

数 据 结 构

课程设计报告

学 院 计算机工程学院 班 级 软件1912

姓 名 游嘉升 学 号 201921122050

成 绩 指导老师 王俊玲

2020年 7 月 1 日

**课程设计答辩记录表**

|  |  |
| --- | --- |
| **课程设计题目及要求** | |
| **题目**  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。  **要求**  (1)每条航线所涉及的信息有：起点站名，终点站名、航班号、飞机号、飞行日期、乘员定额、余票量、已订票的客户名单(包括姓名、订票量及所订的座位号)以及等候替补的客户名单(包括姓名、所需票量)；  (2)系统能实现的操作和功能如下：  1)录入：可以录入航班情况，录入的航班数据可存于内存中，每天结束工作时内存中的数据可通过“保存录入航班数据”添加存入文件中；  提示：使用合适的结构组织，使它方便按“起点站名”与“终点站名”被查找。设一般航班记录数据文件中将保存近10000条记录。  2)加载：将文件中保存的航班数据至内存中。  3)清除：将日期超过当前日期的航班记录清除。  4)查询航线：根据旅客提出的“起点站名”与“终点站名”输出下列信息：航班号、飞机号、星期几飞行，最近一天航班的日期和余票额；  5)承办订票业务：根据客户提出的要求(航班号、订票数额)查询该航班票额情况，若尚有余票，则为客户办理订票手续，输出座位号，并将订票结果保存文件中。若已满员或余票额少于订票额，则需重新询问客户要求。若需要，可登记排队候补，候补名单将保存于内存中，选择合适的数据结构以方便候补业务的处理；  6)承办退票业务：根据客户提供的情况(日期、航班)，为客户办理退票手续，然后查询该航班是否有人排队候补，首先询问排在第一的客户，若所退票额能满足他的要求，则为他办理订票手续，否则依次询问其他排队候补的客户。  7)保存航班信息至文件中。  8)保存订票名单。  9)保存候补名单 | |
| **完成情况：** | |
| 基本数据结构设计 |  |
| 算法设计及流程图 |  |
| 输入输出设计 |  |
| 代码规范及注释 |  |
| 数据测试及调试 |  |
| 代码理解与说明 |  |
| 评语 |  |
| 综合评分 |  |

目 录

[第 1 章 课程设计目的与分析 1](#_Toc44628557)

[1.1 课程设计目的 1](#_Toc44628558)

[1.2课程设计分析 1](#_Toc44628559)

[第 2 章 数据结构的设计与原理 2](#_Toc44628560)

[2.1 数据结构定义 2](#_Toc44628561)

[2.1.1 航班信息结构 2](#_Toc44628562)

[2.1.2 票务信息结构与乘客结构 3](#_Toc44628563)

[2.1.3 根据地点，航班号，时间定义的用于索引航班结点的数据结构 3](#_Toc44628564)

[2.1.4 全局参数定义 4](#_Toc44628565)

[2.2 设计原理 5](#_Toc44628566)

[第3章 功能模块化设计 6](#_Toc44628567)

[3.1 航班结点插入 6](#_Toc44628568)

[3.2 航班记录录入功能 8](#_Toc44628569)

[3.3 航班数据加载功能 9](#_Toc44628570)

[3.4 航班记录清除功能 10](#_Toc44628571)

[3.5 查询航班功能 11](#_Toc44628572)

[3.6 订票业务功能 12](#_Toc44628573)

[3.7 退票业务功能 13](#_Toc44628574)

[3.8 保存航班信息功能 14](#_Toc44628575)

[第 4 章 测试分析 15](#_Toc44628576)

[4.1 测试程序执行情况 15](#_Toc44628577)

[4.2 测试过程中遇到的问题解决与待解决的问题 27](#_Toc44628578)

[第 5 章 心得体会 28](#_Toc44628579)

[第 6 章 参考文献 29](#_Toc44628580)

第 1 章 课程设计目的与分析

## 1.1 课程设计目的

1、训练学生灵活应用所学数据结构知识，独立完成问题分析，结合数据结构理论知识，编写程序求解指定问题。

2.初步掌握软件开发过程的问题分析、系统设计、程序编码、测试等基本方法和技能。

3.提高综合运用所学的理论知识和方法独立分析和解决问题的能力。

4.训练用系统的观点和软件开发一般规范进行软件开发，巩固、深化学生的理论知识，提高编程水平，并在此过程中培养他们严谨的科学态度和良好的工作作风。

## 1.2课程设计分析

航空客运订票系统是面向用户的一个系统，所以需要设计良好的数据结构来保证数据的存放、删除与查找的速度，保证系统工作的运行效率。需要有一定的自我容错能力，保证系统的稳定性和安全性。还需要设计一套清楚的服务逻辑，做到界面简单易懂，方便操作。

实现的功能分有七个模块：①航班记录的录入②航班数据的加载③航班记录的清除④航班的查询⑤航班的订票业务⑥航班的退票业务⑦航班数据的保存。



图1.2-1 航空客运订票系统功能

第 2 章 数据结构的设计与原理

## 2.1 数据结构定义

### 2.1.1 航班信息结构

struct Plane\_Node {

string start, end;

string FlightNum;

string PlaneNum;

string date;

string cap, lecap;

Plane\_Node\* saplnext = NULL, \* safnnext = NULL,\*satmnext=NULL;

Plane\_Node\* saplpre = NULL, \* safnpre = NULL,\*satmpre=NULL;

Tickets\_Node\* ticket;

Plane\_Node() {

ticket = new Tickets\_Node;

}

};

该结构用于存储飞机航班基本信息，起点、终点、航班号、飞机号、飞行日期、乘载量、余票量都采用了string类进行存储，数据的存储读取快捷，比较方便，也利于正误性的检测。

Plane\_Node指针域可以使每个结点可以以相同时间(same time)、航班号(same flightnumber)、地点(same place)为关键词的双链表，六个指针域贯穿三条关键链。

Tickets\_Node是指向该航班票务信息结构的的指针域。

Plane\_Node()为该结构的构造函数，构造一个新的结点时需要开辟一个新的票务信息结构。

### 2.1.2 票务信息结构与乘客结构

struct Passenger\_Node {

string name;

int seat=-1;

};

struct Tickets\_Node {

Passenger\_Node\* Buyer[500] = {0};

queue<int> Refund;

queue<Passenger\_Node\*> Waiter;

};

Tickets\_Node结构为票务信息表，每个航班结点都会有该结点。Buyer在Tickets\_Node中以Passenger\_Node的结构数组来存储购票者信息。Refund以队列结构暂时记录退票的座位号，Waiter为候补者的Passenger\_Node队列，暂时记录候补者的信息。

### 2.1.3 根据地点，航班号，时间定义的用于索引航班结点的数据结构

struct Flight\_Start {

string start;

Flight\_End\* FENode = NULL;

Flight\_Start\* left = NULL, \* right = NULL;

int height;

};

struct Flight\_End {

string End;

Plane\_Node\* data = NULL;

Flight\_End\* left = NULL, \* right = NULL;

int height;

};

struct Flight\_Num {

Plane\_Node\* data = NULL;

Flight\_Num\* left = NULL, \* right = NULL;

int height;

};

struct Flight\_Time {

string date;

int count=0;

Plane\_Node\* data = NULL;

Flight\_Time\* next=NULL;

};

四个结构定义分别用于以地点、航班号、时间为关键词建立的AVL树与链表。

Flight\_Start与Flight\_End是以地点为关键词建立的AVL树的结点结构，其中start与end的string类用于存放地名，Flight\_Start结构中的FENode用于链接用于查找终点的Flight\_End树，而Flight\_End中的data指针域用于链接相同起点与终点的航班链。

Flight\_Num是以航班号为关键词建立的AVL树的结点结构，其中data指针域用于链接相同航班号的航班链。

Flight\_Time是以时间为关键词建立的链表的结构结点，date为该链的飞行日期，data指针域用于链接相同飞行时间的航班链，count用于记录次日期总共的航班数量。

### 2.1.4 全局参数定义

Plane\_Node\* Search\_r = NULL;

int FlightCount = 0;

