**2024级《程序设计基础课程设计》总结报告（第8组）**

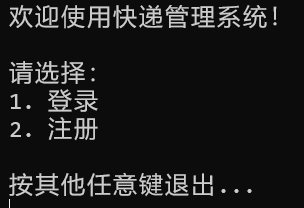
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **学号** | **姓名** | **性别** | **班级** | **具体任务分工** | **所占比例** | **成绩** |
| 25230232 | 王劭屹 | 男 | 212401 | 框架设计、数据结构（结构体、链表）、主函数与登录、管理员、测试、bug修复、非法输入判定、实验报告 | 40% |  |
| 21240102 | 张淋萱 | 女 | 212401 | 快递员一系列功能、部分管理员功能、管理员代码测试 | 20% |  |
| 21240103 | 邢嘉益 | 女 | 212401 | 用户核心功能、用户代码测试、实验报告编写 | 20% |  |
| 21240104 | 方宇芬 | 女 | 212401 | 快递平台一系列功能、部分用户功能、快递平台和快递员测试 | 20% |  |

提交日期：2025年3月29日

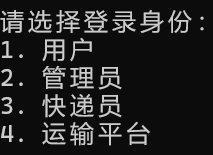
**报告正文**

1. 总体介绍
2. 项目简介 ………………………………………………………………… 4
3. 整体逻辑 ………………………………………………………………… 4
4. 数据结构
5. 各类表结构体 …………………………………………………………… 5
6. 通用链表 ………………………………………………………………… 7
7. 文件与链表映射 ………………………………………………………… 8
8. 主函数
9. 初始化与数据恢复 ……………………………………………………… 9
10. 界面展示与选择操作 …………………………………………………… 10
11. 数据更新与链表释放 …………………………………………………… 11
12. 快递平台
13. 预处理 …………………………………………………………………… 12
14. 写入、修改、删除待发货快递 ………………………………………… 12
15. 匹配快递员发货 ………………………………………………………… 14
16. 查询发货信息、查看库存情况 ………………………………………… 15
17. 注销账号 ………………………………………………………………… 15
18. 快递员
19. 预处理、弹窗 …………………………………………………………… 15
20. 查询任务、查询任务包裹信息 ………………………………………… 16
21. 确认完成任务 …………………………………………………………… 16
22. 拒绝当前任务、隐身功能 ……………………………………………… 17
23. 注销账号 ………………………………………………………………… 17
24. 管理员
25. 预处理、弹窗 …………………………………………………………… 18
26. 入库操作 ………………………………………………………………… 18
27. 用户管理 ………………………………………………………………… 19
28. 货架管理、库存盘点 …………………………………………………… 20
29. 查看反馈 ………………………………………………………………… 22
30. 处理用户寄件、修改寄件信息 ………………………………………… 22
31. 查看业务统计、查看快递单修改记录 ………………………………… 23
32. 提示包裹滞留用户前来取件 …………………………………………… 23
33. 查看驿站账户 …………………………………………………………… 24
34. 处理拒收包裹 …………………………………………………………… 24
35. 用户
36. 预处理、弹窗 …………………………………………………………… 26
37. 取件、查询取件信息 …………………………………………………… 26
38. 寄件、查询寄件信息、修改寄件信息、取消寄件 …………………… 27
39. 反馈问题 ………………………………………………………………… 29
40. 好友功能 ………………………………………………………………… 29
41. 代取 ……………………………………………………………………… 29
42. 修改个人信息、查询服务记录 ………………………………………… 30
43. 注销账号 ………………………………………………………………… 30
44. 拒收 ……………………………………………………………………… 30
45. 消息功能 ………………………………………………………………… 31
46. 项目中遇到的问题
47. 输入及缓冲区问题 ……………………………………………………… 31
48. 代码复用率低问题 ……………………………………………………… 32
49. 分文件编写思路问题 …………………………………………………… 32
50. 编码问题、路径问题 …………………………………………………… 33
51. 图形化问题 ……………………………………………………………… 34
52. 释放堆区内存问题 ……………………………………………………… 34
53. **总体介绍**
54. 项目简介

开始界面：



在实现了课程基础的要求上，我们不但丰富了用户和驿站的功能，还引入了快递员、快递平台两个操作角色，使得项目更加丰富趣味、贴近现实。



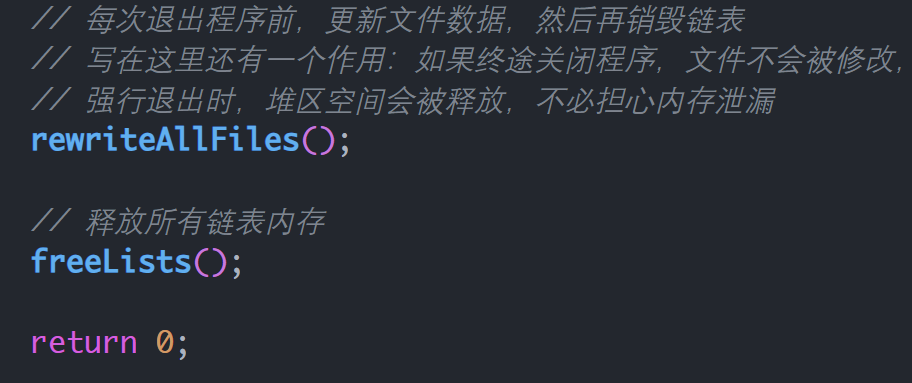
1. 整体逻辑

系统采用链表来存储各类数据，通过文件进会触发信息发送行数据持久化。用户在登录或注册后，根据不同的角色进入相应的功能模块。**系统在启动时从文件中读取数据恢复链表，在退出时将链表数据更新到文件中。**各个角色的操作主要围绕链表的增删改查展开，以实现数据的同步更新。

