数学与计算机学院

数据结构课程设计报告

2019 ~2020 学年第一学期

课 程 设 计 题 目 仓库管理

学 生 姓 名 _____

指 导 教 师 _____

评 定 成 绩 A

填 写 时 间 2020.11.28

1 题目与要求

1.1 问题描述

某电子公司仓库中有若干批次的同一种电脑，按价格、数量来存储。
要求实现功能:

1. 初始化 n 批不同价格电脑入库；
2. 出库：销售 m 台价格为 p 的电脑；
3. 入库：新到 m 台价格为 p 的电脑；
4. 盘点：电脑的总台数，总金额，最高价，最低价，平均价格。

注：每个数据元素含有价格与数量；同一价格的电脑存储为一个数据元素。

1.2 本系统涉及的知识点

仓库管理系统可以使用顺序表、有序表、单链表、有序循环链表等实现，本设计采用有序表实现。

所谓有序表，是指这样的线性表，其中所有的元素以递增或递减方式有序排列。

首先要指出的是，有序表是基于顺序表而延伸出来的一种数据结构，其共同点是用一组地址连续的存储单元依次存储线性表的数据元素。

其次，是有序表的独特之处，它其中的数据元素按照一定的顺序有序排列。

仓库管理系统适合采用有序表，原因是可以按照商品的价格或数量进行有序排列，方便用户比对价格、数量。

1.3 功能要求

1. 初始化仓库：初始化 n 批不同型号的电脑入库；
2. 入库：新到 m 台价格为 p 的电脑；
3. 出库：销售 m 台价格为 p 的电脑；
4. 盘点仓库：列出仓库的关键数据：电脑的总台数、总金额、最高价、平均价格等；

按照有序表的特点以及所使用的编程语言（C++）的特性，本程序还提供了以下功能：

5. 查询某一型号的电脑的价格、数量；
6. 重新对仓库数据按照一定规则排序；
7. 导出仓库数据到外部文件；
8. 从外部文件导入数据，以初始化仓库。

2 功能设计

2.1 数据结构定义

一、基本数据元素：电脑

```
typedef struct computer {
    char type[50]; // 型号
    double price;  // 价格
    int number;    // 数量
```

```
} Computer, ElemType;
```

基本数据元素电脑（Computer/ElemType）采用结构体表示，用于存储某一类电脑的信息：型号（type[50]）、价格（price）、库存数量（number）。

二、数据结构：有序表

```
typedef struct {
    ElemType *elem;      // 基地址
    int length;          // 当前有效数据的个数
    int listsize;        // 当前存储容量
    bool isInit{false}; // 有序表是否已经初始化
    int sortWay{1};      // 有序表的排序方式
} SqList;
```

数据结构有序表（SqList）由以下几个部分组成：

1. 数组指针 elem 指示有序表的基地址；
2. length 指示有序表的当前有效数据个数（长度）；
3. listsize 指示有序表当前分配的存储空间大小，一旦因插入数据元素（Computer）而空间不足时，可进行再分配；
4. isInit 指示有序表是否已经初始化（即是否有一个确定的基地址）；
5. sortWay 指示有序表的排序方式，按照值的不同，对应的有序表排序方式也不同。本程序具体设计了以下四种排序方式：1-按照价格升序、2-按照价格降序、3-按照数量升序、4-按照数量降序。

整体的数据结构如下图所示：

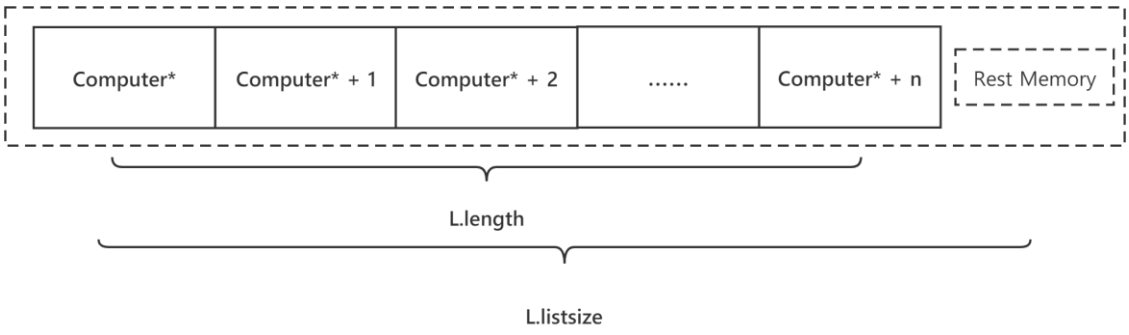

图 1.1 有序表数据结构

2.2 模块图

一、入库

入库有两种方式，一是在仓库中已有和待入库电脑型号相同的数据，此时，检查给出的价格是否与仓库中一致，若一致，同意用户的入库操作。然后只需更改仓库中此种电脑型号的数量。

