电子信息与工程学院

《ACM》和《数据结构》小组课程设计

系　　别 电子信息类

专　　业 电信

成　　员 李春涛 宋奇育 鞠思淼

年　　级 2023 级

指导教师 柴铭

2023-2024 学年　第 2 学期

航空客运订票系统设计报告

程序设计选择题目：1.航空客运订票系统（难度系数 100%）；
小组成员：李春涛 宋奇育 鞠思淼
李春涛 23211131
宋奇育 23211136
鞠思淼 23211130

一、 需求分析

【运行环境】

Devc++

问题描述：航空客运订票的业务活动包括：查询航线、客票预订和办理退票等。试设计一个航空客运订票系统，以使上述业务可以借助计算机来完成。

【基本需求】

（1） 每条航线所设计的信息有：航班 ID、起点站名、终点站名、飞机号、飞行周日（星期几）、成员定额、总余票量、经济舱余票量、商务舱余票量、已订票的客户名单、经济舱候补客户名单、商务舱候补客户名单。
其中，经济舱候补客户名单和商务舱候补客户名单中的包括以下几个域：姓名、身份证、所需票量、预定票的等级

（2） 作为示意系统，数据放在内存中；

（3） 系统能实现的操作和功能如下：

① 录入航班信息：自由输入航班信息，包括航班号、起终点、乘客数等到系统中，可一次输入多个航班信息，数据存储到变量中。

② 查看所有航班信息。

③ 清除航班信息：选择航班 ID，将系统中某航班删去，每次删除一个航班；

④ 查询模块：输入起点站和终点站进行搜索。

⑤ 订票模块的功能：

A：输入航班 ID 进行订票，当航班 ID 不存在或者不合法时要求重新输入航班 ID，然后输入订票数量和等级，数量和等级要合法，订票数目要大于 0，等级输入 1 代表经济舱，输入其他数据商务舱。不合法则重新输入，再判断对应等级的票是否足够 ，足够则继续输入客户信息（姓名和身份证），然后订票成功。当对应等级的票数不足时，询问是否改变订票计划，同意改变计划则重新输入航班 ID 进行相应的订票操作。不同意改变计划则继续询问是否排队等候，不同意排队等候则询问是否需要推荐相同航

线的其他航班，询问操作时，输入 Y 或者 y 表示同意，输入 N，n 或者其它数据表示不同意。

B：根据起点和终点查询航班

C：查询所有航班的相关信息并保存在文件中

D:：查询航班的订票情况，输入航班 ID，当航班 ID 不存在或者不合法时要求重新输入航班 ID，根据 ID 查询航班，显示出已订票的客户的姓名、订票数目和仓位等级，为了保密，不能显示客户的身份证。

⑥ 退票模块的功能：

A:先输入航班 ID，然后输入姓名和身份证号码进行验证，验证成功则办理退票手续。然后查询该航班是否有人排队候补，如果有，而且票数够了，则首先询问排在第一的客户，是否需要订票，是则为他办理订票手续，否则出队，依次询问其他排队候补的客户。若刚刚退票的是经济舱，则询问经济舱排队的客户；若是商务舱，则询问商务舱排队的客户。

B：退出系统

⑦ 查询旅客信息：输入航班 ID，显示旅客相关信息，包括姓名、订票数目、起终点等；

⑧ 退出系统。

【测试数据】（为了提高可读性，详细测试数据将在测试结果中一一说明）：

```
//程序一执行就将五个基本航班插入航班链表
//以下两个变量为全局变量
Flight *pFlight; //定义全局指针变量 pFlight，航班链表的头指针

//五个基本航班
Flight flight1[5] = {
        {"北京", "郑州", "A1", "K1250", "星期日", 12, 12, 6},
        {"北京", "包头", "B2", "L6525", "星期三", 12, 12, 6},
        {"北京", "鹤岗", "C3", "K1010", "星期三", 12, 12, 6},
        {"北京", "上饶", "D4", "L6333", "星期二", 12, 12, 6},
        {"北京", "琼海", "E5", "K1210", "星期五", 12, 12, 6},
};
```

【选做内容】

A：当客户所要订的航班票额不足的时候，系统会根据目的地，输出目的地相同的航线，询问客户是否要订票。（已做）

B：将航班信息保存在内存文件中（已做）

C：并对航班序号和旅客序列进行排序（已做）

二、 概要设计：

1. 所要用到的头文件

```
#include <iostream>
#include <cstring>
#include <cstdio>
#include <malloc.h>
#include <fstream>
#include <ctime>
#include <iomanip>
```

2. 宏定义

```
#define OK 1
#define ERROR 0
#define OVERFLOW -1
#define FALSE -1
#define TRUE 1
```

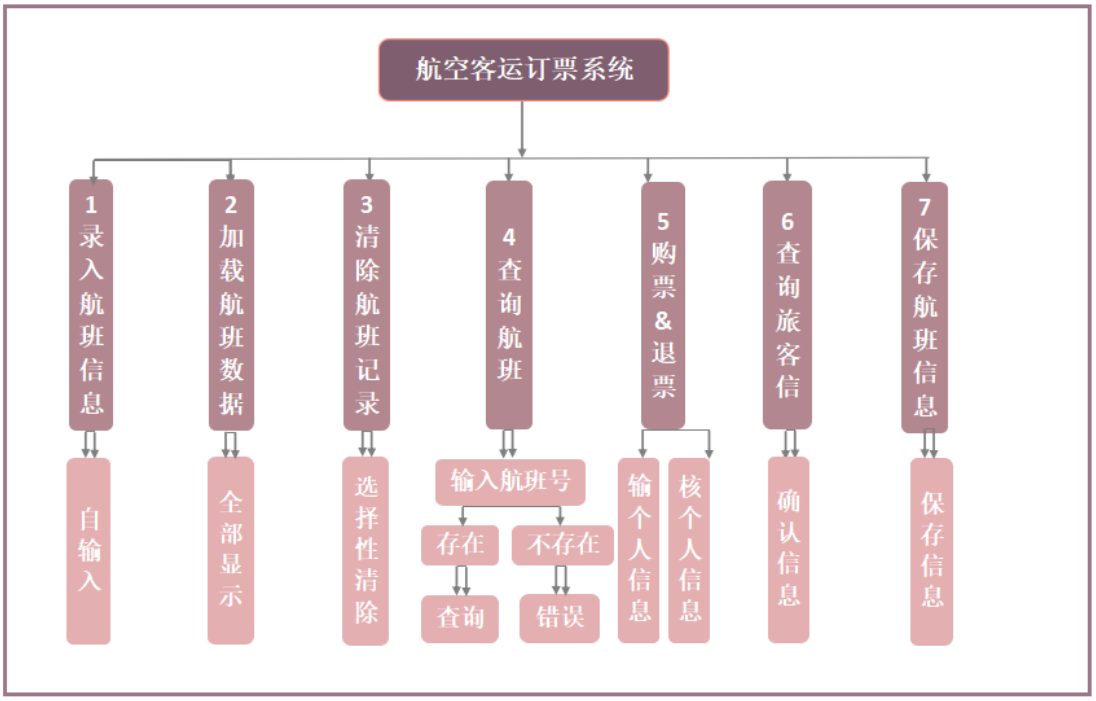
3. 类型定义

```
using namespace std;

typedef int Status;
//航班日期枚举类，星期一到星期天
enum Week{
	Mon = 1, Tues = 2, Wed = 3, Thurs = 4, Fri = 5, Sat = 6, Sun = 7
};

//乘客节点
typedef struct CustomerNode
{
	char name[10];//客户姓名
	int clientTickets;//客户订票量
	char identification[20];//客户身份证号码
	int rank;//舱位等级
	CustomerNode *next;
} CustomerNode, *CusLinkList;

//候补队列中的节点
typedef struct WaitPassenger
{
	char name[10];//姓名
	char identification[20]; //身份证
	int preTickets;//预定的票量
```

```
    struct WaitPassenger *next;
} WaitQNode, *PWait;

//候补队列
typedef struct Queue
{
    PWait front;//等候替补客户名单域的头指针
    PWait rear;//等候替补客户名单域的尾指针
} LinkQueue;

//封装乘客的姓名和订票量和身份证
//用于候补客户出队时把关键字返回
typedef struct NameAndNumAndID
{
    char name[10];//姓名
    char identification[20]; //身份证号码
    int num;//订票量
} NameAndNumAndID;

//航班节点
typedef struct Flight
{
    char startPoint[20];//起点站名
    char destination[20];//终点站名
    char flightCodeID[20];//航班 ID（相当于主键）
    char planeNum[20];//飞机号
    char day[20];//飞行日期（星期几）
    int totalTickets;//乘员定额(总票数)
    int left;//总余票量
    int leftEconomicTicket; //经济票剩余量
    int leftBusinessTicket; //商务票剩余量
    Flight *next;
    CusLinkList cusLinkList;//乘员名单域，指向乘员名单链表的头指针
    LinkQueue waitQueue1;//经济舱候补，等候替补的客户名单域，指向一个队列
    LinkQueue waitQueue2;//商务舱候补，等候替补的客户名单域，指向一个队列

} Flight, FlightNode, *PFlight;
```

4. 接口设计（见 function_declaration.h）

5. 基本的模块调用关系



三、 详细设计：

（一）、 **主页面**：选择功能项;

（二）、 **功能项描述**：

1. 录入航班信息：自由输入航班信息，包括航班号、起终点、乘客数等到系统中，可一次输入多个航班信息，数据存储到变量中；

2. 加载航班数据：全部显示航班信息，包括系统已有航班以及自行输入航班；

3. 清除航班记录：选择航班 ID，将系统中某航班删去，每次删除一个航班；

4. 查询航班：输入起点、终点在系统中查询地点对应的全部航班，并将所有对应航班信息显示；

5. 购票&退票：购票：先输入航班 ID，若航班 ID 正确，进入航班购买系统，输入乘客基本信息以及座位类别及数目进行购票；若航班 ID 不存在，则返回重新输入正确航班 ID;退票：输入退票乘客姓名后核对乘客身份，根据身份证号码核对，若输入正确，退票成功；输入错误号码，退票失败。

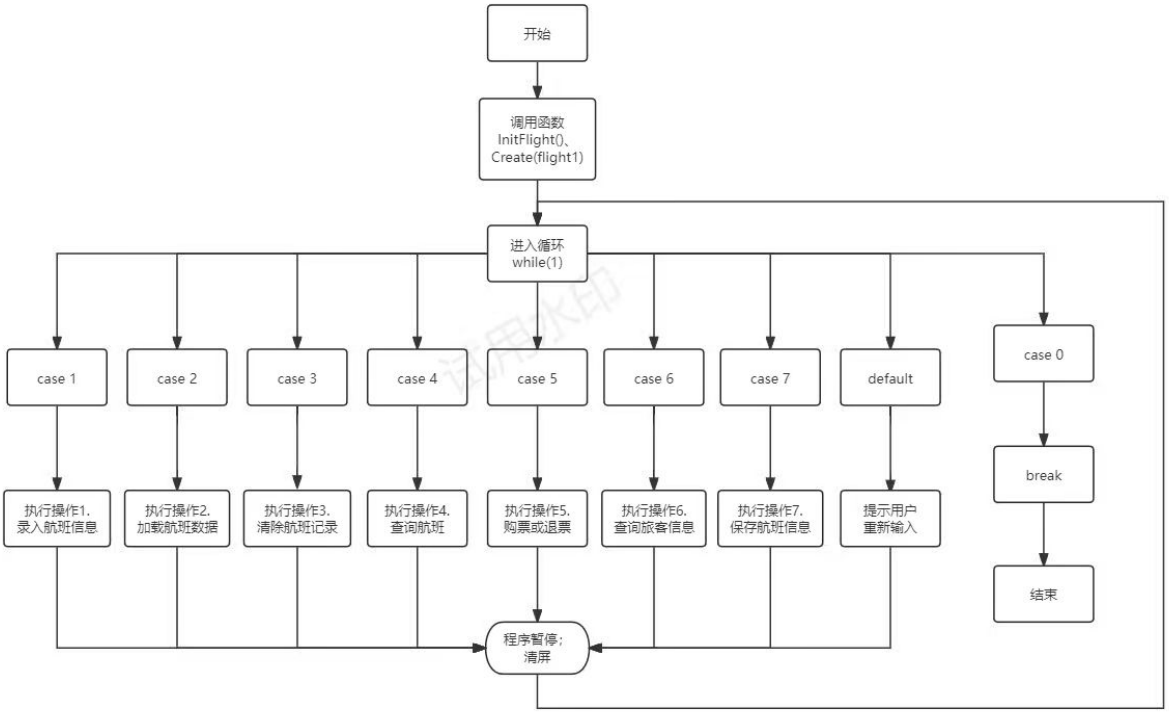
6. 查询旅客信息：输入航班 ID，显示旅客相关信息，包括姓名、订票数目、起终点等；

