附录：程序

源文件 1main.cpp

```
#include "VideoStore.h"
int main() {
    runProgram();   // 启动程序
    return 0;
}
```

源文件 2VideoTape.h

```
#ifndef VIDEOTAPE_H
#define VIDEOTAPE_H

#include <string>
#include <iostream>
using namespace std;

class VideoTape {
private:
    int id;                          // 录像带的 ID
    string movieName;                // 电影名称
    int copiesAvailable;             // 可用副本数

public:
    // 默认构造函数声明
    VideoTape();

    // 带参数的构造函数声明
    VideoTape(int id, const string& movieName, int copiesAvailable);

    // 获取录像带 ID
    int getId() const;
    // 获取电影名称
    string getMovieName() const;

    // 获取可用副本数
    int getCopiesAvailable() const;

    // 设置电影名称
    void setMovieName(const string& newMovieName);

    // 设置可用副本数
    void setCopiesAvailable(int newCopiesAvailable);

    // 显示录像带信息
    void displayInfo() const;
};

#endif // VIDEOTAPE_H
```

源文件 3VideoTape.cpp

```
#include "VideoTape.h"

// 默认构造函数的实现
VideoTape::VideoTape() : id(0), movieName(""), copiesAvailable(0) {}

// 带参数构造函数的实现
VideoTape::VideoTape(int id, const string& movieName, int copiesAvailable)
    : id(id), movieName(movieName), copiesAvailable(copiesAvailable) {}
```

```
// 获取录像带 ID
int VideoTape::getId() const {
        return id;
}

// 获取电影名称
string VideoTape::getMovieName() const {
        return movieName;
}

// 获取可用副本数
int VideoTape::getCopiesAvailable() const {
        return copiesAvailable;
}

// 设置电影名称
void VideoTape::setMovieName(const string& newMovieName) {
        movieName = newMovieName;
}

// 设置可用副本数
void VideoTape::setCopiesAvailable(int newCopiesAvailable) {
        copiesAvailable = newCopiesAvailable;
}

// 显示录像带信息
void VideoTape::displayInfo() const {
        cout << "录像带 ID : " << id << ",  录像带名称 : " << movieName
                << ", 副本数量 : " << copiesAvailable << endl;
}
```

源文件 4SalesRecord.h

```
#ifndef SALESRECORD_H
#define SALESRECORD_H

#include "VideoTape.h"
#include <string>
using namespace std;

class SalesRecord {
private:
    int id;                    // 销售记录 ID
    VideoTape videoTape;   // 关联的录像带对象
    int quantitySold;       // 销售数量
    string saleDate;        // 销售日期
    float price;              // 销售价格

public:

    // 默认构造函数
    SalesRecord();
    // 构造函数
    SalesRecord(int id, VideoTape videoTape, int quantitySold, string saleDate, float
price);

    // Getter 和 Setter 方法
    int getId() const;
    void setId(int id);

    VideoTape getVideoTape() const;
    void setVideoTape(VideoTape videoTape);

    int getQuantitySold() const;
    void setQuantitySold(int quantitySold);

    string getSaleDate() const;
    void setSaleDate(string saleDate);

    float getPrice() const;
    void setPrice(float price);

    // 显示销售信息
    void displaySalesInfo() const;
    float calculateTotalSales() const;
};

#endif // SALESRECORD_H
```

源文件 5SalesRecord.cpp

```
#include "SalesRecord.h"
#include <iostream>
using namespace std;


// 默认构造函数
SalesRecord::SalesRecord()
    : id(0),                              // 销售记录 ID 默认为 0
    videoTape(VideoTape()),               // 使用 VideoTape 的默认构造函数
    quantitySold(0),                      // 销售数量 默认为 0
    saleDate(""),                    // 销售日期 默认为 空字符串
    price(0.0f)                      // 销售价格 默认为 0.0
{
}

// 构造函数
SalesRecord::SalesRecord(int id, VideoTape videoTape, int quantitySold, string saleDate, float price) {
    this->id = id;
    this->videoTape = videoTape;
    this->quantitySold = quantitySold;
    this->saleDate = saleDate;
    this->price = price;
}

// Getter 和 Setter 方法
int SalesRecord::getId() const {
    return id;
}

void SalesRecord::setId(int id) {
    this->id = id;
}

VideoTape SalesRecord::getVideoTape() const {
    return videoTape;
}

void SalesRecord::setVideoTape(VideoTape videoTape) {
    this->videoTape = videoTape;
```

```
}

int SalesRecord::getQuantitySold() const {
        return quantitySold;
}

void SalesRecord::setQuantitySold(int quantitySold) {
        this->quantitySold = quantitySold;
}

string SalesRecord::getSaleDate() const {
        return saleDate;
}

void SalesRecord::setSaleDate(string saleDate) {
        this->saleDate = saleDate;
}

float SalesRecord::getPrice() const {
        return price;
}

void SalesRecord::setPrice(float price) {
        this->price = price;
}

//  显示销售信息
void SalesRecord::displaySalesInfo() const {
        cout << "销售记录 ID : " << id << endl;
        videoTape.displayInfo();
```

```
    cout << "销售数量 : " << quantitySold << endl;
    cout << "销售日期 : " << saleDate << endl;
    cout << "销售价格 : " << price << endl;
}

// 计算销售总额
float SalesRecord::calculateTotalSales() const {
    return quantitySold * price;
}
```

源文件 6PurchaseRecord.h

```
#ifndef PURCHASERECORD_H
#define PURCHASERECORD_H

#include "VideoTape.h"
#include <string>
using namespace std;

class PurchaseRecord {
private:
    int id;                  // 进货记录 ID
    VideoTape videoTape;     // 关联的录像带对象
    int quantity;            // 进货数量
    string purchaseDate;     // 进货日期
    float price;             // 进货价格

public:
    // 默认构造函数
    PurchaseRecord();
    // 构造函数
    PurchaseRecord(int id, VideoTape videoTape, int quantity, string purchaseDate,
float price);

    // Getter 和 Setter 方法