// 恢复链表



// 更新链表



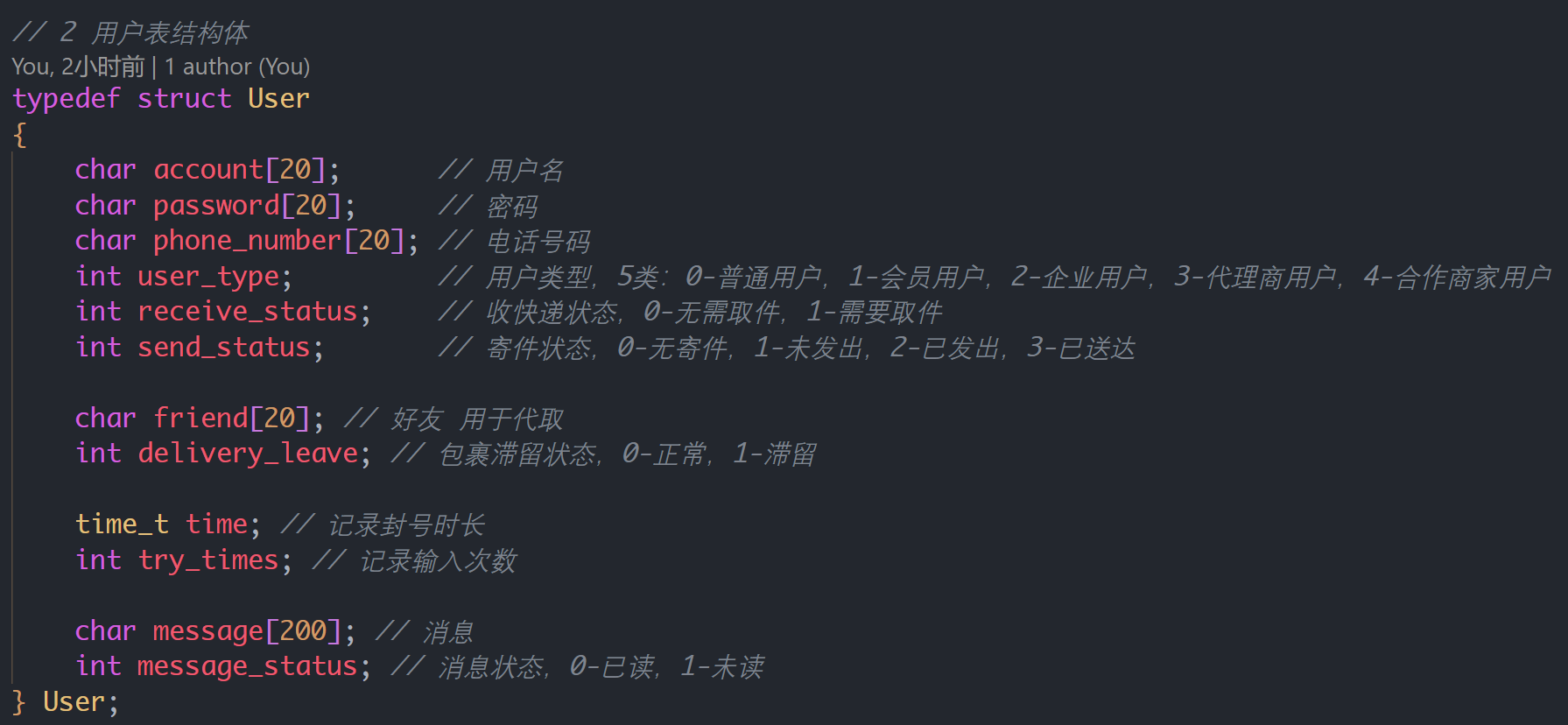
快递的运转可分为两条路径，取决于用户的收件、寄件，方向相反。

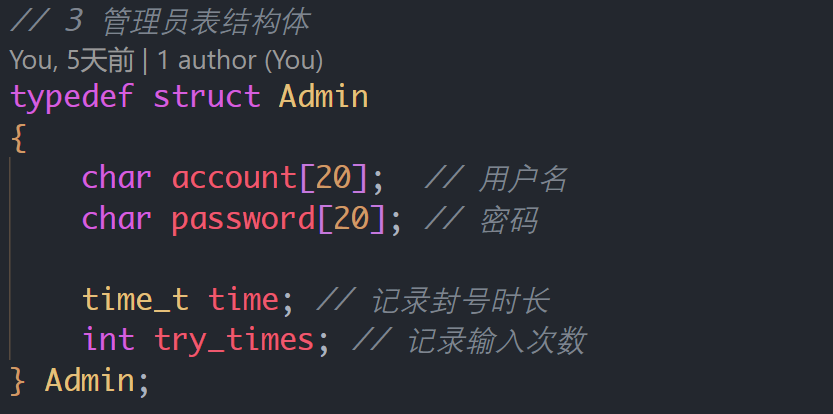
**收件：平台->快递员->驿站->用户**

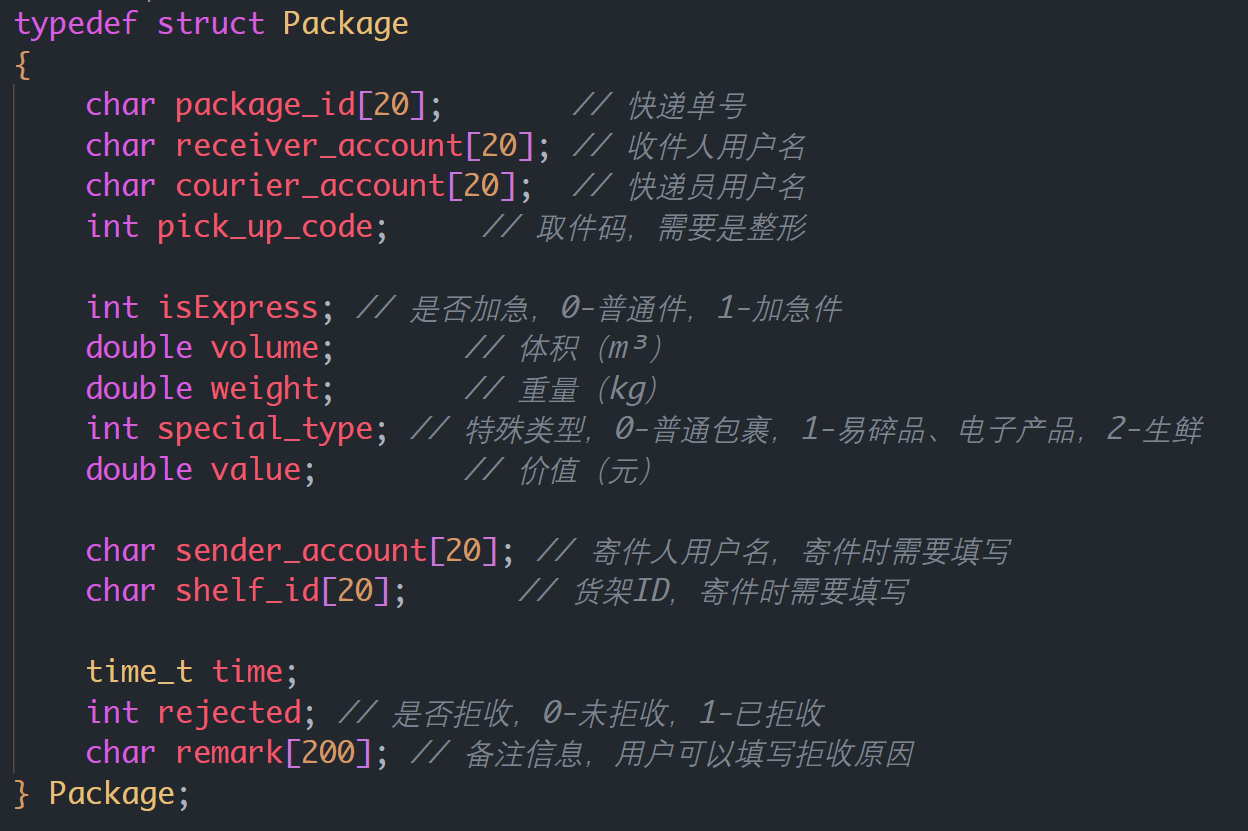
**寄件：用户->驿站->快递员->平台**

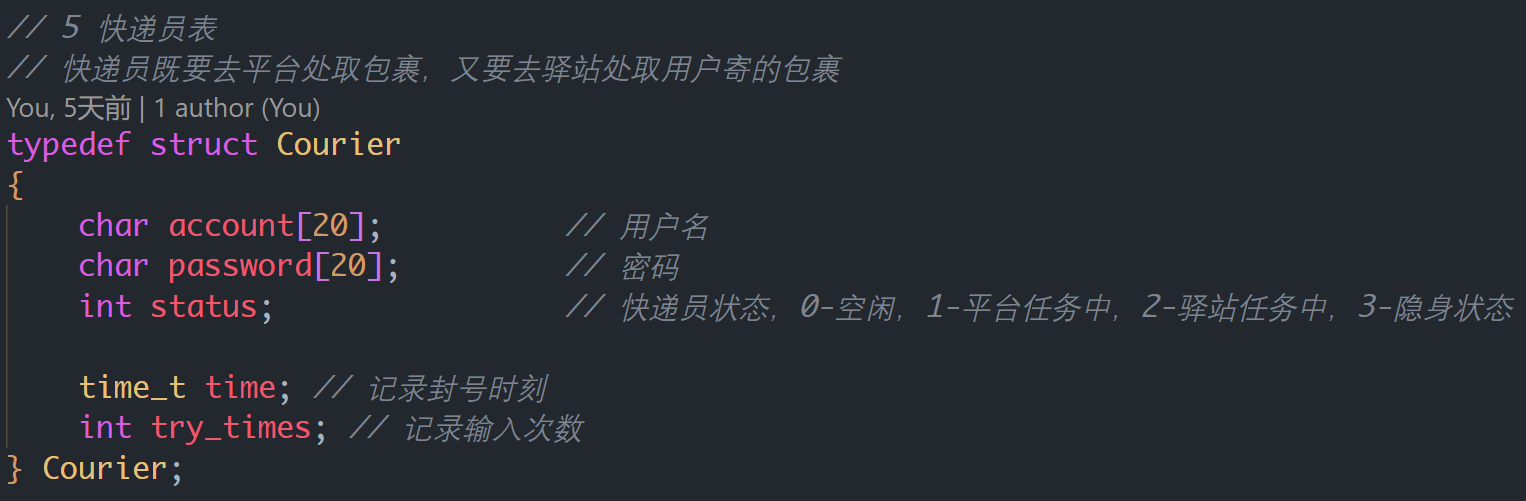
1. **数据结构**
2. 各类表结构体

定义了**用户表、管理员表、包裹表、快递员表、平台表和反馈表**等结构体，用于存储不同类型的信息。结构体的信息后续会储存在链表和文件中。







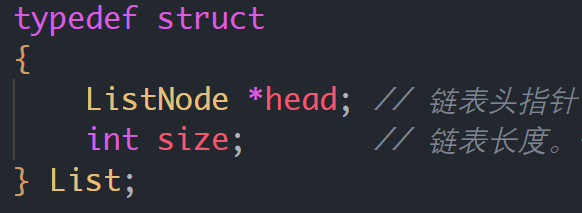


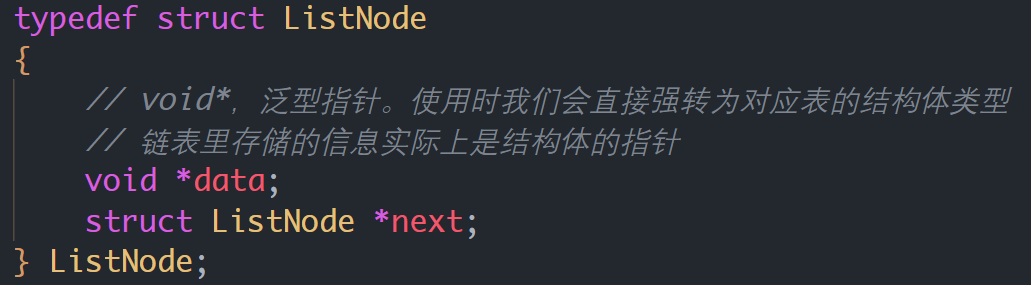




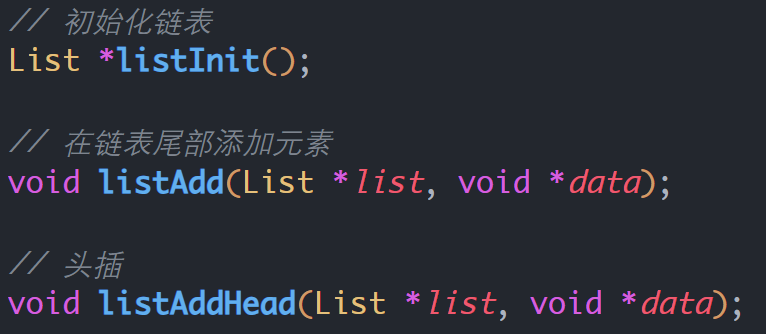
1. 通用链表

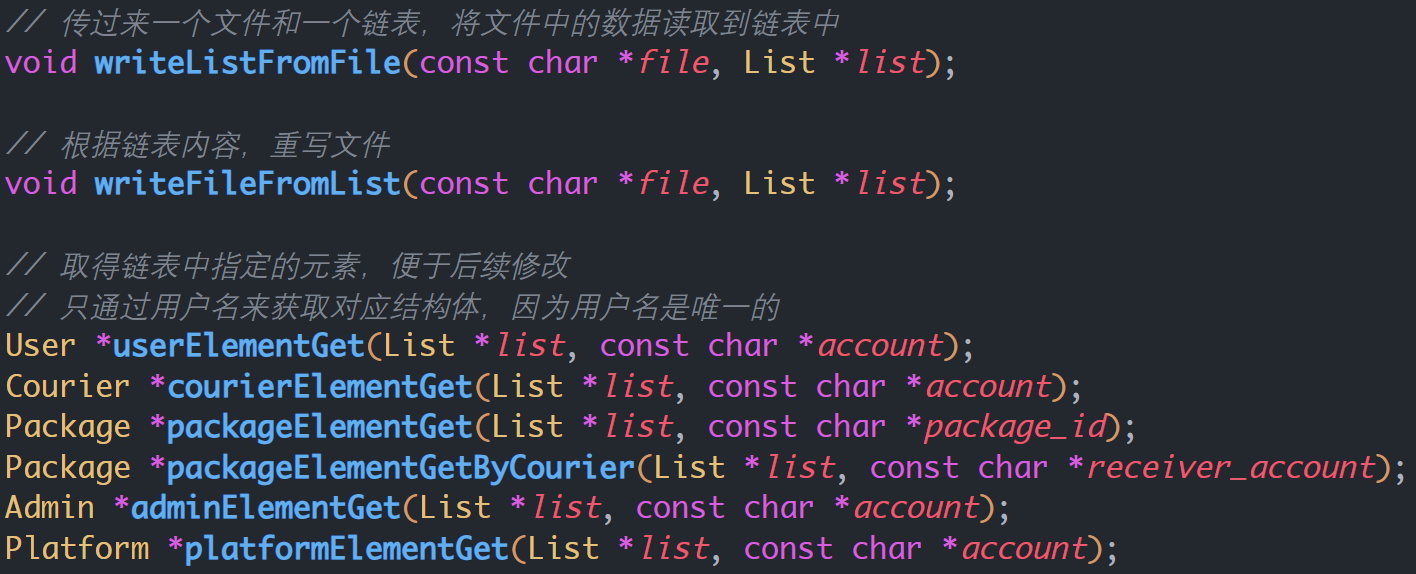
定义了通用链表节点结构体 ListNode 和通用链表结构体 List，用于存储各种类型的数据。**链表采用泛型指针 void\* 存储数据**，方便存储不同类型的结构体指针，避免为上述结构体各自单独设置不同的链表类型。同时，为通用链表提供了一系列方法：如**初始化、头插、尾插、删除、查找、销毁、与文件互相映射**等。





链表操作函数如下：

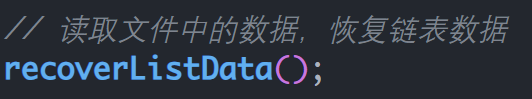






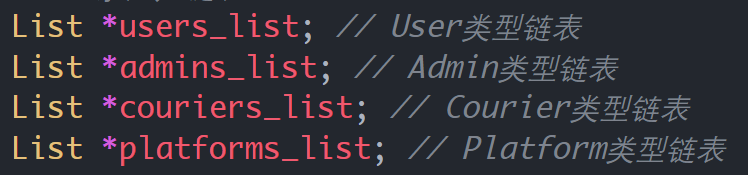