7. 保存航班信息:将航班信息保存到系统中；

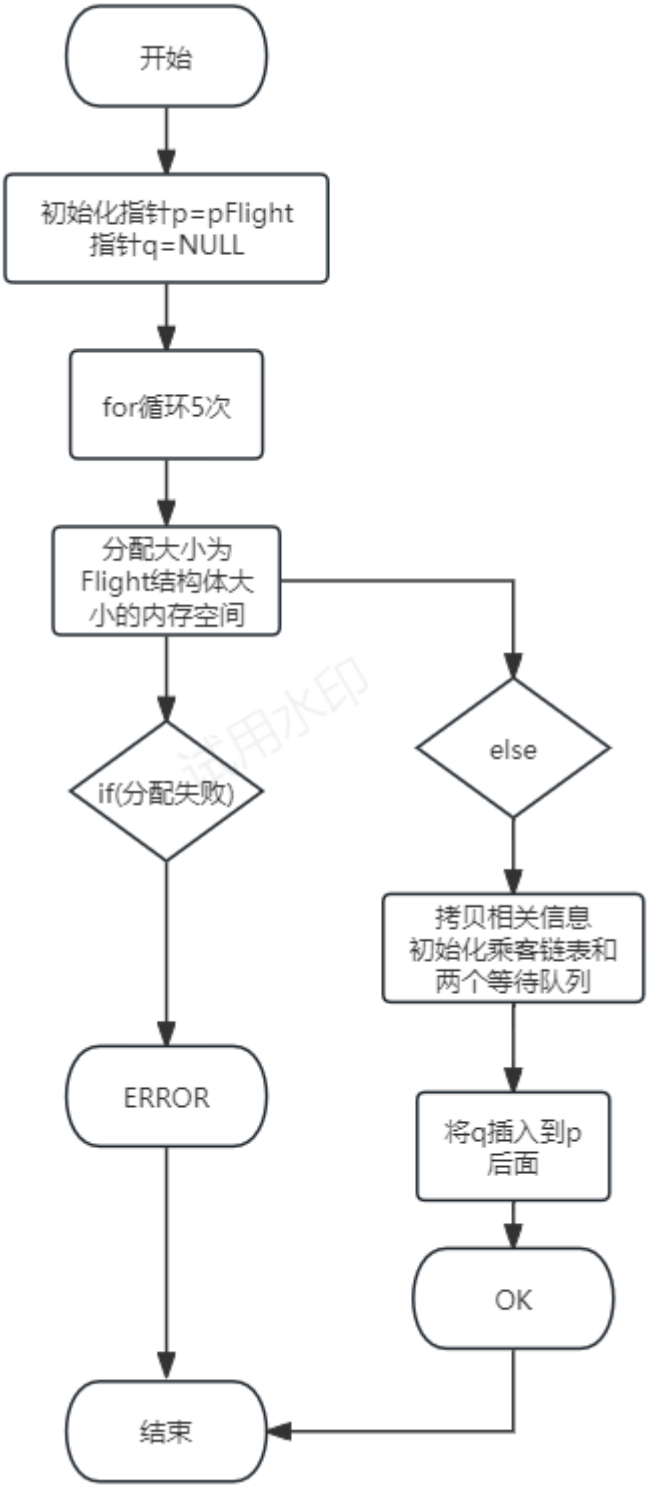
0. 退出系统：结束程序；

（三）、 **流程图**：

1. **main 主函数流程图**：主函数 1

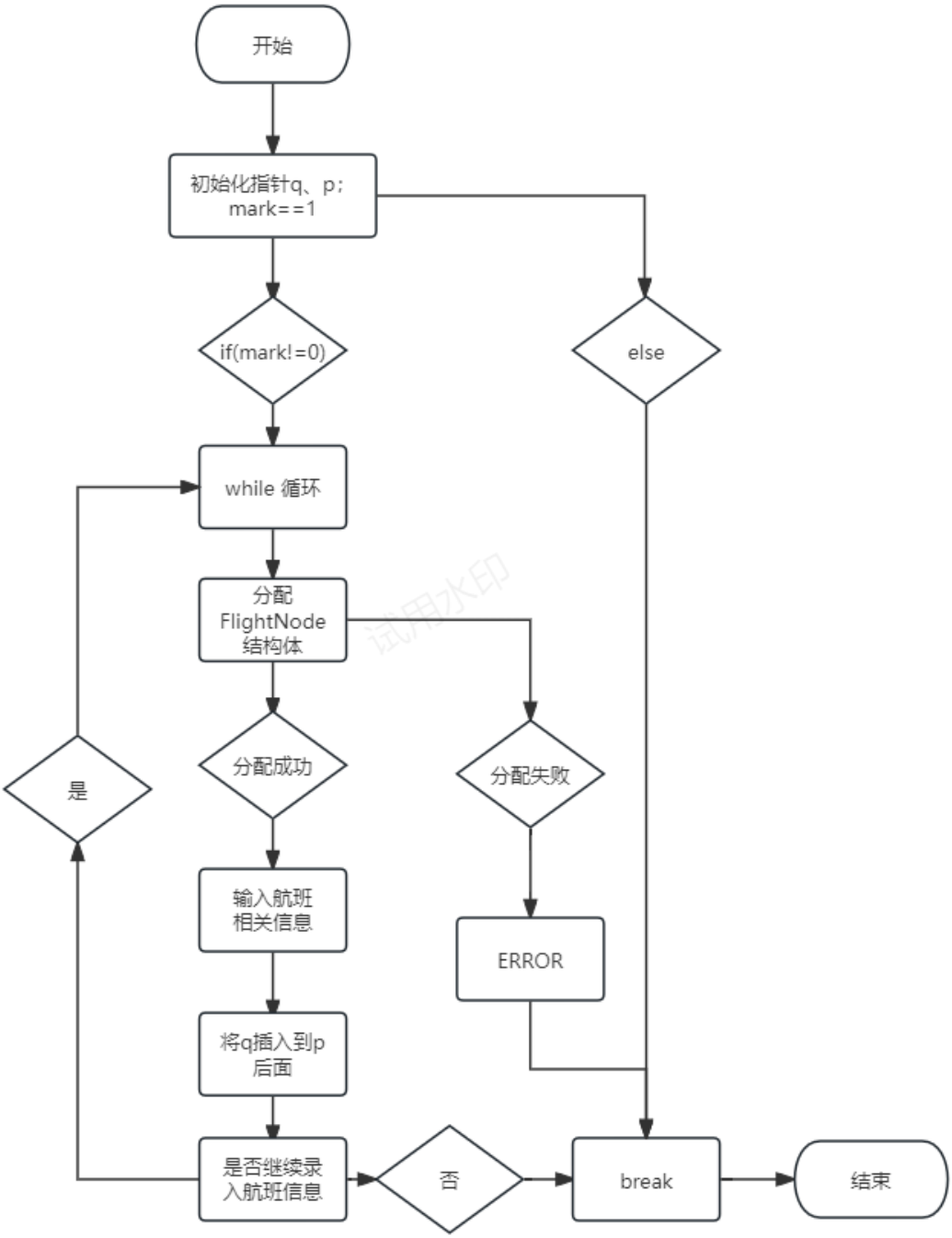


2. **Create 函数流程图**：程序最开始



流程图展示了 create 函数的创建过程，包括了内存分配、信息拷贝、初始化乘客链表等操作。

3. InsertFlight 函数流程图[选择 1：录入航班信息]:

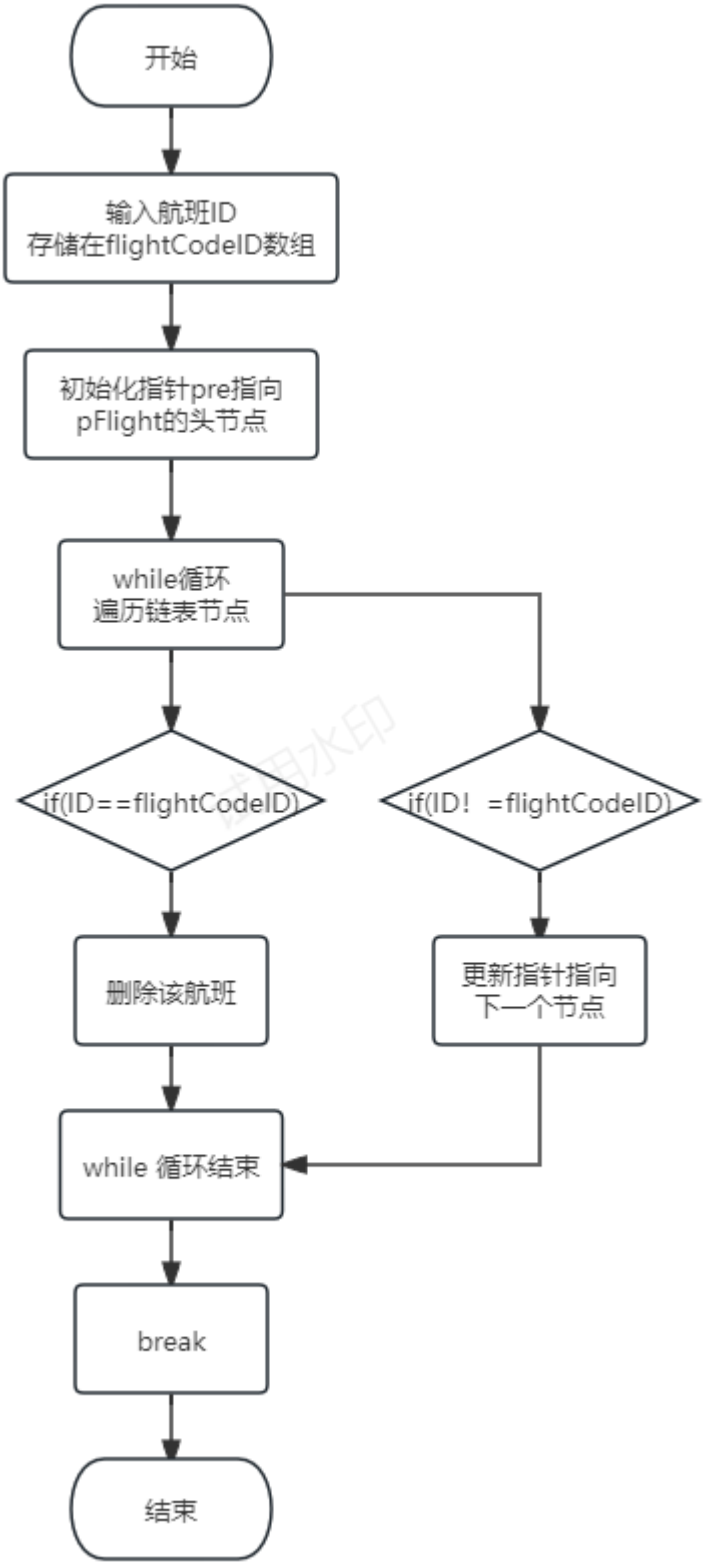


流程图展示了 Insertrlight 函数的逻辑，包括了动态内存分配、用户输入信息、航班 ID 重复检查以及循环录入航班信息的过程。

4. PrintFlightList 函数流程图： 程序 2[加载航班数据]：

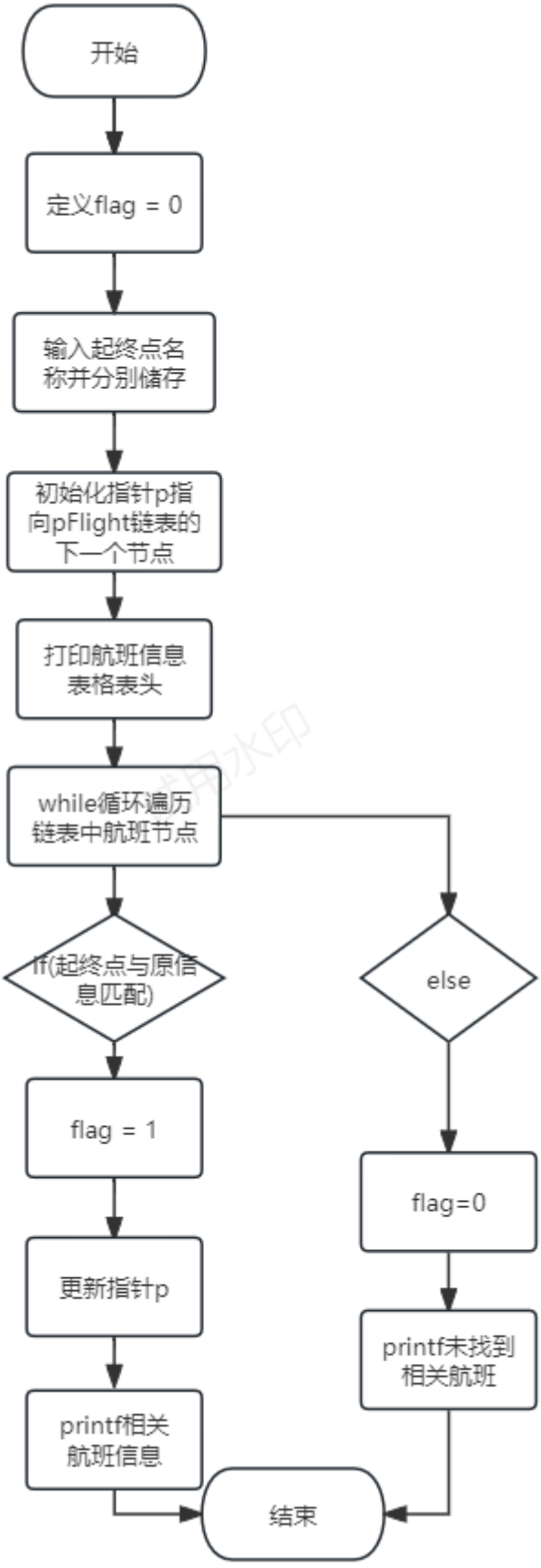
流程图展示了 PrintFlightList 函数的逻辑，包括了对链表进行排序、打印表头、循环打印每个航班节点信息的过程。

5. **DeleteFlight 函数流程图：**程序 3[清除航班记录]：



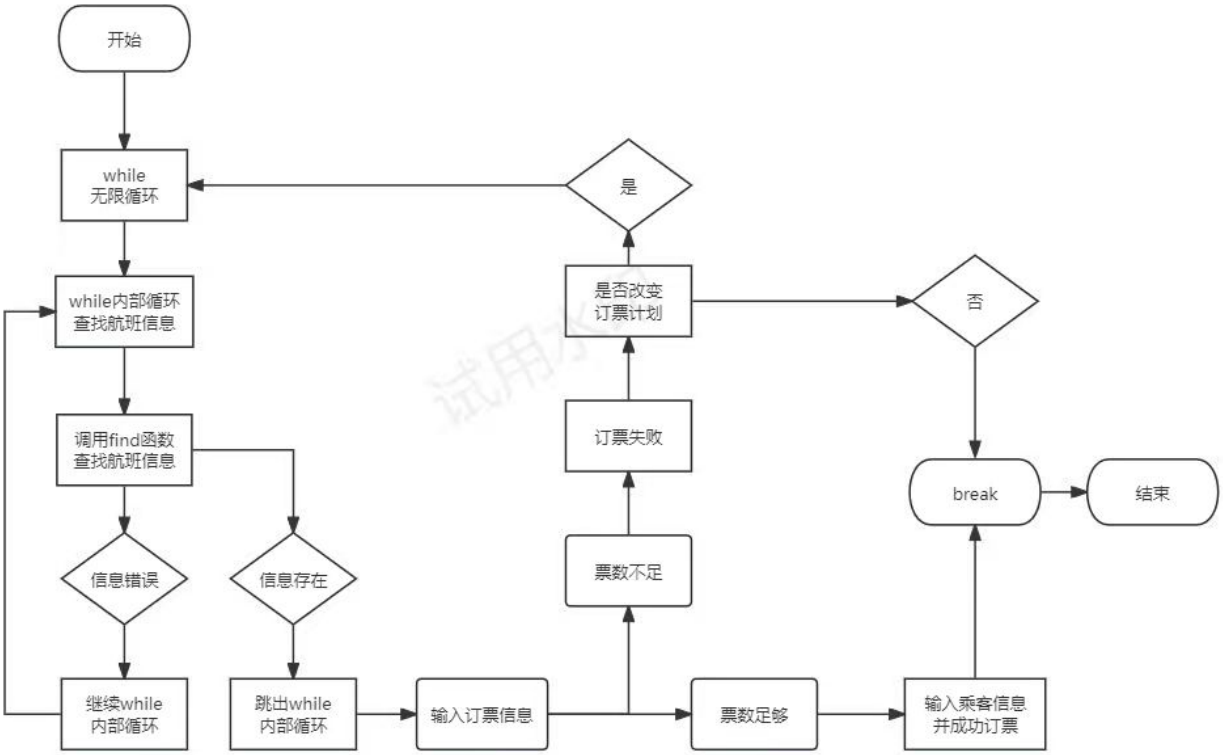
流程图展示了 DeleteFlight 函数的逻辑包括了输入航班 ID、遍历链表查找匹配航班 ID、删除节点并释放内存的过程

5. SearchFlight 函数流程图: 程序 4[查询航班信息]

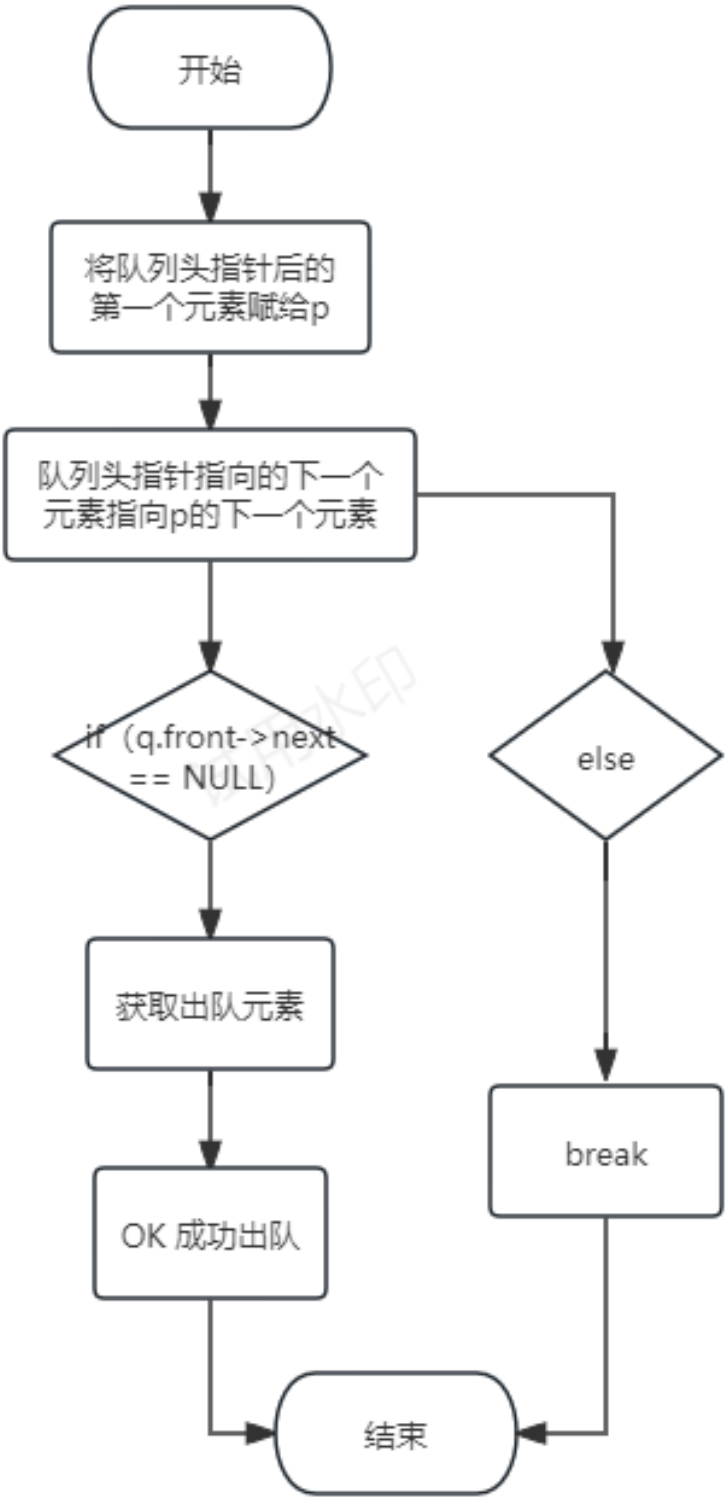


流程图展示了 SearchFlight 函数的逻辑，包括了输入起点站名和终点站名、遍历链表查找匹配航班信息并打印的过程。

6. **BookTickets 函数流程图**：程序 5 上[购票]

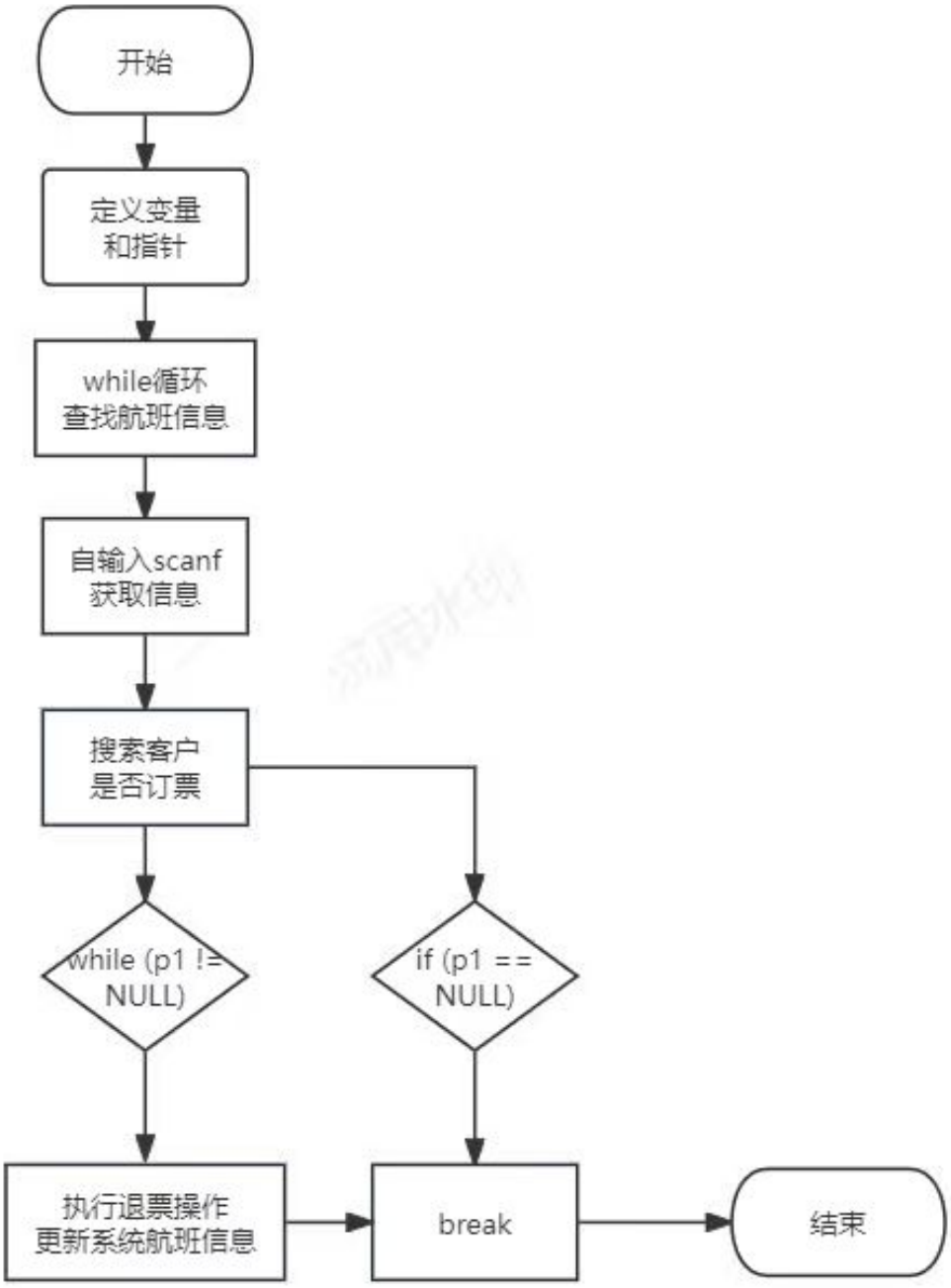


7. **QueueDelete 函数流程图**:程序 5 中[候补]