    int getId() const;
    void setId(int id);

    VideoTape getVideoTape() const;
    void setVideoTape(VideoTape videoTape);

    int getQuantity() const;
    void setQuantity(int quantity);

    string getPurchaseDate() const;
    void setPurchaseDate(string purchaseDate);

    float getPrice() const;
    void setPrice(float price);

    // 显示进货记录信息
    void displayPurchaseInfo() const;
};

#endif // PURCHASERECORD_H
```

源文件 7PurchaseRecord.cpp

```
#include "PurchaseRecord.h"
#include <iostream>
using namespace std;
```

```
// 默认构造函数
PurchaseRecord::PurchaseRecord()
    : id(0),
    videoTape(VideoTape()),  // 默认构造 VideoTape 对象
    quantity(0),
    purchaseDate(""),
    price(0.0f) {
}

// 构造函数
PurchaseRecord::PurchaseRecord(int id, VideoTape videoTape, int quantity, string
purchaseDate, float price) {
    this->id = id;
    this->videoTape = videoTape;
    this->quantity = quantity;
    this->purchaseDate = purchaseDate;
    this->price = price;
}

// Getter 和 Setter 方法
int PurchaseRecord::getId() const {
    return id;
}

void PurchaseRecord::setId(int id) {
    this->id = id;
}

VideoTape PurchaseRecord::getVideoTape() const {
    return videoTape;
}

void PurchaseRecord::setVideoTape(VideoTape videoTape) {
    this->videoTape = videoTape;
}

int PurchaseRecord::getQuantity() const {
    return quantity;
}

void PurchaseRecord::setQuantity(int quantity) {
    this->quantity = quantity;
}

string PurchaseRecord::getPurchaseDate() const {
    return purchaseDate;
}

void PurchaseRecord::setPurchaseDate(string purchaseDate) {
    this->purchaseDate = purchaseDate;
}

float PurchaseRecord::getPrice() const {
    return price;
}

void PurchaseRecord::setPrice(float price) {
    this->price = price;
}
```

```
// 显示进货记录信息
void PurchaseRecord::displayPurchaseInfo() const {
    cout << "进货记录 ID : " << id << endl;
    videoTape.displayInfo();
    cout << "进货数量 : " << quantity << endl;
    cout << "进货日期 : " << purchaseDate << endl;
    cout << "进货价格 : " << price << endl;
}
```

源文件 8VideoStore.h

```
#ifndef VIDEOSTORE_H
#define VIDEOSTORE_H

#include <iostream>
#include <string>
#include "VideoTape.h"
#include "PurchaseRecord.h"
#include "SalesRecord.h"
#include "Inventory.h"
#include <windows.h>
#include <fstream>
#include <sstream>

using namespace std;

// 设置控制台文本颜色
void setColor(int color);
```

```
// 显示菜单
void showMenu();
void runProgram();    // 声明函数
// 快速排序：针对任何类型的容器（例如 VideoTape、PurchaseRecord、SalesRecord）
template <typename T>
void quickSort(LTvector<T>& items, int low, int high) {
        if (low < high) {
                int pivot = partition(items, low, high);    // 找到基准
                quickSort(items, low, pivot - 1);    // 排序基准左边部分
                quickSort(items, pivot + 1, high);    // 排序基准右边部分
        }
}

// 分区操作：通用的分区函数，返回排序后的基准位置
template <typename T>
int partition(LTvector<T>& items, int low, int high) {
        int pivot = items[high].getId();    // 选取最后一个元素作为基准
        int i = (low - 1);

        for (int j = low; j < high; j++) {
                if (items[j].getId() <= pivot) {    // 如果当前元素小于或等于基准
                        i++;
                        swap(items[i], items[j]);    // 交换
                }
        }
        swap(items[i + 1], items[high]);    // 把基准放到正确的位置
        return (i + 1);
}
```

```
// 按照副本数量排序
template <typename T>
void quickSortNum(LTvector<T>& items, int low, int high) {
    if (low < high) {
        int pivot = partitionNum(items, low, high);   // 找到基准
        quickSortNum(items, low, pivot - 1);   // 排序基准左边部分
        quickSortNum(items, pivot + 1, high);   // 排序基准右边部分
    }
}

// 分区操作：按照副本数量排序
template <typename T>
int partitionNum(LTvector<T>& items, int low, int high) {
    int pivot = items[high].getCopiesAvailable();   // 选取最后一个元素作为基准
    int i = (low - 1);

    for (int j = low; j < high; j++) {
        if (items[j].getCopiesAvailable() <= pivot) {   // 如果当前元素小于或等于
基准
            i++;
            swap(items[i], items[j]);   // 交换
        }
    }
    swap(items[i + 1], items[high]);   // 把基准放到正确的位置
    return (i + 1);
}

// 按照价格排序
template <typename T>
void quickSortPri(LTvector<T>& items, int low, int high) {
    if (low < high) {
        int pivot = partitionPri(items, low, high);    //  找到基准
        quickSortPri(items, low, pivot - 1);    //  排序基准左边部分
        quickSortPri(items, pivot + 1, high);    //  排序基准右边部分
    }
}

//  分区操作：按照价格排序
template <typename T>
int partitionPri(LTvector<T>& items, int low, int high) {
    int pivot = items[high].getPrice();    //  选取最后一个元素作为基准
    int i = (low - 1);

    for (int j = low; j < high; j++) {
        if (items[j].getPrice() <= pivot) {    //  如果当前元素小于或等于基准
            i++;
            swap(items[i], items[j]);    //  交换
        }
    }
    swap(items[i + 1], items[high]);    //  把基准放到正确的位置
    return (i + 1);
}

#endif // VIDEOSTORE_H
```

源文件 9VideoStore.cpp

```
#include "VideoStore.h"

//  设置控制台文本颜色
void setColor(int color) {
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);   // 获取控制台句柄
    SetConsoleTextAttribute(hConsole, color);   // 设置文本颜色
}

void showMenu() {
    setColor(7);
    cout << "\n---  录像带商店进销存管理系统  ---" << endl;
    cout << "请输入功能菜单所对应的序号来实现您的需求" << endl;
    cout << "1.  查看录像带记录" << endl;