Search\_r用于存放AVL树搜索后查找到的Plane\_Node结点。

FlightCount用于记录当前储存于内存的航班数量。

## 2.2 设计原理

基于AVL树的添加与查找，链表的添加、查找与删除，每个航班都有其独立的结点存放信息，且能通过索引关键词找到该航班，能够在O[(logN)2]的时间复杂度下以地点索引查找到航班，在O(logN)的复杂度下以航班号找到航班。构造如图2.2-1所示。

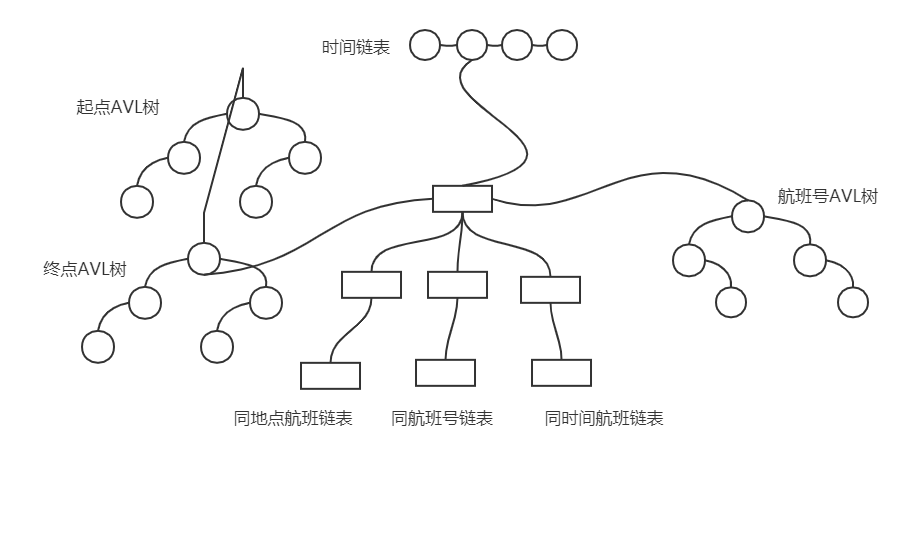


图2.2-1 系统内存数据储存索引构造图

第3章 功能模块化设计

## 3.1 航班结点插入

函数：

航班插入函数：

void Plane\_Insert(Flight\_End\* End, Plane\_Node\* New)

void Plane\_Insert(Flight\_Num\*& FNT, Plane\_Node\* New)

void Plane\_Insert(Flight\_Time\*& FTT, Plane\_Node\* New)

寻找结点插入位置函数：

Flight\_Num\* FN\_Insert(Flight\_Num\*& FNT, Plane\_Node\* New)

Flight\_End\* FE\_Insert(Flight\_End\* End, string end, Plane\_Node\* New)

Flight\_Start\* FS\_Insert(Flight\_Start\*& Start, string start, string end, Plane\_Node\* New)

Flight\_Time\* FT\_Insert(Flight\_Time\*& FTT, Plane\_Node\* New)、

各种AVL树旋转函数：

Flight\_Start\* SigleLeftRotation(Flight\_Start\* A)

Flight\_Start\* SigleRightRotation(Flight\_Start\* A)

Flight\_Start\* DoubleleftrightRotation(Flight\_Start\* A)

Flight\_Start\* DoublerightleftRotation(Flight\_Start\* A)

Flight\_End\* SigleLeftRotation(Flight\_End\* A)

Flight\_End\* SigleRightRotation(Flight\_End\* A)

Flight\_End\* DoubleleftrightRotation(Flight\_End\* A)

Flight\_End\* DoublerightleftRotation(Flight\_End\* A)

Flight\_Num\* SigleLeftRotation(Flight\_Num\* A)

Flight\_Num\* SigleRightRotation(Flight\_Num\* A)

Flight\_Num\* DoubleleftrightRotation(Flight\_Num\* A)

Flight\_Num\* DoublerightleftRotation(Flight\_Num\* A)

流程图：

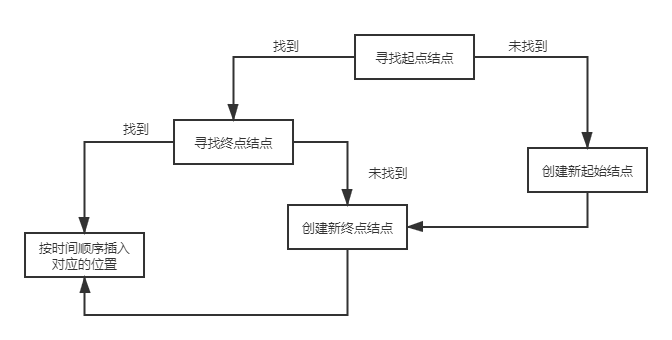


图3.1-1 按地点建立的AVL树插入流程图

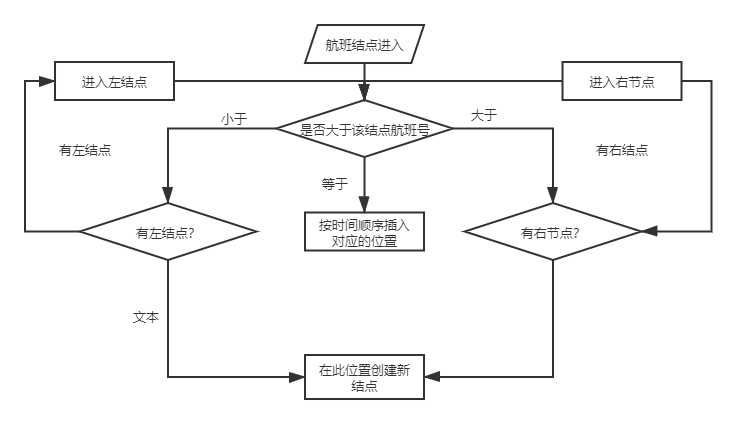


图3.1-2 按航班号建立的AVL树插入流程图

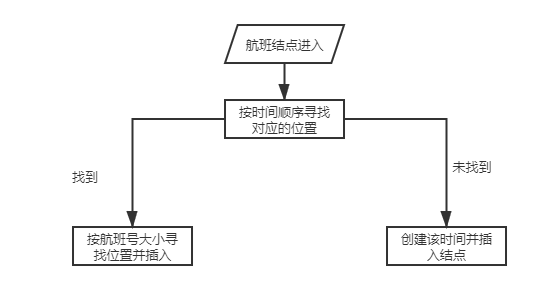


图3.1-3 按时间建立的链表插入流程图

## 3.2 航班记录录入功能

函数：

void Entry(Flight\_Num\*& FNT, Flight\_Start\*& FST, Flight\_Time\*& FTT)

流程图：

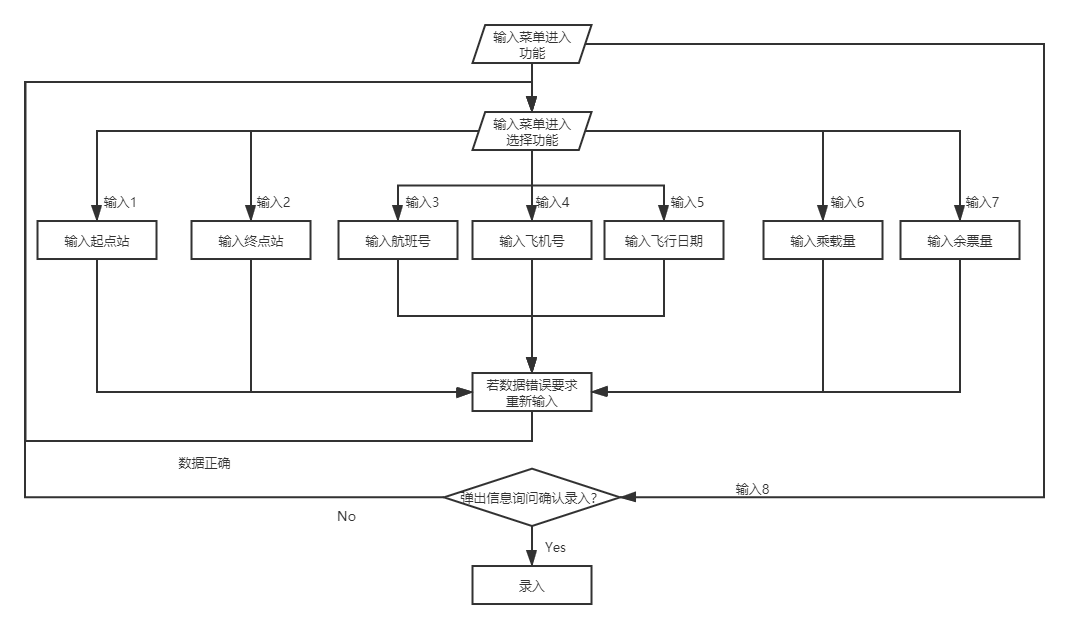


图3.2-1 航班记录录入功能流程图

## 3.3 航班数据加载功能

函数：

void Install\_Buyer(Plane\_Node\* PN)

void Install\_Waiter(Plane\_Node\* PN)

void Install(Flight\_Num\*& FNT, Flight\_Start\*& FST, Flight\_Time\*& FTT)