示意图如下：

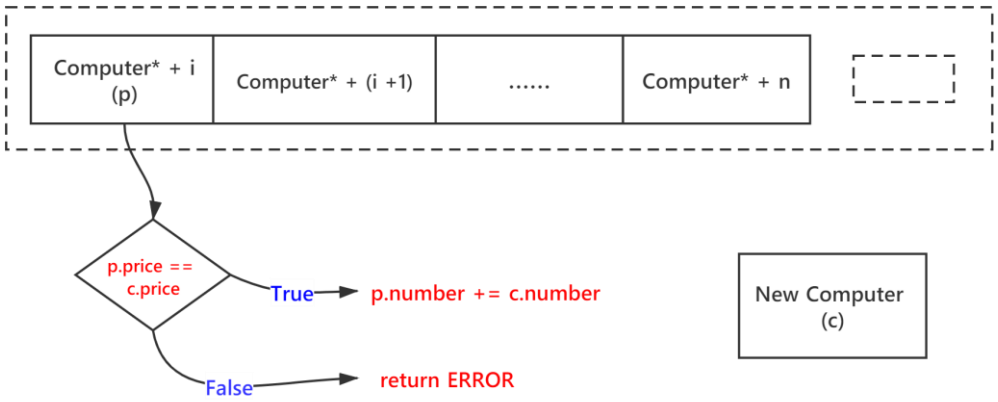

图 2.1 入库仓库中已有型号的电脑

第二种方式，入库一种新型号的电脑，则应该按照有序表的排序方式（sortWay）在正确位置插入元素。

示意图如下：

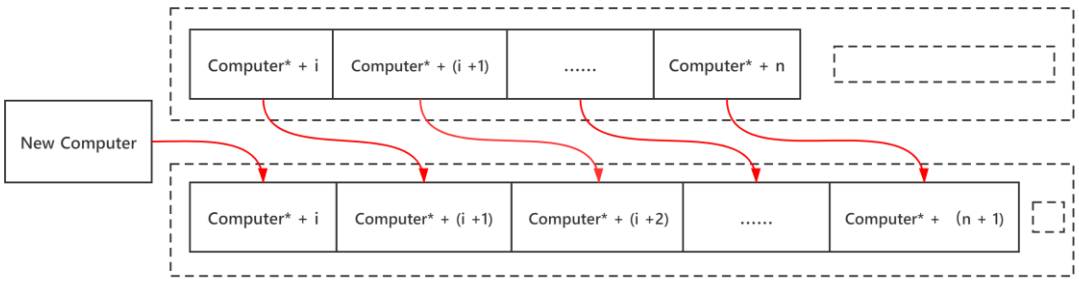

图 2.2 入库一种新型号的电脑

二、出库

可以出库的前提是，在仓库中有待出库型号的电脑且仓库中的库存数量大于等于待出库数量。

第一类情况，待出库电脑的数量在库存中充足（即库存数量大于待出库数量），此时只需更改相应的数据元素。

示意图如下：

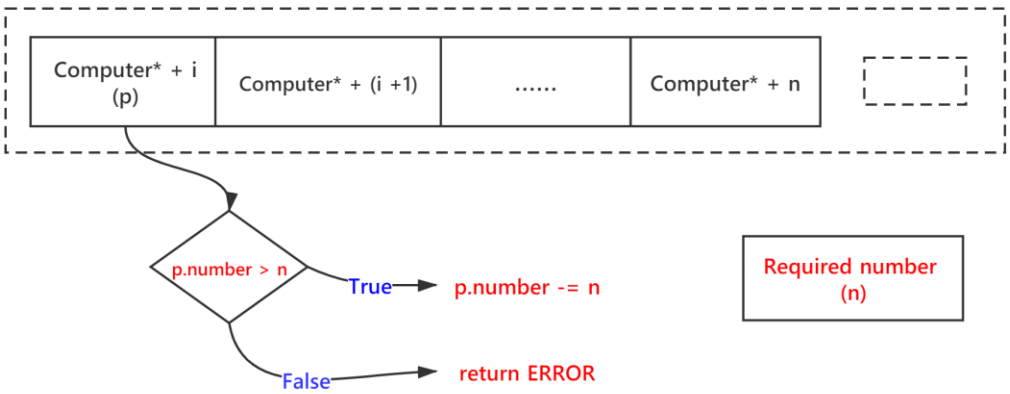

图 2.3 出库电脑数量充足

第二类情况，此种型号的电脑恰好全部出库完，则需要删除相应的数据元素。示意图如下：

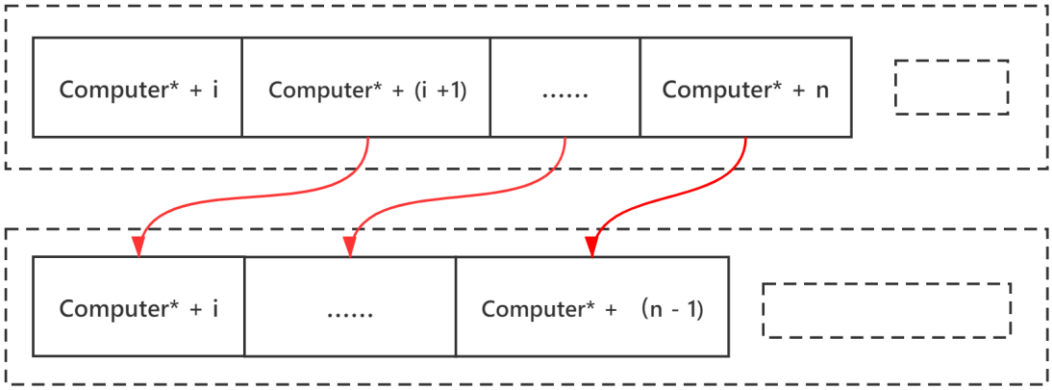

图 2.4 此种型号的电脑恰好全部出库完

3 功能代码

3.1 初始化动态顺序表

有序表的数据结构是基于顺序表实现的，所以在进行一切操作前，应当初始化一个空的动态顺序表。

代码如下：

```
Status InitSqList(SqList &L, int n) {
    auto listsize{((n / 10) + 1) * 10}; // 确定顺序表初始内存占用空间
    L.elem = new ElemType[listsize];    // 分配基地址、顺序表内存
    if (!L.elem)    // 内存不足
        return OVERFLOW;
    L.length = 0;   // 此时顺序表还没有元素，L.length 为0
    L.listsize = listsize;
    return OK;
}
```

有序表初始分配的内存空间（listsize）并没有简单采用一个常数大小（如 10 个 ElemType 字节），这样不用限制用户输入的电脑类型最大个数，对用户友好。具体是通过给定的 n，计算表达式((n/10)+1)*10，使得初始分配的内存为 10 的整数倍大小的 ElemType 字节，比如 n 为 12，表达式((n/10)+1)*10 则为 20。

需要注意的是，初始化动态顺序表时，还暂无数据元素，L.length 应为 0。

3.2 创建（初始化）仓库

创建仓库的前提是，已经初始化过一个空的动态顺序表，这个检测可以在主函数中完成。

创建有序表式的仓库步骤：

1. 输入 n 个数据元素，存储到顺序表中；
2. 对已创建的顺序表按照有序表的排序方式（L.sortWay）进行快速排序。
3. 更新有序表的相应参数（length、isInit、sortWay）

代码如下：

```
void CreateWarehouse(SqList &L, int n, int sort_way) {
    cout << "请输入这" << n << "种电脑各自的型号、单价、总数:(以空格分隔)"
<< endl;
    // 输入 n 个数据元素
    for (int i = 0; i < n; i++)
        cin >> L.elem[i].type >> L.elem[i].price >> L.elem[i].number;
    // 按照 sort_way 对刚创建的顺序表排序
    switch (sort_way) {
        case 1:
            // 按价格升序
            sort(L.elem, L.elem + n, cmp1);
            break;
        case 2:
            // 按价格降序
            sort(L.elem, L.elem + n, cmp2);
            break;
        case 3:
            // 按数量升序
            sort(L.elem, L.elem + n, cmp3);
            break;
        case 4:
            // 按数量降序
            sort(L.elem, L.elem + n, cmp4);
            break;
        default:
            break;
    }
    // 更新 L 的参数
    L.sortWay = sort_way;
    L.length = n;
    L.isInit = true;
}
```

如果使用文件流创建仓库，那么代码略有不同，这部分属于 C++的知识。

不同代码（输入 n 数据元素）如下：

```
string s;
// 从文件流输入 n 个数据元素
for (int i = 0; i < n; i++) {
    ImportFile >> L.elem[i].type >> L.elem[i].price >> L.elem[i].number;
    getline(ImportFile, s);
}
```

3.3 显示仓库

顾名思义，就是将仓库的数据打印出来。

代码如下：

```
// 显示仓库
void PrintWarehouse(SqList L) {
    if (!L.length)
        cout << "当前仓库没有数据!" << endl;
    else {
        cout << "当前仓库数据如下:" << endl;
        string sort_way;
        switch (L.sortWay) {
            case 1:
                sort_way = "(按照价格升序)";
                break;
            case 2:
                sort_way = "(按照价格降序)";
                break;
            case 3:
                sort_way = "(按照数量升序)";
                break;
            case 4:
                sort_way = "(按照数量降序)";
                break;
            default:
                break;
        }
        cout << sort_way << endl;
        cout << "-------------------------------------------------------
" << endl;
        cout << setLeft
             << setw(5) << "序号"
             << setw(20) << "型号"
             << setw(15) << "单价"
             << setw(15) << "数量"
             << endl;
        cout << "-------------------------------------------------------
" << endl;
        for (int i = 0; i < L.length; i++)
            cout << setLeft
                 << setw(5) << i + 1
                 << setw(20) << L.elem[i].type
                 << setw(15) << L.elem[i].price
                 << setw(15) << L.elem[i].number
```

```
                << endl;
        cout << "-----------------------------------------------------
" << endl;
    }
}
```

3.4 入库

入库要考虑两种情况，第一种情况，仓库中已经有和待入库电脑相同型号的数据元素，那么需要先比较价格是否一致（不一致拒绝用户的入库操作），然后更改仓库中此种电脑数据元素的数量值。

