**北京邮电大学课程设计报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **课程设计**  **名称** | 面向对象程序设计实践（C++） | **学 院** | 计算机学院 | **指导教师** | 双锴 |
| 学生姓名 | 陈荣第 | 班 级 | 2019211303 | 学 号 | 2019211205 |
| 课  程  设  计  内  容 | 简要介绍课程设计的主要内容，包括课程设计教学目的、基本内容、实验方法和团队分工等  **教学目的：**让学生了解C++语法知识，明确与C语言的差异与相同，理解如何构造对象，如何面向对象编程。  **基本内容：**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。   1. 账户管理子系统和商品管理子系统。（单机版） 2. 交易管理子系统。（单机版） 3. 电商交易平台。（网络版）   **实验方法：**Socket网络编程，C++封装对象，对平台进行架构设计。 | | | | |
| **学生**  **课程设计**  **报告** | 见程序代码。 | | | | |
| **课**  **程**  **设**  **计**  **成**  **绩**  **评**  **定** | 遵照实践教学大纲并根据以下四方面综合评定成绩：  1、课程设计目的任务明确，选题符合教学要求，份量及难易程度  2、团队分工是否恰当与合理  3、综合运用所学知识，提高分析问题、解决问题及实践动手能力的效果  4、是否认真、独立完成属于自己的课程设计内容，课程设计报告是否思路清晰、文字通顺、书写规范  **评语**:  **成绩**:  指导教师签名：  年 月 日 | | | | |

**题目一：账户管理子系统和商品管理子系统（单机版）**

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：请把所有的用户账户信息写入文件（**要求使用文件存储各类信息，以练习对文件的操作和流的输入输出**），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：请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，请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：请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） 。商品基类请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。

**商品类设计：**

首先设计一个商品基类，将所有种类商品能够拥有的属性放在基类的protected中，保证继承它的派生类也能拥有这些属性。

按要求将计算最终价格的函数设置为虚函数，并写出能够得到商品基本属性的一些函数，用于之后各种平台基础功能的实现。

class Commodity {

public:

Commodity() {};

virtual int getPrice(int n); //虚函数，计算最终价格

string getAccount() const; //得到商品所属账户

string getKind() const; //得到商品类型

string getName() const; //得到商品名称

int getValue() const; //得到商品价格

int getStock() const; //得到商品库存数

bool changeName(string n); //修改商品名称

bool changeValue(int num); //修改商品价格

bool changeStock(int num); //修改商品库存,num>0涨价，num<0降价，最终价格不能小于0；

bool changeDiscount(int num);

protected:

string name; //商品名称

string account; //该商品的所属账户

string kind; //商品种类

int value; //商品价格

int stock; //商品库存量

int discount; //折扣

};

设置三种商品类图书，衣服，食物继承商品基类，并写出各个子类的构造函数，重新覆盖虚函数。

class Food : public Commodity {

public:

Food() {}

Food(vector<string> info);

int getPrice(int n);

};

class Book : public Commodity {

public:

Book() {}

Book(vector<string> info);

int getPrice(int n);

};

class Clothes : public Commodity {

public:

Clothes() {}

Clothes(vector<string> info);

int getPrice(int n);

};

**用户类设计：**

题目要求设置一个用户抽象类，再设计一个消费者类和商家类作为这个用户类的派生类。与用户类同理，将消费者与商家所拥有的共同基本属性一同封装在用户类的protected中，将各种行为函数封装在用户类的public中，按要求将返回用户类型的函数设置为纯虚函数，使用户类为抽象类。

class User {

public:

virtual int getUserKind() const = 0; //得到账号类型

virtual void addCommodity(Commodity\* pcom) = 0; //商家添加商品

virtual int showMyCommodity() = 0; //商家展示它的商品

virtual string changeCommodityInfo() = 0; //改变商品的信息

void changePassword(); //修改密码

int getBalance() const; //获取账号当前余额

bool recharge(int money); //充值

bool consume(int money); //消费

string getAccount() const; //返回账户

string getPassword() const; //返回密码

protected:

string account; //账户

string password; //密码

int balance; //余额

int kind; //账号类型

vector<string> record; //交易记录

void updateUserFile(); //更新用户文件信息

};

设置商家类与消费者类为用户类的派生类，并且商家类在继承原有属性的基础上，添加新属性“自己的商品集合”。用户类中设置的一些纯虚函数为商家类所特有，需要重写函数。两个类都需要各自的构造函数来进行初始化。

class Seller : public User {

private:

list<Commodity\*> coms; //该商家的产品

public:

Seller();

Seller(vector<string>& info, list<Commodity\*>& commodities);

void addCommodity(Commodity\* pcom);

string changeCommodityInfo();

int showMyCommodity();

int getUserKind() const;

};

class Buyer : public User {

public:

Buyer();