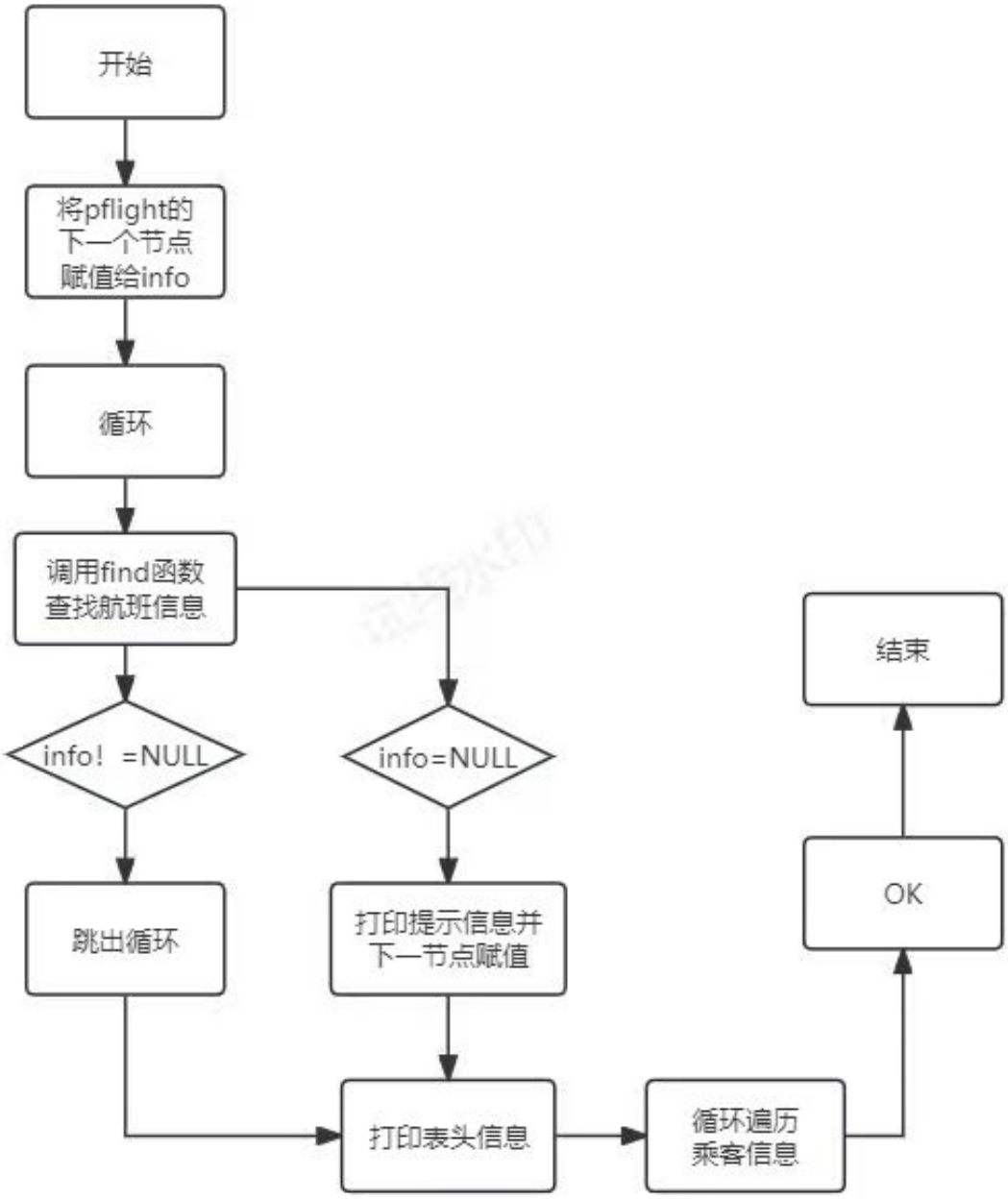
流程图简洁地展示了 QueueDelete 函数的逻辑，包括了判断队列是否为空、出队、获取关键信息和释放内存空间的过程。

8. ReturnTicket 函数流程图：程序 5 下[退票]

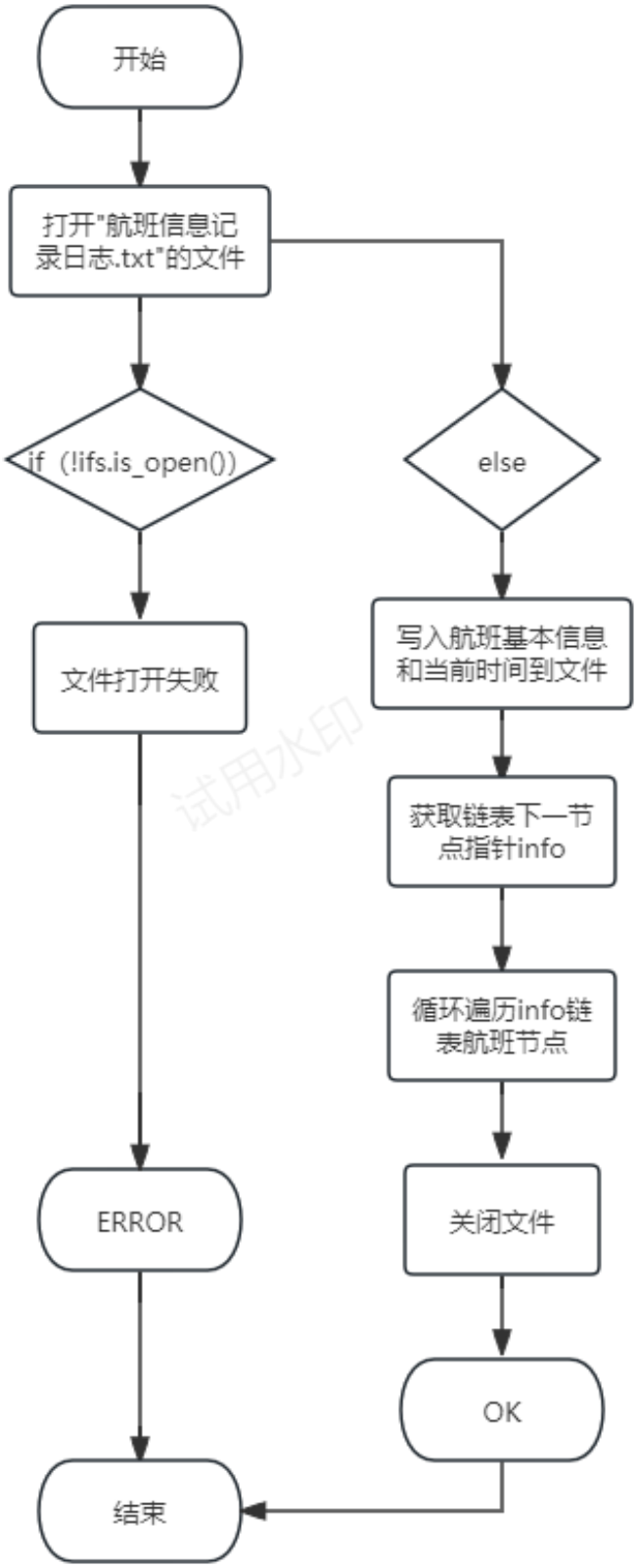


流程图概括了 ReturnTicket 函数的逻辑，包括查找航班信息、搜索客户订票、退票操作、候补订票询问和更新航班信息的过程。

9. Visit_Passenger_Information 函数流程图[选择 6：查询旅客信息]：



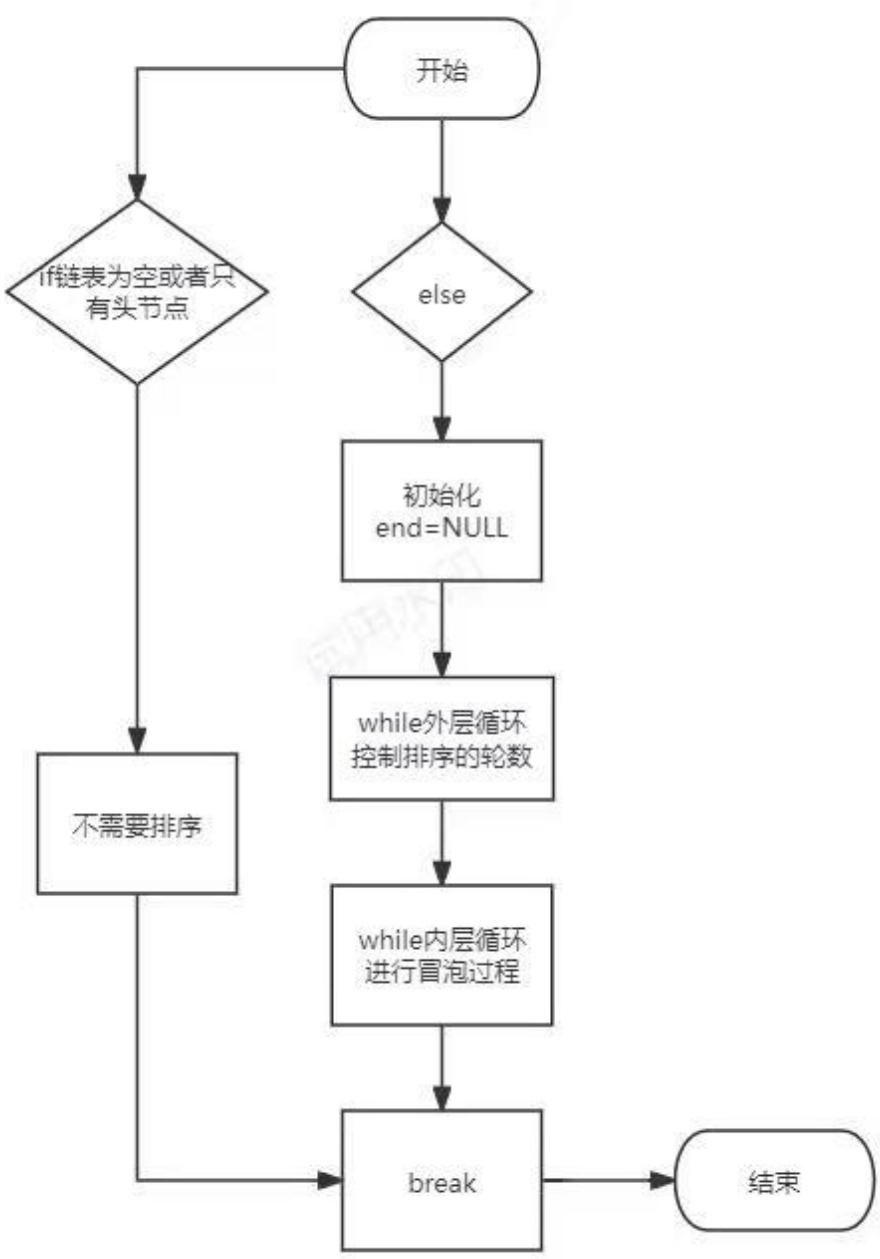
10. Log Flight Information 函数流程图: 程序 7



流程图展示了 Log_Flight_Information 函数的逻辑，包括了打开文件、获取时间信息、写入航班信息到文件并格式化的过程。

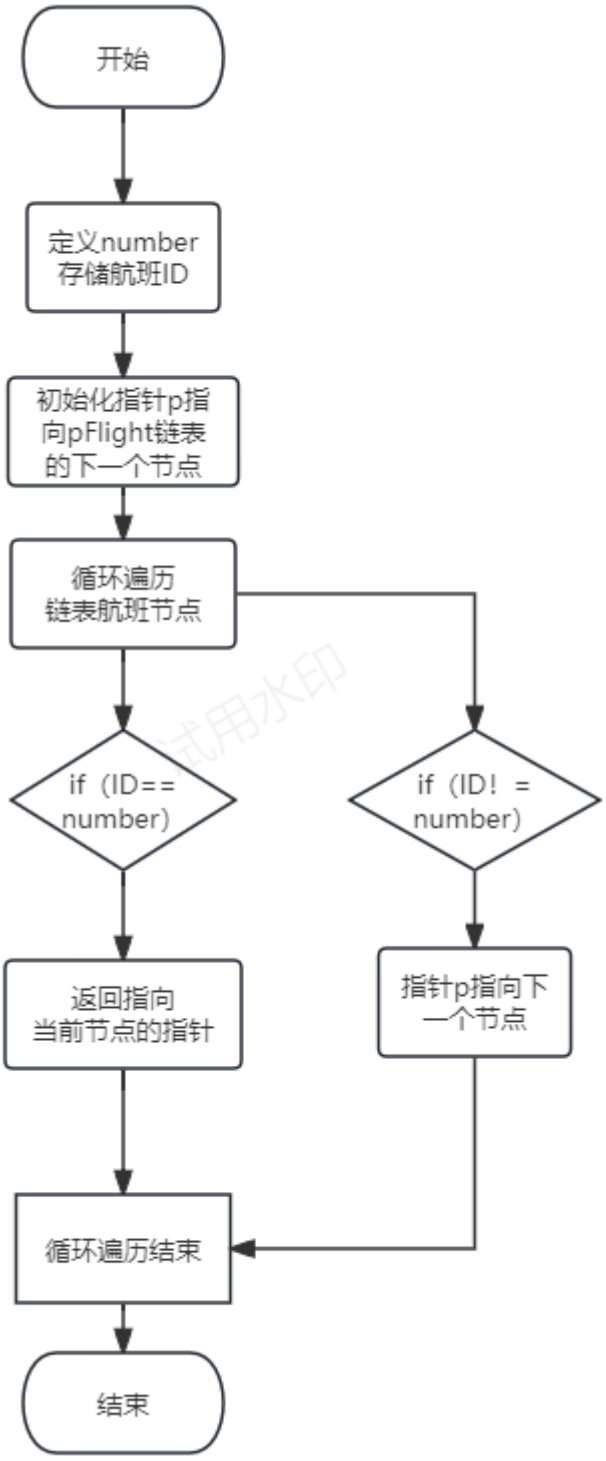
11. 部分重要函数的流程图：

1). bubbleSort_CusLinkList 函数流程图：程序后缀[冒泡排序]：



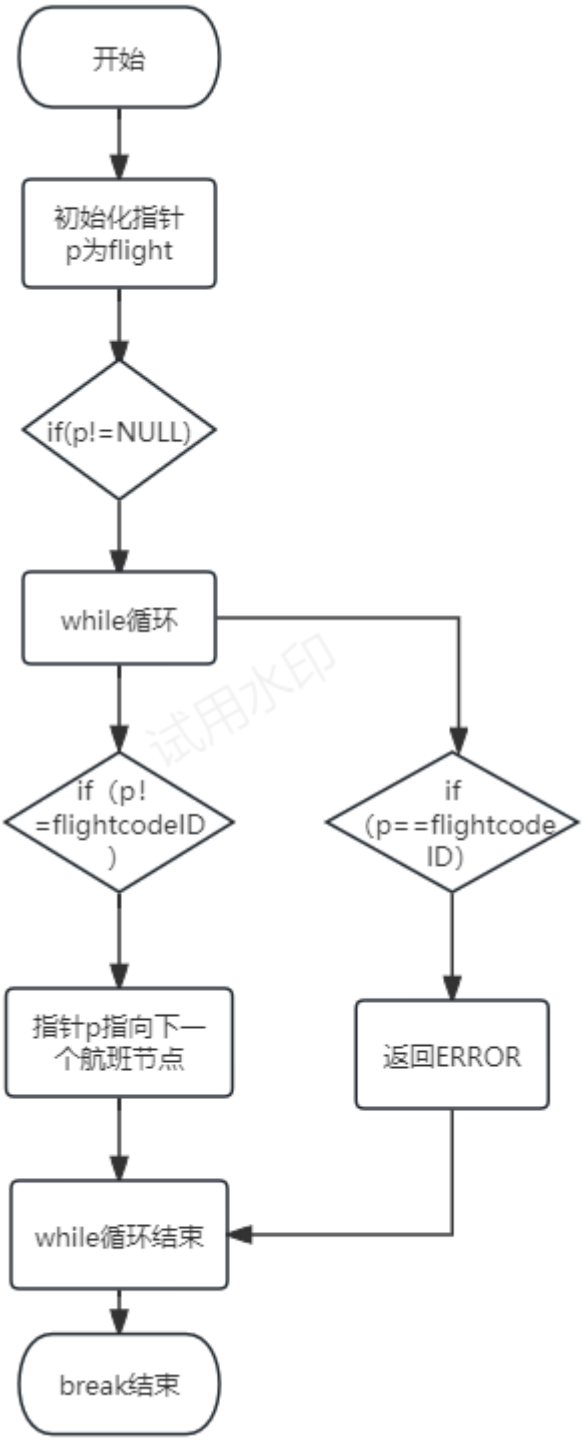
流程图概括了 bubbleSort_CusLinkList 函数的排序过程，包括外层循环控制排序轮数、内层循环进行冒泡比较和节点交换的过程

2). find 函数流程图：程序首



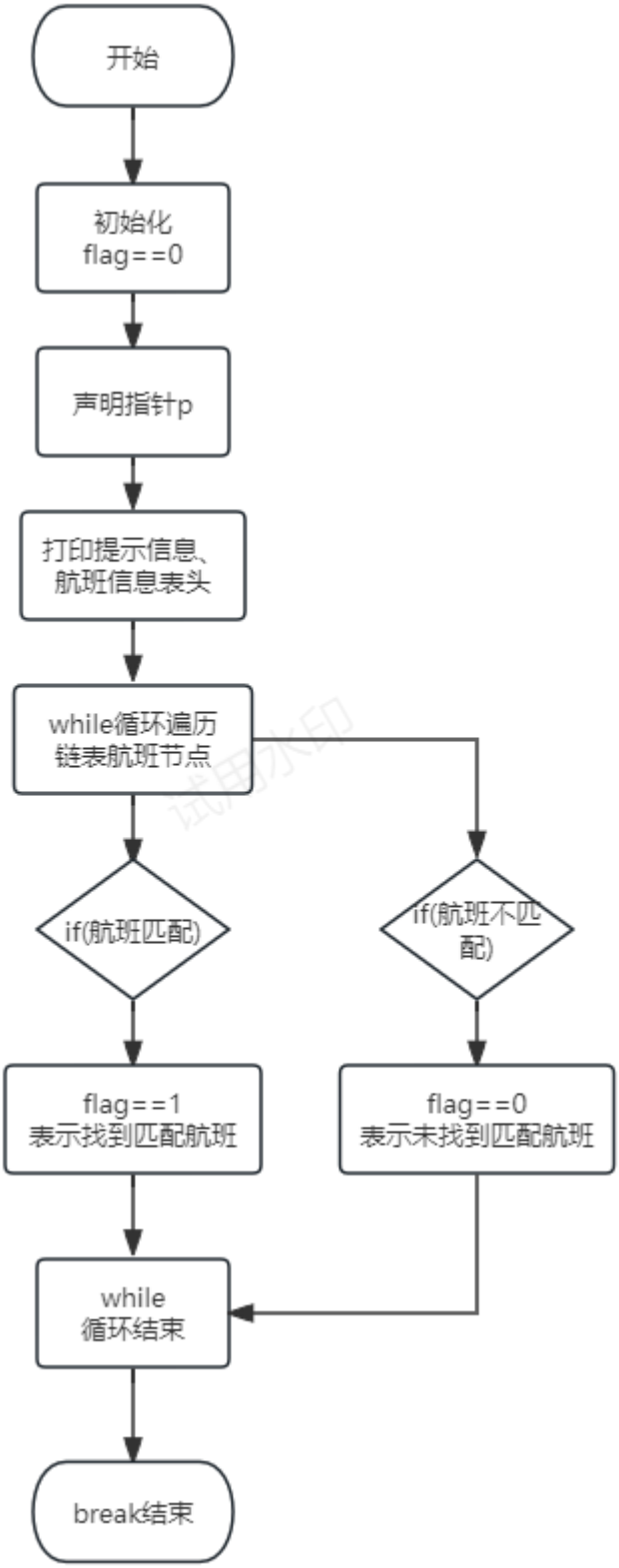
find 函数流程图展示了 find 函数的逻辑，包括了输入航班 ID、遍历链表查找匹配航班 ID 的过程，并返回相应的指针。