1. 文件与链表映射

启动程序后，链表从文件中获取数据；程序使用期间，在链表上进行修改；退出程序销毁链表前，将链表覆盖写入文件。这样一来，实现了通过链表与文件映射来使得数据持久化。

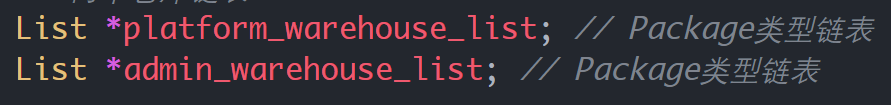


程序中涉及的链表如下：

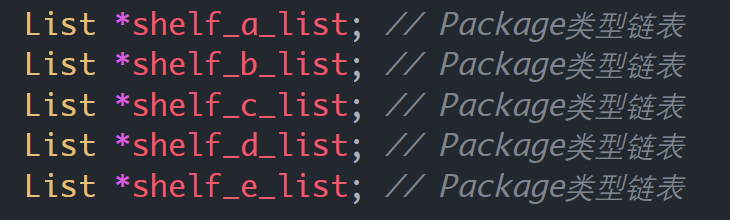
**操作者信息类链表**：用户信息、管理员信息、快递员信息、平台信息链表



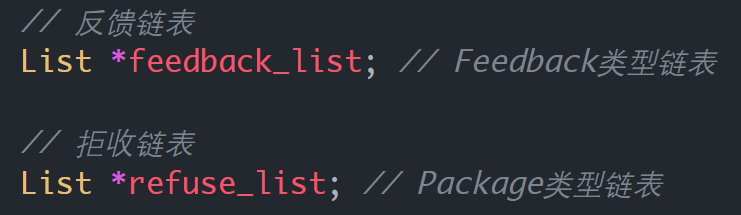
**仓库链表**：驿站仓库链表、快递平台仓库链表



**货架链表**：属于驿站，分为五个，按照包裹属性进行存放

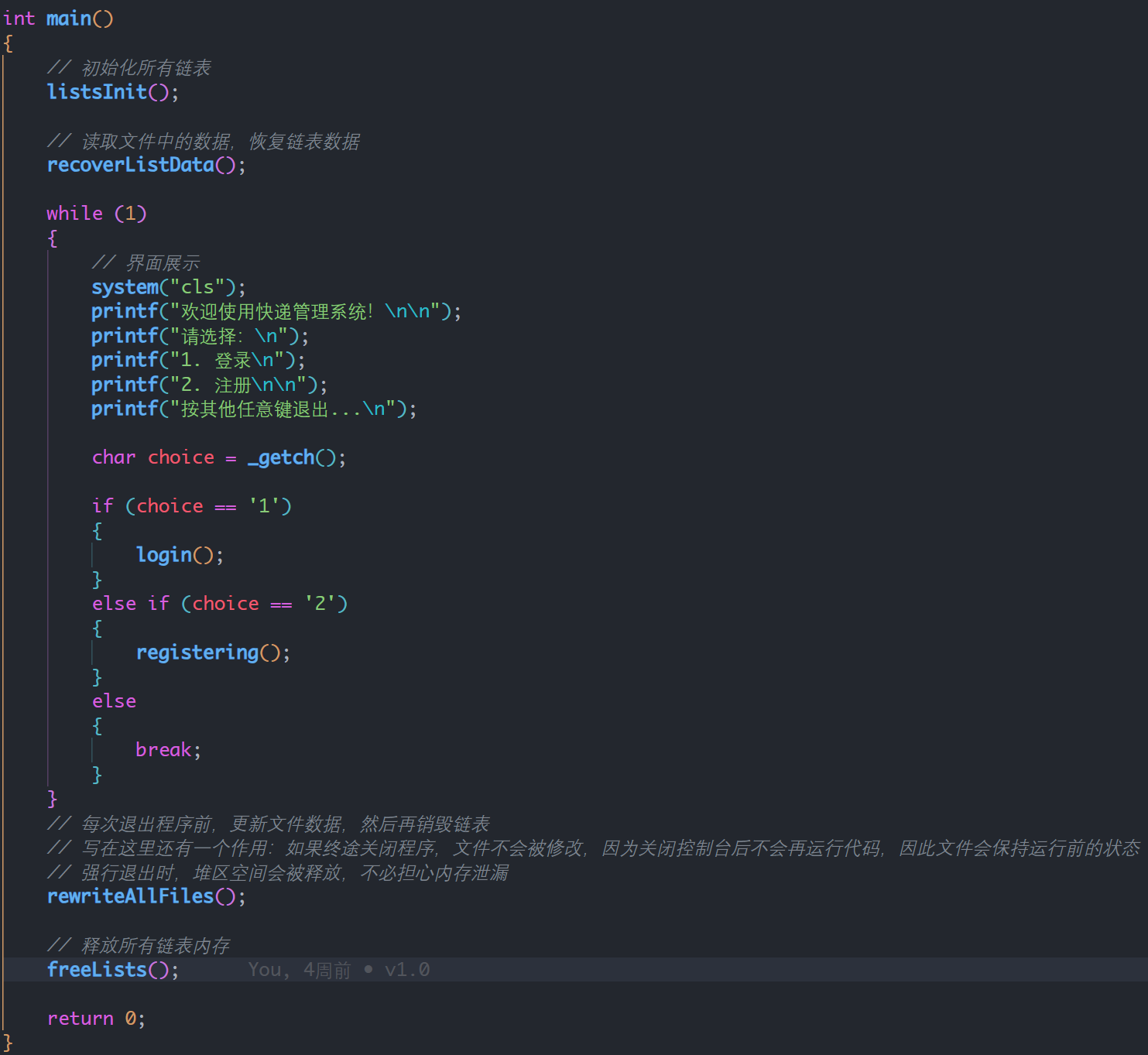


**辅助链表**：推送链表、临时链表、反馈链表、拒收链表



1. **主函数**

main函数尽可能简洁，降低耦合度



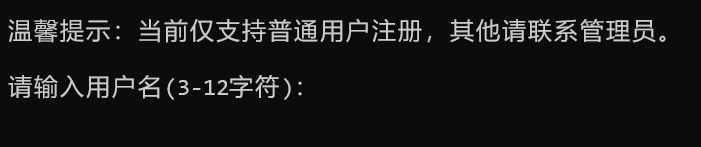
1. 初始化与数据恢复

在 main 函数中，首先定义一系列**全局链表**，并调用 listsInit 函数初始化所有链表，然后调用 recoverListData 函数从文件中读取数据恢复链表。

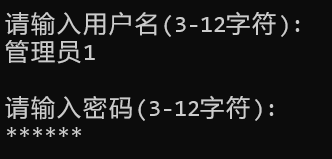
1. 界面展示与选择操作

通过循环展示登录注册界面，根据用户的输入选择登录或注册操作。

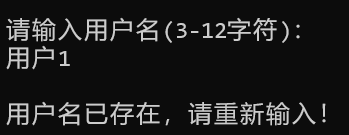
若选择注册，则**只支持用户注册**（这一点是符合逻辑的，管理员、快递员、平台等属于工作账号，需要由管理员统一进行注册）。注册支持重名检测、确认密码检测，注册成功后写入用户信息链表。



