本 科 生 课 程 设 计

实验报告

课 程 名 称 ： 面向对象程序设计（C++）

姓　　名 ： 叶淳瑜

学　　号 ： 2019212686

学　　院 ： 计算机学院

专　　业 ： 计算机类

班　　级 ： 2019211318

指 导 教 师 ： 胡博老师

二〇二一年 六月

目录

一．基础实验 4

1.1 C++基础知识实验 4

1.2 类与对象实验 6

1.2.1 6

1.2.2 8

1.3 继承与派生实验 11

1.4 I/O 流实验 13

1.5 重载实验 14

1.5.1 虚函数 14

1.5.2 运算符重载 16

二．综合实验 17

2.1 实验概述 17

2.1.1 实验设计目标 17

2.1.2 实验要求 17

2.1.3 编写环境 17

2.2 题目一 18

2.2.1 题目要求 18

2.2.2 系统整体设计 19

2.3.1 程序模块细节 20

2.3 题目二 24

2.3.1 题目要求 24

2.3.2 具体设计 25

2.4 题目三 26

2.4.1 题目要求 26

2.4.2 系统整体设计 27

2.4.3 Socket 通信实现 28

三．实验总结 29

3.1 优点与不足.....29
3.1.1 优点.....29
3.1.2 不足.....29
3.2 问题总结与心得体会.....29
3.3 课程建议.....30

一. 基础实验

1.1 C++基础知识实验

本实验要求编写 C++程序完成“矩阵”以下功能：

1. 定义大小为 4×5 的矩阵，矩阵空间采用 new 动态申请，保存在指针中；

```
#define ROW 4
#define COLUMN 5

//Matrix Creation
int **creat()
{
    int **p = new int*[ROW];
    if(!p)
    {
        cout << "Error: out of memory." <<endl;
        exit(1);
    }
    for (int i = 0; i < COLUMN;i++)
    {
        p[i] = new int[COLUMN];
        if(!p[i])
        {
            cout << "Error: out of memory." <<endl;
            exit(1);
        }
    }
    return p;
}
```

对于二维数组，从外到里依次申请内存空间，并在申请空间的时候，使用适当的差错处理，防止内存超限。

2. 定义矩阵初始化函数，可以从 cin 中输入矩阵元素；并且可以通过输出函数将矩阵格式化输出到 cout；

```
void init(int **p)
{
    for (int i = 0; i < ROW;i++)
        for (int j = 0; j < COLUMN;j++)
            cin >> p[i][j];
```

```
}
void print(int **p)
{
    int i, j;
    for (i = 0; i < ROW; i++)
    {
        for (j = 0; j < COLUMN-1;j++)
        {
            cout << p[i][j] << "\t";
        }
        cout << p[i][j] << endl;
    }
}
```

3. 定义矩阵相加、相减的函数，实现两个矩阵运算的功能，结果保存在另一个矩阵中;

```
void addition(int **a, int **b, int **c)
{
    for (int i = 0; i < ROW; i++)
        for (int j = 0; j < COLUMN; j++)
            c[i][j] = a[i][j] + b[i][j];
}
```

4. 定义函数释放矩阵空间。

```
void destroy(int **p)
{
    for (int i = 0; i < COLUMN;i++)
    {
        delete[] p[i];
    }
    delete[] p;
    cout << "Matrix destruction completed." << endl;
}
```

对于二维数组，从里到外依次申请释放空间。

5. 初始化 3 个矩阵并完成相应运算，输出效果如图：

```
Matrix creation completed.
Matrix creation completed.
Matrix creation completed.
please enter Matrix A(4x5):
1 1 1 1 1
2 2 2 2 2
3 3 3 5 4
1 1 1 1 1
please enter Matrix B(4x5):
9 9 9 9 9
1 1 1 1 1
0 0 0 0 0
9 9 9 9 9
The result of A + B is:
10      10      10      10      10
3       3       3       3       3
3       3       3       5       4
10      10      10      10      10
The result of A - B is:
-8      -8      -8      -8      -8
1       1       1       1       1
3       3       3       5       4
-8      -8      -8      -8      -8
```

1.2 类与对象实验

1.2.1

本实验要求编写 C++程序完成“圆形”以下功能：

1. 定义一个 Point 类，其属性包括点的坐标，提供计算两点之间距离的方法。

```
class Point
{
private:
    double x, y;

public:
    Point(double a = 0, double b = 0);
    ~Point();
    void setPoint(double a = 0, double b = 0);
    double getDistance(Point p);
    void print();
};
```

通过设置函数参数来达到默认参数的功能，计算距离的函数实现如下：

```
double Point::getDistance(Point p)
{
    double distance = 0;
    distance = sqrt((x - p.x) * (x - p.x) + (y - p.y) *(y - p.y));
    return distance;
}
```

2. 定义一个圆形类，其属性包括圆心和半径；

```
class Circle
{
private:
    Point center;
    double radius;

public:
    Circle();
    ~Circle();
    void setParameter(Point c, double r);
    double getRadius();
    double calDistance(Circle circle2);
};
```