    cout << "2.  添加录像带记录" << endl;
    cout << "3.  修改录像带记录" << endl;
    cout << "4.  删除录像带记录" << endl;
    cout << "5.  查看进货记录" << endl;
    cout << "6.  添加进货记录" << endl;
    cout << "7.  修改进货记录" << endl;
    cout << "8.  删除进货记录" << endl;
    cout << "9.  查看销售信息" << endl;
    cout << "10. 添加销售信息" << endl;
    cout << "11. 修改销售信息" << endl;
    cout << "12. 删除销售信息" << endl;
    cout << "13. 按照录像带 ID 排序" << endl;
    cout << "14. 按照进货记录 ID 排序" << endl;
    cout << "15. 按照销售记录 ID 排序" << endl;
    cout << "16. 按照录像带副本数排序" << endl;
    cout << "17. 按照进货记录价格排序" << endl;
    cout << "18. 按照销售记录价格排序" << endl;
    cout << "19. 查看所有信息" << endl;
    cout << "20. 显示功能菜单" << endl;
```

```
        cout << "21.  显示格式说明书" << endl;
        cout << "0.    退出系统" << endl;
}

void addVideoTapeRecord(Inventory& inventory) {
        // 添加文件读取功能
        char readChoice;
        cout << "从本地文件读取录像带记录（VedioTape.txt）并添加到系统请输入 y
" << endl;
        cout << "自己手动添加录像带记录请输入 n" << endl;
        cin >> readChoice;

        if (readChoice == 'n' || readChoice == 'N') {
                int id;
                string movieName;
                int copiesAvailable;
                cout << "请输入录像带 ID: ";
                cin >> id;
                cin.ignore();   // 忽略输入缓冲区的换行符
                cout << "请输入电影名称: ";
                getline(cin, movieName);
                cout << "请输入库存数量: ";
                cin >> copiesAvailable;
                if (copiesAvailable < 0) {
                        setColor(4);
                        cout << "库存数量必须是正整数" << endl;
                        return;
                }
        // 检查录像带是否已经存在
        VideoTape* video = inventory.findVideoTapeById(id);
        if (!video) {   // 如果该录像带 ID 不存在，进行添加
            VideoTape newVideo(id, movieName, copiesAvailable);
            inventory.addVideoTape(newVideo);
            setColor(2);
            cout << "录像带记录已添加！" << endl;
        }
        else {
            setColor(4);
            cout << "此 ID 所对应的录像带已存在，无法添加。" << endl;
        }
    }
    else if (readChoice == 'y' || readChoice == 'Y') {
        // 打开文件
        ifstream file("VedioTape.txt");
        if (!file) {
            setColor(4);
            cout << "无法打开文件 VedioTape.txt" << endl;
            return;
        }

        string line;
        while (getline(file, line)) {
            stringstream ss(line);
            int id;
            string movieName;
            int copiesAvailable;
```

```
                // 解析每一行
                ss >> id;   // 读取录像带 ID
                ss.ignore();      // 忽略空格
                getline(ss, movieName, ' ');   // 读取电影名称
                ss >> copiesAvailable;   // 读取库存数量

                if (copiesAvailable < 0) {
                    setColor(4);
                    cout << "库存数量必须是正整数" << endl;
                    return;
                }

                // 检查录像带是否已经存在
                VideoTape* video = inventory.findVideoTapeById(id);
                if (!video) {   // 如果该录像带 ID 不存在，进行添加
                    VideoTape newVideo(id, movieName, copiesAvailable);
                    inventory.addVideoTape(newVideo);
                    setColor(2);
                    cout << "录像带记录已添加！" << endl;
                }
                else {
                    setColor(4);
                    cout << "ID:" << id << "," << "此ID所对应的录像带已存在，无法
添加。" << endl;
                }
            }
        }
}

void searchVideoTape(Inventory& inventory) {
        char readChoice;
        cout << "通过录像带名称检索录像带请输入：  y " << endl;
        cout << "通过录像带唯一 ID 检索录像带请输入：  n" << endl;
        cin >> readChoice;

        if (readChoice == 'n' || readChoice == 'N') { // 根据 ID 查找
            int id;
            cout << "请输入录像带 ID: ";
            cin >> id;
            VideoTape* video = inventory.findVideoTapeById(id);
            if (video) {  // 检查指针是否有效
                setColor(2);
                video->displayInfo();  // 使用 -> 调用成员函数，显示录像带信息
            }
            else {
                setColor(4);
                cout << "未找到该录像带。" << endl;
            }
        }
        else if (readChoice == 'y' || readChoice == 'Y') { // 根据名称查找
            string name;
            cout << "请输入录像带名称: ";
            cin >> name;  // 输入录像带名称

            LTvector<VideoTape>  videos  =  inventory.findVideoTapesByName(name);
// 获取匹配名称的录像带

            if (videos.get_size() > 0) {  // 如果找到了录像带
                setColor(2);  // 设置颜色为绿色
                for (size_t i = 0; i < videos.get_size(); ++i) {
                    videos[i].displayInfo();   // 显示每一个录像带的信息
                }
            }

            else {
                setColor(4);   // 设置颜色为红色
                cout << "未找到该名称的录像带。" << endl;
            }
        }
}

void modifyVideoTapeRecord(Inventory& inventory) {
        int id;
        cout << "请输入要修改的录像带 ID: ";
        cin >> id;
        cin.ignore();   // 忽略输入缓冲区的换行符

        VideoTape* video = inventory.findVideoTapeById(id);
        if (video) {
            string newMovieName;
            int newCopiesAvailable;
            cout << "请输入新的电影名称: ";
            getline(cin, newMovieName);
            cout << "请输入新的库存数量: ";
            cin >> newCopiesAvailable;