第二种情况入库一种新型号的电脑，那么则要插入一个新的数据元素（Computer）到仓库中，并且在插入之前还要检查有序表的内存空间（listsize）是否充足，如果不足，则需要再分配有序表的内存大小，即为顺序表增加一个大小为存储 LISTINCREMENT 个数据元素的空间。然后，根据有序表的排序方式（L.sortWay），找到可以插入元素的地址。插入时，先将该地址及以后的地址全部后移一位，接着将当前地址写入要入库的数据元素。

代码如下：

```
Status Enter(SqList &L, const Computer &c) {
    // 寻找仓库中是否已经有和c.type 同类型的电脑
    for (int i = 0; i < L.length; i++) {
        if (strcmp(c.type, L.elem[i].type) == 0) {
            if (c.price == L.elem[i].price) {   // 检查价格价格是否与
c.price 相等
                L.elem[i].number += c.number;
                return OK;
            } else {
                cout << "提示:" << endl;
                showInfo(L.elem[i]);
                return ERROR;
            }

        }
    }

    // 入库一个新类型的电脑
    if (L.length >= L.listsize) {   // 顺序表占用空间不足
        // 申请新的基地址、内存空间
        auto *newbase = (ElemType *) realloc(L.elem, (L.listsize +
LISTINCREMENT) * sizeof(ElemType));
        if (!newbase)
            return OVERFLOW;
        L.elem = newbase;   // L 的基地址更改为newbase
        L.listsize += LISTINCREMENT;
```

```
    }
    // 按照L.sort_way对插入到顺序表中
    int item{L.length}; // 确定要插入的位序
    for (int i = 0; i < L.length; i++) {
        switch (L.sortWay) {
            case 1: {
                if (c.price < L.elem[i].price) {
                    item = i;
                    goto change_sq;
                }
            }
                break;
            case 2: {
                if (c.price > L.elem[i].price) {
                    item = i;
                    goto change_sq;
                }
            }
                break;
            case 3: {
                if (c.number < L.elem[i].number) {
                    item = i;
                    goto change_sq;
                }
            }
                break;
            case 4: {
                if (c.number > L.elem[i].number) {
                    item = i;
                    goto change_sq;
                }
            }
                break;
        }
    }
    change_sq:
    ElemType *q = &L.elem[item];
    // 将q及q以后的元素全部后移一位
    for (ElemType *p = L.elem + L.length - 1; p >= q; p--)
        *(p + 1) = *p;
    // 将新元素赋给q
    *q = c;
    L.length++;
```

```
    return OK;
}
```

要注意的是如果是入库一种新型号的电脑，完成操作后，应当将有序表的长度加 1（L.length++）。

3.5 出库

出库要考虑以下几种情况：

1、 仓库中是否有待出库电脑的型号的相应数据元素，如果没有找到相应的电脑型号，则应当拒绝用户的出库操作；

2、 仓库中现有库存数量是否充足，满足出库要求，如果库存数量小于待出库数量，则应当拒绝用户的出库操作；

3、 如果库存数量大于待出库数量，那么只需更改相应数据元素的数量值；

4、 如果库存数量恰好等于待出库数量，那么此时这种型号的电脑应从仓库中“剔除”。具体操作是，找到该数据元素的地址，将该地址及以后的地址全部前移一位。最后，更新 L.length 参数。

代码如下：

```
Status Out(SqList &L, const char *type, int num) {
    // 确定要出库元素的位序
    int item{L.length + 1};
    for (int i = 0; i < L.length; i++) {
        if (strcmp(type, L.elem[i].type) == 0) {
            item = i;
            break;
        }
    }
    // 没有找到要出库元素的位序，说明在仓库中没有元素，返回错误
    if (item > L.length)
        return -1;

    // 确定要出库元素现有库存的数量是否可以出库
    if (num < L.elem[item].number) {
        // 正常出库
        L.elem[item].number -= num;
        return OK;
    } else if (num == L.elem[item].number) {
        // 全部出库
        cout << "提示:" << endl
             << L.elem[item].type << "型号的电脑已全部出库!" << endl;

        // 将 p 及 p 以后的元素全部后移一位
        for (ElemType *p = &L.elem[item + 1]; p <= &L.elem[L.length -
1]; p++) {
            *(p - 1) = *p;
```

```
        }
        L.length--;
        return OK;
    } else {
        // 库存不足
        cout << "提示:" << endl;
        showInfo(L.elem[item]);
        return ERROR;
    }
}
```

3.6 查询电脑数据

查询电脑数据，只需从基地址开始遍历，直到找到数据元素的型号和待查询的型号相同，那么返回电脑信息。

可以定义一个 item 参数，初始值为 L.length + 1，如果找到数据元素，则将其更新为数据元素的逻辑位序。如果遍历结束，item 都未更新，说明未找到。因为数据元素的最大逻辑位序是有序表的表长（L.length），所以判断是否找到数据元素的条件是：item > L.length。

代码如下：

```
void getInfo(SqList L, const char *type) {
    // 确定要查找的电脑的元素位序
    int item{L.length + 1};
    for (int i = 0; i < L.length; i++) {
        if (strcmp(type, L.elem[i].type) == 0) {
            item = i;
            break;
        }
    }
    if (item > L.length)
        cout << "没有找到" << type << "型号的电脑" << endl;
    else {
        cout << "信息如下:" << endl;
        showInfo(L.elem[item]);
    }
}
```

3.7 盘点仓库

从基地址开始遍历，求得关键数据：总台数（numSum）、总金额（priceSum）、最高价（priceMax）、最低价（priceMin）、平均价（priceAverage）。

代码如下：

```
void Check(SqList L) {
    int numSum{0};
```

```
    double priceSum{0.0}, priceMax{L.elem[0].price},
priceMin{L.elem[0].price}, priceAverage{0.0};
    vector<char *> priceMaxComputer;    // 存储价格最高的电脑的类型名
    vector<char *> priceMinComputer;    // 存储价格最低的电脑的类型名

    if (!L.length) {
        cout << "当前仓库没有数据!" << endl;
    } else {
        // 求总数、总金额、最高价、最低价
        for (int i = 0; i < L.length; i++) {
            numSum += L.elem[i].number;
            priceSum += L.elem[i].price * L.elem[i].number;
            priceMax = max(priceMax, L.elem[i].price);
            priceMin = min(priceMin, L.elem[i].price);
        }
        // 记录价格最高、最低的电脑类型名
        for (int i = 0; i < L.length; i++) {
            if (L.elem[i].price == priceMax)
                priceMaxComputer.push_back(L.elem[i].type);
            if (L.elem[i].price == priceMin)
                priceMinComputer.push_back(L.elem[i].type);
        }
        // 求平均价
        priceAverage = priceSum / numSum;
        // 输出信息
        cout << "盘点数据如下" << endl
             << "电脑的总台数: " << numSum << endl
             << "电脑的总金额: " << priceSum << endl
             << "电脑的最高价: " << priceMax << endl
             << "          + 对应的型号是: ";
        for (const auto &c:priceMaxComputer)
            cout << c << "\t";
        cout << endl;
        cout << "电脑的最低价: " << priceMin << endl
             << "          + 对应的型号是: ";
        for (const auto &c:priceMinComputer)
            cout << c << "\t";
        cout << endl;
        cout << "电脑的平均价格: " << priceAverage << endl;
    }
}
```

这里定义了两个 char*向量（priceMaxComputer、priceMinComputer），用以存储价格最高的电脑的类型名、最低的电脑的类型名，因为可能若干种型号的电脑都是最高价或最低价，所以使用 C++的向量数据结构比较合适。这样，可以显示