Circle 类嵌套了 Point 类，并且实现了计算两个圆的圆心坐标的方法。

3. 判断两圆的位置关系

```
    double sum = circle1.getRadius() + circle2.getRadius();
    double sub = fabs(circle1.getRadius() - circle2.getRadius());
    double distance = circle1.calDistance(circle2);
    //Judge and output the result
    if (distance>sum && fabs(sum-distance)>esp)
        cout << "Two circles lying outside each other." << endl;
    else if (fabs(distance-sum)<esp)
        cout << "Two circles touching each other externally." << endl;
    else if (fabs(distance-sub)<esp)
        cout << "Two circles touching each other internally." << endl;
    else if (distance<sub && fabs(sub-distance)>esp)
        cout << "One circle lying inside another." << endl;
    else
        cout << "Two circles intersecting each other." << endl;
```

4. 创建两个圆形对象，提示用户输入圆心坐标和半径，判断两个圆是否相交，并输出结果如下：

```
The point (0,0) has been created.
Creation of class: Circle.
The point (0,0) has been created.
Creation of class: Circle.
Please enter the coordinates of the center of Circle 1:
0 0
Please enter the radius of Circle 1:
2
The point (0,0) has been created.
A circle with the center at (0,0) and the radius of 2 has been created.
Destruction of class: Point.
请按任意键继续. . .
Please enter the coordinates of the center of Circle 2:
1 1
Please enter the radius of Circle 2:
3
The point (1,1) has been created.
A circle with the center at (1,1) and the radius of 3 has been created.
Destruction of class: Point.
请按任意键继续. . .
Destruction of class: Point.
Destruction of class: Circle.
Destruction of class: Point.
Two circles intersecting each other.
请按任意键继续. . .
Destruction of class: Point.
Destruction of class: Point.
Destruction of class: Circle.
Destruction of class: Point.
Destruction of class: Circle.
Destruction of class: Point.
```

5. 观察圆形对象以及 Point 类成员的构造函数与析构函数的调用。

如图所示，先调用成员对象 point 类的构造函数，再调用封闭类 circle 类的构造函数。当程序结束时，则先调用 circle 类的析构函数，再调用成员对象 point 类的析构函数。即是，当实例化嵌套的类时，会由里及外进行构造，当释放内存时，会由外及里进行析构。

1.2.2

本实验要求编写 C++程序完成“矩阵”类以下功能：

1. 定义一个矩阵类，包含矩阵的参数以及指针。包括构造函数、析构函数输入输出等基本方法。

```
class Matrix
{
private:
    int rows, lines;
    int **head;

public:
    Matrix(int r, int l);       //Constructor
    Matrix(const Matrix &obj); //Copy Constructor
```

```
    ~Matrix();                  //Destructor

    void inputMatrix();
    void outputMatrix();
    Matrix addMatrix(Matrix m);
    Matrix subMatrix(Matrix m);
    Matrix &operator=(const Matrix &m);
};
```

2. 实现拷贝构造函数

```
Matrix::Matrix(const Matrix &m)
{
    rows = m.rows;
    lines = m.lines;
    head = new int *[rows];
    for (int i = 0; i < rows; i++)
    {
        head[i] = new int[lines];
    }
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < lines; j++)
        {
            head[i][j] = m.head[i][j];
        }
    }
    cout << "copy constructor." << endl;
}
```

拷贝构造函数在以下三种情况下被调用：

- 当用一个对象去初始化同类的另一个对象时。
- 如果某函数有一个参数是类的对象。
- 如果函数的返回值是类的对象时。

3. 实现矩阵的加减

```
Matrix Matrix::addMatrix(Matrix m)
{
    Matrix tmp(rows, lines);
    int i, j;
```

```
    for (i = 0; i < rows; i++)
    {
        for (j = 0; j < lines; j++)
        {
            tmp.head[i][j] = head[i][j] + m.head[i][j];
        }
    }
    cout << "Addtion completed." << endl;
    return tmp;
}
```

4. 实现“=”的重载

```
Matrix &Matrix::operator=(const Matrix &m)
{
    //avoid self-assignment
    if (&m != this)
    {
        //delete old elements
        for (int i = 0; i < rows && head != NULL; i++)
        {
            delete[] head[i];
        }
        delete[] head;

        //value assignment
        head = new int *[rows];
        for (int i = 0; i < rows; i++)
        {
            head[i] = new int[lines];
        }
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < lines; j++)
            {
                head[i][j] = m.head[i][j];
            }
        }
    }
    cout << "use overloaded operator \"=\"." << endl;
    return *this;
}
```

首先，重载的返回值是引用类型的，这是为了保证其计算结果可以作为左值，即实现连等的功能。其次，如果类里面含有指针指向动态分配的资源的话，那么自身赋值就可能出错有可能导致把自己给释放了，所以要通过判断是否是 this 来避免自赋值。最后，要记得释放旧元素。

5. 创建三个矩阵，进行运算并输出结果，最后释放内存。输出如下：

```
Please enter the number of rows and line of matrices:
2 2
A 2x2 matrix has been created.
A 2x2 matrix has been created.
A 2x2 matrix has been created.
Please enter the elements of Matrix A1:
1 1 1 1
Input completed.
Please enter the elements of Matrix A2:
3 2 1 2
Input completed.
请按任意键继续. . .
copy constructor.
A 2x2 matrix has been created.
Addtion completed.
use overloaded operator "=".
A3 = A1 + A2
4       3
2       3
```