3). TraverseFlight 函数流程图: 程序预备 1



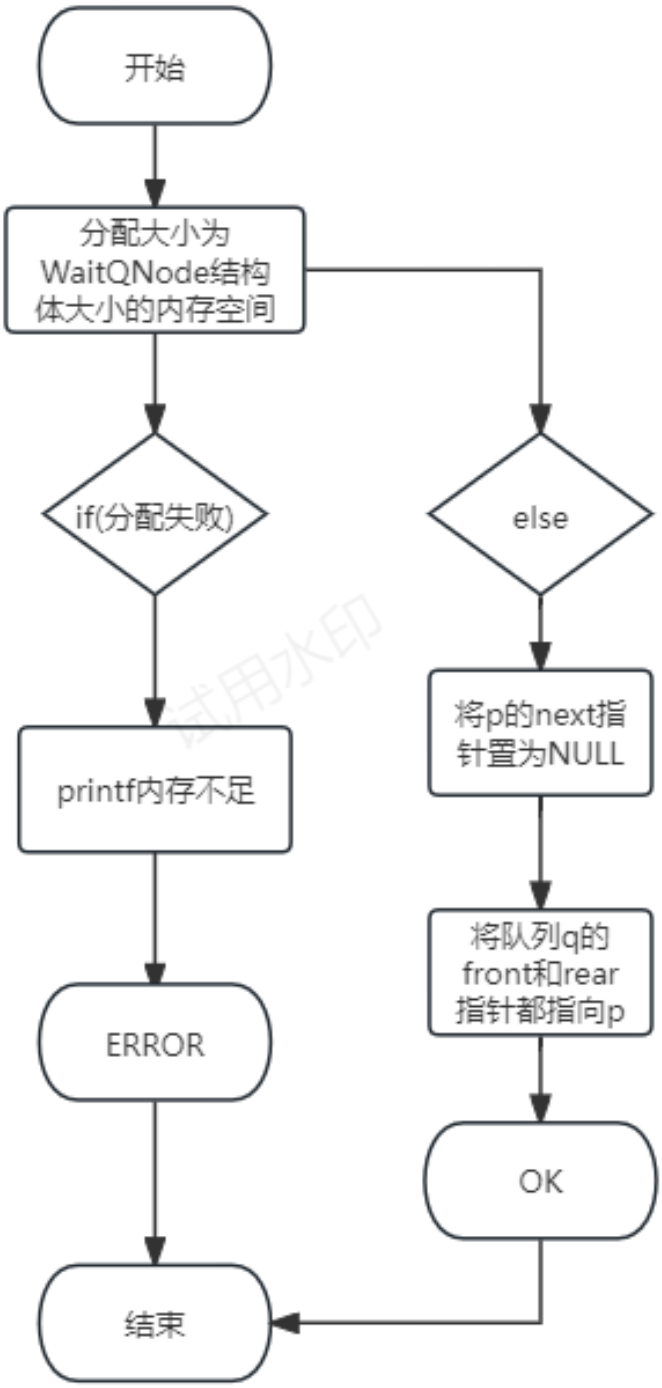
流程图展示了 Traverserlight 函数的遍历过程，检查航班 ID 是否重复的逻辑。

4). RecommendFlight 函数流程图: 程序预备 2



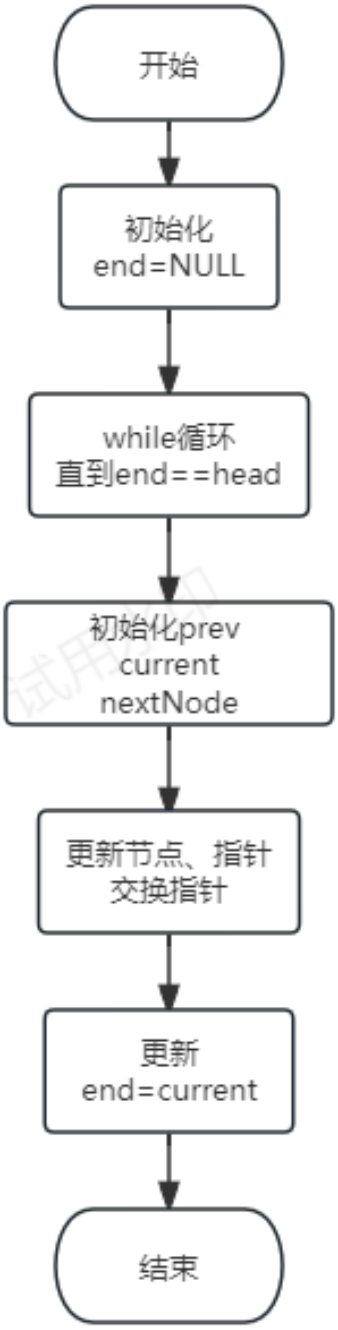
流程图展示了 RecommendFlight 函数的逻辑，包括了寻找符合条件的航班、展示航班信息以及返回结果的过程。

5). InitQueue 函数流程图: 程序预备 3



流程图展示了 InitQueue 函数的初始化过程，包括了内存分配、指针赋值的操作。

6). bubbleSort_PFlight 函数流程图： 程序预备 4



四、 调试分析

1. 航班采用单链表连接起来，采用头插入法，插入的时间复杂度为 O(1)，删除和查询的时间复杂度为 O(n),空间复杂度都为 O（1）；候补客户使用含有头指针和尾指针的链队列实现，所以入队和出队的时间复杂度和空间复杂度都为 O（1）。

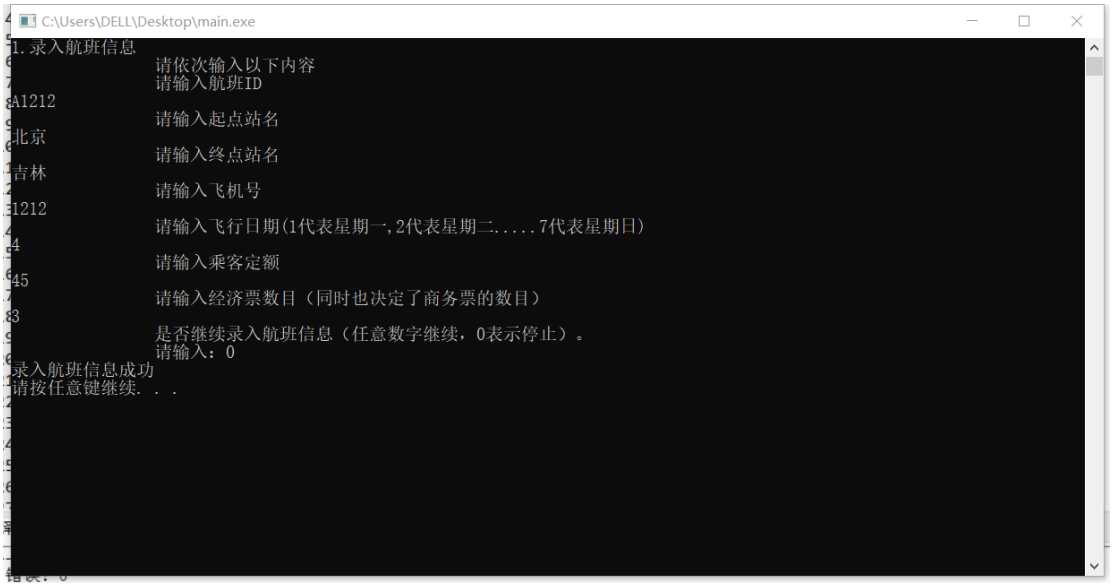
2. 航班和候补队列都要用到单链表，一开始我使用尾插入法的，后来觉得头插入法更简单，所以最后均采用了头插入法插入，为了插入和删除节点方便，使用了带头结点的链表和链队列。
3. 关于舱位等级和剩余票问题，一开始我是采用了票数和舱位等级无关，自己随意输入舱位等级都行，但是后来发现这样并不合理，后来就将总剩余票数分为两部分，一部分是经济舱剩余票数，一部分是商务舱剩余票，这样，在候补客户排队订票时，就也需要分开两条队列，一条是在经济舱队列排队，一条是在商务舱队列排队。
4. 在链表或者队列的最后一个节点的 next 指针域一定要置为 NULL，否则就会出现野指针，可能会导致实验出现错误。
5. 在输入的过程中，要注意参数值的合法性，比如票数不能出现负数，不合法的参数要求重新输入。
6. 在退票的时候原本我是采用了输入航班 ID 和姓名就能退票了，后来觉得不合理，增加了身份证号码验证，验证成功才能退票。
7. 在身份证的输入时有一个明显的不足，因为身份证不是随意输入一系列数字就行了，这里本来应该采用正则表达式进行判断的，但是由于编译器不带<regex.h>头文件，无法直接调用里面的函数进行判断，又受限于时间问题，因此身份证号码不够充分。
8. 当票数不足时，要采取多种判断来为客户服务，这就涉及到了很多 if else 的嵌套。
9. 在这次写代码的过程中，出现过很多问题，如指针错误，空指针异常，野指针等等，在处理这些 Bug 的时候，我深刻认识到了断点调试的重要性，只有断点调试，才能快速找出 Bug 并加以改正。
10. 通过这次课设，我加深了对指针、链表和队列的理解，同时也自我感觉动手能力和算法能力有了明显的提升。

五： 测试数据：

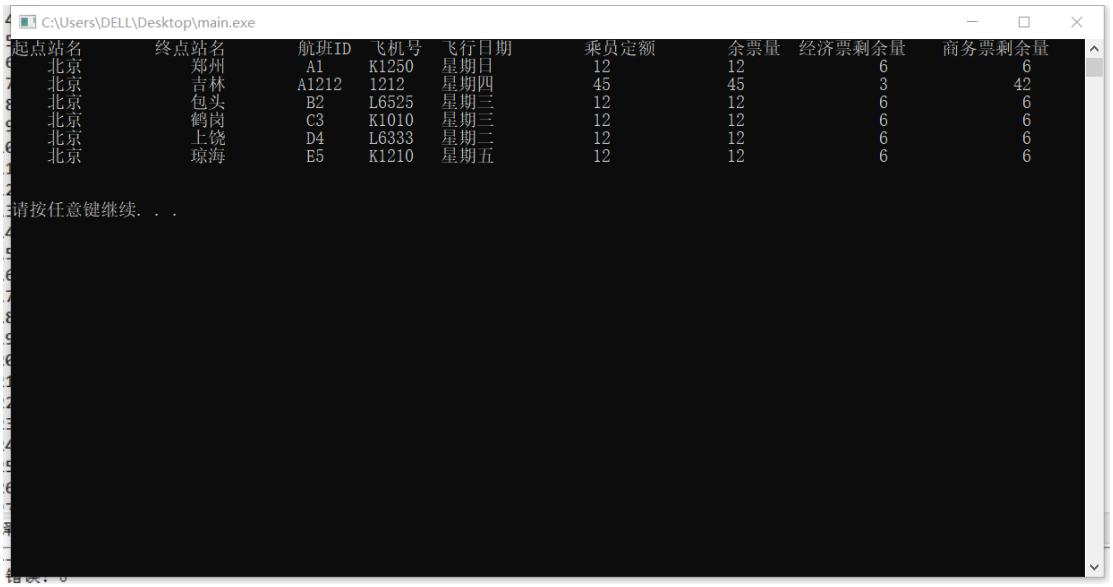
（一）、 测试数据第一组：

1. 菜单界面：输入 0-7 选择相应的功能，输入其他数字要求重新输入。

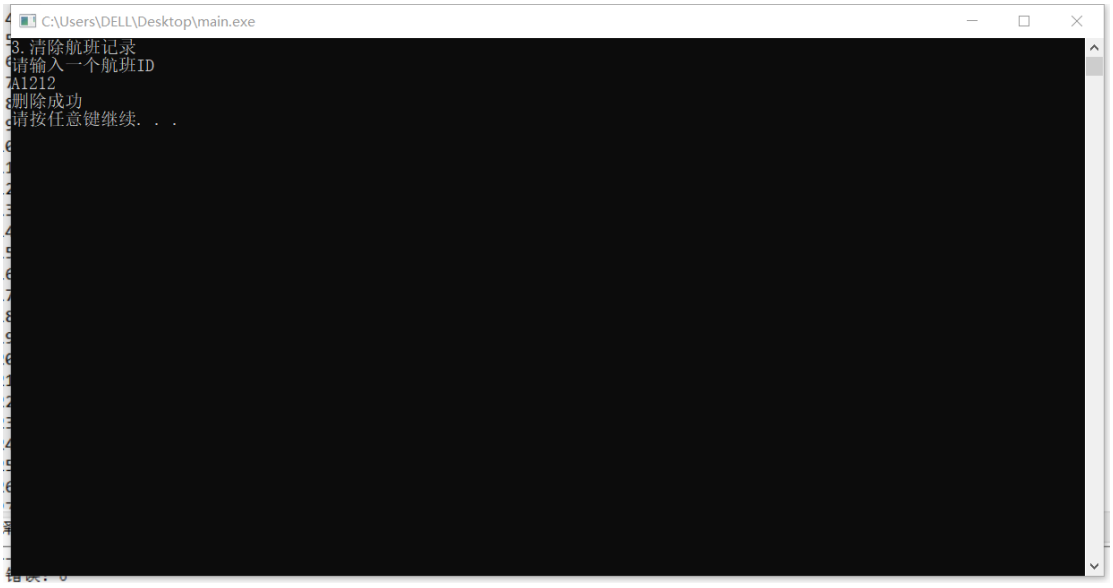
2. 从菜单中输入 1：调用 InsertFlight()函数，进入录入航班系统模块，要求用户依次输入航班 ID、起点站名、终点站名、飞机号、飞行日期、乘客定额、经济票数目（也决定了商务舱数目），在录入完毕后可以选择继续录入或者终止操作，操作完之后按任意键会返回菜单界面。



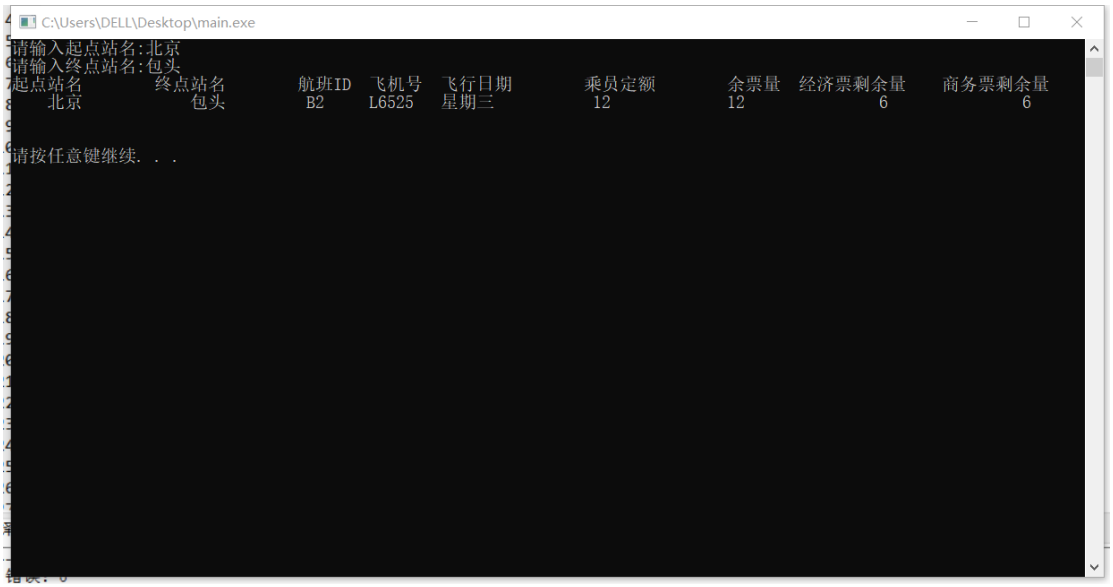
3. 从主菜单中输入 2：调用了 PrintFilghtlist(pFlight)函数，进入加载航班数据模块，其中录入的所有数据将会以飞机号的顺序排序展示，每一条数据包含航班 ID、起点站名、终点站名、飞机号、飞行日期、乘客定额、经济舱票剩余量、商务舱票剩余量。



4. 从主菜单中输入 3：调用了 DeleteFlight()函数，在输入指定航班 ID 后，会清除特定的航班数据，操作完之后按任意键会返回菜单界面。

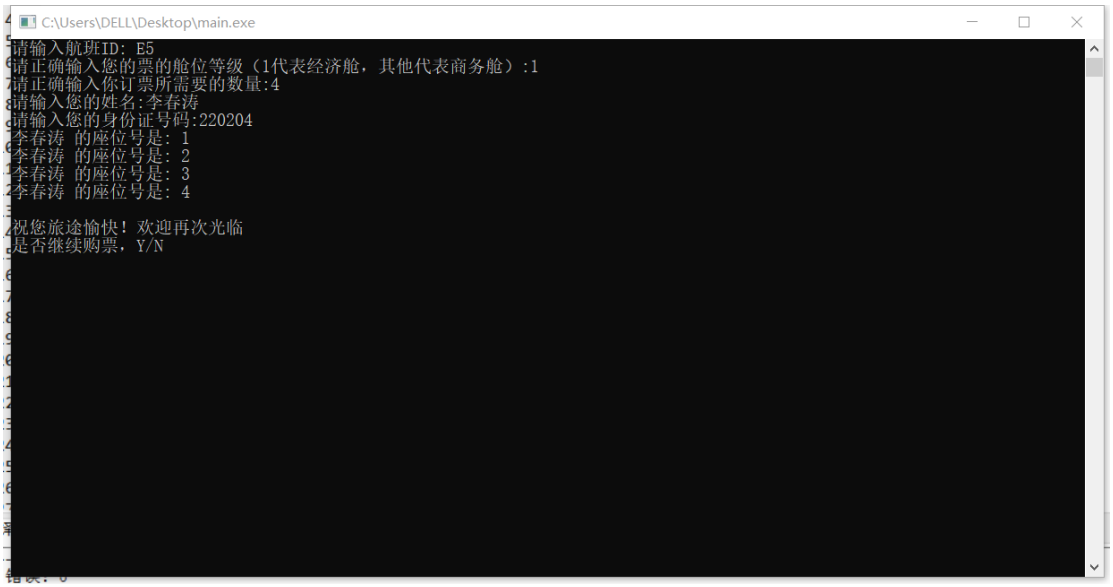


5. 从主菜单中输入 4：调用了 SearchFlight()函数，在输入指定起点站名和终点站名后，如果有相应数据则会显示特定的航班数据，否则显示“对不起，该航班未找到!”并且显示操作完之后按任意键会返回菜单界面。