            if (newCopiesAvailable < 0) {
                setColor(4);
                cout << "库存数量必须是正整数" << endl;
                return;
            }
            video->setMovieName(newMovieName);
            video->setCopiesAvailable(newCopiesAvailable);
            setColor(2);
            cout << "录像带记录已修改！" << endl;
        }
        else {
            setColor(4);
            cout << "找不到该录像带记录！" << endl;
        }
}

void deleteVideoTapeRecord(Inventory& inventory) {
        int id;
        cout << "请输入要删除的录像带 ID: ";
        cin >> id;
        if (inventory.findVideoTapeById(id)) {
            inventory.removeVideoTape(id);
            setColor(2);
            cout << "录像带记录已删除！" << endl;
        }
        else {
            setColor(4);
            cout << "找不到该录像带记录！" << endl;
        }
}

void viewPurchaseRecords(Inventory& inventory) {
        char readChoice;
        cout << "通过进货日期检索进货记录请输入：  y " << endl;
        cout << "通过进货记录唯一 ID 检索请输入：  n" << endl;
        cin >> readChoice;

        if (readChoice == 'n' || readChoice == 'N') {
            int id;
            cout << "请输入进货记录 ID: ";
            cin >> id;
            PurchaseRecord* record = inventory.findPurchaseRecordById(id);
            if (record) {   // 检查指针是否有效
                setColor(2);
                record->displayPurchaseInfo();   // 使用 -> 调用成员函数，显示进
货记录信息
            }
            else {
                setColor(4);
                cout << "未找到该进货记录。" << endl;
            }
        }
        else if (readChoice == 'y' || readChoice == 'Y') {
            // 查询进货记录的日期
            string purchaseDate;
            cout << "请输入进货日期 : ";
            cin >> purchaseDate;   // 输入进货日期

            LTvector<PurchaseRecord> purchases =
inventory.findPurchaseRecordsByDate(purchaseDate);   // 获取匹配日期的进货记
录

            if (purchases.get_size() > 0) {   // 如果找到了匹配的进货记录
```

```
            setColor(2);  // 设置颜色为绿色
            for (size_t i = 0; i < purchases.get_size(); ++i) {
                purchases[i].displayPurchaseInfo();  // 显示每一条进货记录的
信息
            }
        }
        else {
            setColor(4);  // 设置颜色为红色
            cout << "未找到该日期的进货记录。" << endl;
        }
    }
}

void addPurchaseRecord(Inventory& inventory) {
    int id;
    int quantity;
    string purchaseDate;
    float price;
    int videoId;

    cout << "请输入进货记录 ID: ";
    cin >> id;
    cout << "请输入关联的录像带 ID: ";
    cin >> videoId;
    cout << "请输入进货数量: ";
    cin >> quantity;
    cin.ignore();  // 忽略换行符
    cout << "请输入进货日期: ";
    getline(cin, purchaseDate);
```

```
        cout << "请输入进货价格: ";
        cin >> price;

        if (quantity < 0) {
            setColor(4);
            cout << "进货数量必须是正整数" << endl;
            return;
        }
        else if (price < 0) {
            setColor(4);
            cout << "进货价格必须是正数" << endl;
            return;
        }

        PurchaseRecord* record = inventory.findPurchaseRecordById(id);
        if (!record) {   // 若无记录，则可添加
            VideoTape* video = inventory.findVideoTapeById(videoId);
            if (video) {
                PurchaseRecord newPurchase(id, *video, quantity, purchaseDate,
    price);
                inventory.addPurchaseRecord(newPurchase);
                // 更改货品数量
                inventory.purchaseVideoTape(videoId, quantity);
                setColor(2);
                cout << "进货记录已添加！" << endl;
            }
            else {
                setColor(4);
                cout << "录像带 ID 不存在！" << endl;
            }
        }
        else {
                setColor(4);
                cout << "此 ID 所对应的进货记录已存在,无法添加。" << endl;
        }
}

void modifyPurchaseRecord(Inventory& inventory) {
        int id;
        cout << "请输入要修改的进货记录 ID: ";
        cin >> id;
        cin.ignore();    //  忽略换行符

        PurchaseRecord* record = inventory.findPurchaseRecordById(id);
        if (record) {
                int newQuantity;
                float newPrice;
                cout << "请输入新的进货数量: ";
                cin >> newQuantity;
                cout << "请输入新的进货价格: ";
                cin >> newPrice;

                if (newQuantity < 0) {
                        setColor(4);
                        cout << "进货数量必须是正整数" << endl;
                        return;
                }
                else if (newPrice < 0) {
                        setColor(4);
```

```
            cout << "进货价格必须是正数" << endl;
            return;
        }

        // 更新进货记录
        int videoIDPR = record->getVideoTape().getId();   // 获取影片 ID
        int oldQuantity = record->getQuantity();                  // 获取旧的进货数量
        inventory.purchaseVideoTape(videoIDPR, -oldQuantity);    // 撤销旧的进
货数量
        inventory.purchaseVideoTape(videoIDPR, newQuantity);    // 添加新的进
货数量

        record->setQuantity(newQuantity);   // 更新进货数量
        record->setPrice(newPrice);              // 更新进货价格

        setColor(2);
        cout << "进货记录已修改！" << endl;
    }
    else {
        setColor(4);
        cout << "找不到该进货记录！" << endl;
    }
}

void deletePurchaseRecord(Inventory& inventory) {
    int id;
    cout << "请输入要删除的进货记录 ID: ";
    cin >> id;

    PurchaseRecord* record = inventory.findPurchaseRecordById(id);
    if (record) {