1.3 继承与派生实验

本实验要求编写 C++程序完成“形状”的以下功能：

1. 声明一个基类 Shape（形状），其中包含一个方法来计算面积

```
class Shape
{
public:
    Shape();
    ~Shape();

    double getArea();
};
```

2. 从 Shape 派生两个类：矩形类和圆形类，从矩形类派生正方形类

```
class Circle : public Shape
{
public:
    Circle(double r = 0);
    ~Circle();
```

```
    void setRadius(double r);
    double getArea();

private:
    double radius;
};

class Rectangle : public Shape
{
public:
    Rectangle(double len = 0, double wid = 0);
    ~Rectangle();
    void setConfig(double len, double wid);
    double getArea();

private:
    double length, width;
};

class Square : public Rectangle
{
public:
    Square(double len = 0);
    ~Square();
};
```

3. 分别计算面积

```
Constructor of Class: Shape.
The area of shape is 0
请按任意键继续. . .
Constructor of Class: Shape.
Constructor of Class: Circle with radius:4
The area of circle is 50.2655
请按任意键继续. . .
Constructor of Class: Shape.
Constructor of Class: Rectangle with length:8 width:16
The area of rectangle is 128
请按任意键继续. . .
Constructor of Class: Shape.
Constructor of Class: Rectangle with length:32 width:32
Constructor of Class: Square with length:32
The area of square is 1024
请按任意键继续. . .
Destructor of Class: Shape.
Destructor of Class: Circle.
Destructor of Class: Shape.
Destructor of Class: Rectangle.
Destructor of Class: Shape.
Destructor of Class: Square.
Destructor of Class: Rectangle.
Destructor of Class: Shape.
```

4. 观察派生类构造函数、析构函数输出顺序。

从程序执行结果可以看出，在执行一个派生类的构造函数之前，总是先执行基类的构造函数。而在析构时，先执行派生类的析构函数，再调用基类的析构函数。

值得注意的是由于此时没有引入虚函数，所以计算每一个对象的面积时都要调用分别不同类的指针，而不能实现多态，十分不方便，这就显示出下文虚函数的重要性。

```
Shape *s = new Shape();
cout << "The area of shape is " << s->getArea() << endl;
system("pause");

Circle *c = new Circle(4);
cout << "The area of circle is " << c->getArea() << endl;
system("pause");

Rectangle *rec = new Rectangle(8, 16);
cout << "The area of rectangle is " << rec->getArea() << end
system("pause");

Square *sq = new Square(32);
cout << "The area of square is " << sq->getArea() << endl;
system("pause");
```

1.4 I/O 流实验

本实验要求编写 C++程序完成猜价格游戏的以下功能：

1. 假定有一件商品，程序用随机数指定该商品的价格（1-1000 的整数）

```
int  price;
srand(unsigned(time(NULL)));
price = rand() % 1000 + 1;
```

注意：

- rand()产生的是伪随机数字，每次执行时是相同的，若要不同，用函数 srand()初始化它。
- 使用模运算保证价格范围。

2. 提示用户猜价格，并输入：若用户猜的价格比商品价格高或低，对用户作出相应的提示。直到猜对为止，并给出提示。

```
Please enter an integer x (1 <= x <= 1000) :
2
Your guess (2) is smaller than the right price.
9
Your guess (9) is smaller than the right price.
500
Your price (500) is bigger than the right price.
```

3. 注意输入不合法的异常处理

```
try
{
    if (x < MIN || x > MAX || cin.fail())
    {
        throw "Your input is illegal!";
    }
}
catch (const char *msg)
{
    cin.clear(); //reset failbit
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    cout << msg << endl;
    continue;
}
```

当用户输入不合法时（非数字，数字范围不适），程序先抛出异常，通过 catch 接收，先调用 cin.ignore 函数忽略此前用户输入的不合法输入，再使用 continue 语句直接进入下一次 while 循环，提示用户再次输入。具体表现如下：

```
Your input is illegal!
asdf
Your input is illegal!
0
Your input is illegal!
80000
Your input is illegal!
```

1.5 重载实验

1.5.1 虚函数

本实验要求针对题目 3 的“形状”类，编写 C++程序完成以下功能：

1. 将【形状】 中的基类计算面积的方法定义为虚函数，比较与【形状（A）】程序的差异

```
class Shape
{
public:
    Shape();
    ~Shape();
    virtual double getArea();
};
```

```
class Circle : public Shape
{
public:
    Circle(double r = 0);
    ~Circle();
    void setRadius(double r);
    double getArea();

private:
    double radius;
};
```

在派生类中重新定义基类中定义的虚函数时，virtual 关键字会告诉编译器不要静态链接到该函数。我们想要的是在程序中任意点可以根据所调用的对象类型来选择调用的函数，这种操作被称为动态链接，或后期绑定。这样我们就不用像实验 3 那样再使用单个的类的指针，而是而是直接通过指向派生类的基类指针来调用成员函数 getArea 获得面积。

2. 将【形状】中的基类定义为抽象类，比较与【形状（A）】程序的差异

```
class Shape
{
public:
    Shape();
    ~Shape();
    virtual double getArea() = 0;
};

class Circle : public Shape
{
public:
    Circle(double r = 0);
    ~Circle();
    void setRadius(double r);
    double getArea();

private:
    double radius;
};
```