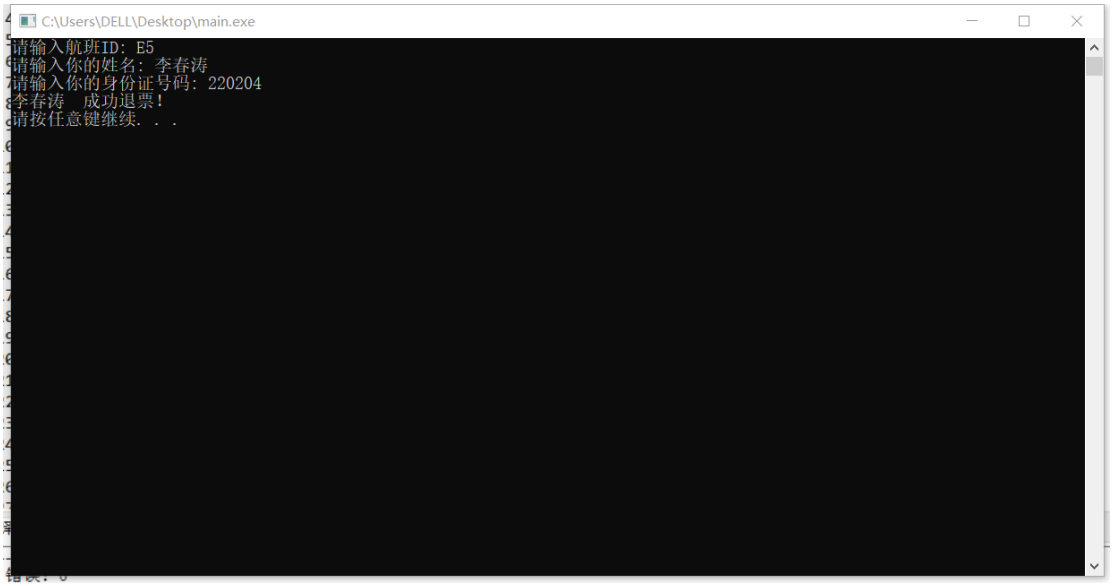
6. 从菜单中输入5：首先根据用户的选择（1或其他）分别进入购票BookTickets()，和退票 ReturnTicket()模块。

在购票模块，输入航班ID查询指定航班，在选择舱位等级、订票数量、姓名、身份证号之后，会根据已有的订票情况分配座位号，并在结束时可以选择是否继续订票。



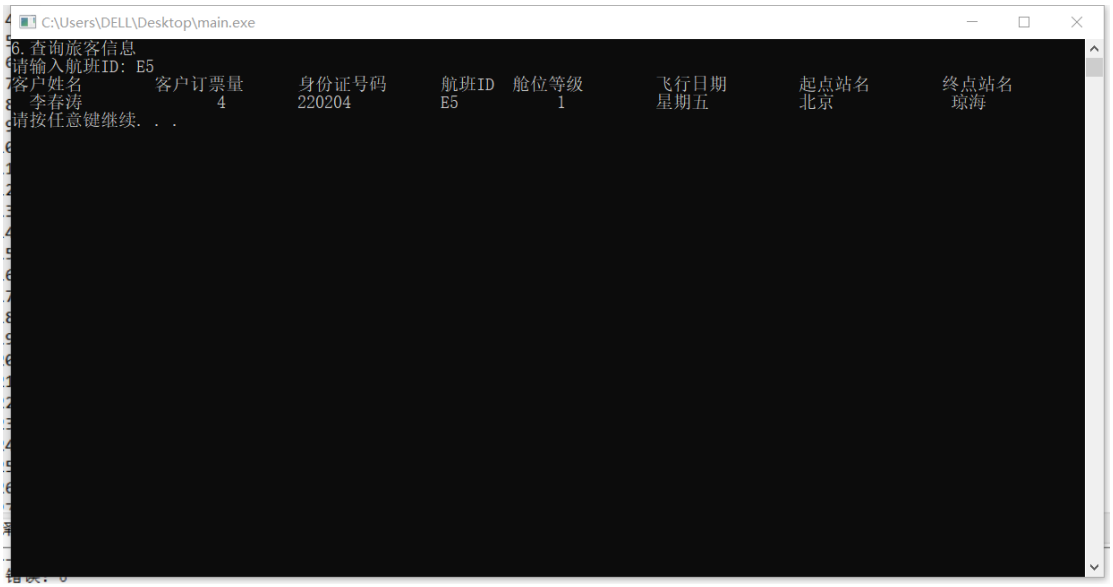
如果在购票时，该航班座位已满，则会提示余票不足，并显示航班信息，并选择是否排队候补购票。

在退票模块，输入航班 ID 查询指定航班，输入姓名和身份证号之后，会将该用户的已购票进行退票。

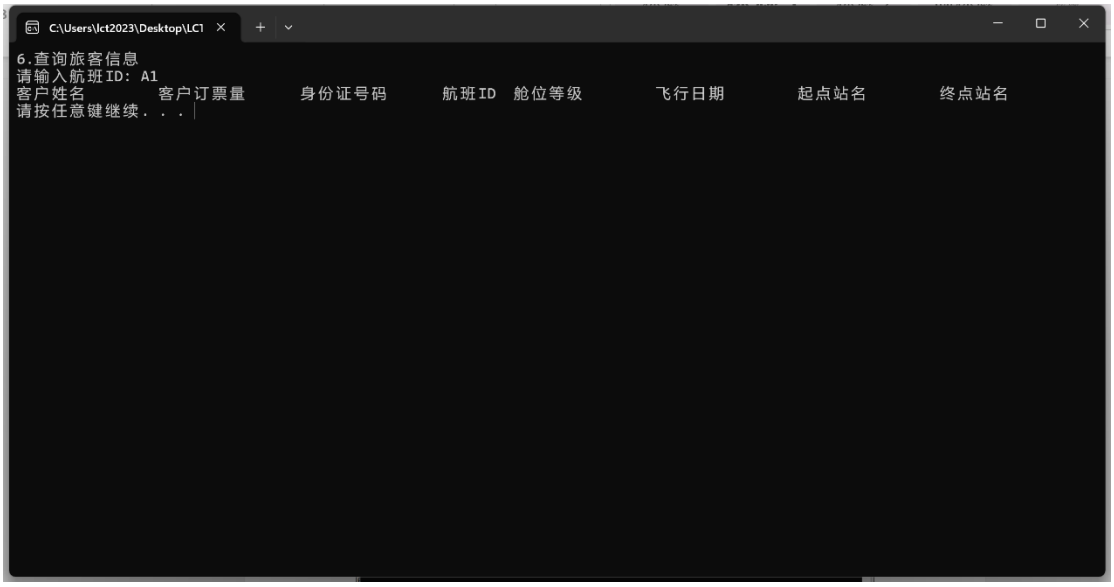


7. 从菜单中输入 6：会进入 Visit_Passenger_Information(pFlight)，查询旅客信息模块。

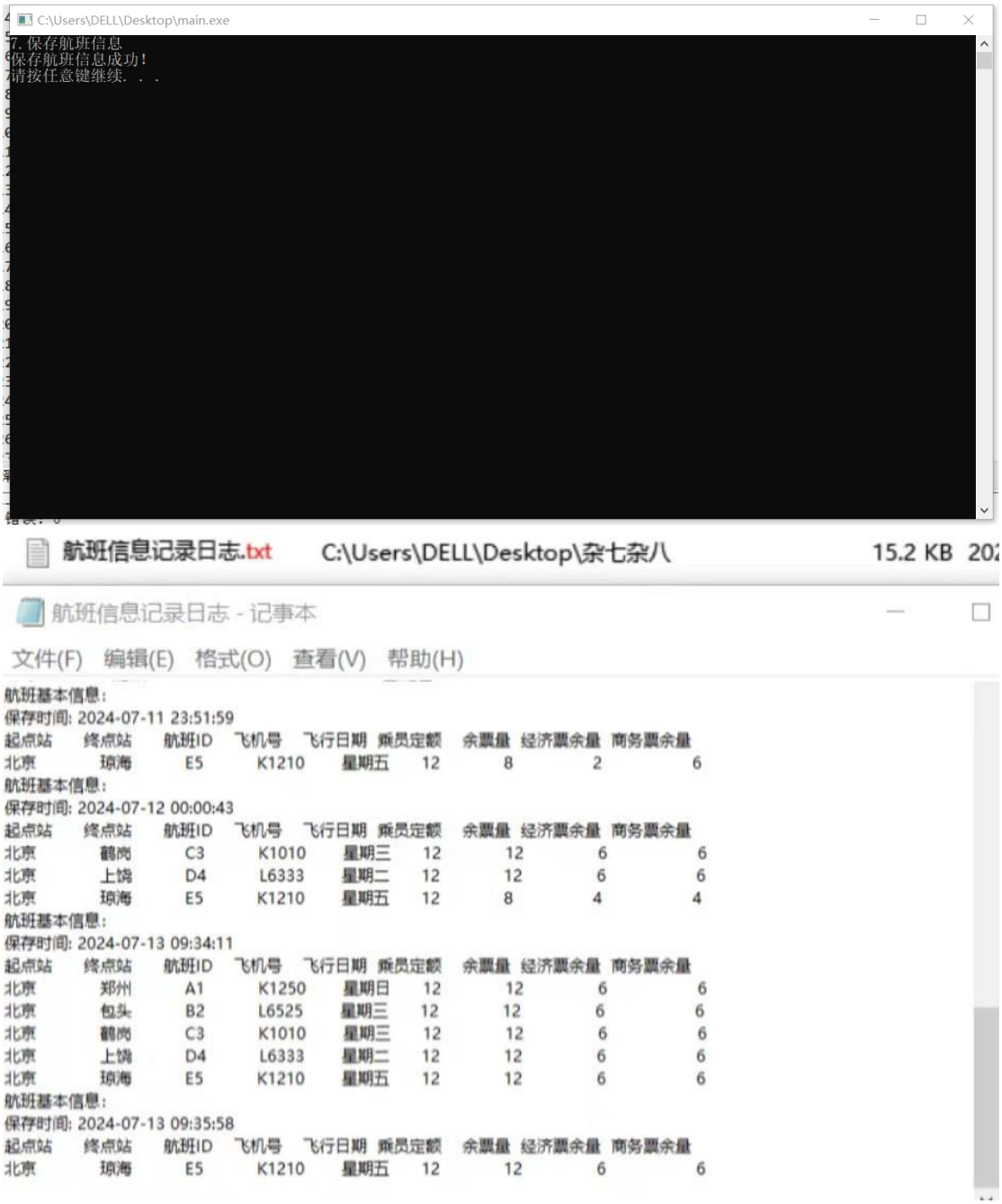
在输入航班 ID 后，会显示该航班的已购票信息，包括客户姓名、客户订票量、身份证号码等信息。



若旅客信息为空，或者所有旅客已经退票，则显示为空。

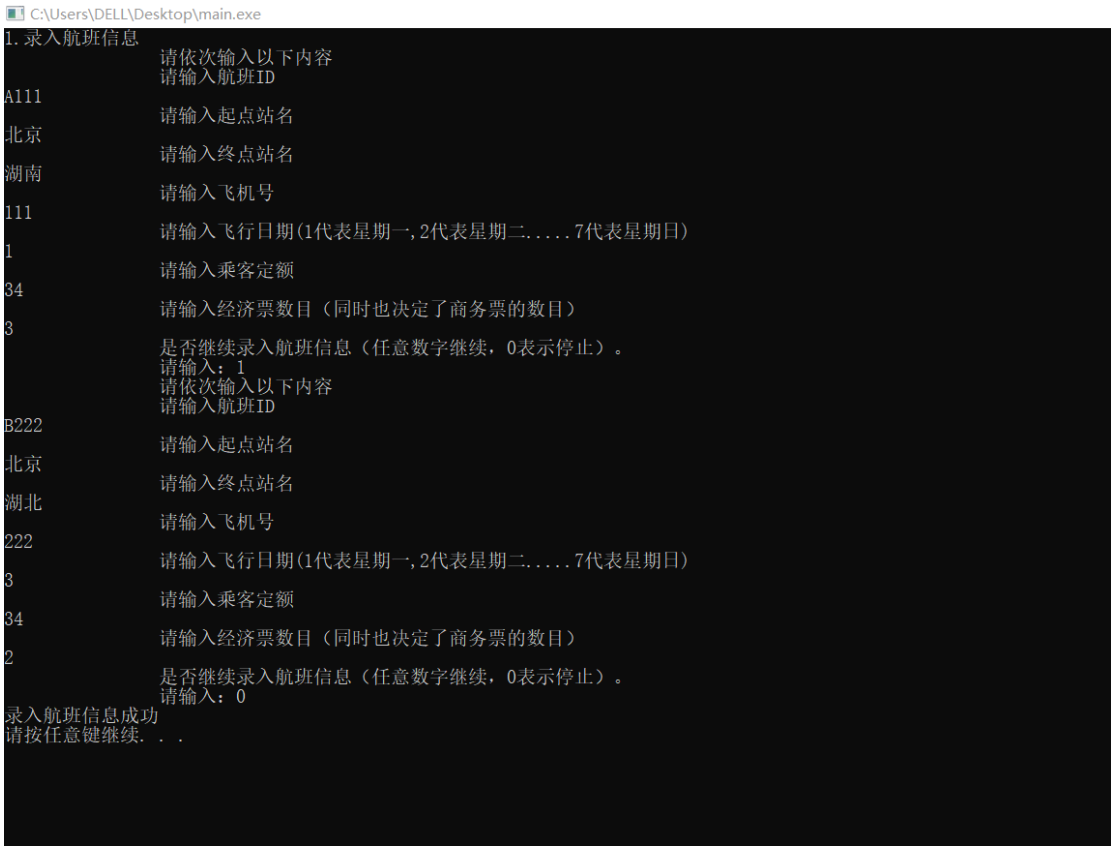


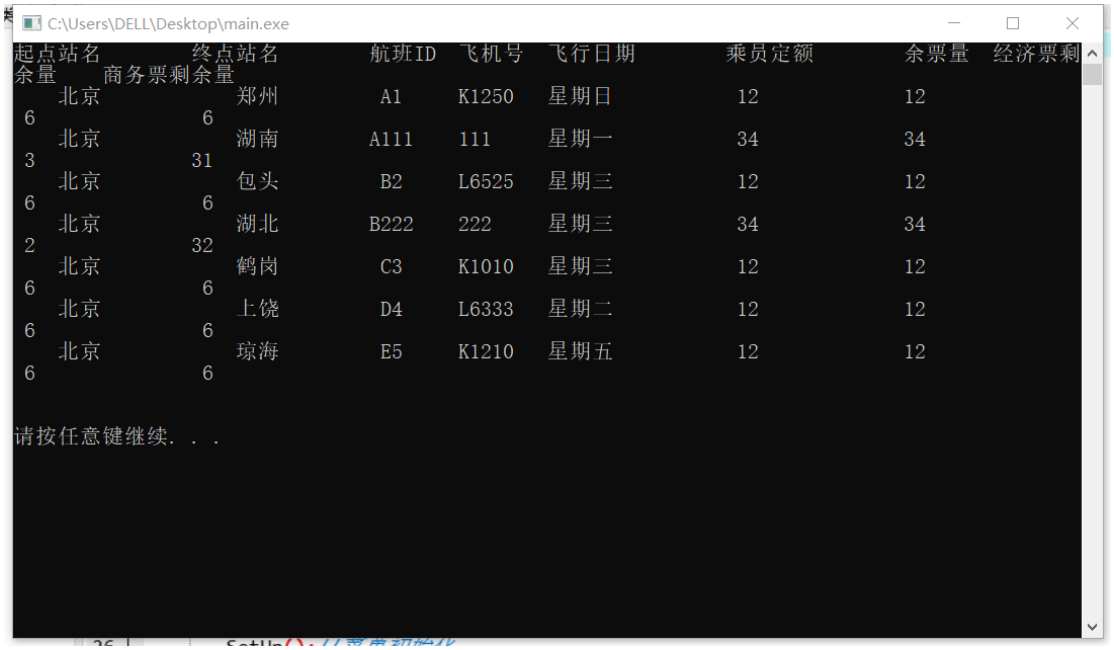
8. 从菜单中输入 7：会进入 Log_Flight_Information(pFlight)，保存航班信息模块。若成功保存，则会显示“保存航班信息成功！”，并将数据保存在名为“航班信息记录日志 txt”的文本文档中，若原本没有此文档，则会新建一个文档。