出最高价、最低价相应的电脑类型名，对用户友好。

3.8 重新对仓库排序

只需按照给定的排序方式，对仓库进行快速排序，然后更改 L.sortWay 参数为相应值。

代码如下：

```
cout << "请输入要重新对仓库排序的主要方式" << endl
     << "1. 按价格升序 2. 按价格降序" << endl
     << "3. 按数量升序 4. 按数量降序" << endl
     << "选择 1-4:";
auto select = input_number(1, 4);
switch (select) {
    case 1:
        sort(L.elem, L.elem + L.length, cmp1);
        break;
    case 2:
        sort(L.elem, L.elem + L.length, cmp2);
        break;
    case 3:
        sort(L.elem, L.elem + L.length, cmp3);
        break;
    case 4:
        sort(L.elem, L.elem + L.length, cmp4);
        break;
    default:
        break;
}
L.sortWay = select;
cout << "重新排序成功!" << endl;
```

其中，4 种排序方式的代码如下：

```
// computer 排序方式：按价格升序
bool cmp1(const Computer &a, const Computer &b) {
    return a.price < b.price;
}

// computer 排序方式：按价格降序
bool cmp2(const Computer &a, const Computer &b) {
    return a.price > b.price;
}

// computer 排序方式：按数量升序
bool cmp3(const Computer &a, const Computer &b) {
    return a.number < b.number;
```

```
}

// computer 排序方式：按数量降序
bool cmp4(const Computer &a, const Computer &b) {
    return a.number > b.number;
}
```

3.9 导出仓库数据到文件

原理和显示仓库大同小异，只是多了文件流的操作。
代码如下：

```
void output(SqList L, const string &filename) {
    // 定义输出文件流 filename.txt
    ofstream OutFile(filename + ".txt");
    // 输出文件头，用于下次输入时识别文件
    OutFile << "Computer Warehouse Data" << endl;
    if (!L.length)
        OutFile << "no data" << endl;
    else {
        OutFile << "The information is as follows:" << endl;
        switch (L.sortWay) {
            case 1:
                OutFile << "Sort Way: 1 (Ascending by price)" << endl;
                break;
            case 2:
                OutFile << "Sort Way: 2 (Descending by price)" << endl;
                break;
            case 3:
                OutFile << "Sort Way: 3 (Ascending by number)" << endl;
                break;
            case 4:
                OutFile << "Sort Way: 4 (Descending by number)" << endl;
                break;
            default:
                break;
        }
        OutFile << "Total Type: " << L.length << endl;

        // 输出数据信息
        for (int i = 0; i < L.length; i++)
            OutFile << L.elem[i].type << " "
                    << L.elem[i].price << " "
                    << L.elem[i].number
                    << endl;
```

```
        // 关闭文件
        OutFile.close();
    }
}
```

4 调试与测试

注意：本设计使用的编译器是 G++ 8.1.0，在 Windows 平台上编译、运行成功。编译命令为：“g++ main.cpp sqlist.cpp -o main”（不含引号）。

4.1 调试分析

一、创建仓库

为方便用户操作，本程序可以导出仓库数据到文件，然后在下次操作时，可以用此文件做仓库的初始化。这是在考虑到程序的友好性后增加的功能。

无论是手动创建还是从文件导入，方法是大同小异的。

本模块时间复杂度：$O(nlogn)$（快速排序$O(nlogn)$）

二、入库、出库、查询、盘点仓库

时间复杂度均为：$O(n)$

考虑到程序的友好性，对入库、出库、查询的拒绝操作均给出具体信息。比如，在出库时如果库存数量小于待出库数量，则先显示当前库存，然后提示库存数量不足。

三、其他

菜单和主函数设计中，由于经常需要检测输入数字是否在某一范围内，使得生成大量重复代码。于是，将这些代码设计成函数，增加了代码的可读性。并且考虑到，用户的误输入（非数字）会导致程序崩溃，设计了正则表达式匹配，检测用户输入。

函数代码如下：

```
// 输入 from 至 end 范围内的数字
int input_number(int from, int end) {
    auto select{0};
    string input;
    regex r("[0-9]*");  // 正则表达式:数字 0-9，可以出现多次
    while (true) {
        cin >> input;
        bool isNumber = regex_match(input,r);
        if (!isNumber)  // 如果 input 和正则表达式匹配
            cout << "输入错误，请输入数字" << from << "-" << end << ":";
        else {
            select = atoi(input.c_str());
            if (select < from || select > end)
                cout << "输入错误，重新输入" << from << "-" << end << ":";
            else
                break;
        }
```

```
    }
    return select;
}
```

另外一个是，如果用户在初始化仓库的情况下，进行入库、出库等操作，亦会导致程序崩溃。所以，后来在有序表中增加了一个参数 isInit，用于指示当前有序表是否已经初始化。在每次操作之前，检测是否已经初始化，这样避免了这一类型的崩溃。

4.2 用户手册

启动本程序后，按照菜单的提示进行操作即可。

注意：

1、 第一步必须先初始化仓库，否则，您无法进行其他的操作，将返回“仓库还未初始化!”。

2、 初始化仓库时，可以选择“2.从外部文件导入”，但前提是，该外部文件要存在，且是以前程序生成的导出合法文件。否则，不存在的文件将提示“没有文件/文件无法打开”，不是合法文件将提示“文件内容不符合要求”，这会导致初始化仓库失败。所以，请检查您输入的文件名和文件内容！

3、 建议在入库操作之前先显示仓库，确保增加仓库中某一电脑型号的数量时，输入的价格和仓库中一致；

4、 建议在出库操作之前先显示仓库，确保仓库中有您想要出库的电脑型号，以及其数量充足；

5、 您可以选择菜单项“7.重新对仓库排序”，按照您喜爱的方式对仓库排序，本程序提供了四种主要排序方式：1-按照价格升序、2-按照价格降序、3-按照数量升序、4-按照数量降序

6、 建议每次结束程序前，导出仓库数据到文件，这样可以保存仓库数据，方便下次操作。

4.3 测试过程

本设计给出了一个输入文件 info.txt

内容如下：

```
Computer Warehouse Data
The information is as follows:
Sort Way: 1 (Ascending by price)
Total Type: 12
RedmiBook-14 3299 30
Dell-G5 5699 14
Surface-Pro-7 5788 20
Lenovo-Air14 5999 17
Lenovo-R7000 6057 12
Lenovo-Yoga14 6299 8
Lenovo-Pro14 6299 10
MiBook-Pro-15.6 6999 13
```

```
Dell-G3 6999 15
Dell-XPS13 8888 5
MacBook-Pro-13 9199 5
MacBook-Pro-16 17399 3
```

下面是各种操作的输出结果（截图）：

1. 菜单显示

```
欢迎使用电脑仓库管理系统!

<---------------显示菜单---------------
1. 初始化仓库
2. 显示仓库
3. 入库
4. 出库
5. 查询
6. 盘点仓库
7. 重新对仓库排序
8. 导出仓库数据到文件
0. 结束程序
---------------显示菜单--------------->
```

图 4.1 菜单显示

2. 初始化仓库

```
输入0-8:1
选择创建仓库的方式
1.手动输入   2.从外部文件导入
选择1-2:2
请输入文件名(无需文件名后缀):info
导入成功!
按照文件要求：仓库数据按照价格升序排序。
您可以选择菜单2以显示仓库。
```