虚函数固然方便，在很多情况下，基类本身生成某些对象是不合情理的。所以引入纯虚函数的概念，将函数定义为纯虚函数，则编译器要求在派生类中必须予以重写以实现多态性。

同时，含有纯虚拟函数的类称为抽象类，它不能被实例化，即不能生成对象。这样就很好地解决了上述问题。

1.5.2 运算符重载

本实验要求对 Point 类重载++和--运算符，编写 C++程序完成以下功能：

1. 实现前置++，--

```
Point &Point::operator++()
{
    this->x++;
    this->y++;
    return *this;
}

Point &Point::operator--()
{
    this->x--;
    this->y--;
    return *this;
}
```

2. 实现后置++，--

```
Point Point::operator++(int)
{
    Point temp = *this;
    this->x++;
    this->y++;
    return temp;
}

Point Point::operator--(int)
{
    Point temp = *this; //copy constructor
    this->x--;
    this->y--;
    return temp;
}
```

注意：

- C++规定，后置运算符重载要多写一个 int 类型的参数。

- 和赋值运算符一样，前置的++，--为了不改变符号的特性，即可以作为左值，所以返回的是引用类型。

3. 为了方便输出，实验者还重载了<<操作符

```
ostream &operator<<(ostream &out, const Point &p)
{
    out << "(" << p.x << "," << p.y << ")";
    return out;
}
```

二. 综合实验

2.1 实验概述

2.1.1 实验设计目标

本次 C++综合设计实验要求实验者使用 C++语言，基于面向对象的程序设计方法，设计并实现一个简单的电商交易平台，提供用户管理、商品管理、交易管理等功能。

本作业包含三个题目，使得该电商交易平台功能逐步增强。前两个题目为单机版，运行时体现为一个进程。第三个题目为网络版，要求采用 CS 结构，客户端和服务器端为不同的进程。

2.1.2 实验要求

程序设计要求

1. 如有必要的友元函数，需说明每个友元函数的不可替代性。
2. 除主函数和必要的友元函数外，不允许出现任何一个非类成员函数。
3. 任何不改变对象状态的成员函数均需显式标注 const。

代码规范性要求

1. 代码需遵循课件提出的编码规范要求。
2. 通过开发环境自动生成的界面类代码，全部数据成员和成员函数需在类声明时加以注释，函数体内的必要步骤要加以注释。
3. 其他全部类代码的数据成员和成员函数的声明和实现均需加以注释，成员函数的必要步骤要加以注释。

2.1.3 编写环境

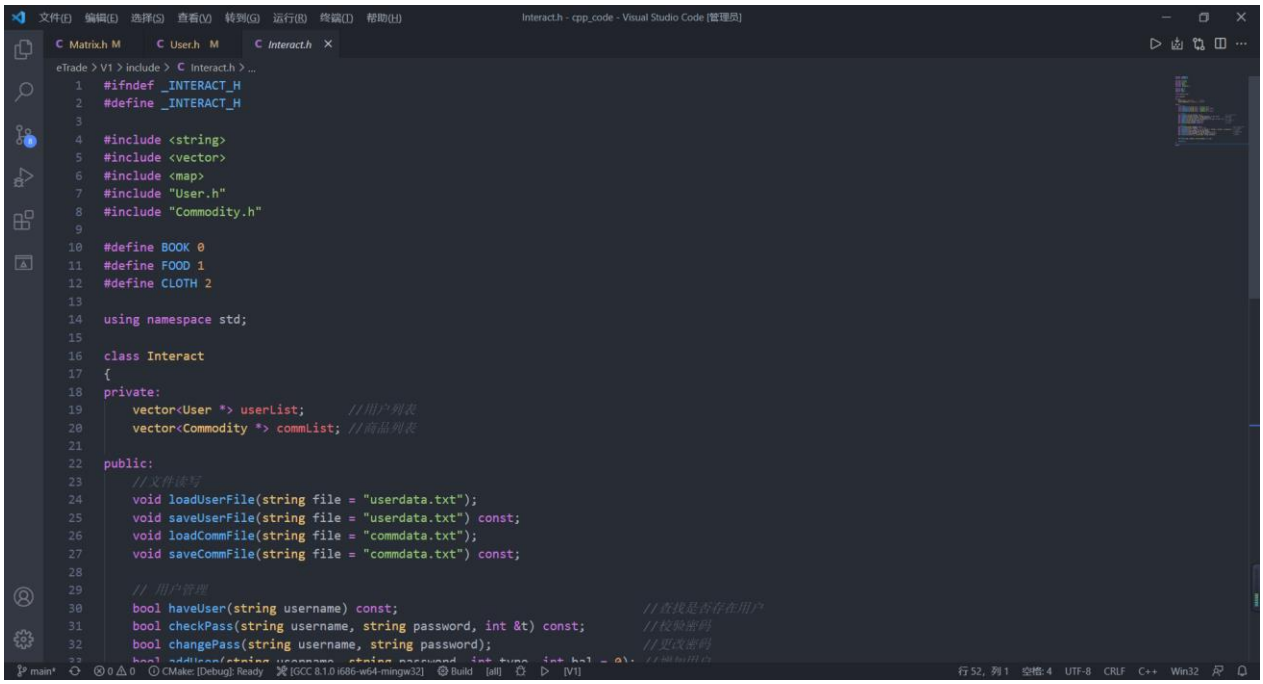
操作系统：Windows10

开发平台：Visual Studio Code

编程语言及标准：C++11

编译器：MinGW

编译工具：CMake



2.2 题目一

2.2.1 题目要求

具体功能：

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，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，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。