流程图：

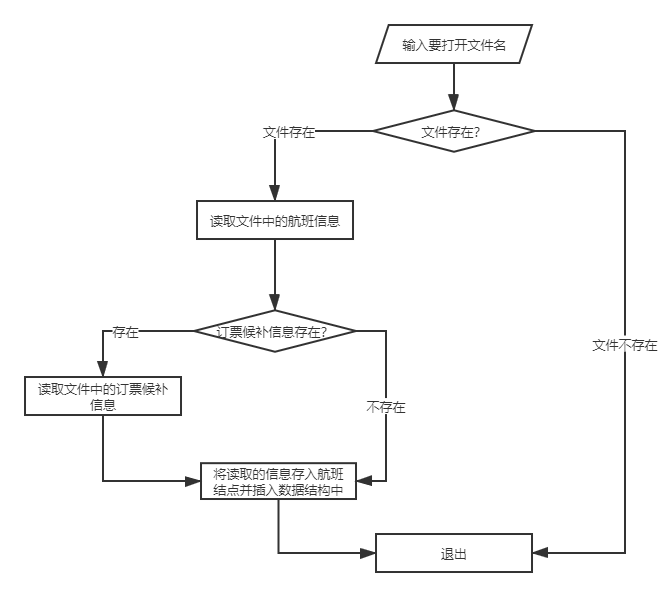


图3.3-1 航班数据加载功能流程图

## 3.4 航班记录清除功能

函数：

void Del\_Flight(Flight\_Num\* FNT, string date)

void Clear\_Flight(Flight\_Num\* FNT, string date)

void Clear(Flight\_Num\* FNT)

流程图：

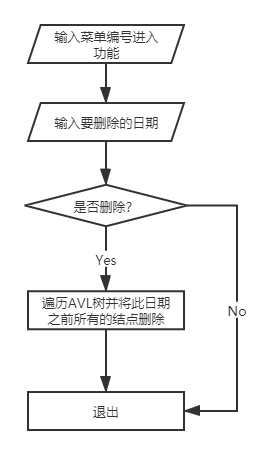


图3.4-1 航班记录清除功能流程图

## 3.5 查询航班功能

函数：

void SearchFlight\_E(Flight\_End\* FET, string end)

void SearchFlight\_E(Flight\_End\* FET, string end)

void SearchFlight\_S(Flight\_Start\* FST, string start, string end)

流程图：

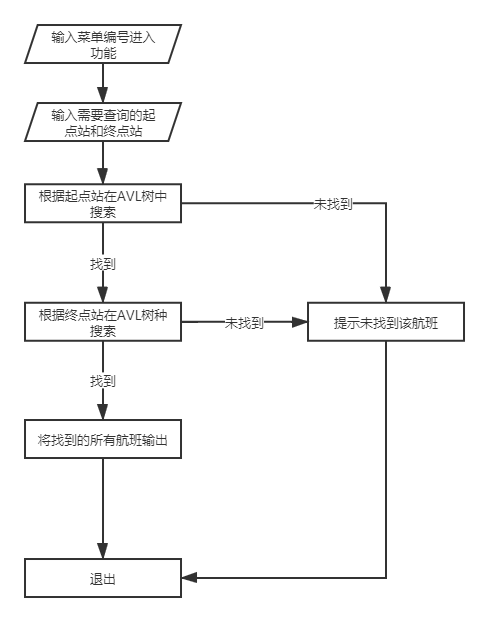


图3.5-1 查询航班功能的流程图

## 3.6 订票业务功能

函数：

void SearchFlight\_N(Flight\_Num\* FNT, string FlightNum)

void Book\_Tickets(Plane\_Node\* PN, int tickets)

void Wait\_Tickets(Plane\_Node\* PN)

int Check\_No(int max, int mode)

int CheckTicket\_N(Plane\_Node\* Buy)

void Book(Flight\_Num\* FNT)

流程图：

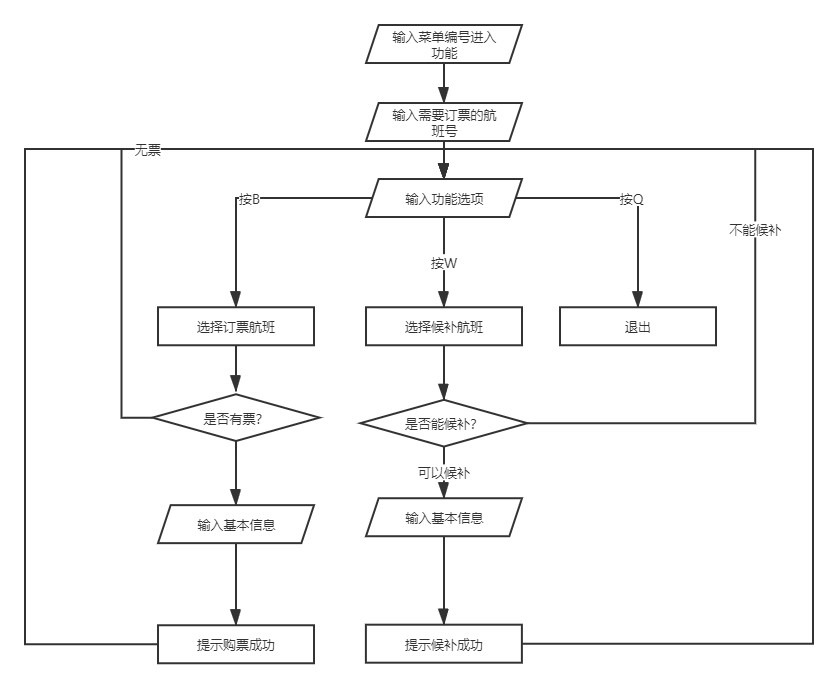


图3.6-1 订票业务功能的流程图

## 3.7 退票业务功能

函数：

void Refound\_Tickets(Plane\_Node\* PN)

void Refund(Flight\_Num\* FNT)

流程图：

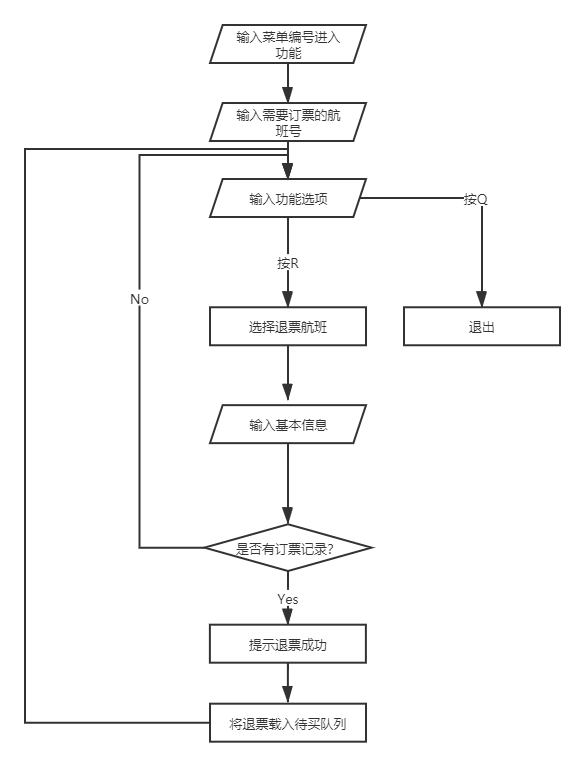


图3.7-1 退票业务功能流程图

## 3.8 保存航班信息功能

函数：

void Del\_Flight(Flight\_Time\* FTT)

void Save\_Buyer(Plane\_Node\* PN)

void Save\_Waiter(Plane\_Node\* PN)

void Save\_Flight(Flight\_Time\* FTT, FILE\* FP)

void Save(Flight\_Time\* FTT)