图 4.2 初始化仓库

3. 显示仓库

```
输入0-8:2
当前仓库数据如下:
(按照价格升序)
-------------------------------------------------------
序号 型号                单价          数量
-------------------------------------------------------
1    RedmiBook-14        3299          30
2    Dell-G5             5699          14
3    Surface-Pro-7       5788          20
4    Lenovo-Air14        5999          17
5    Lenovo-R7000        6057          12
6    Lenovo-Yoga14       6299          8
7    Lenovo-Pro14        6299          10
8    MiBook-Pro-15.6     6999          13
9    Dell-G3             6999          15
10   Dell-XPS13          8888          5
11   MacBook-Pro-13      9199          5
12   MacBook-Pro-16      17399         3
-------------------------------------------------------
```

图 4.3 显示仓库

4. 入库

入库 Dell-G5 5699 2：

```
输入0-8:3
请输入要入库的电脑型号、单价、总数:
Dell-G5 5699 2
入库成功!
```

图 4.4 入库（1）

入库后显示仓库：

序号	型号	单价	数量
1	RedmiBook-14	3299	30
2	Dell-G5	5699	16
3	Surface-Pro-7	5788	20
4	Lenovo-Air14	5999	17
5	Lenovo-R7000	6057	12
6	Lenovo-Yoga14	6299	8
7	Lenovo-Pro14	6299	10
8	MiBook-Pro-15.6	6999	13
9	Dell-G3	6999	15
10	Dell-XPS13	8888	5
11	MacBook-Pro-13	9199	5
12	MacBook-Pro-16	17399	3

图 4.5 入库（2）

入库 Huawei-MateBook-X 8999 2：

```
输入0-8:3
请输入要入库的电脑型号、单价、总数:
Huawei-MateBook-X 8999 2
入库成功!
```

图 4.6 入库（3）

入库后显示仓库：

序号	型号	单价	数量
1	RedmiBook-14	3299	30
2	Dell-G5	5699	16
3	Surface-Pro-7	5788	20
4	Lenovo-Air14	5999	17
5	Lenovo-R7000	6057	12
6	Lenovo-Yoga14	6299	8
7	Lenovo-Pro14	6299	10
8	MiBook-Pro-15.6	6999	13
9	Dell-G3	6999	15
10	Dell-XPS13	8888	5
11	Huawei-MateBook-X	8999	2
12	MacBook-Pro-13	9199	5
13	MacBook-Pro-16	17399	3

图 4.7 入库（4）

5. 出库

出库 Huawei-MateBook-X 2:

```
输入0-8:4
请输入要出库的电脑型号、数量:
Huawei-MateBook-X 2
提示:
Huawei-MateBook-X型号的电脑已全部出库!
出库成功!
```

图 4.8 出库（1）

出库后显示:

序号	型号	单价	数量
1	RedmiBook-14	3299	30
2	Dell-G5	5699	16
3	Surface-Pro-7	5788	20
4	Lenovo-Air14	5999	17
5	Lenovo-R7000	6057	12
6	Lenovo-Yoga14	6299	8
7	Lenovo-Pro14	6299	10
8	MiBook-Pro-15.6	6999	13
9	Dell-G3	6999	15
10	Dell-XPS13	8888	5
11	MacBook-Pro-13	9199	5
12	MacBook-Pro-16	17399	3

图 4.9 出库（2）

6. 查询

查询 Lenovo-Pro14

```
输入0-8:5
请输入要查询的电脑型号:Lenovo-Pro14
信息如下:
```

型号	单价	现有库存数量
Lenovo-Pro14	6299	10

图 4.10 查询

7. 盘点仓库

```
输入0-8:6
盘点数据如下
电脑的总台数：154
电脑的总金额：932567
电脑的最高价：17399
        + 对应的型号是：MacBook-Pro-16
电脑的最低价：3299
        + 对应的型号是：RedmiBook-14
电脑的平均价格：6055.63
```

图 4.11 盘点仓库

8. 重新对仓库排序

按数量升序对仓库排序：

```
输入0-8:7
请输入要重新对仓库排序的主要方式
1．按价格升序 2．按价格降序
3．按数量升序 4．按数量降序
选择1-4:3
重新排序成功!
```

图 4.12 重新对仓库排序（1）

显示重新排序后的仓库：

序号	型号	单价	数量
1	MacBook-Pro-16	17399	3
2	Dell-XPS13	8888	5
3	MacBook-Pro-13	9199	5
4	Lenovo-Yoga14	6299	8
5	Lenovo-Pro14	6299	10
6	Lenovo-R7000	6057	12
7	MiBook-Pro-15.6	6999	13
8	Dell-G3	6999	15
9	Dell-G5	5699	16
10	Lenovo-Air14	5999	17
11	Surface-Pro-7	5788	20
12	RedmiBook-14	3299	30

图 4.13 重新对仓库排序（2）

5 总结

通过本次课程设计，在数据结构课程中所学到的顺序表基础上，实现了有序表，并通过基本数据元素（Computer）和数据结构（SqList）的设计，实现了简易的仓库管理系统。

在本次课程设计中，也学习到很多的 C++的知识，如 STL 中数据结构与算法的应用（vector、sort）、文件流（fstream）。

通过设计输入输出文件的功能，解决了以往繁琐的数据输入步骤。同时，也增加了程序的友好性。

通过输入检测、正则表达式匹配，解决了非法输入导致程序崩溃的问题。

但限于编程能力，本程序仍存在某些问题，如初始化时，没有对输入相同型号的数据做处理，这是程序的改进方向。

6 参考文献

[1] 严蔚敏，吴伟民.数据结构（C 语言版）[M]. 北京：清华大学出版社，1997.

[2] 严薇敏，吴卫民.数据结构题集（C 语言版）[M].北京：清华大学出版社，1997.

[3] [美]Y.Danie Liang. C++程序设计.[M].北京：机械工业出版社，2015.

[4] Michael Wong，IBM XL 编译器中国开发团队.深入理解 C++11：C++11 新特性解析与应用.[M]. 北京：机械工业出版社，2013.

附录

源程序文件名清单，以及每个文件中的程序代码：

SqList.h （有序表头文件）

```
#ifndef SQLIST_H
#define SQLIST_H
#include <string>
#include <fstream>
using namespace std;
/********************宏定义********************/
#define OK 1        // 正确
#define ERROR 0     // 错误
#define OVERFLOW -2 // 溢出
#define LISTINCREMENT 5    // 顺序表存储空间的分配增量
/********************宏定义********************/
/********************数据结构定义********************/
using Status = int; // 状态参数
// 基本元素: computer
typedef struct computer {
    char type[50];  // 型号
    double price;   // 价格
    int number;     // 数量
} Computer, ElemType;
// 顺序表: SqList
```

```
typedef struct {
    ElemType *elem;     // 基地址
    int length;         // 当前有效数据的个数
    int listsize;       // 当前存储容量(单位是 sizeof(ElemType))
    bool isInit{false}; // 有序表是否已经初始化
    int sortWay{1};     // 有序表的排序方式
} SqList;
/********************数据结构定义********************/
/********************函数声明********************/
// 初始化一个空的动态顺序表
Status InitSqList(SqList &L, int n);
// 创建仓库
void CreateWarehouse(SqList &L, int n, int sort_way);
// 从文件流创建仓库
void CreateWarehouse(SqList &L, int n, int sort_way, ifstream &ImportFile);
// 显示仓库
void PrintWarehouse(SqList L);
// 入库
Status Enter(SqList &L, const Computer &c);
// 出库
Status Out(SqList &L, const char *type, int num);
// 盘点仓库
void Check(SqList L);
// computer 排序方式：按价格升序
bool cmp1(const Computer &a, const Computer &b);
// computer 排序方式：按价格降序
bool cmp2(const Computer &a, const Computer &b);
// computer 排序方式：按数量升序
bool cmp3(const Computer &a, const Computer &b);
// computer 排序方式：按数量降序
bool cmp4(const Computer &a, const Computer &b);
// 输出仓库数据到文件中
void output(SqList L, const string &filename);
// 从外部文件导入数据到仓库
Status Import(SqList &L, const string &filename);
// 检查外部文件头
Status getFileHead(ifstream &ImportFile, int &sort_way, int &total_type);
// 获取电脑信息
void getInfo(SqList L, const char *type);
// 显示电脑信息
void showInfo(Computer c);
/********************函数声明********************/
#endif //SQLIST_H
```