要求：

1. 账户至少包含账号名、密码、账户余额、账户类型（商家/消费者）等信息。且要求账户名唯一。

2. 所有的用户账户信息需写入文件。

3. 至少设计一层继承体系。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。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。

4. 电商平台上至少有三类商品，每类商品中至少有三个具体的商品，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。

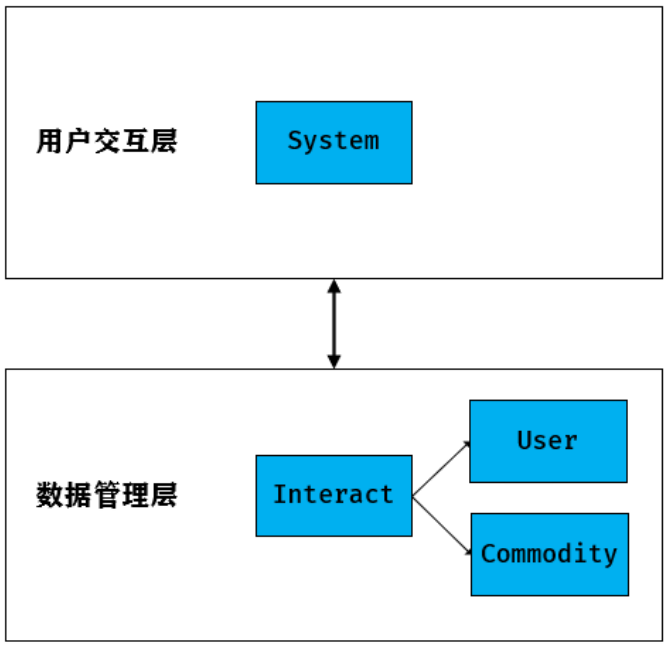
5. 至少设计一层继承体系。设计一个商品基类，让商品子类继承它。商品基类至少具有一个虚函数 getPrice()用于计算具体商品的价格。

6. 支持对同一品类下所有产品打折的活动。

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。

2.2.2 系统整体设计

实验者针对题目一进行分析后设计了四个类，分别为：User（用户类）、Commodity（商品类）、Interact（交互类）以及 System（IO系统类），如下图所示。



其中，System类负责展示用户界面，接收用户命令，输出执行结果等用户交互工作。User类和Commodity类分别是用户和商品信息的最小数据存储单元。Interact类是前端和后端的接口，设置有存储用户与商品信息的数据结构，能够从前端接受命令，并且返回执行结果。此外，其还有读写文件，维护数据等功能。

2.3.1 程序模块细节

System

System 类负责展示用户界面，接收用户命令，输出执行结果等用户交互等工作。同时记录了当前电商平台的信息（当前用户信息，是否登录）

```
class System
{
private:
    string username; //当前用户名
    string password; //当前用户密码
    int type;        //当前用户类型
    bool logged;     //是否登录

    Interact *interact;

public:
    int getType() const { return type; }
    string getName() const { return username; }
    void setName(string n) { username = n; }
    string getPassword() const { return password; }
    void setPassword(string pwd) { password = pwd; }
    int exeCommand(); //执行命令

    System(string un, string pwd);
};
```

在读取用户命令的时候，程序用C++中STL库中的map记录用户命令格式，并用switch语句来循环读取指令，执行指令。部分命令如下：

```
cmdList["help"] = cmdVal::help;
cmdList["sign_up"] = cmdVal::sign_up;
cmdList["sign_in"] = cmdVal::sign_in;
cmdList["list_user"] = cmdVal::list_user;
cmdList["list_comm"] = cmdVal::list_comm;
cmdList["search_comm"] = cmdVal::search_comm;
cmdList["quit"] = cmdVal::quit;
cmdList["sign_out"] = cmdVal::sign_out;
```

输出格式如下：

```
> help
Commands are as below :
--------------------System--------------------
help : helpful information
sign_up : user sign up
sign_in : user sign in
sign_out : user sign out
change_pass : user change password
list_user : list the info of your acount
list_comm : list all commodity
search_comm : search for commodity
quit : quit the eTrade system
-------------------Consumer-------------------
charge : consumer recharge balance
buy : consumer buy commodity
-------------------Merchant-------------------
add_comm : merchant add commodity
change_quantity : change quantity of commodity
change_price : change price of commodity
change_discout : change discount of commodity
----------------------------------------------
```

User 类

User类是用户信息的最小数据存储单元，储存了用户名、密码、余额、用户类型等信息，为了实现买家和卖家的不同用户类型，设置Merchant、Consumer子类，User作为基类被其公有继承。具体实现如下：

```
class User
{
private:
    string username;
    string password;
    double balance;

public:
    int type;
    virtual int getUserType() const = 0;            //获得用户类型
    double getBalance() const { return balance; }   //获得余额
    void setBalance(double b) { balance = b; }      //设置余额
```