        // 更新库存数量
        int videoIDPR = record->getVideoTape().getId(); // 获取影片 ID
        int oldQuantity = record->getQuantity();
        inventory.purchaseVideoTape(videoIDPR, -oldQuantity); // 撤销进货数量

        // 删除进货记录
        inventory.removePurchaseRecord(id);
        setColor(2);
        cout << "进货记录已删除！" << endl;
    }
    else {
        setColor(4);
        cout << "找不到该进货记录！" << endl;
    }
}

void viewSalesRecord(Inventory& inventory) {
    char readChoice;
    cout << "通过销售日期检索销售信息请输入：  y" << endl;
    cout << "通过销售记录唯一 ID 检索请输入：  n" << endl;
    cin >> readChoice;

    if (readChoice == 'n' || readChoice == 'N') {
        int id;
        cout << "请输入销售记录 ID: ";
        cin >> id;
        SalesRecord* record = inventory.findSalesRecordById(id);
        if (record) {   // 检查指针是否有效
            setColor(2);
```

```
                record->displaySalesInfo();    // 显示销售信息
            }
            else {
                setColor(4);
                cout << "未找到该销售记录。" << endl;
            }
        }
        else if (readChoice == 'y' || readChoice == 'Y') {
            string saleDate;
            cout << "请输入销售日期: ";
            cin >> saleDate;    // 输入销售日期

            LTvector<SalesRecord> sales = inventory.findSalesRecordsByDate(saleDate);
// 获取匹配日期的销售记录

            if (sales.get_size() > 0) {    // 如果找到了匹配的销售记录
                setColor(2);    // 设置颜色为绿色
                for (size_t i = 0; i < sales.get_size(); ++i) {
                    sales[i].displaySalesInfo();    // 显示每一条销售记录的信息
                }
            }

            else {
                setColor(4);    // 设置颜色为红色
                cout << "未找到该日期的销售记录。" << endl;
            }
        }
    }

void deleteSalesRecord(Inventory& inventory) {
    int id;
    cout << "请输入要删除的销售记录 ID: ";
    cin >> id;

    SalesRecord* record = inventory.findSalesRecordById(id);
    if (record) {
        // 更新库存数量
        int videoIDPR = record->getVideoTape().getId();    // 获取影片 ID
        int oldQuantity = record->getQuantitySold();
        inventory.salesVideoTape(videoIDPR, -oldQuantity);    // 撤销销售数量

        // 删除销售信息
        inventory.removeSalesRecord(id);
        setColor(2);
        cout << "销售信息已删除！" << endl;
    }
    else {
        setColor(4);
        cout << "找不到该销售信息！" << endl;
    }
}


void addSalesRecord(Inventory& inventory) {
    int id, quantitySold, videoId;
    string saleDate;
    float price;

    cout << "请输入销售记录 ID: ";
    cin >> id;
```

```
cout << "请输入关联的录像带 ID: ";
cin >> videoId;
cout << "请输入销售数量: ";
cin >> quantitySold;
cin.ignore();    //  忽略换行符
cout << "请输入销售日期: ";
getline(cin, saleDate);
cout << "请输入销售价格: ";
cin >> price;

if (quantitySold < 0) {
        setColor(4);
        cout << "销售数量必须是正整数" << endl;
        return;
}
else if (price < 0) {
        setColor(4);
        cout << "销售价格必须是正数" << endl;
        return;
}

SalesRecord* record = inventory.findSalesRecordById(id);
if (!record) {    //  检查销售记录是否已存在
        VideoTape* video = inventory.findVideoTapeById(videoId);
        if (video) {
                SalesRecord newSales(id, *video, quantitySold, saleDate, price);
                inventory.addSalesRecord(newSales);
                //  更新录像带库存数量
                int i = inventory.salesVideoTape(videoId, quantitySold);
                if (i) {
```

```
                inventory.removeSalesRecord(id);
            }
            else {
                setColor(2);
                cout << "销售信息已添加！ " << endl;
            }
        }
        else {
            setColor(4);
            cout << "录像带 ID 不存在！ " << endl;
        }
    }
    else {
        setColor(4);
        cout << "此 ID 所对应的销售记录已存在,无法添加。" << endl;
    }
}

void modifySalesRecord(Inventory& inventory) {
    int id;
    cout << "请输入要修改的销售记录 ID: ";
    cin >> id;
    cin.ignore();    //  忽略换行符

    SalesRecord* record = inventory.findSalesRecordById(id);
    if (record) {
        int newQuantitySold;
        float newPrice;
        cout << "请输入新的销售数量: ";
        cin >> newQuantitySold;
        cout << "请输入新的销售价格: ";
        cin >> newPrice;

        if (newQuantitySold < 0) {
            setColor(4);
            cout << "销售数量必须是正整数" << endl;
            return;
        }
        else if (newPrice < 0) {
            setColor(4);
            cout << "销售价格必须是正数" << endl;
            return;
        }

        // 更新库存数量
        int videoIDPR = record->getVideoTape().getId();   // 获取影片 ID
        int oldQuantity = record->getQuantitySold();
        inventory.salesVideoTape(videoIDPR, -oldQuantity);   // 撤销原销售数量
        int i = inventory.salesVideoTape(videoIDPR, newQuantitySold);   // 设置新
销售数量
        if (i) {
            inventory.salesVideoTape(videoIDPR, oldQuantity);   // 还原原销售
数量
            setColor(4);
            cout << "无法修改！" << endl;
            cout << "销售数量必须不大于录像带数量！" << endl;
        }
        else {
            // 更新销售信息
            record->setQuantitySold(newQuantitySold);
            record->setPrice(newPrice);
            setColor(2);
            cout << "销售信息已修改！" << endl;
        }
    }
    else {
        setColor(4);
        cout << "找不到该销售信息！" << endl;
    }
}




void runProgram() {
    setColor(3);