SqList.cpp （有序表函数文件）

```
#include <iostream>
#include <iomanip>
#include <algorithm>
#include <vector>
#include <cstdlib>
#include <string>
#include <cstring>
#include <fstream>
#include "sqlist.h"

using namespace std;

#define setLeft setiosflags(ios::left)

// 初始化一个空的动态顺序表
Status InitSqList(SqList &L, int n) {
    auto listsize{((n / 10) + 1) * 10}; // 确定顺序表初始内存占用空间
    L.elem = new ElemType[listsize];    // 分配基地址、顺序表内存
    if (!L.elem)    // 内存不足
        return OVERFLOW;
    L.length = 0;   // 此时顺序表还没有元素，L.length 为0
    L.listsize = listsize;
    return OK;
}

// 创建仓库
void CreateWarehouse(SqList &L, int n, int sort_way) {
    cout << "请输入这" << n << "种电脑各自的型号、单价、总数:(以空格分隔)" << endl;
    // 输入 n 个数据元素
    for (int i = 0; i < n; i++)
        cin >> L.elem[i].type >> L.elem[i].price >> L.elem[i].number;
    // 按照 sort_way 对刚创建的顺序表排序
    switch (sort_way) {
        case 1:
            // 按价格升序
            sort(L.elem, L.elem + n, cmp1);
            break;
        case 2:
            // 按价格降序
            sort(L.elem, L.elem + n, cmp2);
            break;
        case 3:
            // 按数量升序
            sort(L.elem, L.elem + n, cmp3);
```

```
            break;
        case 4:
            // 按数量降序
            sort(L.elem, L.elem + n, cmp4);
            break;
        default:
            break;
    }
    // 更新L的参数
    L.sortWay = sort_way;
    L.length = n;
    L.isInit = true;
}

// 从文件流创建仓库
void CreateWarehouse(SqList &L, int n, int sort_way, ifstream &ImportFile) {
    string s;
    // 从文件流输入n个数据元素
    for (int i = 0; i < n; i++) {
        ImportFile >> L.elem[i].type >> L.elem[i].price >> L.elem[i].number;
        getline(ImportFile, s);
    }
    // 按照sort_way对刚创建的顺序表排序
    switch (sort_way) {
        case 1:
            sort(L.elem, L.elem + n, cmp1);
            break;
        case 2:
            sort(L.elem, L.elem + n, cmp2);
            break;
        case 3:
            sort(L.elem, L.elem + n, cmp3);
            break;
        case 4:
            sort(L.elem, L.elem + n, cmp4);
            break;
        default:
            break;
    }
    // 更新L的参数
    L.sortWay = sort_way;
    L.length = n;
    L.isInit = true;
}
```

```
// 显示仓库
void PrintWarehouse(SqList L) {
    if (!L.length)
        cout << "当前仓库没有数据!" << endl;
    else {
        cout << "当前仓库数据如下:" << endl;
        string sort_way;
        switch (L.sortWay) {
            case 1:
                sort_way = "(按照价格升序)";
                break;
            case 2:
                sort_way = "(按照价格降序)";
                break;
            case 3:
                sort_way = "(按照数量升序)";
                break;
            case 4:
                sort_way = "(按照数量降序)";
                break;
            default:
                break;
        }
        cout << sort_way << endl;
        cout << "-------------------------------------------------------" << endl;
        cout << setLeft
             << setw(5) << "序号"
             << setw(20) << "型号"
             << setw(15) << "单价"
             << setw(15) << "数量"
             << endl;
        cout << "-------------------------------------------------------" << endl;
        for (int i = 0; i < L.length; i++)
            cout << setLeft
                 << setw(5) << i + 1
                 << setw(20) << L.elem[i].type
                 << setw(15) << L.elem[i].price
                 << setw(15) << L.elem[i].number
                 << endl;
        cout << "-------------------------------------------------------" << endl;
    }
}
```

```
// 入库
Status Enter(SqList &L, const Computer &c) {
    // 寻找仓库中是否已经有和c.type同类型的电脑
    for (int i = 0; i < L.length; i++) {
        if (strcmp(c.type, L.elem[i].type) == 0) {
            if (c.price == L.elem[i].price) {   // 检查价格价格是否与c.price相等
                L.elem[i].number += c.number;
                return OK;
            } else {
                cout << "提示:" << endl;
                showInfo(L.elem[i]);
                return ERROR;
            }

        }
    }

    // 入库一个新类型的电脑
    if (L.length >= L.listsize) {   // 顺序表占用空间不足
        // 申请新的基地址、内存空间
        auto *newbase = (ElemType *) realloc(L.elem, (L.listsize + LISTINCREMENT) *
sizeof(ElemType));
        if (!newbase)
            return OVERFLOW;
        L.elem = newbase;   // L的基地址更改为newbase
        L.listsize += LISTINCREMENT;
    }
    // 按照L.sort_way对插入到顺序表中
    int item{L.length}; // 确定要插入的位序
    for (int i = 0; i < L.length; i++) {
        switch (L.sortWay) {
            case 1: {
                if (c.price < L.elem[i].price) {
                    item = i;
                    goto change_sq;
                }
            }
                break;
            case 2: {
                if (c.price > L.elem[i].price) {
                    item = i;
                    goto change_sq;
                }
            }
```

```
                break;
            case 3: {
                if (c.number < L.elem[i].number) {
                    item = i;
                    goto change_sq;
                }
            }
                break;
            case 4: {
                if (c.number > L.elem[i].number) {
                    item = i;
                    goto change_sq;
                }
            }
                break;
        }
    }
    change_sq:
    ElemType *q = &L.elem[item];
    // 将q及q以后的元素全部后移一位
    for (ElemType *p = L.elem + L.length - 1; p >= q; p--)
        *(p + 1) = *p;
    // 将新元素赋给q
    *q = c;
    L.length++;
    return OK;
}

// 出库
Status Out(SqList &L, const char *type, int num) {
    // 确定要出库元素的位序
    int item{L.length + 1};
    for (int i = 0; i < L.length; i++) {
        if (strcmp(type, L.elem[i].type) == 0) {
            item = i;
            break;
        }
    }
    // 没有找到要出库元素的位序，说明在仓库中没有元素，返回错误
    if (item > L.length)
        return -1;

    // 确定要出库元素现有库存的数量是否可以出库
    if (num < L.elem[item].number) {
```

```
        // 正常出库
        L.elem[item].number -= num;
        return OK;
    } else if (num == L.elem[item].number) {
        // 全部出库
        cout << "提示:" << endl
             << L.elem[item].type << "型号的电脑已全部出库!" << endl;