// 密码隐蔽性：

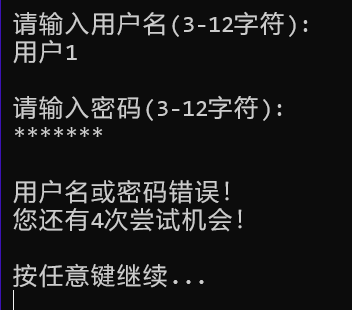


// 重名检测

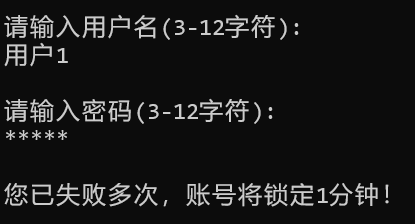


若选择登录，则进入二级菜单，从四个角色中选择自己要登录的。**总共有5次输入机会**，若密码全部错误，会封锁该用户账户一分钟。封锁逻辑是基于时间戳实现的。

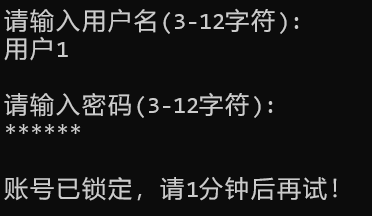
// 错误提示



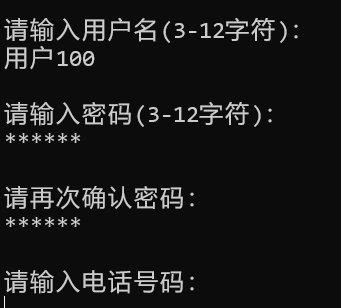
// 封锁账号：



此时，即使正确输入密码：



// 确认密码机制：



四个角色在各自的.c文件中，各自定义了一个**全局临时结构体指针**，并在其.h文件里进行了extern声明。在登录界面，如果登录成功，会使这个临时指针指向该账号对应的结构体，这样**在登录期间就可以通过临时结构体，快速访问自身的部分数据了**。

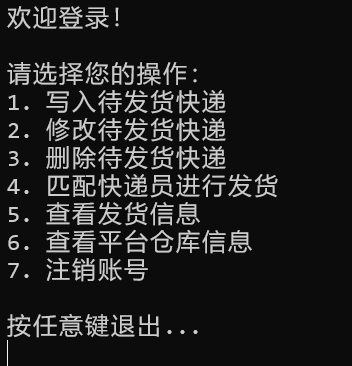
1. 数据更新与链表释放

在退出程序前，调用 rewriteAllFiles 函数将链表数据更新到文件中，然后调用freeLists函数释放所有链表的内存。这个释放函数，同时将 **List 结构体**、各**节点 ListNode 结构体**，以及节点中指向数据的**泛型指针 void\* 指向的数据**也释放掉，完全避免了内存泄漏。

//以用户释放链表为例

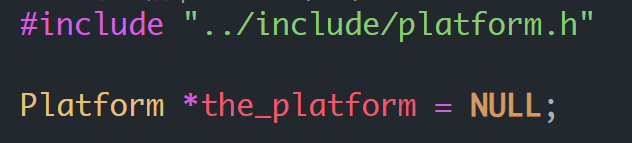


1. **快递平台**



1. 预处理

定义全局变量——临时结构体指针

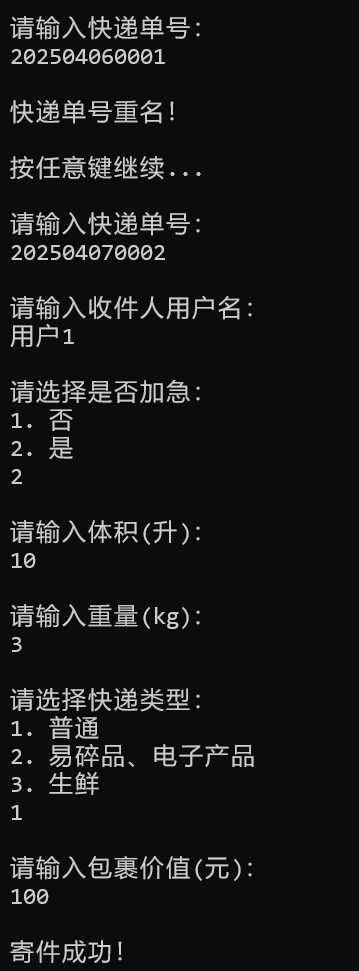


1. 写入、修改、删除待发货快递

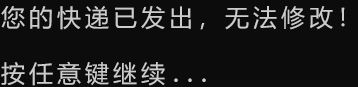
这里的写入，是“用户收件”这条路径的源头，所有用户的收件都来源于平台的写入（固然，在现实网购中，这个源头应该商家对接的仓库，本程序简化了这个过程，将其抽象为“单个平台”）。写入的链表是“平台仓库链表”。

在**发货之前**，可以对这些待发货的快递信息进行修改，或是删除快递。

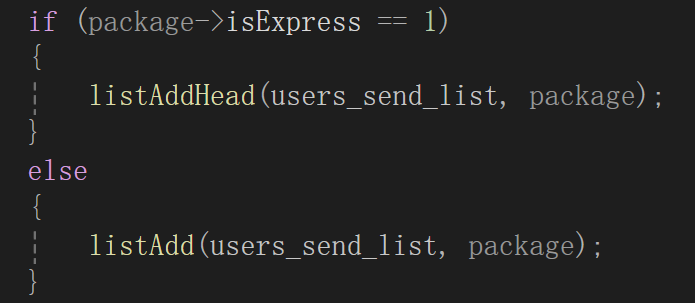
如图，快递单号有重名检测。



//如图，发货后无法修改信息



1. 匹配快递员发货

快递属性中有“加急”这一选项。如果是加急包裹，则**优先**匹配快递员（实现逻辑为，写入时，加急包裹使用**头插**，进而能先匹配上快递员）。

如果没有快递员或没有库存，则无法发货。

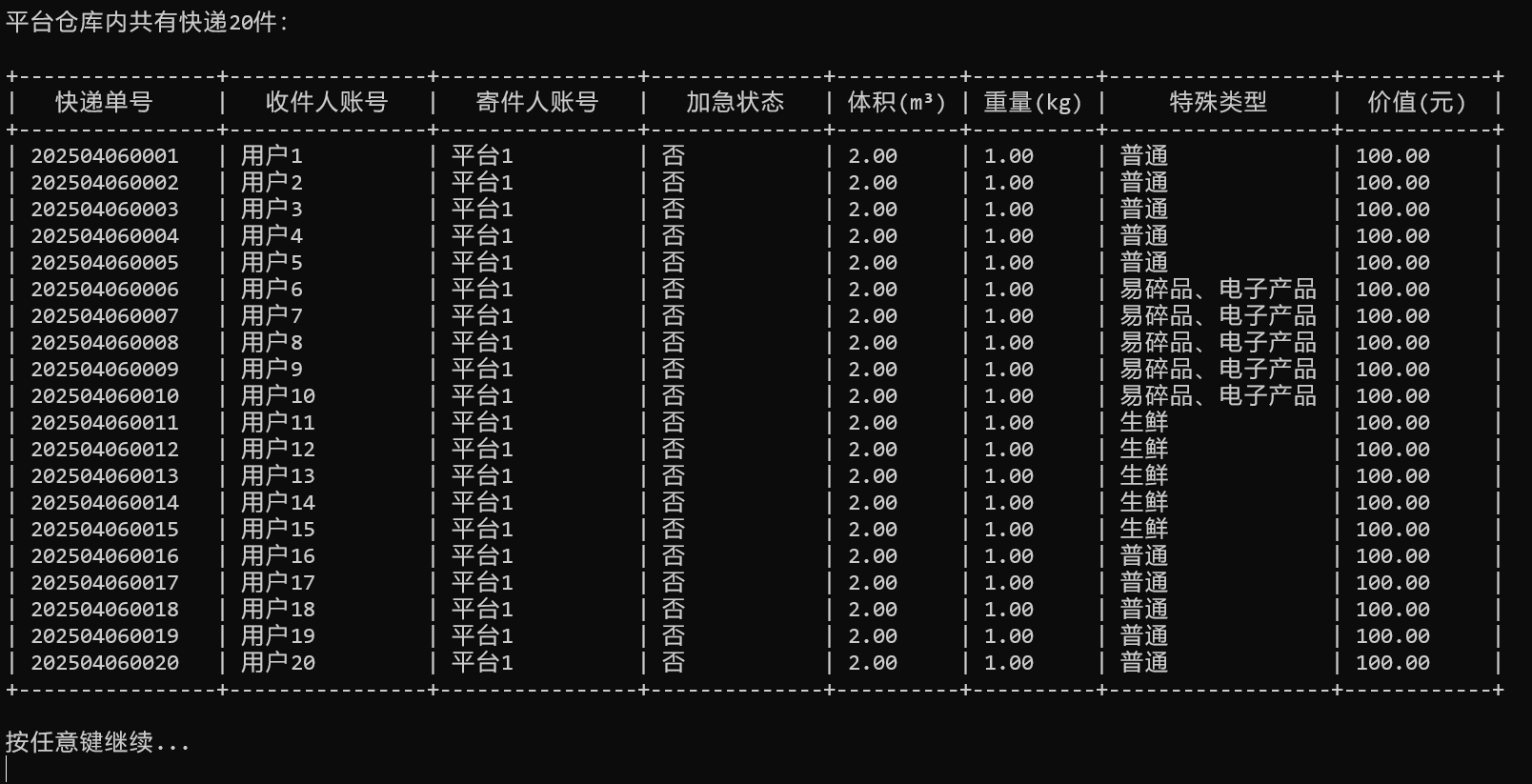
包裹属性中的体积、重量等指标，决定快递员单次运送快递的上限。如果超过上限，则会匹配到下一个快递员。

对于成功匹配到快递员的快递，会删除掉“平台仓库链表”中的对应节点，并添加到“快递员推送链表”中，该链表会在快递员登录后发挥作用。这两个链表中储存的都是包裹类结构体，而包裹结构体含有“快递员”这一数据，可以映射到负责运输它的快递员。



1. 查询发货信息、查看库存情况

简单的遍历链表并打印信息。



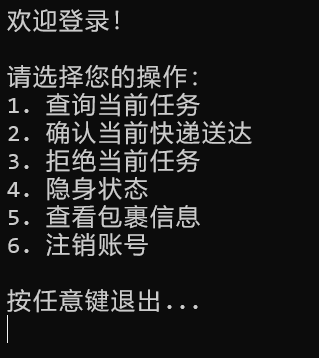
1. 注销账号

逻辑上，因为是商业合作关系，并不能由平台单方面决定账号删除，仍然需要由管理员统一来执行注销。

//注销提示



1. **快递员**



1. 预处理、弹窗

预处理：创建全局结构体指针、临时链表。遍历“快递员推送链表”，如果其上的节点对应的快递员为当前登录的快递员，则把“推送链表”上的节点浅拷贝到“临时链表”上来；如果是其他快递员，则无需处理。