流程图：

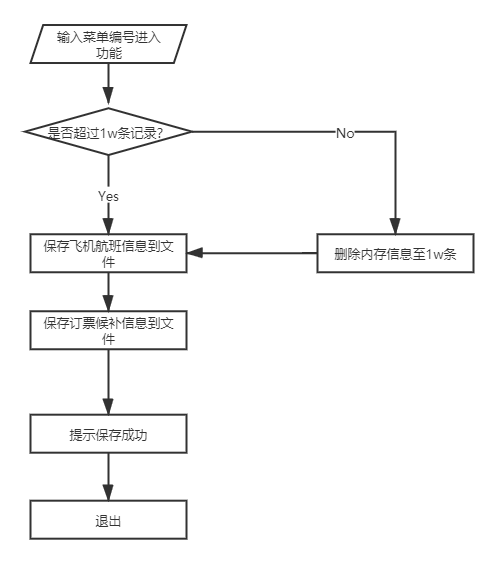


图3.8-1 保存航班信息流程图

第 4 章 测试分析

## 4.1 测试程序执行情况

1.菜单初始化：



图4.1-1 菜单初始化测试

2.航班录入功能：

测试数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 起点站 | 终点站 | 航班号 | 飞机号 | 飞行日期 | 乘载量 | 余票量 |
| 北京 | 福州 | JD5555 | P0123 | 20200701 | 100 | 100 |