        // 将p及p以后的元素全部后移一位
        for (ElemType *p = &L.elem[item + 1]; p <= &L.elem[L.length - 1]; p++) {
            *(p - 1) = *p;
        }
        L.length--;
        return OK;
    } else {
        // 库存不足
        cout << "提示:" << endl;
        showInfo(L.elem[item]);
        return ERROR;
    }
}

// 盘点仓库
void Check(SqList L) {
    int numSum{0};
    double priceSum{0.0}, priceMax{L.elem[0].price}, priceMin{L.elem[0].price},
priceAverage{0.0};
    vector<char *> priceMaxComputer;    // 存储价格最高的电脑的类型名
    vector<char *> priceMinComputer;    // 存储价格最低的电脑的类型名

    if (!L.length) {
        cout << "当前仓库没有数据!" << endl;
    } else {
        // 求总数、总金额、最高价、最低价
        for (int i = 0; i < L.length; i++) {
            numSum += L.elem[i].number;
            priceSum += L.elem[i].price * L.elem[i].number;
            priceMax = max(priceMax, L.elem[i].price);
            priceMin = min(priceMin, L.elem[i].price);
        }
        // 记录价格最高、最低的电脑类型名
        for (int i = 0; i < L.length; i++) {
            if (L.elem[i].price == priceMax)
                priceMaxComputer.push_back(L.elem[i].type);
```

```
            if (L.elem[i].price == priceMin)
                priceMinComputer.push_back(L.elem[i].type);
        }
        // 求平均价
        priceAverage = priceSum / numSum;
        // 输出信息
        cout << "盘点数据如下" << endl
             << "电脑的总台数: " << numSum << endl
             << "电脑的总金额: " << priceSum << endl
             << "电脑的最高价: " << priceMax << endl
             << "          + 对应的型号是: ";
        for (const auto &c:priceMaxComputer)
            cout << c << "\t";
        cout << endl;
        cout << "电脑的最低价: " << priceMin << endl
             << "          + 对应的型号是: ";
        for (const auto &c:priceMinComputer)
            cout << c << "\t";
        cout << endl;
        cout << "电脑的平均价格: " << priceAverage << endl;
    }
}

// computer 排序方式：按价格升序
bool cmp1(const Computer &a, const Computer &b) {
    return a.price < b.price;
}

// computer 排序方式：按价格降序
bool cmp2(const Computer &a, const Computer &b) {
    return a.price > b.price;
}

// computer 排序方式：按数量升序
bool cmp3(const Computer &a, const Computer &b) {
    return a.number < b.number;
}

// computer 排序方式：按数量降序
bool cmp4(const Computer &a, const Computer &b) {
    return a.number > b.number;
}

// 输出仓库数据到文件中
```

```
void output(SqList L, const string &filename) {
    // 定义输出文件流filename.txt
    ofstream OutFile(filename + ".txt");
    // 输出文件头，用于下次输入时识别文件
    OutFile << "Computer Warehouse Data" << endl;
    if (!L.length)
        OutFile << "no data" << endl;
    else {
        OutFile << "The information is as follows:" << endl;
        switch (L.sortWay) {
            case 1:
                OutFile << "Sort Way: 1 (Ascending by price)" << endl;
                break;
            case 2:
                OutFile << "Sort Way: 2 (Descending by price)" << endl;
                break;
            case 3:
                OutFile << "Sort Way: 3 (Ascending by number)" << endl;
                break;
            case 4:
                OutFile << "Sort Way: 4 (Descending by number)" << endl;
                break;
            default:
                break;
        }
        OutFile << "Total Type: " << L.length << endl;

        // 输出数据信息
        for (int i = 0; i < L.length; i++)
            OutFile << L.elem[i].type << " "
                    << L.elem[i].price << " "
                    << L.elem[i].number
                    << endl;
        // 关闭文件
        OutFile.close();
    }
}

// 从外部文件导入数据到仓库
Status Import(SqList &L, const string &filename) {
    // 定义输入输入文件流filename.txt
    ifstream InFile;
    InFile.open(filename + ".txt");
    // 检查文件是否存在
```

```
    if (!InFile.is_open()) {
        InFile.close();
        return ERROR;
    }
    auto sort_way{0}, n{0};
    // 检查文件头是否正确，同时返回仓库排序方式、电脑类型总数
    switch (getFileHead(InFile, sort_way, n)) {
        case OK:
            break;
        case 2:
            return 2;
        case 3:
            return 3;
    }
    if (!InitSqList(L, n))
        return OVERFLOW;
    else
        // 调用函数CreateWarehouse 创建仓库
        CreateWarehouse(L, n, sort_way, InFile);
    InFile.close();
    return OK;
}

// 检查外部文件头
Status getFileHead(ifstream &ImportFile, int &sort_way, int &total_type) {
    string s;
    getline(ImportFile, s);
    if (s != "Computer Warehouse Data") {
        ImportFile.close();
        return 2;
    }
    getline(ImportFile, s);
    if (s == "no data") {
        ImportFile.close();
        return 3;
    }
    if (s != "The information is as follows:") {
        ImportFile.close();
        return 2;
    }
    ImportFile >> s;
    if (s != "Sort") {
        ImportFile.close();
        return 2;
```

```
    }
    ImportFile >> s;
    if (s != "Way:") {
        ImportFile.close();
        return 2;
    }

    ImportFile >> sort_way;
    if (sort_way < 1 || sort_way > 4)
        return 2;
    getline(ImportFile, s);

    ImportFile >> s;
    if (s != "Total") {
        ImportFile.close();
        return 2;
    }
    ImportFile >> s;
    if (s != "Type:") {
        ImportFile.close();
        return 2;
    }

    ImportFile >> total_type;
    if (total_type < 1)
        return 2;

    getline(ImportFile, s);
    return OK;
}

// 获取电脑信息
void getInfo(SqList L, const char *type) {
    // 确定要查找的电脑的元素位序
    int item{L.length + 1};
    for (int i = 0; i < L.length; i++) {
        if (strcmp(type, L.elem[i].type) == 0) {
            item = i;
            break;
        }
    }
    if (item > L.length)
        cout << "没有找到" << type << "型号的电脑" << endl;
    else {
```

```
        cout << "信息如下:" << endl;
        showInfo(L.elem[item]);
    }
}

// 显示电脑信息
void showInfo(Computer c) {
    cout << "-------------------------------------------------" << endl
         << setLeft << setw(20) << "型号" << setw(15) << "单价" << setw(15) << "现有库存数量" << endl
         << "-------------------------------------------------" << endl
         << setLeft << setw(20) << c.type << setw(15) << c.price << setw(15) << c.number << endl
         << "-------------------------------------------------" << endl;
}
```