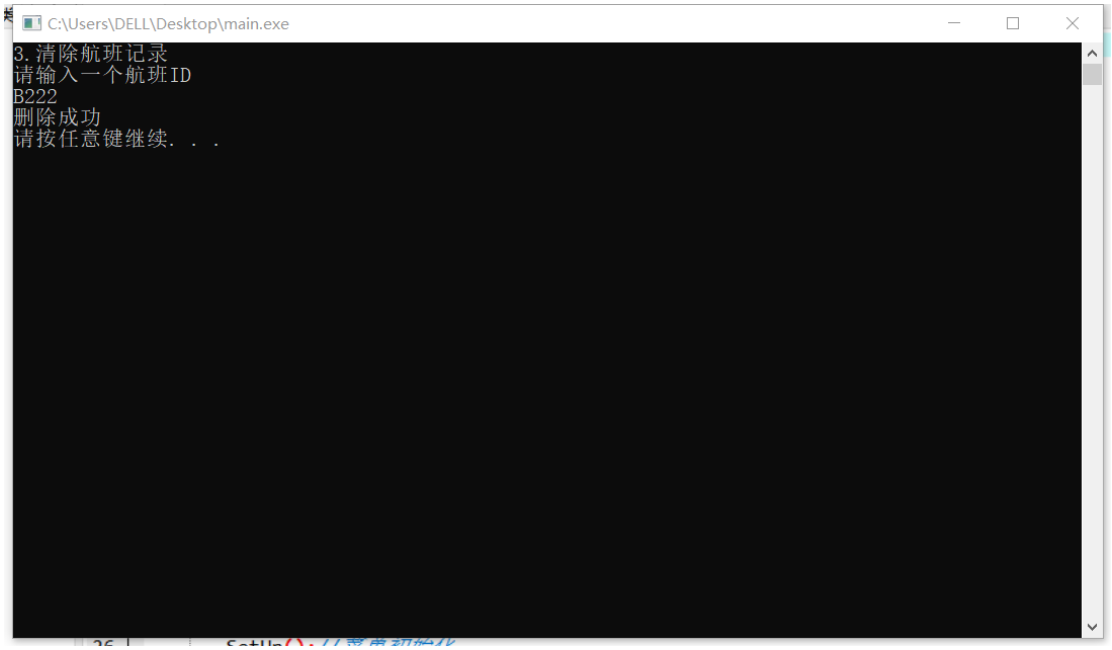


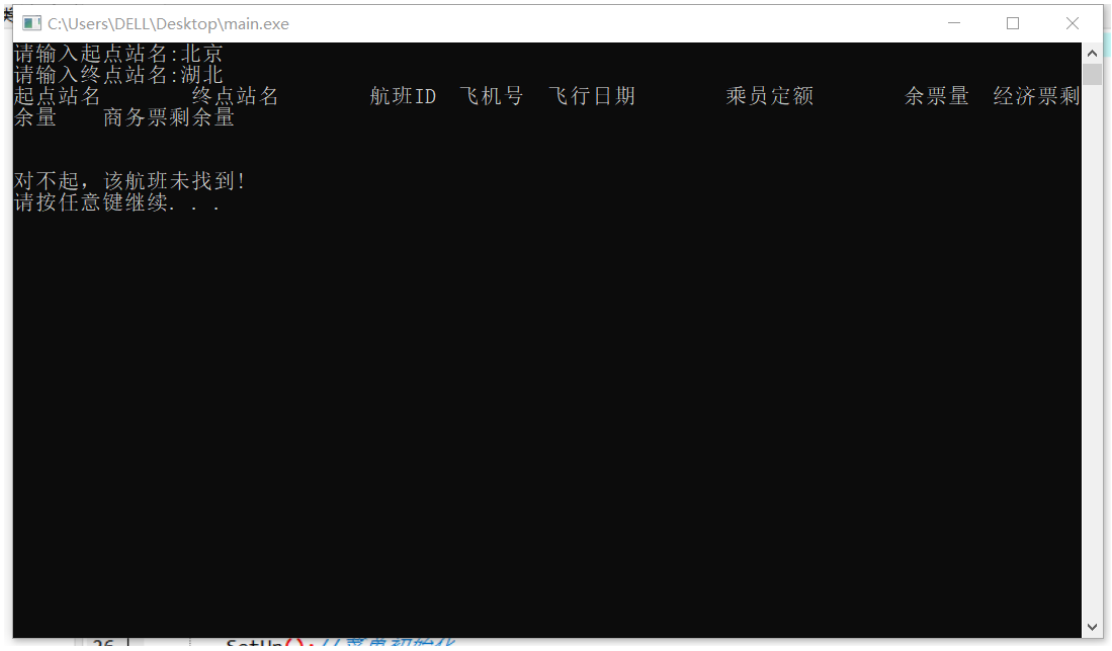
9. 从菜单中输入 0：结束程序，并在按任意键后退出程序。

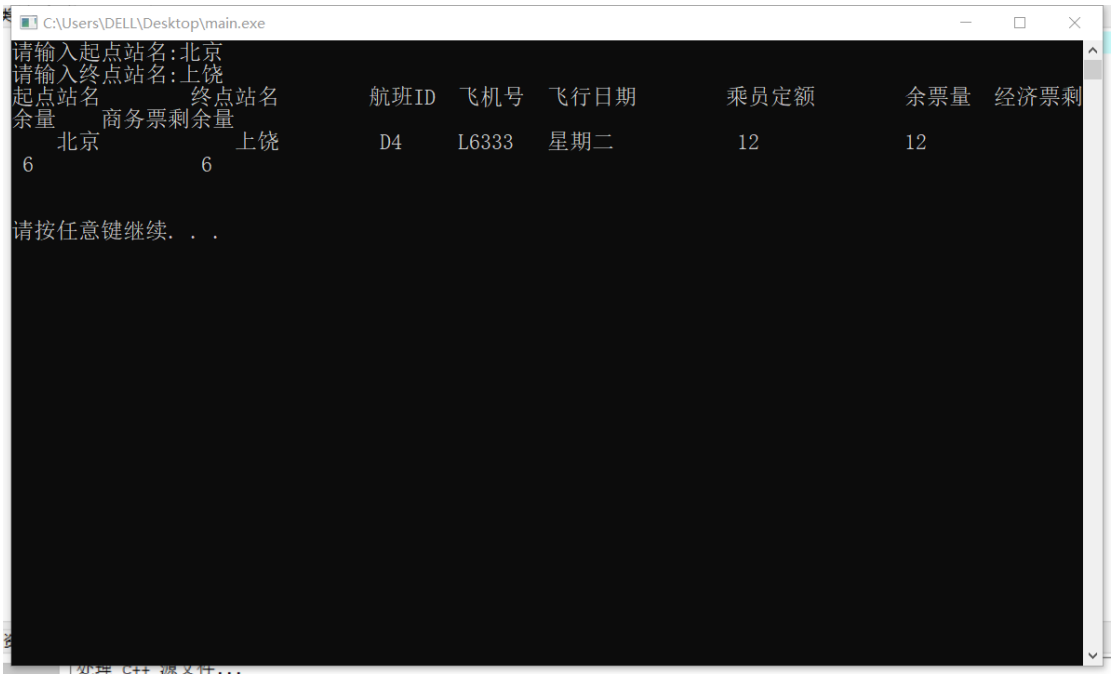
（二）、 测试数据第二组：（和第一组的数据相对应）


C:\Users\DELL\Desktop\main.exe
1.录入航班信息
请依次输入以下内容
请输入航班ID
A111
请输入起点站名
北京
请输入终点站名
湖南
请输入飞机号
111
请输入飞行日期(1代表星期一,2代表星期二.....7代表星期日)
1
请输入乘客定额
34
请输入经济票数目（同时也决定了商务票的数目）
3
是否继续录入航班信息（任意数字继续，0表示停止）。
请输入：1
请依次输入以下内容
请输入航班ID
B222
请输入起点站名
北京
请输入终点站名
湖北
请输入飞机号
222
请输入飞行日期(1代表星期一,2代表星期二.....7代表星期日)
3
请输入乘客定额
34
请输入经济票数目（同时也决定了商务票的数目）
2
是否继续录入航班信息（任意数字继续，0表示停止）。
请输入：0
录入航班信息成功
请按任意键继续. . .

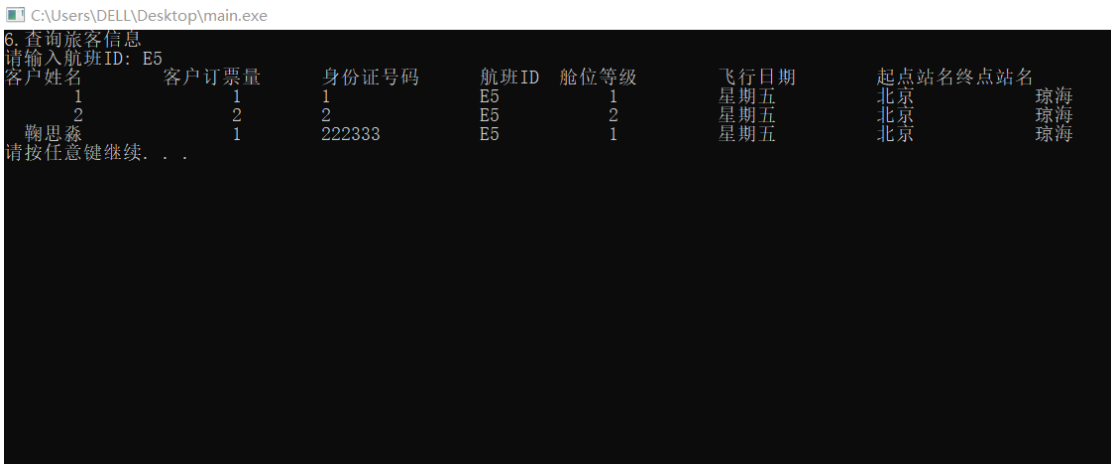

C:\Users\DELL\Desktop\main.exe
起点站名 终点站名 航班ID 飞机号 飞行日期 乘员定额 余票量 经济票剩余量 商务票剩余量
北京 郑州 A1 K1250 星期日 12 12 6 6
北京 湖南 A111 111 星期一 34 34 3 31
北京 包头 B2 L6525 星期三 12 12 6 6
北京 湖北 B222 222 星期三 34 34 2 32
北京 鹤岗 C3 K1010 星期三 12 12 6 6
北京 上饶 D4 L6333 星期二 12 12 6 6
北京 琼海 E5 K1210 星期五 12 12 6 6
请按任意键继续. . .

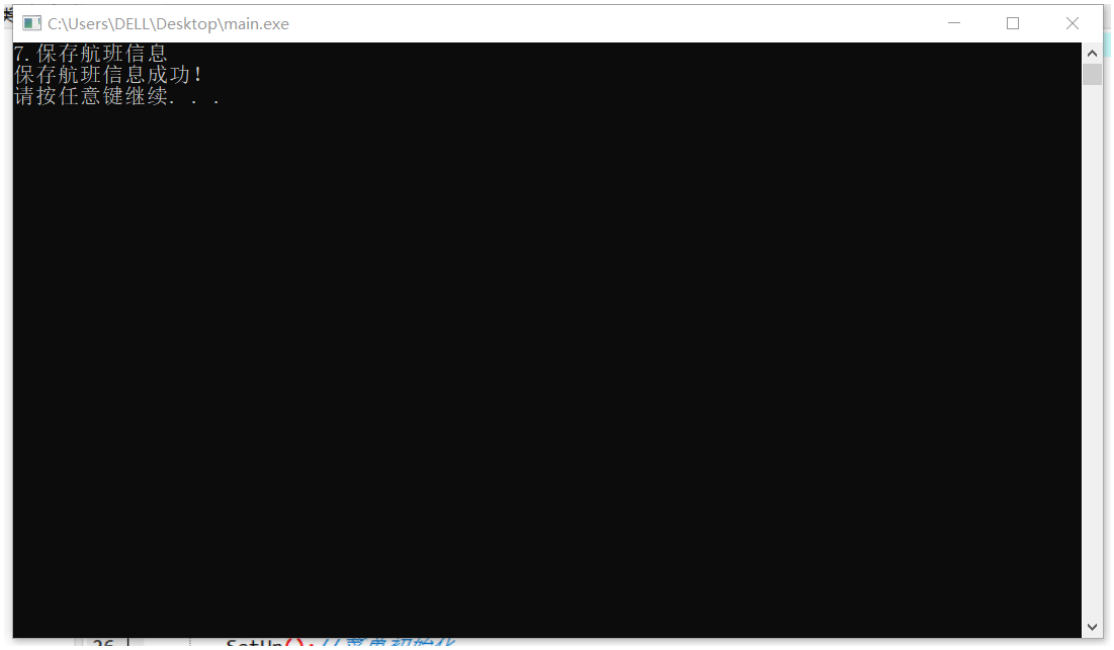

C:\Users\DELL\Desktop\main.exe
3.清除航班记录
请输入一个航班ID
B222
删除成功
请按任意键继续. . .

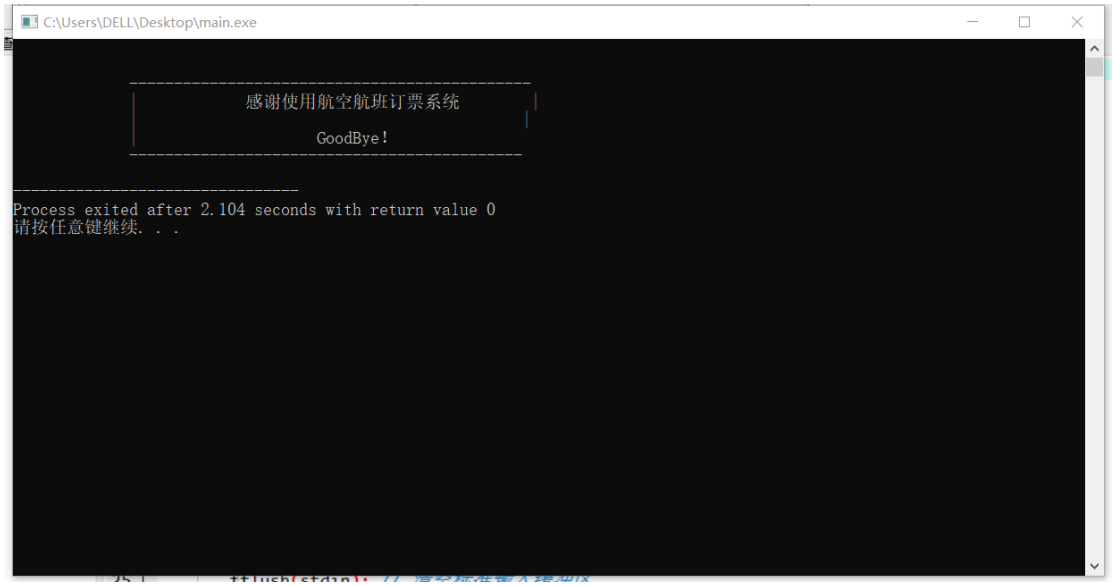

C:\Users\DELL\Desktop\main.exe
请输入起点站名:北京
请输入终点站名:湖北
起点站名 终点站名 航班ID 飞机号 飞行日期 乘员定额 余票量 经济票剩余量 商务票剩余量
对不起，该航班未找到!
请按任意键继续. . .


C:\Users\DELL\Desktop\main.exe
请输入起点站名:北京
请输入终点站名:上饶
起点站名 终点站名 航班ID 飞机号 飞行日期 乘员定额 余票量 经济票剩余量 商务票剩余量
北京 上饶 D4 L6333 星期二 12 12 6 6
请按任意键继续. . .


C:\Users\DELL\Desktop\main.exe
请输入航班ID: B222
没有这个航班, 请重新输入
请输入航班ID: A111
没有这个航班, 请重新输入
请输入航班ID: E5
请正确输入您的票的舱位等级（1代表经济舱，其他代表商务舱）:1
请正确输入你订票所需要的数量:1
请输入您的姓名:鞠思淼
请输入您的身份证号码:222333
鞠思淼 的座位号是: 1
祝您旅途愉快！欢迎再次光临
是否继续购票，Y/N


C:\Users\DELL\Desktop\main.exe
6.查询旅客信息
请输入航班ID: E5
客户姓名 客户订票量 身份证号码 航班ID 舱位等级 飞行日期 起点站名终点站名
1 1 1 E5 1 星期五 北京 琼海
2 2 2 E5 2 星期五 北京 琼海
鞠思淼 1 222333 E5 1 星期五 北京 琼海
请按任意键继续. . .


C:\Users\DELL\Desktop\main.exe
7.保存航班信息
保存航班信息成功!
请按任意键继续. . .


C:\Users\DELL\Desktop\main.exe
感谢使用航空航班订票系统
GoodBye!
Process exited after 2.104 seconds with return value 0
请按任意键继续. . .

六、 数据分析：

通过程序，系统可以输入录入航班信息、加载航班数据。清除航班记录，用户乘客可以查询航班、购票退票。查看个人航班信息，以满足用户对出行航班的需求。
在航空客运订票系统程序菜单中清晰地展示了每一项特定功能，包括录入航班信息等。用户可以通过 5.6.7 功能对航班票进行购买、退订以及查看确认。
各种功能项运用指针、循环、冒泡排序、链表等 C 语言知识，详细流程见流程图。

七、 收获与体会：

通过这次小组合作《ACM》和《数据结构》课程设计，我们收获了许多知识、经验与能力，有了深刻的体会感悟。
①巩固和加深对老师课堂讲授内容的理解，提高综合运用所学知识的能力：
课堂上老师讲授了许多数据结构相关知识，要想真正掌握这些知识，上机练习是必不可少的，学习数据结构不能止步于理解函数结构基本内容，更重要的是提升上机操作能力，检验自己是否消化所学数据结构内容，检验程序是否能够顺利运行，这次大作业完成过程中我们的上机实践能力得到了很好的锻炼和提升；
②提高自主解决问题能力：
将写好的程序进行上机编译时，会有许多未曾料想的错误或警告，通过自主尝试解决问题，能够快速有效提升程序编写修改能力。发现问题、解决问题、顺利运行，每一个步骤环节都可以逐步提高我们自主解决问题能力和数据结构程序开发能力；
③提高深入思考能力：
从接触课题问题开始，到逐步构思程序大体结构、增添航班信息各种功能项、完善细节、初步运行、最终优化程序，我们根据所学知识、依据经验、参考教材、查找资料，解决每个航班票务环节出现的问题，一步步完成《ACM》和《数据结构》大作业，独立思考、深入研究、分析问题的能力得到了很大的提高；
④提高运用数据结构知识解决实际问题的能力：
此次航空客运订票系统课程设计是把我们所学的数据结构理论知识进行总结并应用于实践的成果，我们需要从实际情况的具体问题入手，设想实际情况在航班预定、退票等输入信息过程中可能出现的一系列问题，包括航班星期输入错误、输入错误未订票乘客信息、查找出发地到目的地的所有航班、航班退订等，尽可能全面高效地解决实际问题，完成此次程序巩固了我们学习的数据结构知识，提升我们用知识理论分析实际问题的能力，进而提高运用数据结构知识解决实际问题的能力；
⑤提高数据结构综合素质：
这次数据结构程序设计航空客运订票系统，让我们对数据结构有了进一步的认识。我们通过这次大作业的完成，更加熟悉的运用数据结构相关程序的操作，掌握更多的数据结构知识，为我们以后学习编程丰富了许多实际经验；
⑥提高小组合作能力：
从确定《ACM》和《数据结构》作业课题为航空客运订票系统开始，我们小组成员进行了有效沟通讨论与任务分配，遇到问题及时沟通，讨论解决方案，线上线下优化细节，敲定最终方案，完成本次作业。此次作业我们倾注了大量的努力汗水与真诚用心，过程中收获理论知识，提升各项能力，最终交出一份满意的《ACM》和《数据结构》大作业，感恩！

八、 任务分工：

①李春涛：程序选择 5、6 部分代码　整合调整 统筹管理　代码修改
②宋奇育：程序选择 1、2、7 部分代码　系统页面制作　代码修改
③鞠思淼：程序选择 0、3、4 部分代码　文稿制作