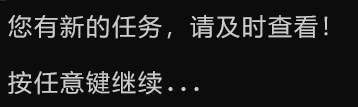
临时链表销毁时机：当前快递员退出账号时销毁，但只能销毁 List 结构体和 ListNode 结构体，不能释放 void\* 指向的内存，因为本程序的所有数据都是**浅拷贝**的，如果一方释放了数据，那其他链表都会受影响。因此数据的释放统一在main函数结束前执行。

而且，浅拷贝也是有必要的。程序中涉及到大量对数据的修改，这些修改应当是**全局同步**的。如果是深拷贝，反而不利于这种全局同步的实现。

值得注意的是，推送链表的节点不能立即删除。如果快递员未完成任务就下线了的话，那临时链表就会随之销毁；此时如果推送链表对应的节点也被删了，那这个任务就“永久性消失了”，这是恶性bug。

弹窗的机制则是，如果推送链表中含有“属于自己负责的包裹”，即临时链表不为空，那么就弹出“任务提示”的弹窗。

弹窗样式如下：



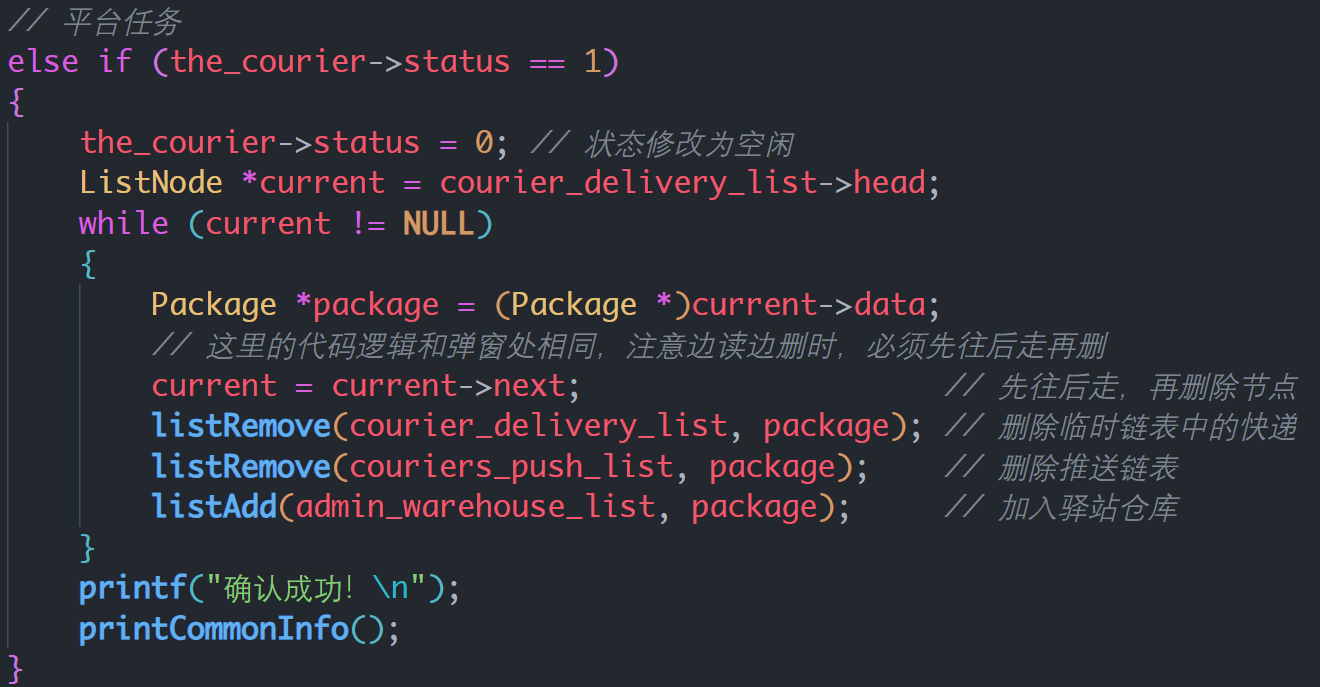
1. 查询任务、查询任务包裹信息

简单遍历临时链表以查询任务、获得包裹信息。



1. 确认完成任务

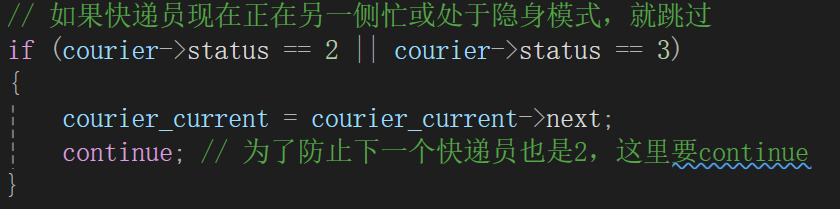
现实中，快递员到了目的地之后，就可以视为“任务完成”；但是在程序中，没法直接去获知这一点。因此，**让快递员自己去确认完成，类似“打卡”**。随后，包裹对应的节点就会加入到“驿站仓库链表”，临时链表和快递员推送链表中的对应节点就可以没有后顾之忧地删除了。



1. 拒绝当前任务、隐身功能

执行拒绝后，临时链表和推送链表中的节点会删掉，重新加入到“平台仓库链表”中，相当于执行了逆操作；隐身功能开启后，匹配快递员时无法再自动匹配到该快递员（如果快递员工资是固定发放的，那这个功能无论如何都会被要求删掉了，哈哈哈）。

//平台派单会跳过隐身状态



1. 注销账号

同快递平台一样，快递员属于商业合作性质的账号，自己不具备注销权。

1. **管理员**

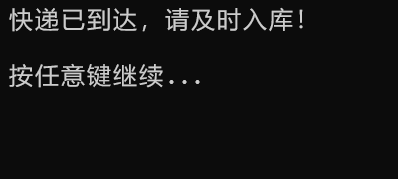


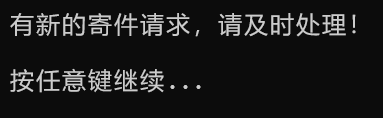
1. 预处理、弹窗

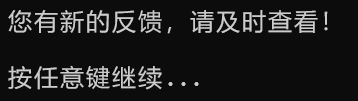
创建全局结构体指针，弹窗分为以下模块：**快递入库提醒**（若快递员将包裹送达，则提醒管理员执行入库）、**货架容量警告提醒**（若某货架容量超过80%，则对应货架发出警示）、**反馈处理提醒、寄件处理提醒。**

另外，驿站有一个独立的账户，账户收益来自于用户寄件付费。同时，若涉及补偿等操作，这部分支出也来自于账户。

// 弹窗样式如下：



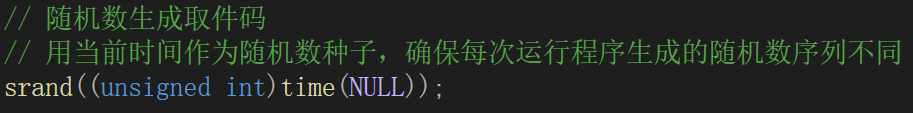




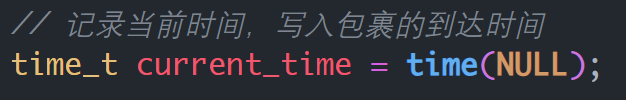
1. 入库操作

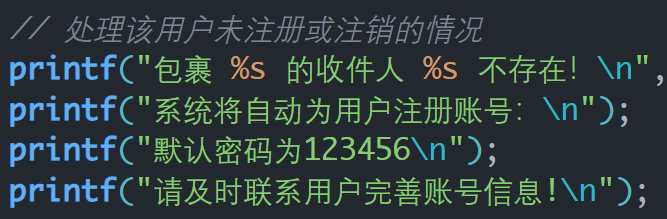
入库的主体操作是，将“驿站仓库链表”中快递员送来的快递，智能分配到各货架中；智能分配逻辑如下：**先判断是否为“生鲜”**，是则直接放入E货架；**再判断是否为易碎品**，是则直接放入D货架；然后是**大件重件**，若体积大于0.**5dm**³或重量大于10kg，直接放入C货架；剩下的AB货架为**普通货架**，若此前的指标都不满足，则入普通货架。

放入货架时，通过随机数种子随机生成一个四位数的取件码；货架编号则根据当前该货架的容量，通过除模运算，生成有序且唯一的货架编号。



同时生成一个时间戳，记录包裹入库时间，为包裹滞留提供判断依据。

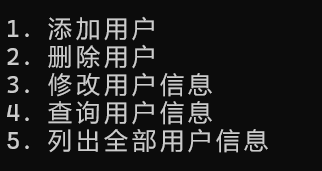


放入对应货架后，将包裹从“驿站仓库链表”中删除。找到包裹对应的用户，将包裹加入“用户推送链表”（机理类似快递员推送链表，用户登录账号后发挥作用）。如果该用户未注册，那将**自动为包裹收件人创建账号**（不能直接将包裹丢弃，因此选择了自动创建账号的机制）。

1. 用户管理

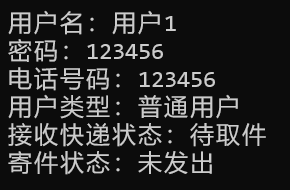
二级菜单中，支持**添加**用户、**删除**用户、**修改**用户信息、**查询**用户信息、**展示用户信息**。

之前提到，除“用户”外，快递平台、快递员都不支持自己注册和注销。这里添加、删除、修改可对其进行统一操作。其中删除操作需要注意，**如果用户此时有未取的快递，则不允许删除**（之前提过，如果快递收件人未注册账号，则自动注册。本程序在取件方面的逻辑是，只要用户的快递在，那用户的账号就必须存在）。