Buyer(vector<string>& info);

int getUserKind() const;

//对消费者无用

void addCommodity(Commodity\* pcom) {}

string changeCommodityInfo() { return ""; }

int showMyCommodity() { return 0; }

};

**平台类设计：**

平台类中的函数供主函数调用为了去完成各种用户需求，题目一中的需求主要有用户注册或登录、修改账户密码、余额管理、添加商品、展示平台信息、搜索平台商品、商品信息管理，还有必需的查看商品菜单功能，退出平台程序功能。

平台类的基本属性应该有存储所有商品的链表，还有当前登录的用户指针，若有用户登录则设置给该指针赋值，若无用户登录该设置一直为空指针。

还有一些函数仅供类中函数调用，因此应该将这些函数封装在平台类的私有属性当中。

class Trade {

public:

Trade();

User\* getUser();

void printInitialInterface(); //打印未登录初始界面

void printUserInterface(); //打印登录界面

void buyCommodity(); //购买单个商品

bool userSignIn(); //用户登录

bool userSignUp(); //用户注册

void addCommodity(); //添加商品

void showAllCommodity(); //显示所有商品信息

void searchCommoddity(); //搜索商品

void changeCommodityInfo(); //改变商品信息

~Trade();

private:

list<Commodity\*> commodities; //存储商品

User\* user; //保存当前登录用户

//使得输入密码时屏幕上出现的是\*\*\*\*\*

string inputPassword(int type);

//KMP算法比较子字符串实现搜索功能

bool KMPcompare(string s, string t);

//更新对应商品类别的文件信息

void updateCommodityFile(string filename, string kind);

};

**主函数架构：**

根据菜单中的选项使得用户每循环一次输入一个菜单选项choose，根据输入选项进入switch去调用相关函数完成相应的功能。由于已登录和未登录的菜单不同，因此在进入switch之前需要判断当前是否已有用户登录，即判断用户指针是否为空，若为空则未登录，不为空已登录；还要设置一个bool变量exit标记当前用户输入是否是退出选项，由于主函数的循环设置为死循环，若用户选择退出，那么应该使exit = true,并且exit = true时执行break退出循环，结束程序。

程序执行框图：

开始

平台初始化，变量初始化

输入菜单选项

是否登录

否 是

是否是退出选项

是否是退出选项

是

否 否

执行登录后选项对应的操作

执行未登录选项对应的操作

退出循环

结束程序

**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，请实现以下功能：

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。

**商品类：**

题目二基于题目一的代码进行改造，商品类属性及其功能函数并未改变。

**用户类：**

由于新功能需要支持进行购物车管理，因此在用户的基本属性当中需要增加新属性购物车，购物车中需要存储选中的商品及其对应的数量。出于简便原因，我将存储订单的链表也归到了购物车的类中，每个订单中需要存储要支付的多个商品以及此订单中所有商品的总金额。

为了简化购物车中的每一项都是商品及其数量，构造一个Item类，专门用来存储指向对应商品的指针，及其要加入购物车的对应数量。

购物车中除了商品及其数量之外，还需要能够打印车中内容的函数，使用户对其进行查询时能够调用，订单也需要查询，因此也需要一个打印函数供平台调用。

项目类（指向商品的指针和数量）：

class Item {

public:

Item() {

nums = 0;

pcom = nullptr;

}

Commodity\* pcom;

string name; //商品名称

string acc; //所属账号

string kind; //商品种类

int nums;

};

订单类：

class Order {

public:

Order() {

totalMoney = 0;

}

Order(vector<Item>& a, int& b) {

for (int i = 0; i < a.size(); ++i) {

items.push\_back(a[i]);

}

totalMoney = b;

}

void showOrder(); //展示订单

vector<Item> items; //订单中的商品

int totalMoney; //总金额

};

购物车类：

class Cart {

public:

void showCart(); //展示购物车中所有商品

void showOrders(); //展示所有订单

list<Order> orders; //订单可以有多个

list<Item> items; //购物车中存储的商品及对应数量

};

用户基类添加购物车：

添加前： 添加后：



**平台类：**

根据新增功能在平台类中添加新的成员函数，管理购物车，生成订单，网上支付。

class Trade {

public:

Trade();

User\* getUser();

void printInitialInterface(); //打印未登录初始界面

void printUserInterface(); //打印登录界面

void buyCommodity(); //购买单个商品

bool userSignIn(); //用户登录

bool userSignUp(); //用户注册

void addCommodity(); //添加商品

void showAllCommodity(); //显示所有商品信息

void searchCommoddity(); //搜索商品

void changeCommodityInfo(); //修改商品信息

void manageCart(); //管理购物车

void makeOrders(); //生成订单

void onlinePayment(); //网上支付

~Trade();

private:

list<Commodity\*> commodities; //存储商品

User\* user; //保存用户

string inputPassword(int type); //输入密码屏幕回显\*\*\*\*\*\*\*\*\*\*\*

bool KMPcompare(string s, string t); //KMP算法实现搜索

//更新对应商品的文件

void updateCommodityFile(const string& filename, const string& kind);

void addComToCart(); //加入购物车商品

void subComToCart(); //删除购物车商品

};