    cout << "欢迎使用录像带商店进销存管理系统" << endl;
    cout << "制作人：梁桐     班级：计算机 2203 " << endl;
    cout << "密码是我的学号：" ;
    setColor(6);
    cout << "2209060322" << endl;
    setColor(3);
    cout << "------------------------------------------" << endl;
    cout << "请输入密码启动程序，祝您使用愉快！" << endl;
    long long passWord;
    cin >> passWord;
    if (passWord != 2209060322) {
        setColor(4);
```

```
		cout <<"密码错误！ " << endl;
	}
	else {
		// 创建库存对象
		Inventory inventory;
		showMenu();   // 显示功能菜单
		while (true) {
			setColor(3);
			cout << "输入 20 显示功能菜单" << endl;
			cout << "输入 21 显示格式说明书" << endl;
			cout << "请输入数字选择你要使用的功能：" << endl;
			setColor(7);
			int choice;
			cin >> choice;   // 获取用户选择

			switch (choice) {
			case 1: {
				searchVideoTape(inventory);   // 调用提取的函数
				break;
			}
			case 2: { // 添加录像带记录
				addVideoTapeRecord(inventory);
				break;
			}
			case 3: { // 修改录像带记录
				modifyVideoTapeRecord(inventory);
				break;
			}
			case 4: { // 删除录像带记录
				deleteVideoTapeRecord(inventory);
    break;
}
case 5: { // 查看进货记录
    viewPurchaseRecords(inventory);
    break;
}
case 6: { // 添加进货记录
    addPurchaseRecord(inventory);
    break;
}
case 7: { // 修改进货记录
    modifyPurchaseRecord(inventory);
    break;
}
case 8: { // 删除进货记录
    deletePurchaseRecord(inventory);
    break;
}
case 9: { // 查看销售信息
    viewSalesRecord(inventory);
    break;
}
case 10: { // 添加销售信息
    addSalesRecord(inventory);
    break;
}
case 11: { // 修改销售信息
    modifySalesRecord(inventory);
    break;
}
```

```
case 12: { //  删除销售信息
        deleteSalesRecord(inventory);
        break;
}
case 13: { //  通过录像带 ID 排序
        inventory.sortVideoTapeById();
        break;
}
case 14: {
        //按照进货记录 ID 排序
        inventory.sortPurchById();
        break;
}
case 15: {
        //按照销售记录 ID 排序
        inventory.sortSaleById();
        break;
}
case 16: {
        //按照录像带副本数排序
        inventory.sortVideoTapeByNum();
        break;
}
case 17: {
        //按照进货价格排序
        inventory.sortPurchByPri();
        break;
}
case 18: {
        //按照销售价格排序
        inventory.sortSaleByPri();
        break;
}
case 19: { //  查看所有记录
        setColor(2);
        inventory.displayInventory();
        break;
}
case 20: {
        showMenu();    //  显示功能菜单
        break;
}
case 21: {
        setColor(6);
        cout << "----------------------------------------" << endl;
        cout << "---格式说明书---" << endl;
        cout << "所有功能选择输入必须为正整数" << endl;
        cout << "所有 ID 必须为正整数" << endl;
        cout << "所有价格须为正实数" << endl;
        cout << "所有副本数量输入必须为正整数" << endl;
        cout << "所有录像带名称中空格使用‘-’字符替换" << endl;
        cout << "所有日期格式须为 xxxx-yy-zz" << endl;
        cout << "VedioTape.txt 文件内每行内容格式应为" << endl;
        cout << "ID  电影名称  副本数" << endl;
        cout << "VedioTape.txt文件内不得出现非ASCII码字符" << endl;
        cout << "----------------------------------------" << endl;
        break;
}
```

```
            case 0: { // 退出
                setColor(2);
                cout << "谢谢使用，程序退出！" << endl;
                return;
            }
            default:
                setColor(4);
                cout << "无效输入，请重新选择！" << endl;
                break;
            }
        }
    }
}
```

源文件 10LTvector.h

```
#ifndef LTVECTOR_H
#define LTVECTOR_H

#include <stdexcept>
#include <cstddef> // for size_t

template <typename T>
class LTvector {
private:
    T* data;          // 动态分配的数组
    size_t capacity;  // 容量
    size_t size;      // 当前元素个数

    void resize(size_t new_capacity);

public:
    LTvector();                                  // 默认构造函数
    LTvector(const LTvector& other);        // 拷贝构造函数
    LTvector& operator=(const LTvector& other); // 赋值运算符
    ~LTvector();                                 // 析构函数

    void push_back(const T& value);          // 添加元素
    void pop_back();                             // 移除最后一个元素
    T& operator[](size_t index);              // 下标运算符
    const T& operator[](size_t index) const;

    size_t get_size() const;                  // 获取当前大小
    size_t get_capacity() const;             // 获取容量
    bool empty() const;                          // 判断是否为空
    void clear();                                // 清空所有元素
};

// 默认构造函数
template <typename T>
LTvector<T>::LTvector() : data(nullptr), capacity(0), size(0) {}

// 拷贝构造函数
template <typename T>
LTvector<T>::LTvector(const LTvector& other)
    : data(nullptr), capacity(other.capacity), size(other.size) {
    data = new T[capacity];
    for (size_t i = 0; i < size; ++i) {
        data[i] = other.data[i];
    }
}
```

```
// 赋值运算符
template <typename T>
LTvector<T>& LTvector<T>::operator=(const LTvector& other) {
    if (this != &other) {
        delete[] data;
        capacity = other.capacity;
        size = other.size;
        data = new T[capacity];
        for (size_t i = 0; i < size; ++i) {
            data[i] = other.data[i];
        }
    }
    return *this;
}

// 析构函数
template <typename T>
LTvector<T>::~LTvector() {
    delete[] data;
}

// 动态调整容量
template <typename T>
void LTvector<T>::resize(size_t new_capacity) {
    T* new_data = new T[new_capacity];
    for (size_t i = 0; i < size; ++i) {
        new_data[i] = data[i];
    }
    delete[] data;
    data = new_data;
```

```