图4.1-2 航班记录录入功能测试



图4.1-3 测试航班已录入

3.加载航班数据：

测试数据：

见附录1：测试0401.txt，测试0501.txt，测试0601.txt，测试0701.txt

（共10800条数据信息,4个月，一月30天，一天90条）

数据生成源代码见附录2：日期生成.cpp



图4.1-4 航班数据加载功能测试

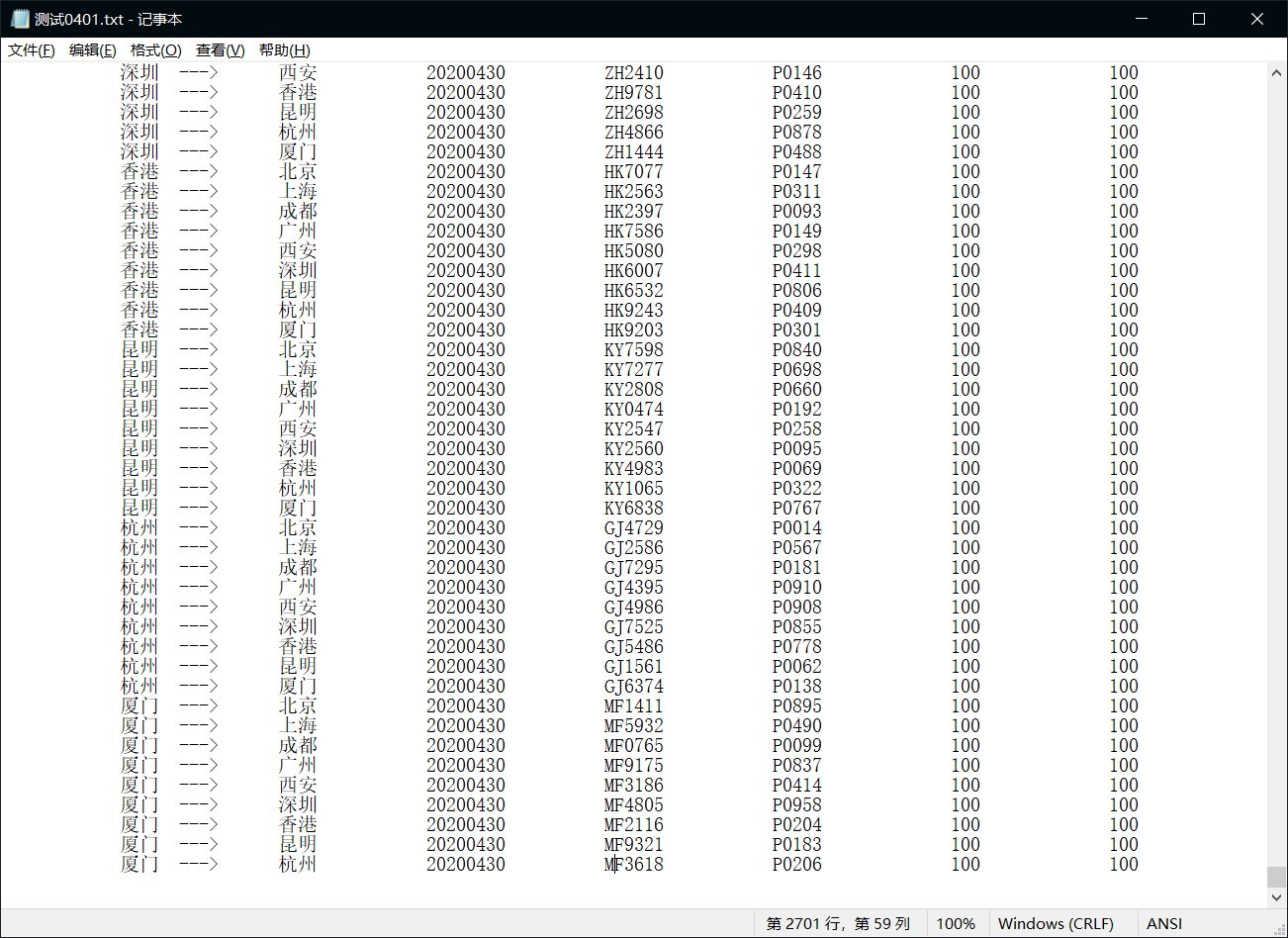


图4.1-5 航班数据加载测试文件1