九、　附源程序清单：

//C++语言代码【main+其他】

一．function_declaration.h

```
//
/**
*********************************************************************************
***
   * @file               : function_declaration.h
   * @author             : lct2023
   * @brief              : None
   * @attention          : None
   * @date               : 2024/6/2
*********************************************************************************
***
   */
//
#include "function.cpp"

/**
   *   @function     SetUp
   *   @brief        菜单初始化
*/
void SetUp();


/**
 *  查询模块 打印 info 航班的基本信息
 * @param info
 */
void Display(Flight *info);


 /**
    *   @function     InitFlight
```

```
    *    @brief          初始化移 pFlight 为头结点的空航班链表,录入航班信息和增加航班后将航班结点插入该链表
     */
void InitFlight();


/**
 * @brief 初始化已订票乘客指针链表
 * @param cusLinkList 航班中乘员链表的头指针
 * @return 函数执行状态
 */
Status InitCusLinkList(CusLinkList &cusLinkList);


/**
 * @brief 初始化带头结点的链队列
 * @param q 链队的头结点
 */
Status InitQueue(LinkQueue &q);


/**
 * @brief 将 flight1 的 6 个航班用头插入法插入到 pFlight 的链表中
 * @param flight1 里面存有六个基本航班
 * @return 返回操作是否成功
 */
Status Create(PFlight flight1);


/**
 * @brief 增加航班时输入日期的辅助函数（1 代表星期一，7 代表星期日）
 * @param day1 传进来的 1-7 中的一个
 * @param day 数组类变量，可以返回回去给航班的日期变量
 * @return 返回操作状态，输入是否合法
 */
Status IputDay(int day1, char day[]);


/**
 * @brief 插入航班时遍历航班，防止航班 ID 重复（航班 ID 相当于主键）
 * @param flight 航班的头结点
 * @param flightCodeID   需要进行遍历查找航班号
 * @return   返回是否重复
 */
```

```
Status TraverseFlight(Flight *flight, char flightCodeID[]);


/**
   * @brief 将新的航班结点插入到航班链表中，
   * @return 返回操作是否成功
   */
Status InsertFlight();


/**
 * @brief 查询模块，打印全部航班信息
 * @param pflight 传入的是航班链表的头指针
 */
void PrintFilghtlist(Flight *pflight);


/**
   * @brief 删除节点
   * @return 返回操作是否成功
   */
Status DeleteFlight();


/**
 *@brief 根据客户提出的起点，终点站名输出航线信息
 */
void SearchFlight();


/**
 * @brief 入队，增加排队等候的客户名单域
 * @param q 带头结点的链队列
 * @param name   等候客户的名字
 * @param amount       等候客户所需机票的数量
 * @param identification       等候客户的身份证号
 * @return   返回成功入队后的等候客户链的头结点
 */
LinkQueue Appendqueue(LinkQueue &q, char name[], int amount, char identification[]);


/**
 * @brief 根据自己输入的航班 ID 查询并以指针形式返回
 * @return   航班指针
```

```
 */
Flight *find();


/**
 * 订票成功之后，将乘客信息插入到对应航班的订成员名单域中（链表）
 * @param head 乘客名单域头指针
 * @param amount 该乘客订票的数量
 * @param name 乘客的姓名
 * @param rank 订票的等级
 * @return  乘客链表头指针
 */
CusLinkList insertlink(CusLinkList &head, int amount, char name[], char identification[], int rank);


/**
 * @brief 输出 p 节点的航班信息
 * @param p 航班节点
 */
void FlightInfo(Flight *p);


/**
 * @brief 查找是否有同一路线的航班
 * @param destination    目的地
 * @param pflight 原航班，用来判断和新搜到的航班是否一样
 * @return 找不到就返回 FALSE，否则返回 TRUE
 */
Status RecommendFlight(char startPoint[], char destination[], Flight *pflight);


/**
 * @brief 订票模块
 */
void BookTickets();


/**
 * @param Q    Q 为候补订票客户的队列
 * @param NameAndNumAndIDAndID 候补客户的姓名和订票数目，出队时，将姓名和关键字和身份证返回
 * @return
 */
```

```
Status QueueDelete(LinkQueue &q, NameAndNumAndID &NameAndNumAndID);


/**
 * @brief 退票功能模块
 */
void ReturnTicket();


/**
   *   @function     Log_Flight_Information
   *   @brief        将航班信息存进文件"航班信息记录日志.txt"
   *   @param          pflight 航班头指针
   *   @return       函数执行状态
*/
Status Log_Flight_Information(Flight *pflight);

/**
   *   @function     Visit_Passenger_Information
   *   @brief        访问旅客信息
   *   @param          pflight 航班头指针
   *   @return       函数执行状态
*/
Status Visit_Passenger_Information(Flight *pflight);


/**
 * @brief 退出程序模块界面
 */
void GoodbyeFace();


#ifndef DATA_STRUCTURE_PRACTICE_FUNCTION_DECLARATION_H
#define DATA_STRUCTURE_PRACTICE_FUNCTION_DECLARATION_H

#endif //DATA_STRUCTURE_PRACTICE_FUNCTION_DECLARATION_H
```

