无搜索结果……\n";

return;

}

**优化输出格式​​：**

仅在找到商品时打印表头和详细信息，避免冗余输出

### 核心收获

* **1. ​​面向对象编程思想的深入理解​​**

通过本次实验，我深刻掌握了面向对象编程的三大核心特性：

* ​**​封装性​**​：将矩阵运算、用户账户、商品管理等功能封装成独立的类（如Matrix、Account、Products），隐藏内部实现细节，仅暴露必要接口。
* ​**​继承与多态​**​：构建了Shape基类与Rectangle/Circle子类的继承体系，并通过虚函数（如Area()）实现运行时多态，支持不同子类的差异化行为。
* ​**​抽象与接口设计​**​：通过纯虚函数（如getUserType()）定义抽象基类，强制子类实现统一接口，提升代码扩展性。
* **2. ​​C++核心技术的实践应用​​**
* ​**​动态内存管理​**​：手动实现矩阵的new/delete操作，理解深拷贝与浅拷贝的区别，并通过"三大件"（构造、拷贝、析构）避免内存泄漏。
* ​**​运算符重载​**​：为Point类重载++运算符，掌握前后置运算符的差异及哑元参数的作用。
* ​**​STL容器应用​**​：使用unordered\_map管理用户数据、vector存储商品列表，熟悉了STL的高效用法。
* **3. ​​工程化开发能力的提升​​**
* ​**​防御性编程​**​：对所有用户输入（如金额、密码）进行严格验证，确保系统健壮性。例如，通过正则表达式检查金额格式（^\d+(\.\d{1,2})?$）。
* ​**​数据持久化​**​：采用CSV文件存储用户和商品信息，实现readInInformation()和outMessege()方法完成数据读写，理解文件IO操作的关键点。
* ​**​异常处理机制​**​：针对文件读写失败、库存不足等场景设计错误提示，如cerr << "矩阵大小不匹配"并配合exit(114514)终止程序。
* **4. ​​调试与优化经验​​**
* ​**​调试技巧​**​：通过在构造函数/析构函数中添加日志输出（如cout << "Shape构造"），直观跟踪对象生命周期和资源管理。
* ​**​性能优化​**​：发现矩阵输出对齐问题后，引入<iomanip>的setw(12)优化显示格式；通过B+树索引（实验扩展部分）提升商品搜索效率。
* **5. ​​电商系统的完整设计能力​​**
* ​**​模块化设计​**​：将系统拆分为账户管理、商品管理、交易管理三大子系统，符合"高内聚低耦合"原则。
* ​**​设计模式应用​**​：使用工厂模式（通过UserBehavior派生不同用户类型）和策略模式（商品打折活动），提升代码复用性。
* ​**​网络通信基础​**​：设计基于Socket的CS架构协议（[消息类型][数据长度][具体数据]），为网络版扩展奠定基础。
* **6. ​​工程规范意识​​**
* ​**​代码可读性​**​：通过有意义的变量命名（如row/col替代i/j）、关键逻辑注释，避免矩阵行列混淆等问题。
* ​**​现代C++实践​**​：认识到std::vector和智能指针（如unique\_ptr）相比原生指针的优势，为后续改进指明方向。