查询信息方面，提供了**两种查询方式**：一是列出全部信息，二是根据用户名来搜索。

// 按用户名搜索



// 直接列出所有用户

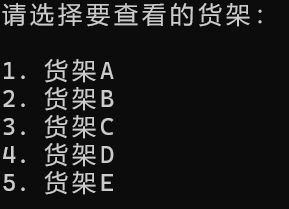


这部分功能代码复用性较差，代码量较为冗余。由于四个角色的信息都有细微差异，因此没法直接复用。

1. 货架管理、库存盘点

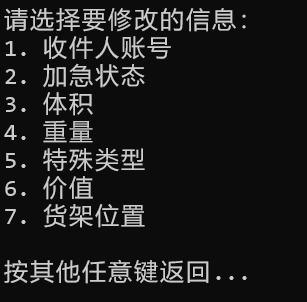
货架管理的二级菜单分为：**查看**货架信息、**修改**货架信息。

其中，查看货架信息，可以分别查看5个货架的全部快递。

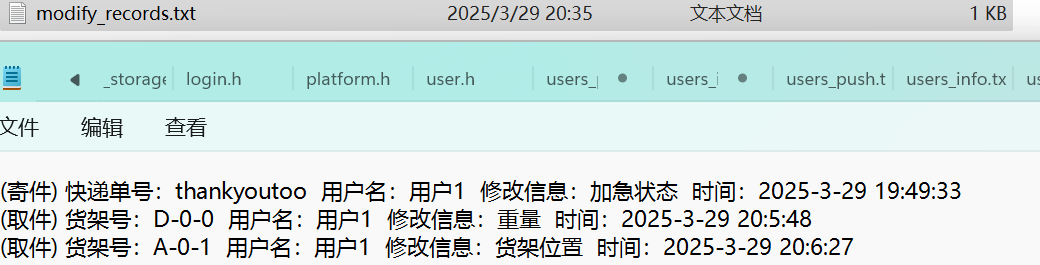


修改货架信息，输入快递单号，对快递进行映射。修改项中允许修改货架位置（前提是新货架未满）。虽然之前提到货架分配是智能的，但难免遇到实际情况与算法相悖的情况。货架号作为包裹结构体的属性也会随之更改。之前提到整个程序的链表节点都是浅拷贝的，所以数据都是全局统一更新的。

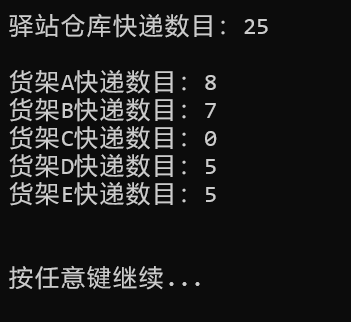
可修改选项：



对快递单的修改，会记录到文件 modify\_record.txt 中永久保存，后续也会提到查看该信息的功能。



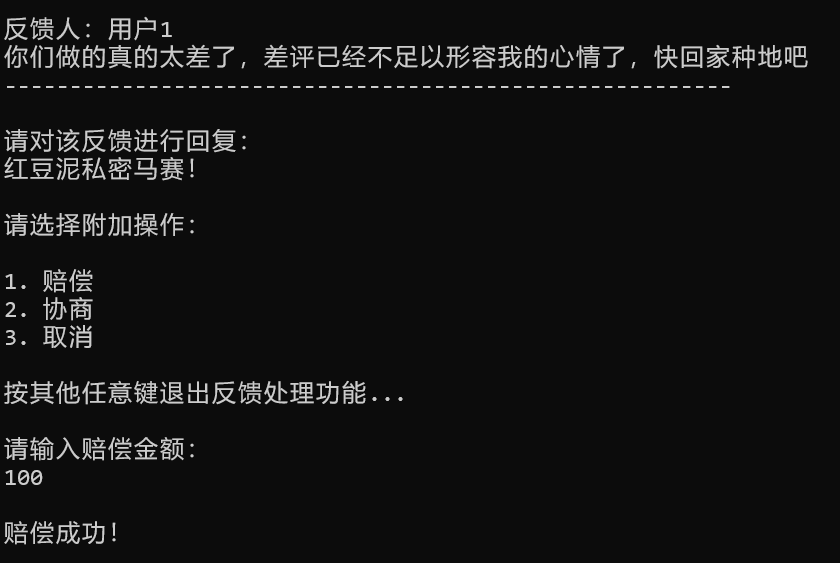
库存盘点则是直接列出当前所有货架中快递的数量及各货架的情况。如果达到仓库预警提示要求，这里也会列出。