**主函数架构：**

主函数架构基本与题目一相似基本不变，仅仅只是登录之后新增了几个菜单选项。程序执行顺序也与题目一程序框图一致。

**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。

网络版CS架构分为客户端和服务端，在客户端与服务端之间地通信选择地是Socket中TCP协议。由于服务端只能有一个，而客户端可以同时有多个连接，因此我选择使用多线程来处理多个连接地问题。

**客户端设计：**

客户端使用两个线程来处理，一个是用来写请求的线程，一个是用来读入处理响应包的线程。使用平台类来封装一系列函数，但由于对函数指针的理解不够，两个线程函数需要函数指针来实现，放在类中时无法绑定类中的函数，因此放到了类外。

**平台类设计：**

class Trade {

public:

bool getSendBuf(char\* buf); //构造发送请求

bool solveRecvBuf(char recvbuf[]); //分析得到的回答

void printInitialInterface(); //打印未登录界面

void printUserInterface(); //打印登录界面

void getResult(char buf[]); //得到回复消息

void buyCommodity(char\* buf, int& offset); //购买单个商品

void changePassword(char\* buf, int& offset);//改变密码

void rechargeMoney(char\* buf, int& offset); //充值

void userSignIn(char\* buf, int& offset); //用户登录

void userSignUp(char\* buf, int& offset); //用户注册

void addCommodity(char\* buf, int& offset); //添加商品

void showCommodity(char buf[], int type); //显示所有商品信息

void searchCommoddity(char\* buf,int& offset); //搜索商品

void changeCommodityInfo(char\* buf, int& offset);//修改商品信息

void manageCart(char\* buf, int& offset); //管理购物车

void showCart(char buf[]); //查看购物车

void makeOrders(char\* buf, int& offset); //生成订单

void showOrders(char buf[]); //查看订单

void onlinePayment(char\* buf, int& offset); //网上支付

};

**全局变量以及两个类外线程函数：**

extern int getNum(); //正确地输入一个数字，对不正常输入进行报错处理

extern bool isSignIn; //是否登录地标志

#define BUF\_SIZE 1024 //缓冲区大小

void ErrorHandler(char\* message); //调试

unsigned WINAPI WriteHandler(void\* param); //处理写操作

unsigned WINAPI ReadHandler(void\* param); //处理读进程

const int PORT = 8000; //端口号

//事件数组用来控制线程的读写

HANDLE evts[2];

//读取的标准输入的长度，用来接收客户端数据的最大值

int strLen = 0;

Trade myTrade; //平台声明

**主函数架构:**

与服务器建立连接，建立连接成功之后，根据输入的选项构造要发送的包，再等待响应包的到达，响应包到达之后平台对其进行处理打印相关信息。

与服务器连接成功之后，建立两个线程，并且开始构造发包和收包处理的两个线程开始如下步骤：

HANDLE hThreads[2];

//创建线程用于写和读

hThreads[0] = (HANDLE)\_beginthreadex(NULL, 0, WriteHandler, &sock, 0, NULL);

hThreads[1] = (HANDLE)\_beginthreadex(NULL, 0, ReadHandler, &sock, 0, NULL);

//创建事件，都为manual-reset模式，控制写的事件起初是有信号状态，读的事件起初是无信号状态

evts[0] = CreateEvent(NULL, true, true, NULL);

evts[1] = CreateEvent(NULL, true, false, NULL);

WaitForMultipleObjects(2, hThreads, true, INFINITE);

CloseHandle(evts[0]);

CloseHandle(evts[1]);

由于对线程掌握也不是太清楚，都是查阅网上资料以及询问同学，调试时也出现了一些问题，最终总体而言基本可以通信。

程序框图：

开始

退出程序

接受包打印提示信息

写线程休眠

开始读线程

构造包发送

读线程休眠

开始写线程

创建读写线程

连接成功

连接服务端

否

是

构造包失败

构造包成功

**服务端设计：**

服务端只有一个，必须能够实现同时对多个客户端进行处理，因此需要用多线程来处理多个客户端，每监听到一个连接请求并且建立与客户端的连接就给这个客户端构建一个线程。在各自的线程中，服务端与各个客户端进行收包发包处理，完成各个客户端的请求，每个线程中都有一个用户的指针指向登录的用户，每接受一次包就更新一次用户信息，保证每次操作后用户信息正确。如果接收到客户端的包出现错误，删除该用户并且删除对应的进程。