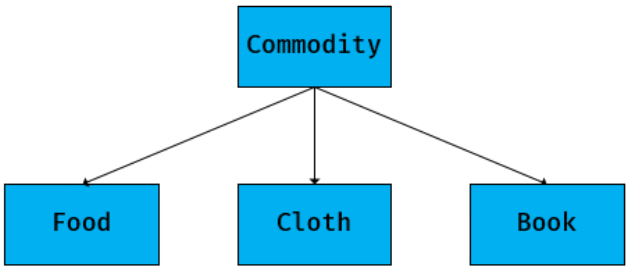
```
    string getName() const { return username; }     //获得用户名
    void setName(string nm) { username = nm; }      //设置用户名
    string getPassword() const { return password; } //获得密码
    void setPassword(string pw) { password = pw; }  //设置密码
    User(string n, string pw, double b);            //构造函数
    virtual ~User() {}
};

class Consumer : public User
{
public:
    int getUserType() const { return type; } //获得用户类型
    Consumer(string n, string pw, double b);
};

class Merchant : public User
{
public:
    int getUserType() const { return type; } //获得用户类型
    Merchant(string n, string pw, double b);
};
```

值得注意的是由于各种用户分类的不同，为了实现多态，实验者将getUserType函数设置为纯虚函数，方便使用基类指针进行访问。

Commodity 类

Commodity类与User类类似，作为商品信息的最小储存单位，储存了商品名、所有者、数量、价格、折扣、描述等基本信息。为了实现商品的分类，设置food、cloth、book子类，公有继承Commodity类，在三个子类中分别存储相应的折扣。

其中，值得注意的是，由于各种商品分类的折扣不同，为了实现多态，实验者将getDiscount、setDiscount都设置为纯虚函数，方便使用基类指针进行访问。



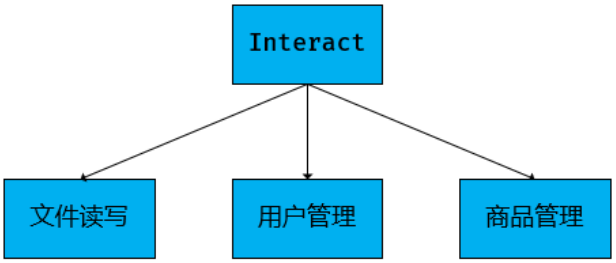
Interact 类

Interact类是前端和后端的接口，设置有存储用户与商品信息的数据结构，能够从前端接受命令，并且返回执行结果。此外，其还有读写文件，维护数据等功能。

数据存储在 STL 库的动态大小数组的顺序容器 vector 中，便于实现数据的动态增减，数据类型分别是基类指针。由于在实现 User 和 Commodity 的时候，我们已经使用纯虚函数实现了多态，于是这里我们可以很容易地通过基类指针访问子类函数。

```
class Interact
{
private:
    vector<User *> userList;       //用户列表
    vector<Commodity *> commList; //商品列表

public:
    //文件读写
    …

    // 用户管理
    …

    // 商品管理
    …

    // 购物
    bool buy(string userName, string commName, int num);

    ~Interact();
};
```

Interact类分别实现了文件读写、用户管理、商品管理三个功能。



用户与商品数据永久储存在文件中。分别按照给定顺序存储数据。

eTrade > commdata.txt

```
T-shirt cloth test3 135.8 1 1100 for_Summer_Day
skirt cloth test3 220 1 1234 for-beautiful-girls
pajamas cloth test3 123 1 2233 love-sleeping
Cocacola food hyl 2.5 1 12345 happy-water
hotpot food hyl 102.2 1 1223 beat-sichuan
bread food hyl 3.8 1 203 simple-but-useful
CSAPP book ycy 123.2 0.5 1234 beast-book
CPrimer book ycy 1.8 0.5 20000 to-read
ThreeBody book ycy 12 0.5 1300 best-sci
```

eTrade > userdata.txt

```
ycy 123 1 1000
hyl 123 1 222
test1 123 1 0
test2 666 0 999
test3 233 1 100000
```

用户管理与商品管理均通过改变vector来实现。所有函数的定义如下：

```
//文件读写
void loadUserFile(string file = "userdata.txt");
void saveUserFile(string file = "userdata.txt") const;
void loadCommFile(string file = "commdata.txt");
void saveCommFile(string file = "commdata.txt") const;

// 用户管理
bool haveUser(string username) const;                                    //查找是否存在用户
bool checkPass(string username, string password, int &t) const;          //校验密码
bool changePass(string username, string password);                       //更改密码
bool addUser(string username, string password, int type, int bal = 0); //增加用户
bool getUserInfo(string username) const;                                 //获得用户信息
bool addBal(string userName, double b);                                  //增加余额
bool deBal(string userName, double b);                                   //减少余额

// 商品管理
bool haveComm(string commName) const;                                                    //查找是否存在商品
bool listComm(string commName = "*") const;                                              //列出商品信息
bool addComm(double p, double d, int q, string n, string o, string t, string des); //添加新商品
bool delComm(string commName, string userName);                                          //删除商品
bool setQuantity(string commName, string userName, int q);                               //更改商品数量
bool setPrice(string commName, string userName, double p);                               //设置价格
bool setDiscount(double p, string type, string userName);                                //设置折扣

// 购物
bool buy(string userName, string commName, int num);
```