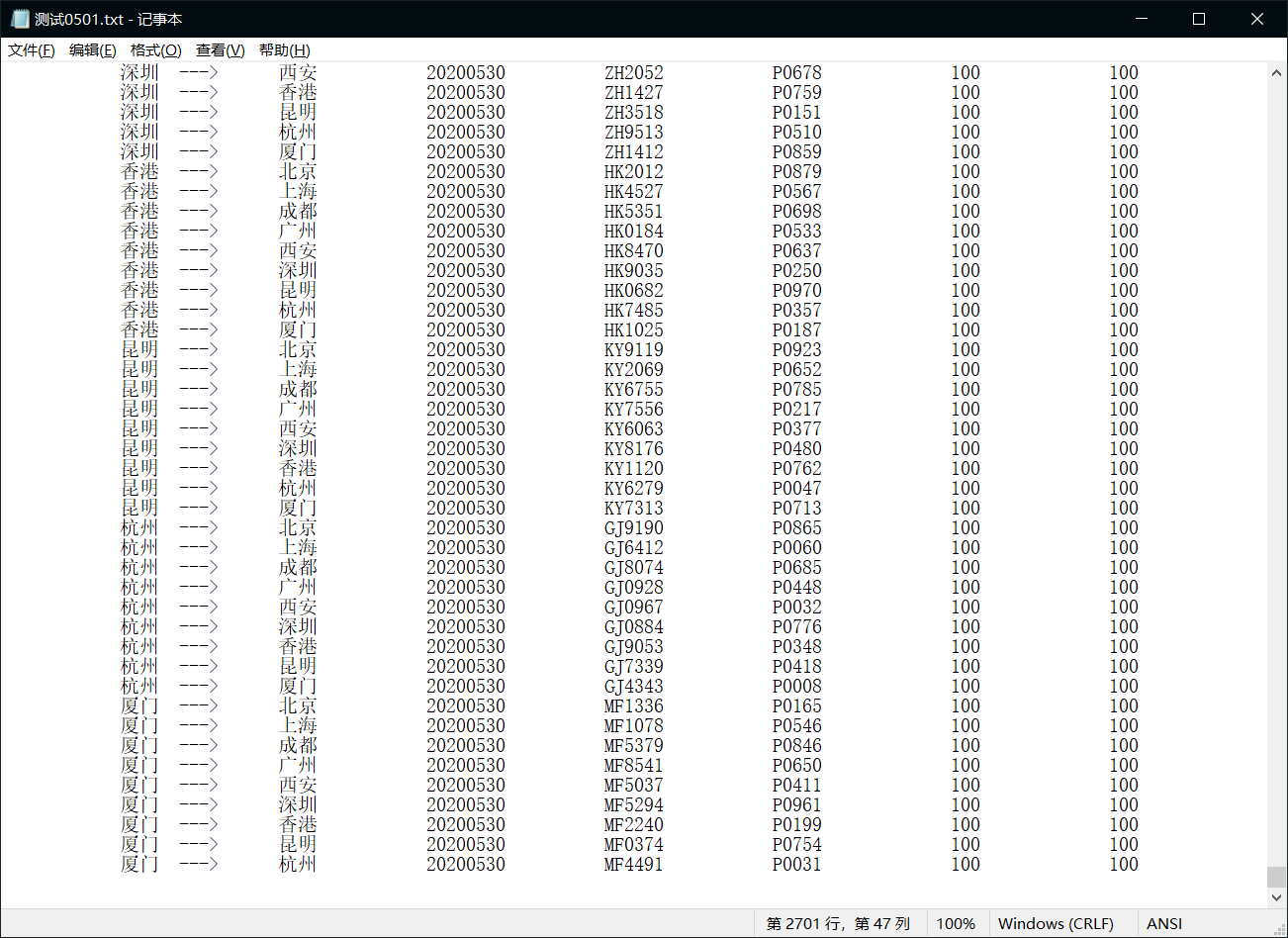


图4.1-6 航班数据加载测试文件2

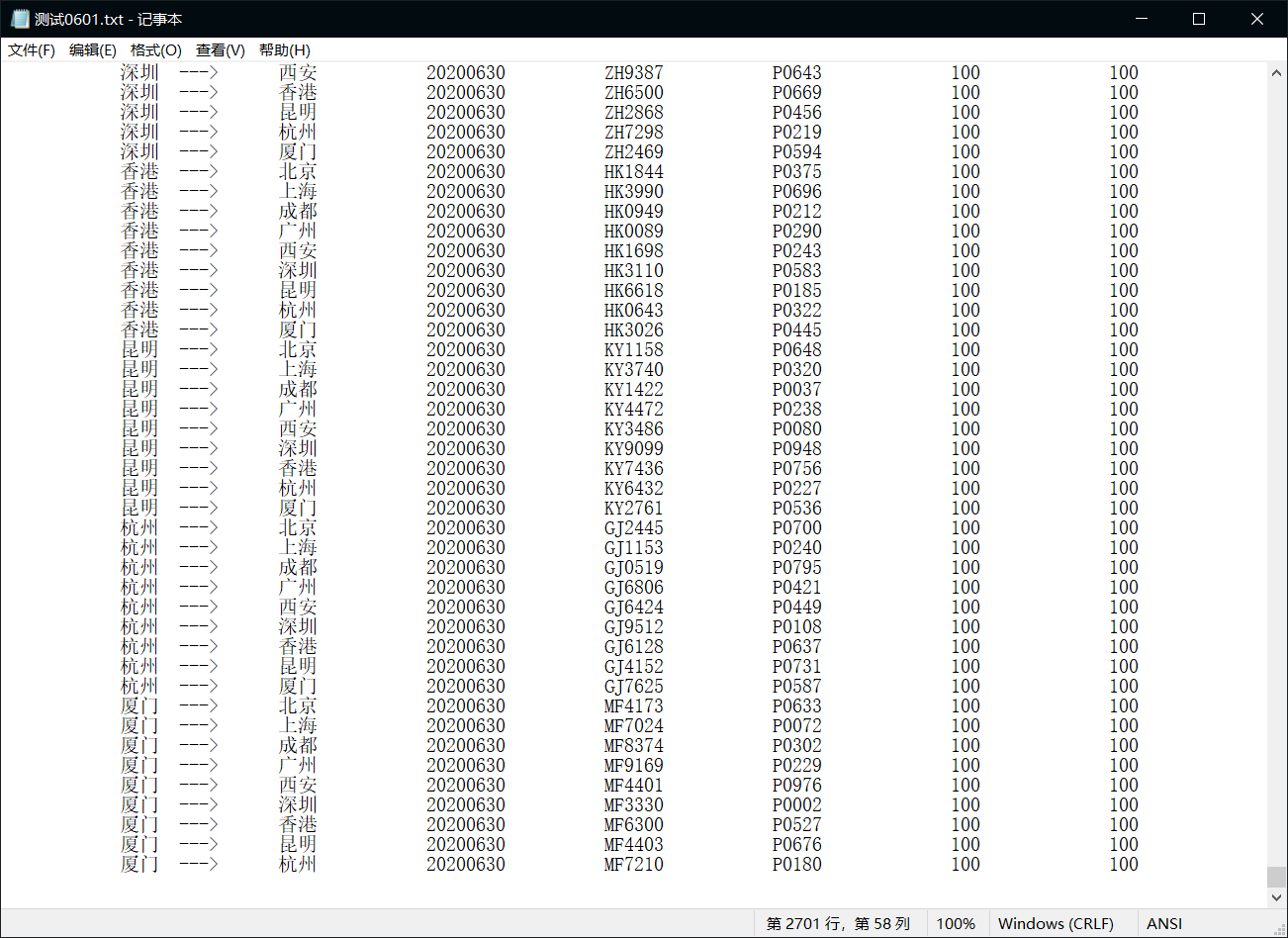


图4.1-7 航班数据加载测试文件3

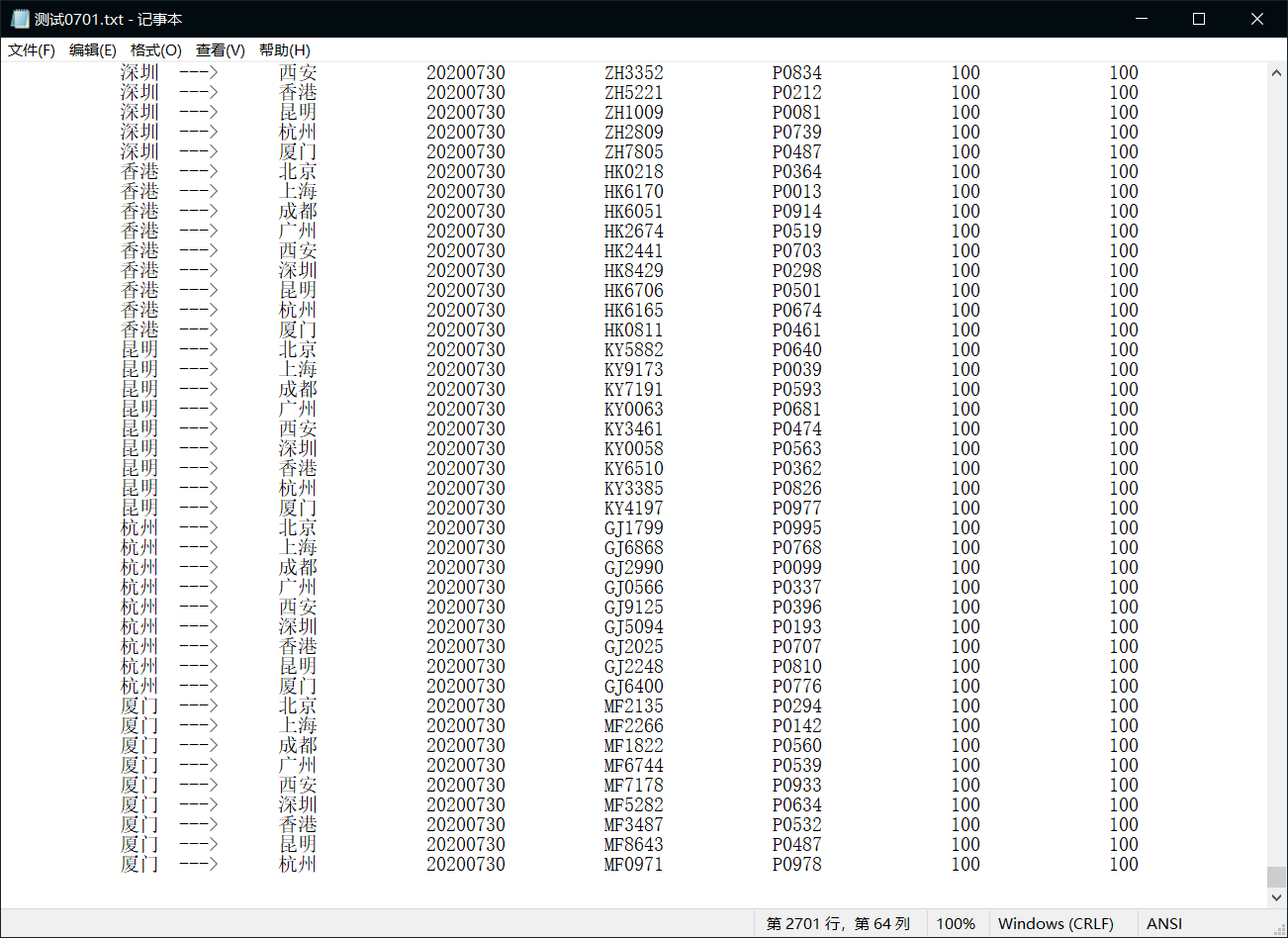


图4.1-8 航班数据加载测试文件4

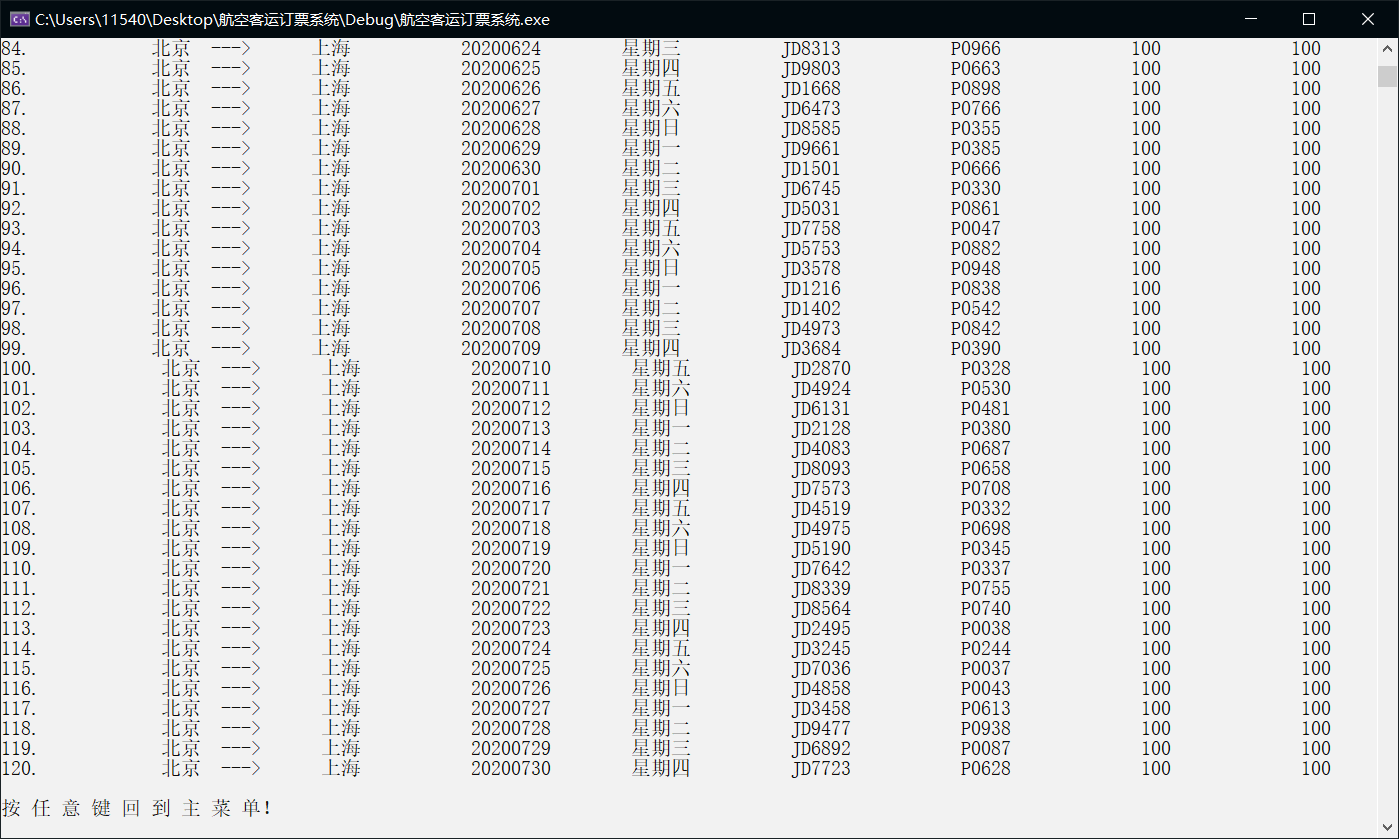