服务端的商品类和用户类沿用了题目二中的类，因此基本没有改变。

平台类：

class Trade {

public:

Trade();

//处理收到的包并构造要响应的包

void handleBuf(char recvbuf[], char\* sendbuf, User\*& user);

//购买单个商品

void buyCommodity(char\* sendbuf, User\*& user, char recvbuf[], int& offset);

//用户登录

void userSignIn(char\* sendbuf,User\*& user, char recvbuf[], int& offset);

//用户注册

bool userSignUp(char\* sendbuf, char recvbuf[], int& offset);

//添加商品

void addCommodity(char\* sendbuf, User\*& user, char recvbuf[], int& offset);

//显示所有商品信息

void showAllCommodity(char\* sendbuf, int& offset);

//搜索商品

void searchCommoddity(char\* sendbuf, int& offset, char recvbuf[]);

//修改商品信息

void changeCommodityInfo(char\* sendbuf, User\*& user, char recvbuf[], int& offset);

//管理购物车

void manageCart(char\* sendbuf, User\*& user, char recvbuf[], int& offset);

//生成订单

void makeOrders(char\* sendbuf, User\*& user, char recvbuf[], int& offset);

//网上支付

void onlinePayment(char\* sendbuf, User\*& user, char recvbuf[], int& offset);

//更新当前用户数据

bool updateMyData(User\*& user);

~Trade();

private:

//存放商品

list<Commodity\*> commodities;

//回显\*\*\*\*\*

string inputPassword(int type);

//KMP实现搜索

bool KMPcompare(string s, string t);

//更新商品文件

void updateCommodityFile(const string& filename, const string& kind); //加入购物车商品

void addComToCart(char\* sendbuf, User\*& user, char recvbuf[], int& offset);

//删除购物车商品

void subComToCart(char\* sendbuf, User\*& user, char recvbuf[], int& offset);

};

**全局变量及线程函数：**

#define BUF\_SIZE 1024 //缓冲区大小

#define THREAD\_SIZE 20 //最多能够创建的线程个数

const int PORT = 8000; //端口号

//创建线程函数

unsigned WINAPI ClntHandler(void\* param);

//存放客户端socket的数组

SOCKET clntSocks[THREAD\_SIZE];

//锁

HANDLE mtx;

//总的客户端连接的个数

int clntCnt = 0;

//定义平台

Trade myTrade;

程序框图：

开始

发送响应包

为连接用户创建一个线程

接收包并构造响应包

建立一个连接

初始化并创建套接字并开始监听

主线程一直监听连接

删除该用户所在的线程

接收包为无效包

**文件准备格式：**

首先需要创建一个用户文件夹和商品文件夹如图：



对于用户文件夹，我们无需再改变，直接在程序中注册或登录。

对于商品文件夹，我们还要创建每种商品的txt文件，程序中仅支持book、clothes、food种类，如下图：



用户可以在程序中注册一个商家账户来添加对应种类的商品。

**过程中的问题，教训，感想以及存在的问题**

**对于判断输入错误：**

起初写前两个实验时，仅仅只对输入菜单选项进行了错误处理，由于我认为输入很多，如果各个输入都进行判错会过于麻烦，在先验收了前两次之后，第三个实验我想要对所有输入进行判错，因此我写了一个能够判断输入错误的函数，输入正确格式才会对其进行接收，这样当需要输入的时候，直接调用此函数就可以解决许多问题。

**对于设计方面的问题：**

商品类与用户类的按要求进行设计，基本没有出现特别大的问题。但是起初题目二中我将购物车设置在了平台类当中，对于单机版而言并无问题。但当我要设计网络版时，由于每个客户端都需要一个购物车，将购物车放在平台类中明显是错误的，因此后续过程中我将购物车以及订单修改到了用户类当中。

还有在要实现平台类的过程当中，平台类的函数可能需要修改用户类的属性，而用户类的属性是私有的，因此有些功能的实现应该卸载用户类当中，而不是平台类。

**存在的问题及如何改进：**

在设计过程中，由于设计问题，我在实现最终网络版时发现我的服务端不能及时更新当前的信息，在支付失败之后，虽然文件中的商品个数未扣除，但内存中的个数已扣除，如果不及时更新，那么遇到多次此问题该商品个数可能为0。因此我认为应该设置一个时间阈值，每隔此段时间，显示的调用平台的构造函数一次，对平台的所有商品信息进行更新。

还有注册账号时如果在注册过程中强制关闭程序，可能会导致保存了一个注册信息不完全的账户文件，这会使登录该账户出现错误，目前还没有解决方案。

总而言之，此次电商平台的设计与实现仍然有一些潜在的问题，但基本能够实现设计所要求的功能。学习完成这个课程设计让我也对C++这门语言和面向对象编程有了更深层次的了解和认识，相信这会让我以后使用C++能够更加的心应手。