2.3 题目二

2.3.1 题目要求

具体功能：

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。

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。

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。

2.3.2 具体设计

相对于题目一，题目二要求增加的功能只有购物车功能与订单功能。以下是具体的实现。

在 User 类的 Consumer 子类中设置 cart 与 order 变量，数据类型都是 vector<pair<string, int>>，pair 是将 int 和 string 类型的数据整合成一个二元组，形成一一对应的关系。在购物车与订单中，分别存储了商品名称与商品数量。并且使用 vector 动态调整商品种类。每个买家都有一个对应的 cart 与 order，这样就能实现对每个买家的购物车与订单的操作。

```
class Consumer : public User
{
public:
    vector<pair<string, int>> cart;  //购物车
    vector<pair<string, int>> order; //订单列表
    double bill = 0;                 //订单金额

    void addCart(string commName, int amount); //加入购物车
    int getUserType() const { return type; }   //获得用户类型
    Consumer(string n, string pw, double b);
};
```

在 Interact 类中，增加对购物车与订单的相应操作。

首先，为了保证用户数据不丢失，实验者将数据存在本地文件中，通过函数进行读写操作。

```
void loadCartFile(string file = "cartdata.txt");
void saveCartFile(string file = "cartdata.txt") const;
```

其次，新增订单与购物车管理函数。

```
//购物
bool buy(string userName, string commName, int num);        //购买单个物品
bool addCart(string userName, string commName, int q);      //添加购物车
bool delCart(string userName, string commName, int q = 0); //删除购物车
bool changeCart(string userName, string commName, int q);  //更改购物车
void listCart(string userName) const;                       //输出购物车信息
void clearCart(string userName);                            //清空购物车
bool payOrder(string userName);                             //支付订单
bool creatOrder(string userName);                           //生成订单
void listOrder(string userName);                            //输出订单信息
bool delOrder(string userName);                             //删除订单
```

此处，实验者遇到了一个难题，就是 cart 与 order 都是 Consumer 类特有的，而 Interact 中存储的是 User 基类的指针，常规方法无法访问到 Consumer 的这两个数据。实验者查阅资料后发现可以通过 dynamic_cast 来将基类指针转换成子类指针，来实现相应的功能。如图所示：

```
for (auto uit : userList) //遍历用户列表
{
    auto conIt = dynamic_cast<Consumer *>(uit);
```

这样，我们就能继续使用题目一的数据结构，在用户列表中循环访问每一个用户的购物车与订单信息。

2.4 题目三

2.4.1 题目要求

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：

具体功能：

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。

要求：

1. 网络版需要实现的功能的要求与单机版要求一致。

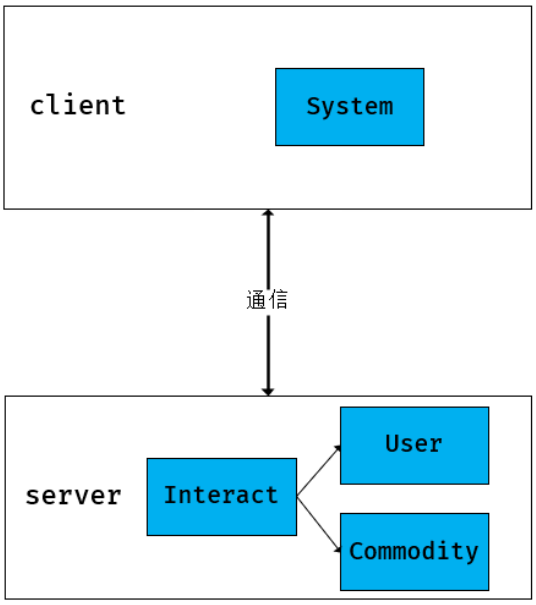
2. 要求采用传统 CS 结构而非 BS 结构，客户端与服务器系统之间使用 socket 进行通信，不能使用 rpc 框架。

3. 支持一定的错误场景处理能力。

2.4.2 系统整体设计

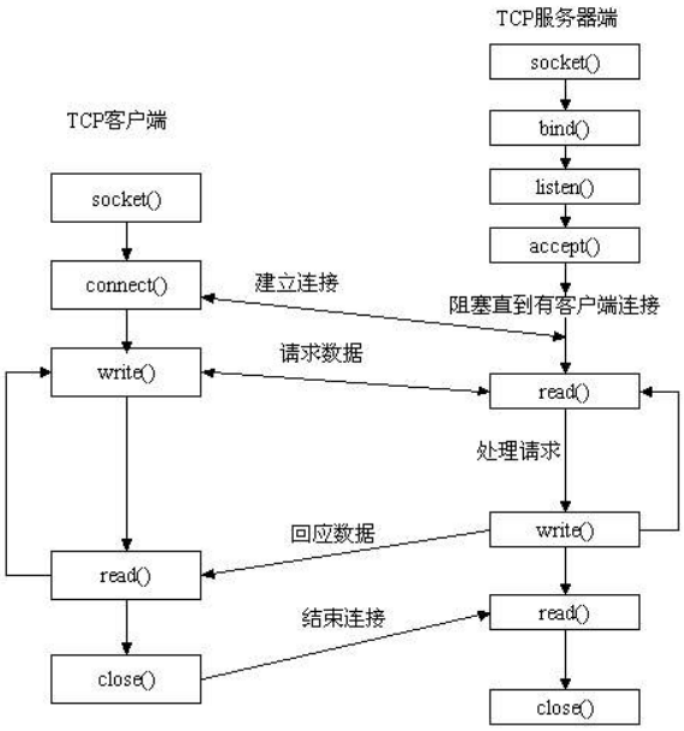
由于模块设计的合理性，第三题相比于前两题只添加了的服务器功能，因此第三题将只对网络功能做详细说明。在线系统采用的是 CS 模式。离线状态下，服务器端预先读取 txt 文件中的商品和用户信息。当开启服务器端进程时，并接受到客户端请求的时候，执行相应的计算命令，将其请求的数据发送回客户端