图4.1-9 航班数据已加载

3.清除航班记录

测试数据：20200402

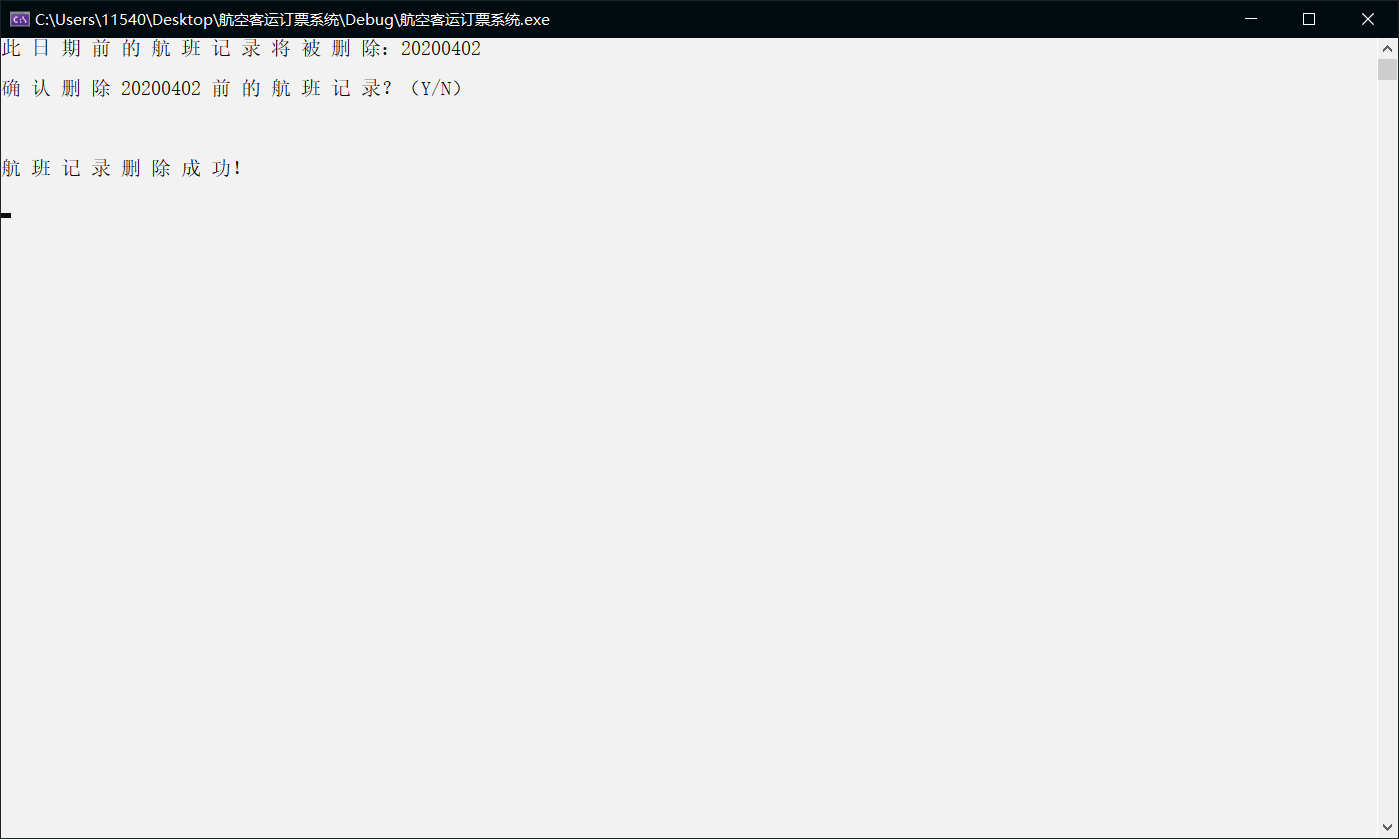


图4.1-10 清除航班记录功能测试

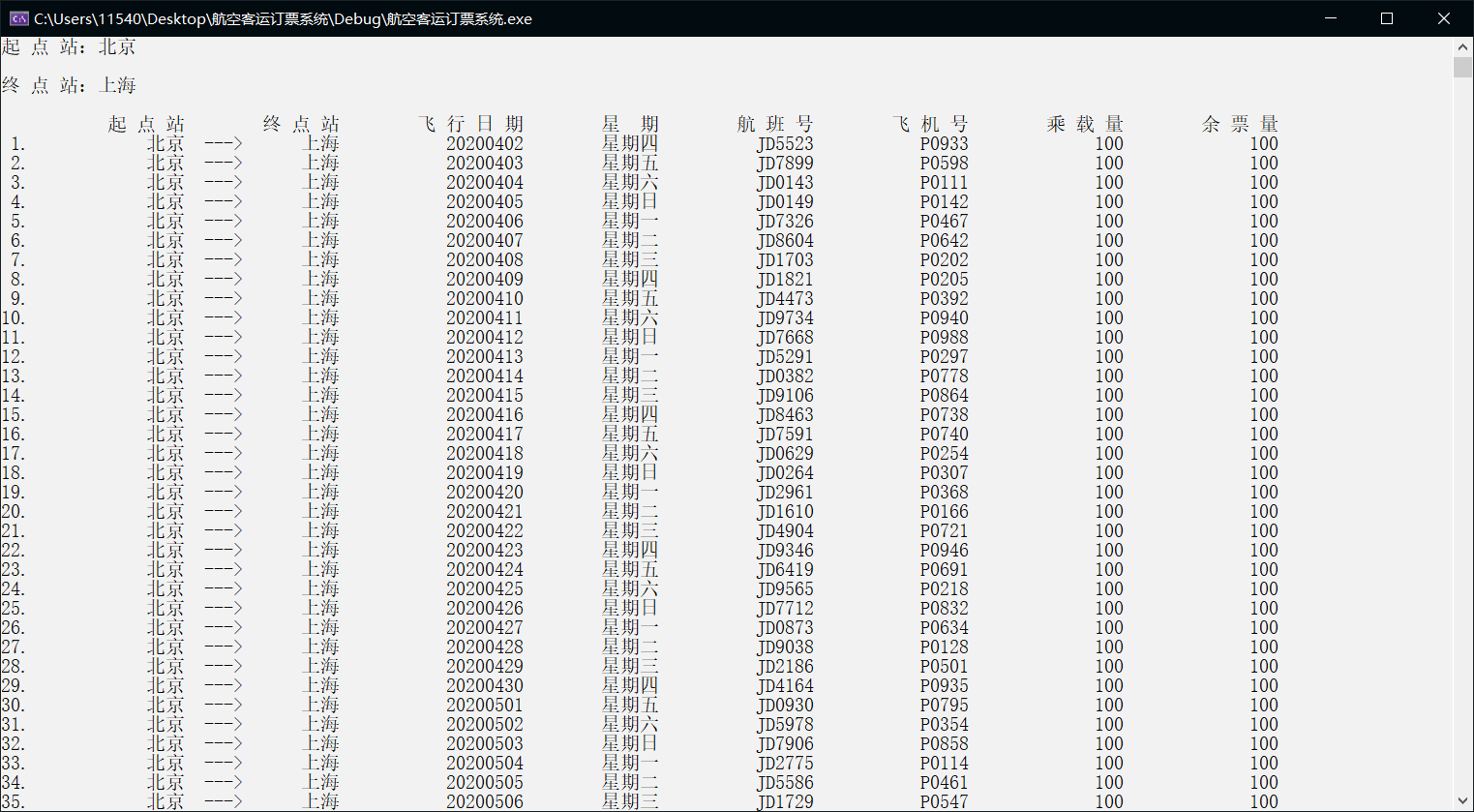


图4.1-11 20200402前记录已清除

4.查询航班

测试数据：

起点站：北京

终点站：上海

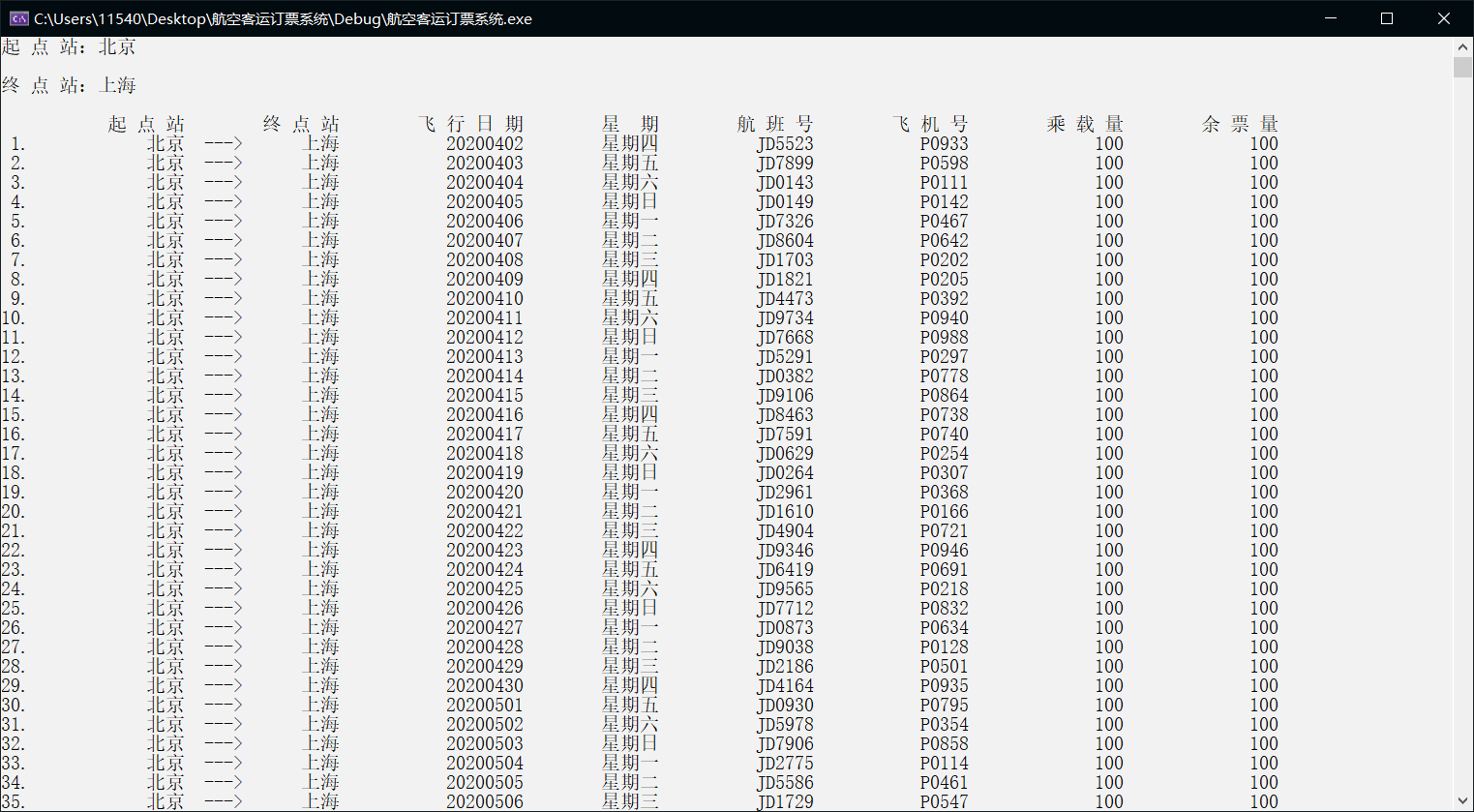