        capacity = new_capacity;
}

// 添加元素
template <typename T>
void LTvector<T>::push_back(const T& value) {
        if (size == capacity) {
                resize(capacity == 0 ? 1 : capacity * 2);
        }
        data[size++] = value;
}

// 移除最后一个元素
template <typename T>
void LTvector<T>::pop_back() {
        if (empty()) {
                throw std::out_of_range("Pop from empty LTvector");
        }
        --size;
}

// 下标运算符（可修改）
template <typename T>
T& LTvector<T>::operator[](size_t index) {
        if (index >= size) {
                throw std::out_of_range("Index out of range");
        }
        return data[index];
}
```

```
// 下标运算符（只读）
template <typename T>
const T& LTvector<T>::operator[](size_t index) const {
    if (index >= size) {
        throw std::out_of_range("Index out of range");
    }
    return data[index];
}

// 获取当前大小
template <typename T>
size_t LTvector<T>::get_size() const {
    return size;
}

// 获取容量
template <typename T>
size_t LTvector<T>::get_capacity() const {
    return capacity;
}

// 判断是否为空
template <typename T>
bool LTvector<T>::empty() const {
    return size == 0;
}

// 清空所有元素
template <typename T>
void LTvector<T>::clear() {
    size = 0;
}

#endif // LTVECTOR_H
```

源文件 11Inventory.h

```
#ifndef INVENTORY_H
#define INVENTORY_H

#include "VideoTape.h"
#include "PurchaseRecord.h"
#include "SalesRecord.h"
#include "LTvector.h"
using namespace std;

class Inventory {
private:
    LTvector<VideoTape> videoTapes;                  // 录像带列表
    LTvector<PurchaseRecord> purchaseRecords;    // 进货记录列表
    LTvector<SalesRecord> salesRecords;            // 销售记录列表

public:
    // 添加录像带、进货记录、销售记录的方法
    void addVideoTape(VideoTape video);
    void removeVideoTape(int videoId);
    void addPurchaseRecord(PurchaseRecord record);
    void removePurchaseRecord(int recordId);
    void addSalesRecord(SalesRecord record);
    void removeSalesRecord(int recordId);
    // 查找录像带、进货记录、销售记录的方法
    VideoTape* findVideoTapeById(int videoId);
    LTvector<VideoTape> findVideoTapesByName(const string& videoName);

    // 在 Inventory 类中添加根据进货日期查询进货记录的函数
    LTvector<PurchaseRecord> findPurchaseRecordsByDate(const string& purchaseDate);

    // 在 Inventory 类中添加根据销售日期查询销售记录的函数
    LTvector<SalesRecord> findSalesRecordsByDate(const string & saleDate);

    PurchaseRecord* findPurchaseRecordById(int recordId);
    SalesRecord* findSalesRecordById(int recordId);

    // 修改录像带、进货记录、销售记录的方法
    void modifyVideoTape(int videoId, string newMovieName, int newCopiesAvailable);
    void modifyPurchaseRecord(int recordId, int newQuantity, float newPrice);
    void modifySalesRecord(int recordId, int newQuantitySold, float newPrice);

    // 显示库存信息
    void displayInventory() const;

    // 进货录像带，增加副本数
    void purchaseVideoTape(int videoId, int quantity);
    // 卖货录像带，增加副本数
    int salesVideoTape(int videoId, int quantity);
    //按照录像带 ID 排序
    void sortVideoTapeById();
    //按照进货记录 ID 排序
    void sortPurchById();
    //按照销售记录 ID 排序
    void sortSaleById();
    //按照录像带副本数排序
    void sortVideoTapeByNum();
    //按照进货价格排序
    void sortPurchByPri();
    //按照销售价格排序
    void sortSaleByPri();
};

#endif // INVENTORY_H
```

源文件 12Inventory.cpp

```
#include "VideoStore.h"
using namespace std;

// 添加录像带
void Inventory::addVideoTape(VideoTape video) {
    videoTapes.push_back(video);
}

// 删除录像带
void Inventory::removeVideoTape(int videoId) {
    for (size_t i = 0; i < videoTapes.get_size(); ++i) {
        if (videoTapes[i].getId() == videoId) {
            // 使用手动移位操作来模拟 std::vector 的 erase 功能
            for (size_t j = i; j < videoTapes.get_size() - 1; ++j) {
                videoTapes[j] = videoTapes[j + 1];
            }
            videoTapes.pop_back(); // 移除最后一个重复元素
            return;
        }
    }
}


// 添加进货记录
void Inventory::addPurchaseRecord(PurchaseRecord record) {
    purchaseRecords.push_back(record);
}

// 删除进货记录
void Inventory::removePurchaseRecord(int recordId) {
    for (size_t i = 0; i < purchaseRecords.get_size(); ++i) {
        if (purchaseRecords[i].getId() == recordId) {
            // 使用手动移位操作来模拟 std::vector 的 erase 功能
            for (size_t j = i; j < purchaseRecords.get_size() - 1; ++j) {
                purchaseRecords[j] = purchaseRecords[j + 1];
            }
            purchaseRecords.pop_back(); // 移除最后一个重复元素
            return;
        }
    }
}


// 添加销售记录
void Inventory::addSalesRecord(SalesRecord record) {
    salesRecords.push_back(record);
}

// 删除销售记录
void Inventory::removeSalesRecord(int recordId) {
    for (size_t i = 0; i < salesRecords.get_size(); ++i) {
        if (salesRecords[i].getId() == recordId) {
            // 使用手动移位操作来模拟 std::vector 的 erase 功能
            for (size_t j = i; j < salesRecords.get_size() - 1; ++j) {
                salesRecords[j] = salesRecords[j + 1];
            }
            salesRecords.pop_back(); // 移除最后一个重复元素
            return;
        }
    }
}

// 查找录像带
VideoTape* Inventory::findVideoTapeById(int videoId) {
    for (size_t i = 0; i < videoTapes.get_size(); ++i) {
        if (videoTapes[i].getId() == videoId) {
            return &videoTapes[i];
        }
    }
    return nullptr;
}

// 根据录像带名称查找录像带