为了实现多用户同步的要求，所有修改数据的操作，都要求在操作结束后与服务器及时同步。模块间关系如下。



按照题目一的架构， System类负责展示用户界面，接收用户命令，输出执行结果等用户交互工作，并通过Socket通信请求服务器端数据。Interact类是前端和后端的接口，设置有存储用户与商品信息的数据结构，能够从前端通过Socket通信接受命令，并且返回执行结果。以此实现网络版的全过程

2.4.3 Socket 通信实现



实验者在服务器与客户端之间通过发送与接收字符数组来交换数据，从而实现客户端的请求与服务端的响应。

```
send(clientFd, buffSend, MAXBUF, 0);
recv(clientFd, buffRecv, MAXBUF, 0);
```

部分 Socket 代码如下：

```
// 初始化DLL
WSADATA wsaData;
WSAStartup(MAKEWORD(2, 2), &wsaData);

// 生成套接字
SOCKET serverFd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
sockaddr_in myAddr;
memset(&myAddr, 0, sizeof(myAddr));
myAddr.sin_family = PF_INET;
myAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
myAddr.sin_port = htons(atoi(port.c_str()));
bind(serverFd, (SOCKADDR *)&myAddr, sizeof(myAddr));

listen(serverFd, 20);
```

三．实验总结

3.1 优点与不足

3.1.1 优点

1. 模块划分合理

实验者在程序设计之初就认真考虑模块设计问题，并规划出较好的模块划分方案。这使得在做后一个题目的时候不用对先前的架构进行大量调整，在网络编程的时候效率也很高。这些都得益于良好的模块划分。

2. 对异常的处理能力较强

本系统的设计过程中，我始终重视用户的输入输出，以及系统对数据的处理过程。本系统各部分的输入框均进行了限制。这种设计方式使得从根源上断绝了用户输入不合法情况的出现。在其他地方也有许多异常处理，可以有效防止不合法输入。

3.1.2 不足

1．数据安全性有待提升

本系统的数据存储与传输均使用最简单的分隔方式，而未经过其他任何加密，故不管是本地存储的数据还是服务端与客户端之间交互的数据的保密性与安全性较低。

经过分析发现，可通过自己设计或者使用一些已有的、封装良好的库进行加密处理，并且数据可以使用其他格式的文件进行存储来提升数据安全性。

2．用户实验体验不佳

对于电商平台，用户交互体验应该是很重要的一个标准，但是由于时间与自身水平，没有实现可视化，这是很遗憾的。

3.2 问题总结与心得体会

本次实验是我首次面向对象的完整实践。从 C 语言的面向过程到 C++的面向对象的，我在不断的编写程序、修复漏洞、完善算法的过程中进步，并且理解了面向对象的妙处，也感受到了封装、继承、多态的基本特征。

本次 C++综合实验中，我总共花费了 51 天分别完成了三个题目的程序设计，并适当设计了部分扩展功能。我在程序编写的过程中并非一帆风顺的，由于在以前的学习过程中很少使用到 Socket 编程，因此在设计网络版的过程中遇到了许多通信以及并发处理层面的问题。经过查阅大量资料并进行学习，经过大量调试与分析后才最终设计好了通信系统。

C++是计算机学院学生的一门基础课程，也是作为该领域人才所应当掌握的一门编程语言。其作为几门面向对象语言中较底层、较复杂的一门，给程序员提供的编程工具是非常全面的，但由于其提供的编程工具过于全面，因此程序员在编写工作也需要变得谨慎起来。所以在基础实验中，我理解到许多 C++的基础知识，例如虚函数、继承、多态、流等，真正打好了语言基础。

在这基础知识之上，我进入了综合实验。刚开始也是不知所措，后来不多试错之后，我真正体会到了老师上课所讲的一些精华所在，例如全面地考虑类的设计、良好地进行封装与继承等，虽然这种程序设计方法在抽象过程中是困难的，但其对最终组装成为一个完整系统的帮助是巨大的。

从基础实验到综合实验循序渐进的完成，使我对 C++语言有了一个较为基础、全面的认识，这样的合理的课程设计使我受益匪浅。但如果我还要想深入理解，还需在今后的学习生涯中不断开拓，继续实践。

3.3 课程建议

希望可以介绍一些 C++最新的一些应用，以及未来发展趋势。也希望可以稍微介绍可视化与网络编程的知识。