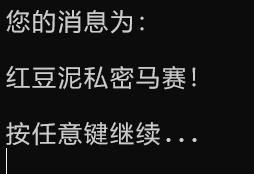
1. 查看反馈

用户的反馈会统一写入文件中，读取后逐条处理，处理方式有赔款、协商或暂不处理，由管理员自行选择。

回复给用户后，用户登录后会有消息提示。



用户端接收消息：



1. 处理用户寄件、修改寄件信息

处理用户寄件即匹配快递员，来执行“用户->驿站->快递员->平台”的路线。匹配快递员的算法与快递平台基本相同，这里不再赘述。

// 管理员为用户的寄件匹配快递员：



在快递寄出之前（匹配到快递员之前），可对快递信息进行修改。

1. 查看业务统计、查看快递单修改记录

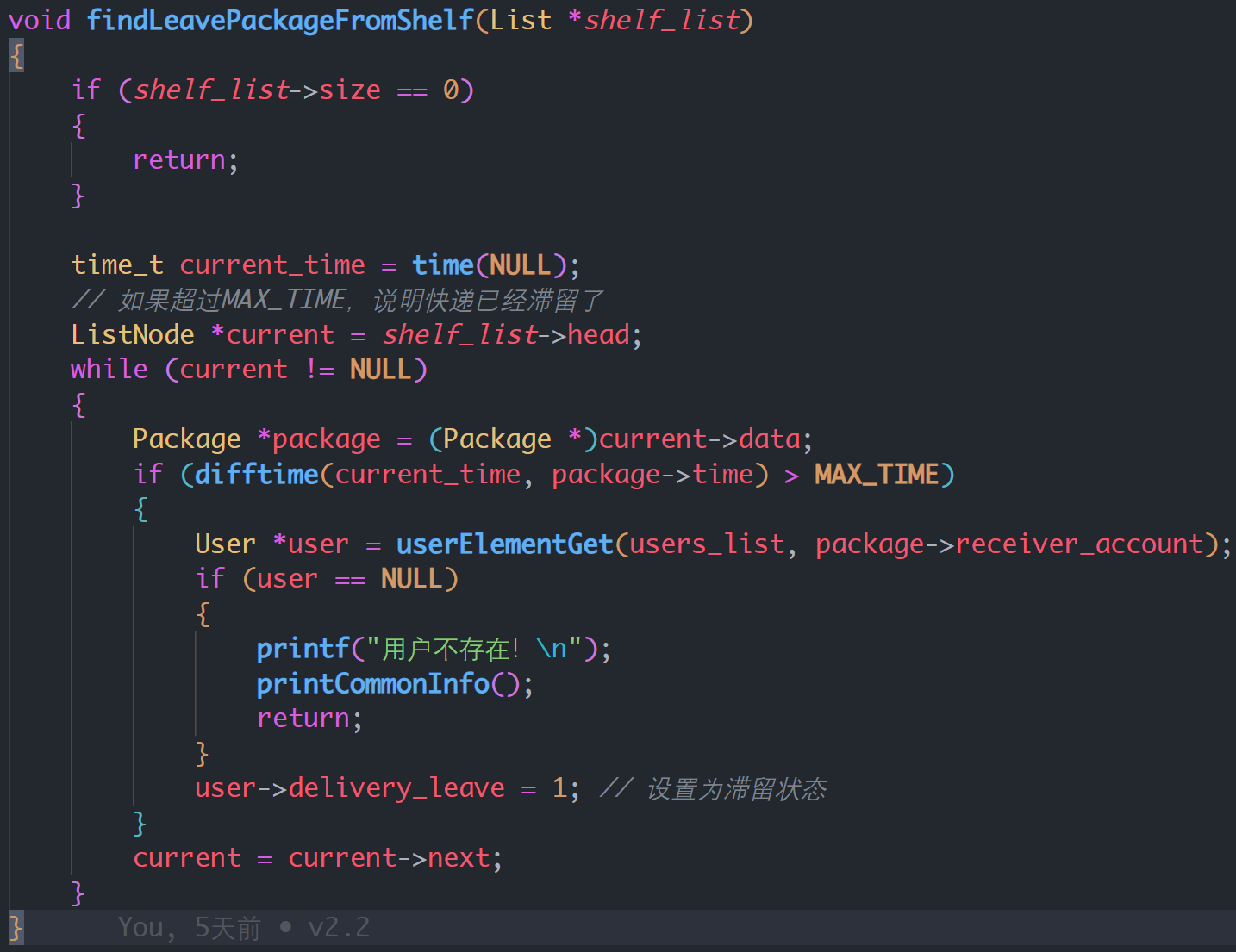
业务统计记录了所有用户的寄件、取件情况，并标注了时间。

修改货架信息操作中，对于快递单的修改也会永久记录，同样也有时间标注。



1. 提示包裹滞留用户前来取件

该操作是自动执行的。当管理员登录账号后，会自动遍历各货架，根据快递的时间戳和当前时间的差值来判断，如果滞留时间超过1天，那么该包裹会被标记为“滞留状态”。



1. 查看驿站账户

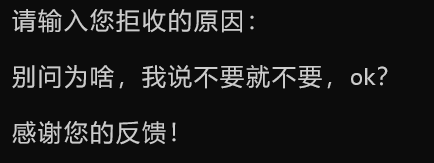
该账户即为之前提到的账户。



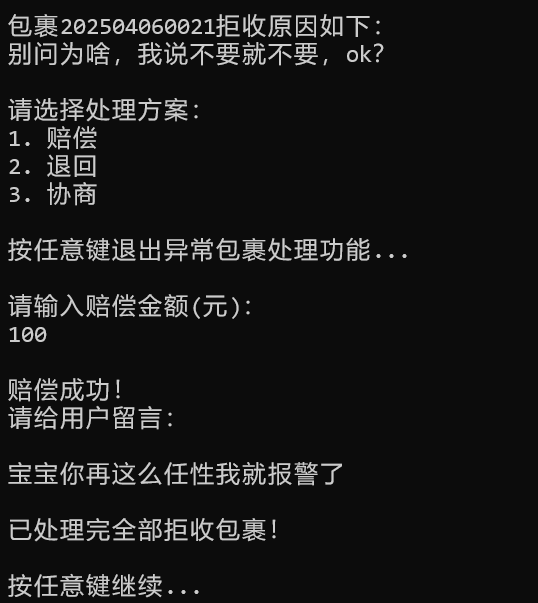
1. 处理拒收包裹

若用户拒收快递，对应包裹会加入到拒收链表中来，需要管理员进一步处理。

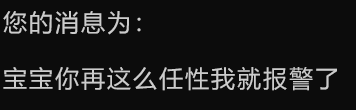
用户拒收：



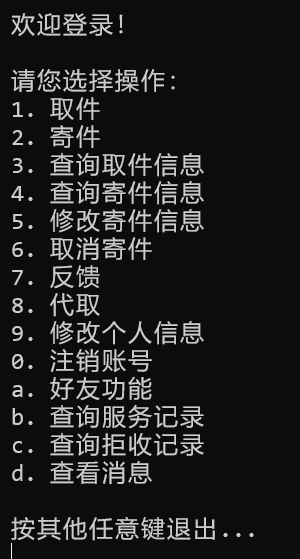
管理员处理：



用户消息接收：



1. **用户**



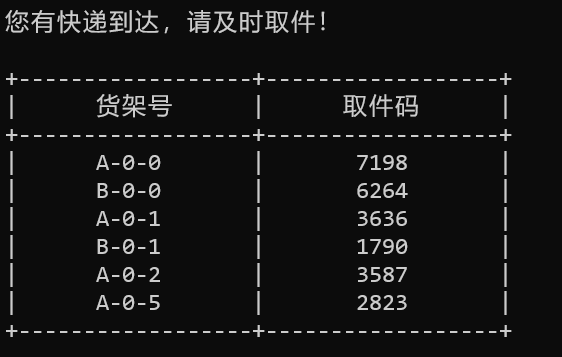
1. 预处理、弹窗

预处理：创建全局结构体指针、全局临时链表、静态全局注销变量。

遍历“用户推送链表”，若节点中包裹的收件人为此时登录的用户，则将节点加入到临时链表中。这里推送链表的节点不能提前删除，具体原因同快递员处。

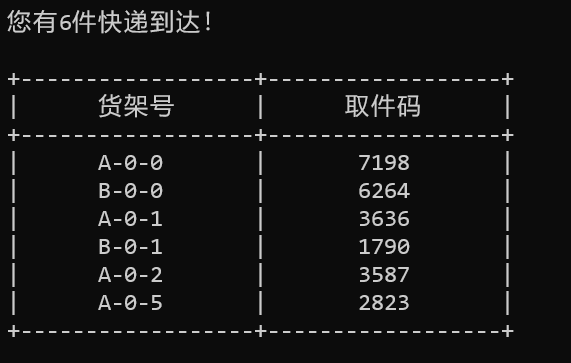
弹窗：**滞留提示弹窗**、**取件弹窗、消息通知弹窗、寄件状态弹窗**。取件弹窗中会给出取件码和快递货架编号。

// 取件弹窗样式



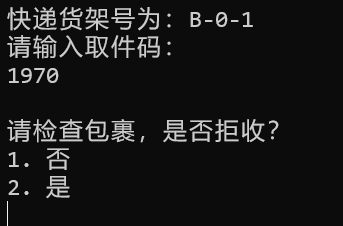
1. 取件、查询取件信息

查询取件信息，可以再次查询到取件码和快递货架编号。



取件的逻辑为：必须当前登录者为快递收件人，且取件码输入正确，才能取到快递。因此在程序方面，不存在“误取”的情况（除非被盗取了账号）。

// 如果正确输入取件码，会来到拒收判断

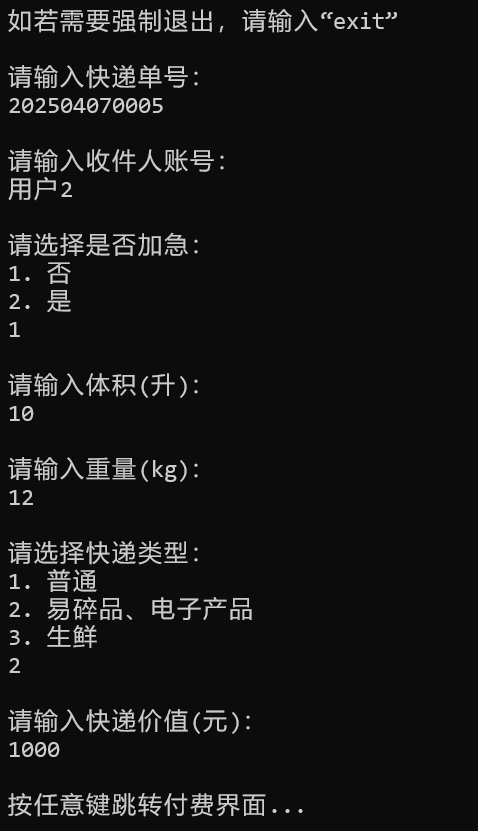


成功取件后，将该包裹节点从“临时链表”、“用户推送链表”、包裹对应的“货架链表”，同时写入行为文件，供管理员查询。



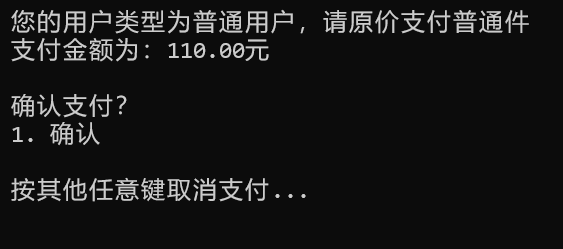
1. 寄件、查询寄件信息、修改寄件信息、取消寄件

寄件的逻辑与快递平台的写入寄件信息逻辑类似，之前也提过二者分别是相反路线的“源头”。寄件信息填写完后，跳转到**支付**界面。



支付费用取决于两方面：一是寄件包裹的属性，二是用户的类型。

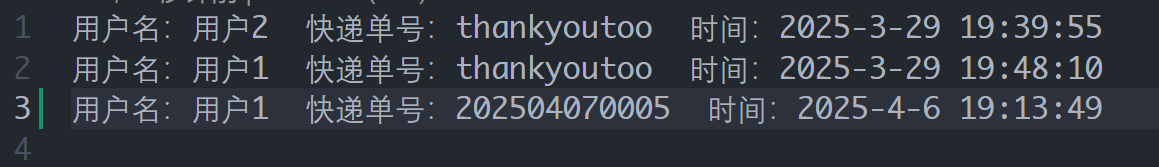
// 寄件的消费会转到驿站的账户上



包裹属性方面，根据包裹的重量、体积、材质等来确定其费用，**若快递为加急件，则需要支付双倍费用**，费用为double类型并保留两位小数。

用户分为五类：**普通用户、会员用户、企业用户、代理商用户、合作商家用户**。其中普通用户**原价支付**，会员用户享受**八折优惠**。企业用户为**月付制度**，合作商家为**年付制度**，代理商主要负责**发展下级用户并获得佣金**。