LTvector<VideoTape> Inventory::findVideoTapesByName(const string& videoName) {
    LTvector<VideoTape> foundVideos;
    for (size_t i = 0; i < videoTapes.get_size(); ++i) {
        if (videoTapes[i].getMovieName() == videoName) {    // 如果录像带名称匹配
            foundVideos.push_back(videoTapes[i]);    // 将该录像带添加到返回的结果中
        }
    }
    return foundVideos;    // 返回找到的录像带列表
}

// 查找进货记录
PurchaseRecord* Inventory::findPurchaseRecordById(int recordId) {
    for (size_t i = 0; i < purchaseRecords.get_size(); ++i) {
        if (purchaseRecords[i].getId() == recordId) {
            return &purchaseRecords[i];
        }
    }
    return nullptr;
}

// 根据进货日期查询进货记录
LTvector<PurchaseRecord> Inventory::findPurchaseRecordsByDate(const string& purchaseDate) {
    LTvector<PurchaseRecord> result;
    for (size_t i = 0; i < purchaseRecords.get_size(); ++i) {
        if (purchaseRecords[i].getPurchaseDate() == purchaseDate) {
            result.push_back(purchaseRecords[i]);
        }
    }
    return result;
```

```
}

// 根据销售日期查询销售记录
LTvector<SalesRecord> Inventory::findSalesRecordsByDate(const string& saleDate) {
    LTvector<SalesRecord> result;
    for (size_t i = 0; i < salesRecords.get_size(); ++i) {
        if (salesRecords[i].getSaleDate() == saleDate) {
            result.push_back(salesRecords[i]);
        }
    }
    return result;
}

// 查找销售记录
SalesRecord* Inventory::findSalesRecordById(int recordId) {
    for (size_t i = 0; i < salesRecords.get_size(); ++i) {
        if (salesRecords[i].getId() == recordId) {
            return &salesRecords[i];
        }
    }
    return nullptr;
}

// 修改录像带信息
void Inventory::modifyVideoTape(int videoId, string newMovieName, int newCopiesAvailable) {
    VideoTape* video = findVideoTapeById(videoId);
    if (video) {
        video->setMovieName(newMovieName);
        video->setCopiesAvailable(newCopiesAvailable);
        }
}

// 修改进货记录
void Inventory::modifyPurchaseRecord(int recordId, int newQuantity, float newPrice)
{
        PurchaseRecord* record = findPurchaseRecordById(recordId);
        if (record) {
                record->setQuantity(newQuantity);
                record->setPrice(newPrice);
        }
}

// 修改销售记录
void Inventory::modifySalesRecord(int recordId, int newQuantitySold, float newPrice)
{
        SalesRecord* record = findSalesRecordById(recordId);
        if (record) {
                record->setQuantitySold(newQuantitySold);
                record->setPrice(newPrice);
        }
}
// 显示库存信息
void Inventory::displayInventory() const {
        cout << "录像带记录信息：  " << endl;
        for (size_t i = 0; i < videoTapes.get_size(); ++i) {
                videoTapes[i].displayInfo();
        }
        cout << endl;
```

```
    cout << "进货记录信息：" << endl;
    for (size_t i = 0; i < purchaseRecords.get_size(); ++i) {
        purchaseRecords[i].displayPurchaseInfo();
        cout << endl;
    }
    cout << endl;

    cout << "销售记录信息：  " << endl;
    for (size_t i = 0; i < salesRecords.get_size(); ++i) {
        salesRecords[i].displaySalesInfo();
        cout << endl;
    }
}

// 进货录像带，增加副本数
void Inventory::purchaseVideoTape(int videoId, int quantity) {
    VideoTape* tape = findVideoTapeById(videoId);   // 查找录像带
    if (tape != nullptr) {
        int currentCopies = tape->getCopiesAvailable();   // 获取当前可用副本数
        tape->setCopiesAvailable(currentCopies + quantity);   // 更新副本数
        cout << "Successfully purchased " << quantity << " copies of \"" <<
tape->getMovieName() << "\"." << endl;
    }
    else {
        cout << "VideoTape with ID " << videoId << " not found." << endl;
    }
}

// 卖货录像带，减少副本数
int Inventory::salesVideoTape(int videoId, int quantitySold) {
    VideoTape* tape = findVideoTapeById(videoId);   // 查找录像带
    if (tape != nullptr) {
        int currentCopies = tape->getCopiesAvailable();   // 获取当前可用副本数
        if (quantitySold <= currentCopies) {
            tape->setCopiesAvailable(currentCopies - quantitySold);   // 更新副
本数
            return 0;
        }
        else {
            setColor(4);
            cout << "出售量大于存货量，无法添加！ " << endl;
            return 1;
        }
    }
    else {
        cout << "VideoTape with ID " << videoId << " not found." << endl;
    }
}

void Inventory::sortVideoTapeById() {
    quickSort(videoTapes, 0, videoTapes.get_size() - 1);   // 使用快速排序
    setColor(2);
    cout << "按照录像带 ID 排序完成！ " << endl;
}

void Inventory::sortPurchById() {
    quickSort(purchaseRecords, 0, purchaseRecords.get_size() - 1);   // 使用快速
排序
    setColor(2);
    cout << "按照进货记录 ID 排序完成！" << endl;
}

void Inventory::sortSaleById() {
    quickSort(salesRecords, 0, salesRecords.get_size() - 1);    // 使用快速排序
    setColor(2);
    cout << "按照销售记录 ID 排序完成！" << endl;
}

void Inventory::sortVideoTapeByNum() {
    quickSortNum(videoTapes, 0, videoTapes.get_size() - 1);    // 使用快速排序
    setColor(2);
    cout << "按照录像带副本数排序完成！" << endl;
}

void Inventory::sortPurchByPri() {
    quickSortPri(purchaseRecords, 0, purchaseRecords.get_size() - 1);    // 使用快
速排序
    setColor(2);
    cout << "按照进货记录价格排序完成！" << endl;
}

void Inventory::sortSaleByPri() {
    quickSortPri(salesRecords, 0, salesRecords.get_size() - 1);    // 使用快速排序
    setColor(2);
    cout << "按照销售记录价格排序完成！" << endl;
}
```