main.cpp （主程序文件）

```
#include <iostream>
#include <string>
#include <algorithm>
#include <regex>
#include "sqlist.h"

using namespace std;

/********************函数声明********************/
// 菜单
int menu();

// 输入from至end范围内的数字
int input_number(int from, int end);

// 输入大于from的数字
int input_number(int from);

/********************函数声明********************/

// 主函数
int main() {
    cout << "欢迎使用电脑仓库管理系统！" << endl;

    // 定义顺序表L
    SqList L;

    // 根据菜单选择实现对应功能
```

```
while (true) {
    switch (menu()) {
        // 初始化仓库
        case 1: {
            // 检查仓库是否已经创建
            if (L.isInit) {
                cout << "仓库已创建过!" << endl;
                break;
            }

            cout << "选择创建仓库的方式" << endl
                 << "1.手动输入  2.从外部文件导入" << endl
                 << "选择 1-2:";
            auto sn = input_number(1, 2);

            if (sn == 1) {  // 手动创建仓库
                cout << "请输入电脑的类型数:";
                auto n = input_number(1);

                if (!InitSqList(L, n)) {
                    cout << "创建仓库失败!" << endl
                         << "原因: 内存不足。" << endl;
                } else {
                    cout << "请输入仓库排序的主要方式" << endl
                         << "1. 按价格升序 2. 按价格降序" << endl
                         << "3. 按数量升序 4. 按数量降序" << endl
                         << "选择 1-4:";
                    auto select = input_number(1, 4);
                    CreateWarehouse(L, n, select);
                    cout << "创建仓库成功!" << endl
                         << "您可以选择菜单 2 以显示仓库" << endl;
                }
            } else { // 从文件创建仓库
                cout << "请输入文件名(无需文件名后缀):";
                string filename;
                cin >> filename;
                switch (Import(L, filename)) {
                    case OK: {
                        string sort_way;
                        cout << "导入成功!" << endl;
                        switch (L.sortWay) {
                            case 1:
                                sort_way = "按照价格升序排序。";
                                break;
```

```
                            case 2:
                                sort_way = "按照价格降序排序。";
                                break;
                            case 3:
                                sort_way = "按照数量升序排序。";
                                break;
                            case 4:
                                sort_way = "按照数量降序排序。";
                                break;
                        }
                        cout << "按照文件要求: 仓库数据" + sort_way << endl;
                        cout << "您可以选择菜单 2 以显示仓库。" << endl;
                    }
                        break;
                    case ERROR:
                        cout << "导入失败!" << endl
                             << "原因: 没有" << filename << ".txt 这个文件 / " <<
filename << ".txt 文件无法打开。" << endl;
                        break;
                    case 2:
                        cout << "导入失败!" << endl
                             << "原因: " << filename << ".txt 文件内容不符合要求。"
<< endl;
                        break;
                    case 3:
                        cout << "导入失败!" << endl
                             << "原因: " + filename + ".txt 文件没有数据。" << endl;
                        break;
                    case OVERFLOW:
                        cout << "导入失败!" << endl
                             << "原因: 内存不足。" << endl;
                        break;
                }
            }
        }
            break;

        // 显示仓库
        case 2:
            // 检查仓库是否已经创建
            if (!L.isInit) {
                cout << "仓库还未初始化!" << endl;
                break;
            }
```

```
                PrintWarehouse(L);
                break;

            // 入库
            case 3: {
                // 检查仓库是否已经创建
                if (!L.isInit) {
                    cout << "仓库还未初始化!" << endl;
                    break;
                }
                cout << "请输入要入库的电脑型号、单价、总数:" << endl;
                Computer c;
                cin >> c.type >> c.price >> c.number;
                switch (Enter(L, c)) {
                    case OK:
                        cout << "入库成功!" << endl;
                        break;
                    case ERROR:
                        cout << "无法入库！" << endl
                             << "原因: 输入的" << c.type << "型号电脑的单价" << c.price
<< "与库存中单价不一致。" << endl;
                        break;
                    case OVERFLOW:
                        cout << "入库失败!" << endl
                             << "原因: 内存不足。" << endl;
                        break;
                }
            }
                break;

            // 出库
            case 4: {
                // 检查仓库是否已经创建
                if (!L.isInit) {
                    cout << "仓库还未初始化!" << endl;
                    break;
                }
                cout << "请输入要出库的电脑型号、数量:" << endl;
                char type[50];
                int num{0};
                cin >> type >> num;
                if (num <= 0) {
                    cout << "无法出库!" << endl
                         << "原因: 数量输入小于等于 0" << endl;
```

```
                break;
            }
            switch (Out(L, type, num)) {
                case OK:
                    cout << "出库成功!" << endl;
                    break;
                case -1:
                    cout << "出库失败!" << endl
                         << "原因: " << type << "型号的电脑不在仓库中!" << endl;
                    break;
                case ERROR:
                    cout << "无法出库!" << endl
                         << "原因: 要出库的数量" << num << "大于库存中的数量。" <<
endl;
                    break;
            }
        }
            break;

        // 查询
        case 5: {
            // 检查仓库是否已经创建
            if (!L.isInit) {
                cout << "仓库还未初始化!" << endl;
                break;
            }
            cout << "请输入要查询的电脑型号:";
            char type[50];
            cin >> type;
            getInfo(L, type);
        }
            break;

        // 盘点仓库
        case 6:
            // 检查仓库是否已经创建
            if (!L.isInit) {
                cout << "仓库还未初始化!" << endl;
                break;
            }
            Check(L);
            break;

        // 对仓库重新排序
```

```
case 7: {
    // 检查仓库是否已经创建
    if (!L.isInit) {
        cout << "仓库还未初始化!" << endl;
        break;
    }
    cout << "请输入要重新对仓库排序的主要方式" << endl
         << "1. 按价格升序 2. 按价格降序" << endl
         << "3. 按数量升序 4. 按数量降序" << endl
         << "选择 1-4:";
    auto select = input_number(1, 4);
    switch (select) {
        case 1:
            sort(L.elem, L.elem + L.length, cmp1);
            break;
        case 2:
            sort(L.elem, L.elem + L.length, cmp2);
            break;
        case 3:
            sort(L.elem, L.elem + L.length, cmp3);
            break;
        case 4:
            sort(L.elem, L.elem + L.length, cmp4);
            break;
        default:
            break;
    }
    L.sortWay = select;
    cout << "重新排序成功!" << endl;
}
    break;

// 导出仓库数据到文件
case 8: {
    // 检查仓库是否已经创建
    if (!L.isInit) {
        cout << "仓库还未初始化!" << endl;
        break;
    }
    cout << "请输入要保存的文件名(无需文件名后缀):";
    string filename;
    cin >> filename;
    output(L, filename);
    cout << "已成功将仓库数据保存在程序所在文件夹的" << filename << ".txt。"
```

```
<< endl;
            }
                break;

            // 结束程序
            case 0:
                cout << "程序结束，谢谢使用！" << endl;
                system("Pause");
                exit(0);
        }
    }
}

// 菜单
int menu() {
    cout << endl
         << "<---------------显示菜单---------------" << endl
         << "1. 初始化仓库" << endl
         << "2. 显示仓库" << endl
         << "3. 入库" << endl
         << "4. 出库" << endl
         << "5. 查询" << endl
         << "6. 盘点仓库" << endl
         << "7. 重新对仓库排序" << endl
         << "8. 导出仓库数据到文件" << endl
         << "0. 结束程序" << endl
         << "---------------显示菜单--------------->" << endl
         << "输入 0-8:";
    return input_number(0, 8);
}

// 输入 from 至 end 范围内的数字
int input_number(int from, int end) {
    auto select{0};
    string input;
    regex r("[0-9]*");  // 正则表达式:数字 0-9，可以出现多次
    while (true) {
        cin >> input;
        bool isNumber = regex_match(input,r);
        if (!isNumber)  // 如果 input 和正则表达式匹配
            cout << "输入错误，请输入数字" << from << "-" << end << ":";
        else {
            select = atoi(input.c_str());
            if (select < from || select > end)
```

```
                cout << "输入错误，重新输入" << from << "-" << end << ":";
            else
                break;
        }
    }
    return select;
}

// 输入大于from的数字
int input_number(int from) {
    auto number{0};
    string input;
    regex r("[0-9]*");  // 正则表达式:数字0-9，可以出现多次
    while (true) {
        cin >> input;
        bool isNumber = regex_match(input,r);
        if (!isNumber)  // 如果input和正则表达式匹配
            cout << "输入错误，请输入数字:";
        else {
            number = atoi(input.c_str());
            if (number < from)
                cout << "输入错误，重新输入:";
            else
                break;
        }
    }
    return number;
}
```

评分表

数学与计算机学院课程设计评分表

课程名称：数据结构

考核项目	考核分数
设计方案的合理性与创造性（15 分）	
设计与调试结果（30 分）	
课程设计报告的质量（30 分）	
答辩陈述与回答问题情况（25 分）	
综合成绩	

教师签名：

日　　期：