二．function.cpp

```
//
/**
   * @file                : function.cpp
   * @author              : lct2023
   * @brief               : None
```

```
   * @attention          : None
   * @date                : 2024/6/2
   */
//
#include <iostream>
#include <cstring>
#include <cstdio>
#include <malloc.h>
#include <fstream>
#include <ctime>
#include <iomanip>

#define OK 1
#define ERROR 0
#define OVERFLOW -1
#define FALSE -1
#define TRUE 1

using namespace std;

typedef int Status;

//航班日期枚举类，星期一到星期天
enum Week{
	Mon = 1, Tues = 2, Wed = 3, Thurs = 4, Fri = 5, Sat = 6, Sun = 7
};

//乘客节点
typedef struct CustomerNode
{
	char name[10];//客户姓名
	int clientTickets;//客户订票量
	char identification[20];//客户身份证号码
	int rank;//舱位等级
	CustomerNode *next;
} CustomerNode, *CusLinkList;

//候补队列中的节点
typedef struct WaitPassenger
{
	char name[10];//姓名
	char identification[20]; //身份证
	int preTickets;//预定的票量
	struct WaitPassenger *next;
```

```
} WaitQNode, *PWait;

//候补队列
typedef struct Queue
{
	PWait front;//等候替补客户名单域的头指针
	PWait rear;//等候替补客户名单域的尾指针
} LinkQueue;

//封装乘客的姓名和订票量和身份证
//用于候补客户出队时把关键字返回
typedef struct NameAndNumAndID
{
	char name[10];//姓名
	char identification[20]; //身份证号码
	int num;//订票量
} NameAndNumAndID;

//航班节点
typedef struct Flight
{
	char startPoint[20];//起点站名
	char destination[20];//终点站名
	char flightCodeID[20];//航班 ID（相当于主键）
	char planeNum[20];//飞机号
	char day[20];//飞行日期（星期几）
	int totalTickets;//乘员定额(总票数)
	int left;//总余票量
	int leftEconomicTicket; //经济票剩余量
	int leftBusinessTicket; //商务票剩余量
	Flight *next;
	CusLinkList cusLinkList;//乘员名单域，指向乘员名单链表的头指针
	LinkQueue waitQueue1;//经济舱候补，等候替补的客户名单域，指向一个队列
	LinkQueue waitQueue2;//商务舱候补，等候替补的客户名单域，指向一个队列

} Flight, FlightNode, *PFlight;

//定义全局指针变量 pFlight，航班链表的头指针
Flight *pFlight;

//五个基本航班
Flight flight1[5] = {
		{"北京", "郑州", "A1", "K1250", "星期日", 12, 12, 6},
		{"北京", "包头", "B2", "L6525", "星期三", 12, 12, 6},
```

```
        {"北京", "鹤岗", "C3", "K1010", "星期三", 12, 12, 6},
        {"北京", "上饶", "D4", "L6333", "星期二", 12, 12, 6},
        {"北京", "琼海", "E5", "K1210", "星期五", 12, 12, 6},
};

// 冒泡排序函数，按照航班 ID 从小到大排序
void bubbleSort_PFlight(PFlight *head) {
    if (*head == NULL || (*head)->next == NULL) {
        return; // 如果链表为空或者只有一个节点，不需要排序
    }

    Flight *end = NULL; // 用于标记已经排好序的部分

    while (end != *head) { // 外层循环控制排序的轮数
        Flight **prev = head; // 前一个节点的指针的指针
        Flight *current = *head; // 当前节点指针
        Flight *nextNode = current->next; // 下一个节点指针

        while (current->next != end) { // 内层循环进行一次完整的冒泡过程
            nextNode = current->next;
            if (strcmp(current->flightCodeID, nextNode->flightCodeID) > 0) { // 比较当前
节点和下一个节点的航班 ID
                // 交换节点的位置
                *prev = nextNode;
                current->next = nextNode->next;
                nextNode->next = current;

                // 交换指针
                Flight *temp = current;
                current = nextNode;
                nextNode = temp;
            }

            // 更新指针，继续下一次比较
            prev = &current->next;
            current = nextNode;
        }

        // 更新已排好序的部分
        end = current;
    }
}
```

```
/**
   *    @function      InitFlight
   *    @brief         初始化移 pFlight 为头结点的空航班链表,录入航班信息和增加航班后将航班结点插入该链表
*/
void InitFlight()
{
      pFlight = (Flight *) malloc(sizeof(Flight));//申请头结点的空间
      if (pFlight == NULL) exit(0);
      pFlight->next = NULL;//将头结点 h 的指针域置为空
}

/**
 * @brief 初始化已订票乘客指针链表
 * @param cusLinkList 航班中乘员链表的头指针
 * @return 函数执行状态
 */
Status InitCusLinkList(CusLinkList &cusLinkList)
{
      CusLinkList q = cusLinkList;
      cusLinkList = (CustomerNode *) malloc(sizeof(CustomerNode));
      cusLinkList->next = NULL;
}

/**
 * @brief 初始化带头结点的链队列
 * @param q 链队的头结点
 */
Status InitQueue(LinkQueue &q)
{
      WaitQNode *p;
      p = (WaitQNode *) malloc(sizeof(WaitQNode));
      if (p == NULL) {
            printf("内存不足\n");
            return ERROR;
      }

      p->next = NULL;
      q.front = q.rear = p;
      return OK;

}

/**
```

```
 * @brief 将 flight1 的 6 个航班用头插入法插入到 pFlight 的链表中
 * @param flight1 里面存有六个基本航班
 * @return 返回操作是否成功
 */
Status Create(PFlight flight1)
{
	Flight *p = pFlight, *q;
	for (int i = 0; i < 5; i++) {
		q = (Flight *) malloc(sizeof(Flight));
		if (q == NULL)
			return ERROR;
		strcpy(q->startPoint, flight1[i].startPoint);
		strcpy(q->destination, flight1[i].destination);
		strcpy(q->flightCodeID, flight1[i].flightCodeID);
		strcpy(q->planeNum, flight1[i].planeNum);
		strcpy(q->day, flight1[i].day);
		q->totalTickets = flight1[i].totalTickets;
		q->left = flight1[i].totalTickets;
		q->leftEconomicTicket = flight1[i].leftEconomicTicket;
		q->leftBusinessTicket = flight1[i].totalTickets - flight1[i].leftEconomicTicket;
		//初始化乘客链表
		InitCusLinkList(q->cusLinkList);

		InitQueue(q->waitQueue1);
		InitQueue(q->waitQueue2);

		q->next = p->next;
		p->next = q;

	}
	return OK;

}

/**
 * @brief 增加航班时输入日期的辅助函数（1 代表星期一，7 代表星期日）
 * @param day1 传进来的 1-7 中的一个
 * @param day 数组类变量，可以返回回去给航班的日期变量
 * @return 返回操作状态，输入是否合法
 */
Status IputDay(int day1, char *day)
{
	switch (day1) {
		case Mon:
```

```
            strcpy(day, "星期一");
            break;
        case Tues:
            strcpy(day, "星期二");
            break;
        case Wed:
            strcpy(day, "星期三");
            break;
        case Thurs:
            strcpy(day, "星期四");
            break;
        case Fri:
            strcpy(day, "星期五");
            break;
        case Sat:
            strcpy(day, "星期六");
            break;
        case Sun:
            strcpy(day, "星期日");
            break;
        default:
            return ERROR;
    }
    return OK;
}


/**
 * @brief 插入航班时遍历航班，防止航班 ID 重复（航班 ID 相当于主键）
 * @param flight 航班的头结点
 * @param flightCodeID  需要进行遍历查找航班号
 * @return  返回是否重复
 */
Status TraverseFlight(Flight *flight, char *flightCodeID)
{
    Flight *p = flight;
    while (p != NULL)
    {
        //当有航班 ID 重复时候，返回 ERROR，
        if (!strcmp(flightCodeID, p->flightCodeID))
        {
            return ERROR;
        }
```

```
            p = p->next;
        }
        //输入的航班 ID 不重复
        return OK;

}


/**
   * @brief 将新的航班结点插入到航班链表中，
   * @return 返回操作是否成功
   */
Status InsertFlight()
{
        FlightNode *q;//定义 q 为新增加的航班结点的指针的形参
        Flight *p = pFlight;
        int mark = 1;

        while (mark != 0) {
            q = (FlightNode *) malloc(sizeof(FlightNode));
            if (q == NULL)
                return ERROR;

            printf("\t\t 请依次输入以下内容\n");
            printf("\t\t 请输入航班 ID\n");
            scanf("%s", q->flightCodeID);
            Status status = TraverseFlight(pFlight, q->flightCodeID);
            if (status == ERROR) {
                printf("该航班 ID 已经存在，请重新输入航班 ID\n");
                continue;
            }
            printf("\t\t 请输入起点站名\n");
            scanf("%s", q->startPoint);
            printf("\t\t 请输入终点站名\n");
            scanf("%s", q->destination);
            printf("\t\t 请输入飞机号\n");
            scanf("%s", q->planeNum);
            printf("\t\t 请输入飞行日期(1 代表星期一,2 代表星期二.....7 代表星期日)\n");
            int day1;
            scanf("%d", &day1);
            while (ERROR == IputDay(day1, q->day)) {
                printf("请输入合法数据(1-7)\n");
                scanf("%d", &day1);
            };
```

```
            printf("\t\t 请输入乘客定额\n");
            scanf("%d", &q->totalTickets);
            q->left = q->totalTickets;

            printf("\t\t 请输入经济票数目（同时也决定了商务票的数目）\n");
            scanf("%d", &q->leftEconomicTicket);
            //商务票数目 = 总票数目 - 经济票数目
            q->leftBusinessTicket = q->totalTickets - q->leftEconomicTicket;
            InitCusLinkList(q->cusLinkList);
            //初始化
            InitQueue(q->waitQueue1);
            InitQueue(q->waitQueue2);

            q->next = p->next;
            p->next = q;

            printf("\t\t 是否继续录入航班信息（任意数字继续，0 表示停止）。\n");
            printf("\t\t 请输入：");
            scanf("%d", &mark);
        }
        return OK;
}

/**
   *   @function     SetUp
   *   @brief        菜单初始化
*/
void SetUp()
{
        cout << endl << endl;
        std::cout << "                ##           /           ## ###### /          ####      ####
/             /          ######" << std::endl;
        std::cout << "                 ##        ///         ##   #         /          #          #   #
/ /           / /        #         " << std::endl;
        std::cout << "                  ##     /// //    ##    #####   /         #         #   #     /
/        /     /       ##### " << std::endl;
        std::cout << "                   ## //    //   ##      #         /         #         #   #    /
/     /        /     #         " << std::endl;
        std::cout << "                    ///         ///       #         /         #         #   #   /
///           /    #         " << std::endl;
        std::cout << "                     /           /        ###### ######   ####      #### /
/             / ######" << std::endl;
```

```
    cout << endl << endl << endl;
    cout << "                                    ☆☆☆☆☆☆☆☆☆☆☆☆☆ Airplane Servervice
System ☆☆☆☆☆☆☆☆☆☆☆☆☆☆" << endl;
    cout << endl;
    cout                                    <<                                    "
┌──────────────────────────────────────────────────────────────────────────────
┐ \n";
    cout << "                                    │          欢迎使用航空客运订票系统
│ \n";
    cout                                    <<                                    "
│──────────────────────────────────────────────────────────────────────────────
│ \n";
    cout << "                                    │ 1.录入航班信息       5.购票&退票
│ \n";
    cout                                    <<                                    "
│──────────────────────────────────────────────────────────────────────────────
│ \n";
    cout << "                                    │ 2.加载航班数据       6.查询旅客信息
│ \n";
    cout                                    <<                                    "
│──────────────────────────────────────────────────────────────────────────────
│ \n";
    cout << "                                    │ 3.清除航班记录       7.保存航班信息
│ \n";
    cout                                    <<                                    "
│──────────────────────────────────────────────────────────────────────────────
│ \n";
    cout << "                                    │ 4.查询航班信息       0.退出系统
│ \n";
    cout                                    <<                                    "
└──────────────────────────────────────────────────────────────────────────────
┘ \n";
}

/**
 * @brief 查询模块 打印 info 航班的基本信息
 * @param info 单个航班信息
 */
void Display_flight_Infomation(Flight *info)
{
    //printf("起点站名\t 终点站名\t 航班 ID\t 飞机号\t 飞行日期\t 乘员定额\t 余票量\t
经济票剩余量\t 商务票剩余量\n");
    printf("%8s\t%8s\t%3s\t%s\t%4s\t\t%3d\t%10d\t%10d\t%10d\n",      info->startPoint,
info->destination, info->flightCodeID,
```

```
            info->planeNum, info->day,
            info->totalTickets, info->left, info->leftEconomicTicket, info->leftBusinessTicket);
}


Status Log_Flight_Information(Flight *pflight) {
    fstream ifs;
    // 追加写入，在原来基础上加了 ios::app
    ifs.open("航班信息记录日志.txt", ios::out | ios::app);

    if (!ifs.is_open()) {
        cout << "文件打开失败" << endl;
        return ERROR;
    }

    // 获取当前时间
    time_t now = time(0);
    tm *ltm = localtime(&now);

    // 格式化时间信息
    char timeStr[50];
    snprintf(timeStr, sizeof(timeStr), "%04d-%02d-%02d %02d:%02d:%02d",
             1900 + ltm->tm_year,
             1 + ltm->tm_mon,
             ltm->tm_mday,
             ltm->tm_hour,
             ltm->tm_min,
             ltm->tm_sec);

    // 输入你想写入的内容
    ifs << "航班基本信息：" << endl;
    ifs << "保存时间: " << timeStr << endl;
    ifs << left << setw(12) << "起点站" << setw(12) << "终点站" << setw(10) << "航班 ID"
        << setw(10) << "飞机号" << setw(10) << "飞行日期" << setw(12) << "乘员定额"
        << setw(8) << "余票量" << setw(12) << "经济票余量" << setw(14) << "商务票余量
" << endl;

    Flight *info = pflight->next;

    bubbleSort_PFlight(&info);
    while (info) {
        ifs << left << setw(16) << info->startPoint
            << setw(14) << info->destination
            << setw(12) << info->flightCodeID
```

```
                << setw(12) << info->planeNum
                << setw(12) << info->day
                << setw(14) << info->totalTickets
                << setw(16) << info->left
                << setw(18) << info->leftEconomicTicket
                << setw(16) << info->leftBusinessTicket << endl;
            info = info->next;
        }

        ifs.close();
        return OK;
}


/**
 * @brief 查询模块，打印全部航班信息
 * @param pflight 传入的是航班链表的头指针
 */
void PrintFilghtlist(Flight *pflight)
{
        Flight *p;
        //带头结点的头指针，所以从下一个指针还是遍历

        p = pflight->next;
        system("cls");

        // 对链表进行排序
        bubbleSort_PFlight(&p);

        printf("起点站名\t 终点站名\t 航班 ID\t 飞机号\t 飞行日期\t 乘员定额\t 余票量\t 经
济票剩余量\t 商务票剩余量\n");
        while (p != NULL) {
            //调用 Display_flight_Infomation 函数打印出每个航班节点的信息
            Display_flight_Infomation(p);
            p = p->next;
        }
        printf("\n\n");
}

/**
   * @brief 删除节点
   * @return 返回操作是否成功
   */
Status DeleteFlight() {
```

```
    char flightCodeID[20];
    printf("请输入一个航班 ID\n");
    scanf("%s", flightCodeID);
    PFlight pre = pFlight;
    PFlight p = pre->next;

    while (p != NULL) {
        if (!strcmp(flightCodeID, p->flightCodeID)) {
            pre->next = p->next;
            free(p);
            return OK;
        }
        pre = p;
        p = p->next;
    }
    return ERROR;
}

/**
 *@brief 根据客户提出的起点，终点站名输出航线信息
 */
void SearchFlight() {
    char startPonit[10];
    char destination[10];
    int flag = 0;
    system("cls");
    printf("请输入起点站名:");
    scanf("%s", startPonit);
    printf("请输入终点站名:");
    scanf("%s", destination);
    struct Flight *p;
    p = pFlight->next;
    printf("起点站名\t 终点站名\t 航班 ID\t 飞机号\t 飞行日期\t 乘员定额\t 余票量\t 经
济票剩余量\t 商务票剩余量\n");

    while (p != NULL) {
        if ((strcmp(startPonit, p->startPoint) == 0) && (strcmp(destination, p->destination)
== 0) ) {
            flag = 1;
            Display_flight_Infomation(p);
        }
        p = p->next;
    }
    printf("\n\n");
```

```
    if (flag == 0)
        printf("对不起，该航班未找到! \n");

}

/**
 * @brief 入队，增加排队等候的客户名单域
 * @param q 带头结点的链队列
 * @param name 等候客户的名字
 * @param amount 等候客户所需机票的数量
 * @param identification 等候客户的身份证号
 * @return 返回成功入队后的等候客户链的头结点
 */
LinkQueue Appendqueue(LinkQueue &q, char name[], int amount, char identification[])
{
    PWait new1;
    new1 = (PWait) malloc(sizeof(WaitQNode));
    strcpy(new1->name, name);
    strcpy(new1->identification, identification);
    new1->preTickets = amount;
    new1->next = NULL;

    q.rear->next = new1;
    q.rear = new1;

    return q;
}


/**
 * @brief 根据自己输入的航班 ID 查询并以指针形式返回
 * @return 航班指针
 */
Flight *find()
{
    char number[10];
    int i = 0;
    int loop;

    printf("请输入航班 ID: ");
    scanf("%s", number);

    //头结点的下一个节点开始遍历
```

```
    Flight *p = pFlight->next;

    while (p != NULL) {
        if (!strcmp(number, p->flightCodeID))
            return p;

        p = p->next;
    }
    return NULL;
}

/**
 * 订票成功之后，将乘客信息插入到对应航班的订成员名单域中（链表）
 * @param head  乘客名单域头指针
 * @param amount  该乘客订票的数量
 * @param name  乘客的姓名
 * @param rank  订票的等级
 * @return    乘客链表头指针
 */
CusLinkList insertlink(CusLinkList &head, int amount, char name[], char identification[], int rank)
{
    //成员名单域新节点 new1
    CusLinkList new1;
    new1 = (CustomerNode *) malloc(sizeof(CustomerNode));
    if (new1 == NULL)
    {
        printf("\n 内存不足\n");
        return NULL;
    }
    //将传入乘客信息赋值给 new1 节点
    strcpy(new1->name, name);
    strcpy(new1->identification, identification);
    new1->clientTickets = amount;
    new1->rank = rank;

    //头插入法加入成员名单域
    new1->next = head->next;
    head->next = new1;

    return head;
}

/**
```

```
 * @brief 输出 p 节点的航班信息
 * @param p 航班节点
 */
void FlightInfo(Flight *p) {
    printf("起点站名\t 终点站名\t 航班 ID\t 飞机号\t 飞行日期\t 乘员定额\t 余票量\t 经济票剩余\t 商务票剩余\n");
    Display_flight_Infomation(p);
    printf("\n\n");

}

/**
 * @brief 查找是否有同一路线的航班
 * @param destination    目的地
 * @param pflight 原航班，用来判断和新搜到的航班是否一样
 * @return 找不到就返回 FALSE，否则返回 TRUE
 */
Status RecommendFlight(char startPoint[], char destination[], Flight *pflight) {
    //标记变量，是否找到同一路线的航班
    int flag = 0;
    system("cls");
    struct Flight *p;
    p = pFlight->next;
    printf("寻找同一路线的航班\n");
    printf("起点站名\t 终点站名\t 航班 ID\t 飞机号\t 飞行日期\t 乘员定额\t 余票量\t 经济票剩余量\t 商务票剩余量\n");

    while (p != NULL)
    {
        //路线相同，且不是同一个航班，标记 flag = 1，表示找到
        if (strcmp(destination, p->destination) == 0 && strcmp(startPoint, p->startPoint) == 0 && p != pflight) {
            flag = 1;
            Display_flight_Infomation(p);
        }
        p = p->next;

    }
    printf("\n\n");

    //没有相同路线的航班
    if (flag == 0)
        return FALSE;
    return TRUE;
```

```
}


/**
 * @brief 订票模块
 */
void BookTickets()
{
	while(1)
	{
		struct Flight *info;
		int amount = 1, rank;
		int tickets; // 剩余的商务票数目或者经济票数目
		char name[10];
		char identification[20];
		int success = 0;

		system("cls");

		while (1)
		{
			// 调用查找航班函数，返回给 info
			info = find();
			if (info != NULL)
			{
				break;
			}
			printf("没有这个航班，请重新输入\n");
		}

		printf("请正确输入您的票的舱位等级（1 代表经济舱，其他代表商务舱）:");
		scanf("%d", &rank);
		if (rank == 1)
		{
			tickets = info->leftEconomicTicket;
		} else
		{
			tickets = info->leftBusinessTicket;
		}

		while (1) {
			printf("请正确输入你订票所需要的数量:");
			scanf("%d", &amount);
			if (amount > 0) {
```

```
                break;
            }
            printf("输入的数量不合法，请重新输入\n");
        }

        if (amount <= tickets)
        {
            printf("请输入您的姓名:");
            scanf("%s", name);
            printf("请输入您的身份证号码:");
            scanf("%s", identification);
            CusLinkList head = info->cusLinkList;
            // 订票成功，插入成员名单链表
            insertlink(head, amount, name, identification, rank);
            for (int i = 0; i < amount; i++) {
                printf("%s 的座位号是: %d\n", name, info->totalTickets - info->left + i +
1);
            }
            info->left -= amount;


            if (rank == 1)
            {
                info->leftEconomicTicket -= amount;
            } else
            {
                info->leftBusinessTicket -= amount;
            }

            printf("\n 祝您旅途愉快！欢迎再次光临\n");
            success = 1;
        }
        else
        {
            printf("该等级的票不足，订票失败，以下为该航班乘客信息，希望对您的
订票有所帮助\n");
            FlightInfo(info);
        }

        while (!success)
        {
            printf("是否改变订票计划？Y/N\n");
            char r;
            getchar(); // 清除缓冲区
```

```
            r = getchar();
            if (r == 'Y' || r == 'y')
            {
                // 改变计划，重新选择航班
                BookTickets();
                return;
            } else
            {
                printf("\n 已经没有更多的票，您需要排队等候吗?(Y/N)\n");
                getchar(); // 清除缓冲区
                r = getchar();
                if (r == 'Y' || r == 'y')
                {
                    // 不改变计划，排队候票
                    printf("\n 请输入您的姓名（排队订票客户）:");
                    scanf("%s", name);
                    printf("\n 请输入您的身份证（排队订票客户）:");
                    scanf("%s", identification);
                    if (rank == 1)
                    {
                        // 进入经济舱排队队列
                        info->waitQueue1 = Appendqueue(info->waitQueue1, name,
amount, identification);
                    } else
                    {
                        // 进入商务舱排队队列
                        info->waitQueue2 = Appendqueue(info->waitQueue2, name,
amount, identification);
                    }
                    printf("\n 排队成功!\n");
                    success = 1;
                } else
                {
                    printf("\n 您选择不排队，谢谢使用！ \n");
                    success = 1;
                }
            }
        }
        printf("是否继续购票，Y/N\n");
        char choice;
        cin>>choice;
        if(choice == 'N' || choice == 'n')
        {
            return;
```

```
            }
        }
}


/**
 * @param Q     Q 为候补订票客户的队列
 * @param NameAndNumAndIDAndID 候补客户的姓名和订票数目，出队时，将姓名和关键字和身份证返回
 * @return
 */
Status QueueDelete(LinkQueue &q, NameAndNumAndID &NameAndNumAndID)
{
    WaitQNode *p;
    p = q.front->next;
    //带头结点的链表，当 front 指针和 rear 指针相等时，相当于队列为空，没有元素出队
    if (q.front == q.rear)
    {
        return ERROR;
    }
    //出队
    q.front->next = p->next;
    //出队到空时， rear = front
    if (q.front->next == NULL)
    {
        q.rear = q.front;
    }

    //返回出队元素的关键信息，包括候补客户的姓名和订票量
    NameAndNumAndID.num = p->preTickets;
    strcpy(NameAndNumAndID.name, p->name);
    strcpy(NameAndNumAndID.identification, p->identification);
    free(p);

    return OK;
}


/**
 * @brief 退票功能模块
 */
void ReturnTicket()
{
```

```
    struct Flight *info;
    int rank;
    CustomerNode *p1, *p2, *head;
    char cusname[10];
    char identification[20];
    system("cls");

    // 查找航班
    while (1)
    {
        info = find();
        if (info != NULL)
        {
            break;
        }
        printf("没有这个航班，请重新输入\n");
    }

    head = info->cusLinkList;
    p1 = head->next;
    printf("请输入你的姓名: ");
    scanf("%s", cusname);
    printf("请输入你的身份证号码: ");
    scanf("%s", identification);

    // 搜索客户是否订票
    p2 = head;
    while (p1 != NULL)
    {
        if ((strcmp(cusname, p1->name) == 0) && (strcmp(identification, p1->identification)
== 0))
        {
            break;
        }
        p2 = p1;
        p1 = p1->next;
    }

    if (p1 == NULL)
    {
        printf("对不起，你没有订过票或姓名和身份证不对应\n");
        return;
    }
    else
```

```
    {
        // 退票成功
        rank = p1->rank;
        p2->next = p1->next;
        info->left += p1->clientTickets;
        if (rank == 1)
        {
            info->leftEconomicTicket += p1->clientTickets;
        }
        else
        {
            info->leftBusinessTicket += p1->clientTickets;
        }
        printf("%s  成功退票！\n", p1->name);
        free(p1);
    }

    LinkQueue queue1 = info->waitQueue1;
    LinkQueue queue2 = info->waitQueue2;
    NameAndNumAndID nameAndNumAndID = {0, 0};

    if (rank == 1)
    {
        //cout<<"有经济舱退票，询问经济舱排队的客户" <<endl;
        // 有经济舱退票，询问经济舱排队的客户
        while (queue1.front->next != NULL && queue1.front->next->preTickets <=
info->leftEconomicTicket) {
            // 从候补客户队列中，出队客户的姓名和订票量用 NameAndNumAndID 返
回
            QueueDelete(info->waitQueue1, nameAndNumAndID);
            int y;
            printf("有经济舱票剩余 ，尊敬的%s ：\n", nameAndNumAndID.name);
            printf("是否确认订票（1 确认订票， 其他数字拒绝订票）\n");
            scanf("%d", &y);

            if (y == 1)
            {
                // 排队订票成功
                for (int i = 0; i < nameAndNumAndID.num; i++)
                {
                    printf(" 排 队 订 票 成 功    %s 的 座 位 号 是 :%d\n",
nameAndNumAndID.name, (info->left) - i);
                }
                // 剩余票减少
```

```
                    info->left -= nameAndNumAndID.num;
                    info->leftEconomicTicket -= nameAndNumAndID.num;
                    // 乘员名单链表插入排队订票成功的客户
                    info->cusLinkList = insertlink(info->cusLinkList, nameAndNumAndID.num,
nameAndNumAndID.name,
                                                   nameAndNumAndID.identification,
rank);
                }
            }
        }
        else
        {
            //cout<<"有商务舱客户退票，询问商务舱排队的客户" <<endl;
            // 有商务舱客户退票，询问商务舱排队的客户
            while (queue2.front->next != NULL && queue2.front->next->preTickets <=
info->leftBusinessTicket)
            {
                // 从候补客户队列中,出队客户的姓名和订票量用 NameAndNumAndID 返
回
                QueueDelete(info->waitQueue2, nameAndNumAndID);
                int y;
                printf("有商务舱票剩余 ，尊敬的%s ： \n", nameAndNumAndID.name);
                printf("是否确认订票（1 确认订票， 其他数字拒绝订票）\n");
                scanf("%d", &y);

                if (y == 1)
                {
                    // 排队订票成功
                    for (int i = 0; i < nameAndNumAndID.num; i++) {
                        printf(" 排 队 订 票 成 功     %s 的 座 位 号 是 :%d\n",
nameAndNumAndID.name, (info->left) - i);
                    }
                    // 剩余票减少
                    info->left -= nameAndNumAndID.num;
                    info->leftBusinessTicket -= nameAndNumAndID.num;
                    // 乘员名单链表插入排队订票成功的客户
                    info->cusLinkList = insertlink(info->cusLinkList, nameAndNumAndID.num,
nameAndNumAndID.name,
                                                   nameAndNumAndID.identification,
rank);
                }
            }
        }
}
```

```
// 冒泡排序函数，按照客户姓名从小到大排序
void bubbleSort_CusLinkList(CusLinkList head) {
    if (head == NULL || head->next == NULL) {
        return; // 如果链表为空或者只有头节点，不需要排序
    }

    CustomerNode *end = NULL; // 用于标记已经排好序的部分

    while (end != head->next) { // 外层循环控制排序的轮数
        CustomerNode *prev = head; // 前一个节点指针，初始为头节点
        CustomerNode *current = head->next; // 当前节点指针
        CustomerNode *nextNode = current->next; // 下一个节点指针

        while (nextNode != end) { // 内层循环进行一次完整的冒泡过程
            if (strcmp(current->name, nextNode->name) > 0) { // 比较当前节点和下一个
节点的姓名
                // 交换节点的位置
                prev->next = nextNode;
                current->next = nextNode->next;
                nextNode->next = current;

                // 交换指针
                CustomerNode *temp = current;
                current = nextNode;
                nextNode = temp;
            }

            // 更新指针，继续下一次比较
            prev = current;
            current = nextNode;
            nextNode = nextNode->next;
        }

        // 更新已排好序的部分
        end = current;
    }
}


Status Visit_Passenger_Information(Flight *pflight)
{
    Flight *info = pflight->next;
```

```
    while (1)
    {
        // 调用查找航班函数，返回给 info
        info = find();
        if (info != NULL)
        {
            break;
        }
        printf("没有这个航班，请重新输入\n");
    }

    CusLinkList head = info->cusLinkList->next;
    bubbleSort_CusLinkList(head);

    printf("客户姓名\t 客户订票量\t 身份证号码\t 航班 ID\t 舱位等级\t 飞行日期\t 起点站名\t 终点站名\n");
    while(head)
    {
        printf("%8s\t%8d\t%s\t%10s\t%6d\t %11s\t %9s\t %8s\n",
head->name, head->clientTickets, head->identification,
                info->flightCodeID,
head->rank ,info->day ,info->startPoint,info->destination);
        head = head->next;
    }
    return OK;
}


/**
 * @brief 退出程序模块界面
 */
void GoodbyeFace()
{
    system("cls");
    printf("\n");
    printf("\n");
    printf("                ----------------------------------------\n");
    printf("                |              感谢使用航空航班订票系统          |\n");
    printf("                |                                        |\n");
    printf("                |                   GoodBye !             \n");
    printf("                ----------------------------------------\n");

}
```

三．**main.cpp**

```
//
/**

*****************************************************************************
***
   * @file                : function_declaration.h
   * @author              : lct2023
   * @brief               : None
   * @attention           : None
   * @date                : 2024/6/2

*****************************************************************************
***
   */
//

#include "function_declaration.h"


int main()
{
      //初始化 pFlight，pFlight 为全局变量。
      InitFlight();
      //用 flight1 中的数据创建初始航班链表，里面含有四个节点
      Create(flight1);

    while(1)
    {
          fflush(stdin); //  清空标准输入缓冲区
          SetUp();//菜单初始化
          cout << "请输入您的选择（0~7）:" << endl;
          int select;
          scanf("%d", &select);
          fflush(stdin); //  清空标准输入缓冲区
          system("CLS");
          switch (select)
          {
                case 1:
                      cout << "1.录入航班信息  "<< endl;
                      if(OK == InsertFlight())
                            cout <<"录入航班信息成功"<< endl;
```

```
        break;
case 2:
        cout << "2.加载航班数据  "<< endl;
        PrintFilghtlist(pFlight);
        break;
case 3:
        cout << "3.清除航班记录  "<< endl;
        if (OK == DeleteFlight())
       {
              printf("删除成功\n");
        }
       else
       {
              printf("没有这个航班，删除失败！\n");
        }
        break;
case 4:
        cout << "4.查询航班  "<< endl;
        SearchFlight();
        break;
case 5:
        cout << "5.购票&退票  "<< endl;
        int choice5;
        printf("请输入您的选择，1 表示购票，其他表示退票\n");
       scanf("%d",&choice5);

      if(choice5 == 1)       BookTickets();
      else   ReturnTicket();

        break;
case 6:
        cout << "6.查询旅客信息  "<< endl;
        Visit_Passenger_Information(pFlight);
        break;
case 7:
         cout << "7.保存航班信息  "<< endl;
         if(OK ==    Log_Flight_Information(pFlight))
         {
               cout << "保存航班信息成功！" << endl;
         }
         else
         {
              cout << "保存航班信息失败！"<< endl;
         }
```

```
                break;
            case 0:
                GoodbyeFace();
                exit(0);
            default:
                cout << "输入错误，请重新输入！"<< endl;
                break;
        }
        system("PAUSE");
        system("CLS");
    }
     return 0;
}
```