支付后寄件成功，并写入行为记录文件。

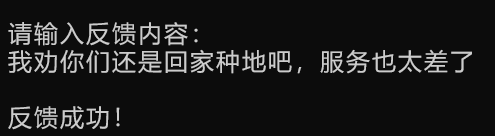


寄件状态分为四种：未寄件、未发货、运输中、已到达。

管理员发货前，状态为“未发货”；快递员运输快递时，状态为“运输中”；送达平台后，状态为“已到达”。

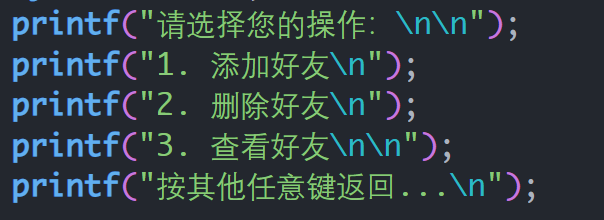
1. 反馈问题

功能很简单，直接输入即可，等待管理员登录后处理其反馈。



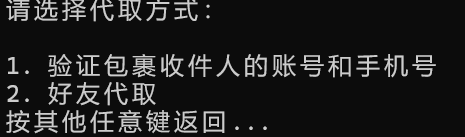
1. 好友功能

这个功能是服务于代取的，确定了好友关系后，可帮助其代取，但仍需对方提供取件码，以防万一。



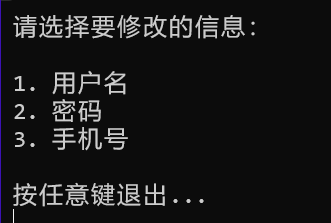
1. 代取

代取有两种方式：一是基于**好友功能**，二是通过**验证快递收件人的用户名和手机号**。二者都需要提供取件码。验证成功后，剩下的逻辑与**取件**处相同。

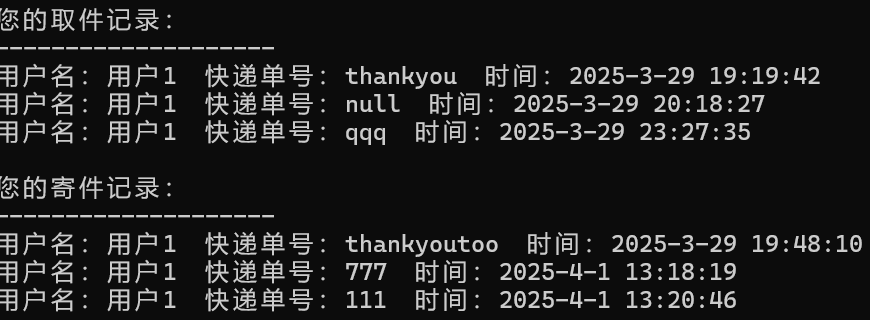


1. 修改个人信息、查询服务记录

修改个人信息，直接通过预处理创建的全局结构体指针对自己进行增删查改即可。



查询服务记录，可以查到自己账号所有的寄件、取件记录。



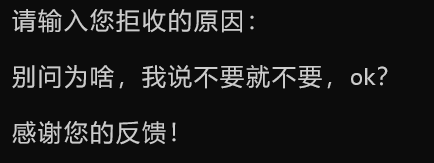
1. 注销账号

与管理员处相同，如果当前有未取的快递，不允许注销；成功注销后，通过静态全局注销变量，可直接退出该账号，**销毁临时链表**，回到登录界面。



1. 拒收

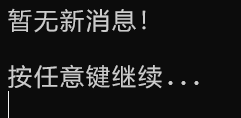
在取件时可以选择拒收，需要提供原因，供管理员处理。



1. 查看消息

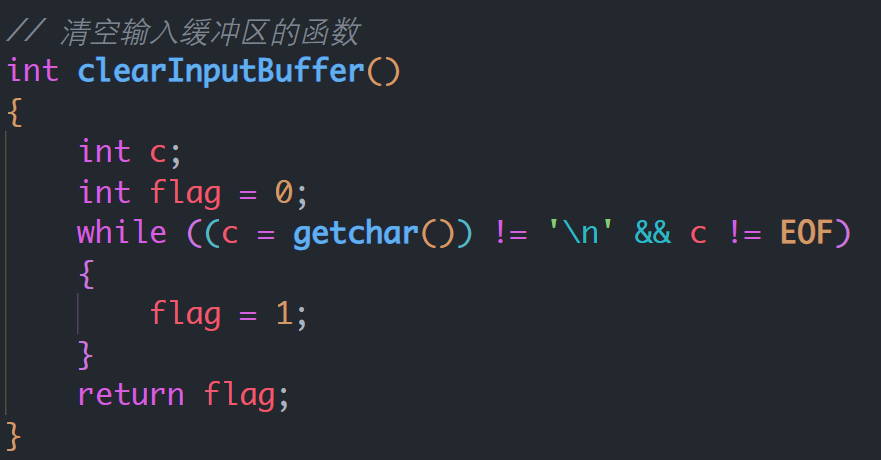
用户提交了反馈或执行了拒收操作后，会收到来自管理员的消息。

无新消息时，会给予相应提示。



1. **项目中遇到的问题**
2. 输入及缓冲区问题

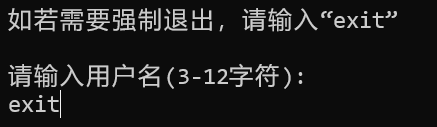
项目中使用到大量的switch语句，一开始其表达式的判断我们选择的输入是int类型，但int类型的抗暴力测试能力太弱，因此换成了char类型。但又遇到了新问题：选项一般是’1’ ’2’等单字符，而输入值又是用getchar接受的，因此如果输入类似“1kkkk”的字符串，getchar只接受第一个字符，依然可以顺利进入1的分支。所以当时下意识的改法是使用strcmp，输入也不再用getchar，但这样一来，switch语句还要改成if-else if-else，太过于繁琐抵消，因此我们选择结合缓冲区操作来解决问题。



缓冲区问题，由于整个程序是int、char、double类型交替输入的，所以必须处理处理缓冲区问题。对于，提供了一个处理缓冲区函数，每次输入完都会调用这个函数，来清空当前的缓冲区的所有内容：对于字符串输入和int类型输入，可以清除掉空白字符；对于单字符输入，可以清除掉空白字符和多余的暴力输入。

第一段中提到的问题，对于getchar接受单字符的选项，如果缓冲区清除掉了除了回车以外的字符，那就说明输入大于1字符，就会被认定为无效输入，执行switch的default分支。

还有一点就是“返回”功能。控制台的交互能力远远差于图形化界面。日常中我们使用某个软件，时刻都可以执行“返回”操作，强制停止当前的进程。在控制台中如何实现这一点呢？我们想了一个略显“丑陋”的办法：在输入界面，如果输入“exit”，即可强制回到上一个界面。缺点就是“exit”本身成为了“禁词”，无法作为账号、密码使用。



实现方法：对于所有输入，最优先进行该判断



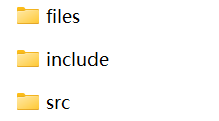
1. 代码复用率低问题

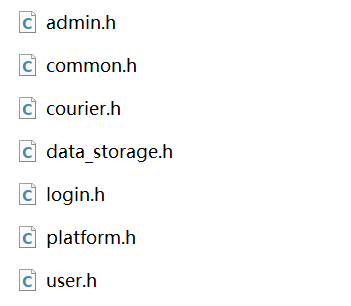
虽然代码量最终到了7000余行，但其中不乏许多代码都是重复的。这个问题是后期才察觉到的，由于相似的功能没有统一封装成函数，导致修改和维护比较困难。应当归因于项目开始前准备不足以及各成员经验不足。

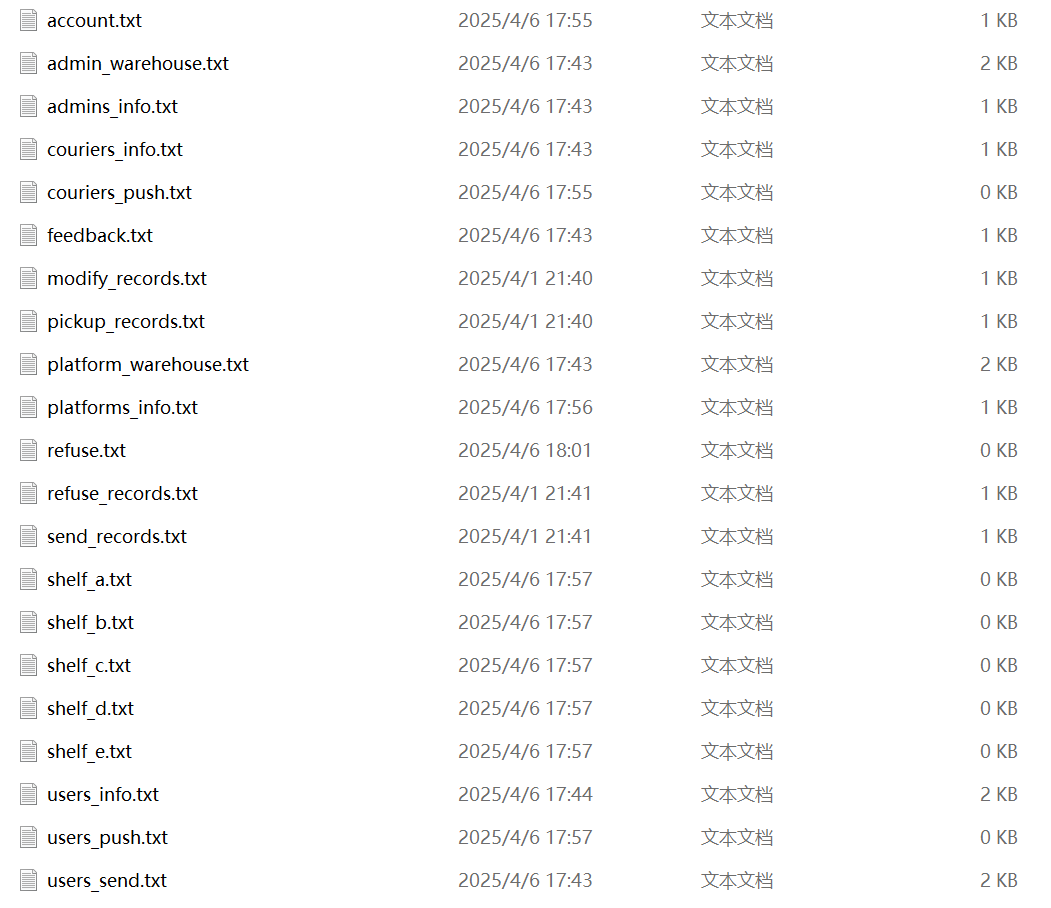
1. 分文件编写问题

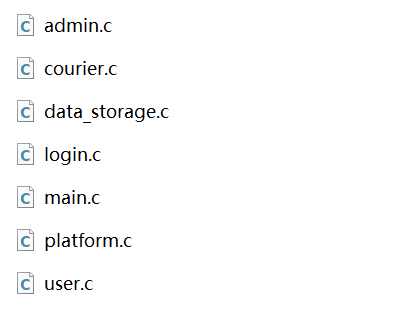
我们文件分得比较少，应当独立出来的功能，如输入、查询等完全可以单独拿出来，这样还有利于代码修改。

// files储存txt文件，include放所有的头文件，src放所有的源文件









1. 编码问题和路径问题

组内成员使用的IDE不同，对字符默认编码格式不同，在VSCode中能顺利运行的代码，在VS中却报了上百个错误。后来经过学习才了解，不同IDE对字符默认编码不同（VSCode默认字符编码格式为UTF-8），以及默认工作路径不同（错误路径会使得文件操作、头文件包含出错）。

最终我们选择的字符编码格式为GBK。

1. 图形化问题

我们组没有进行图形化设计。曾浅尝辄止，先后接触了easyx、SDL、Qt等库。

我们是总体代码完成后，最后才想进行图形化设计。本以为提供几个接口就可以较为顺利解决，但发现需要大量地对原代码进行修改才可以，而我们代码量较大，已经将近6000行，图形化工作量太大。其中easyx和Qt需要将文件后缀改成.cpp，而支持.c的SDL难度又很大，因此最终我们放弃了这个工作。

1. 释放堆区内存问题

本程序中，我们使用的通用链表节点，数据为泛型指针 \*void ，其指向的数据也是储存在堆区的。在此之前没处理过类似的问题，以为“如果释放了节点，那对应泛型指针的数据也会释放”，但后面经过查阅资料，发现并不会释放。操作不当的话，泛型指针指向的数据可能永远访问不到，造成内存泄漏。

对于这个问题，本程序进行了尽可能精细的改善，尽可能避免了内存泄漏问题。

// 对所有的全局链表，都提供合适的释放内存函数