图4.1-12 查询航班功能测试

5.订票业务

测试数据：

航班号：JD5649

订票航班序号：1

订票数：1

购票者姓名：YOU

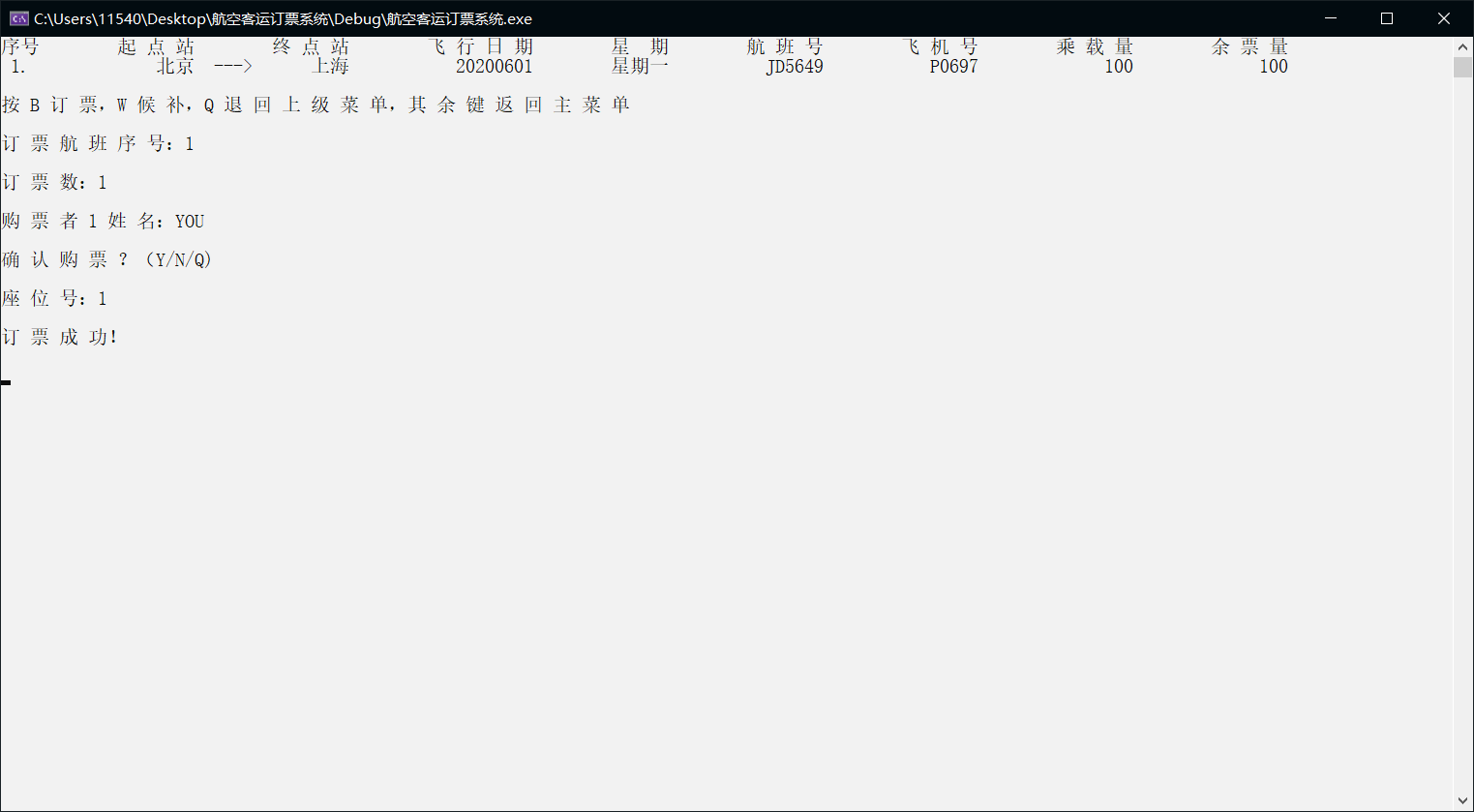


图4.1-13 订票业务功能测试

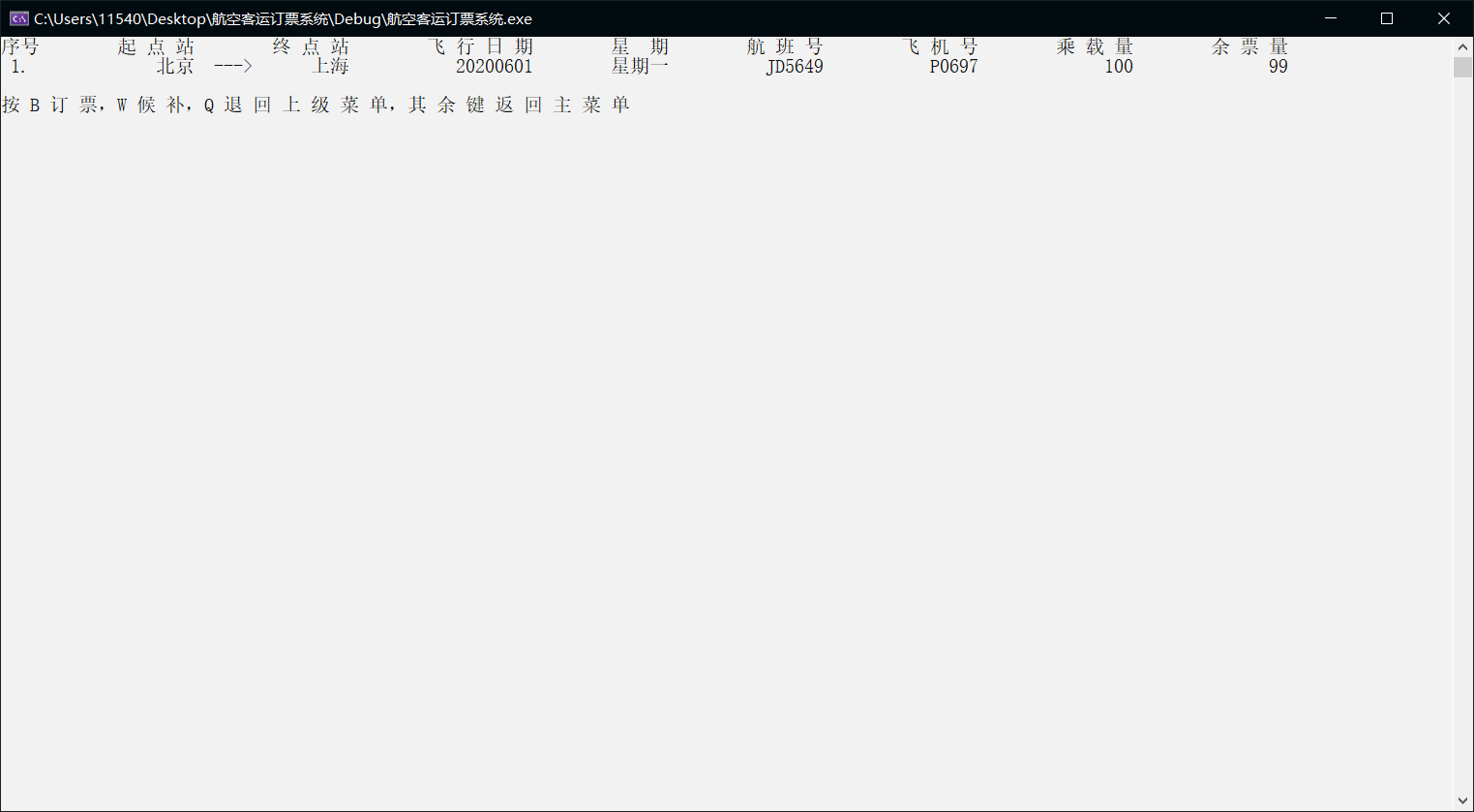


图4.1-14 订票后余票量减少

测试数据：

航班号：JD5649

订票航班序号：1

订票数：101

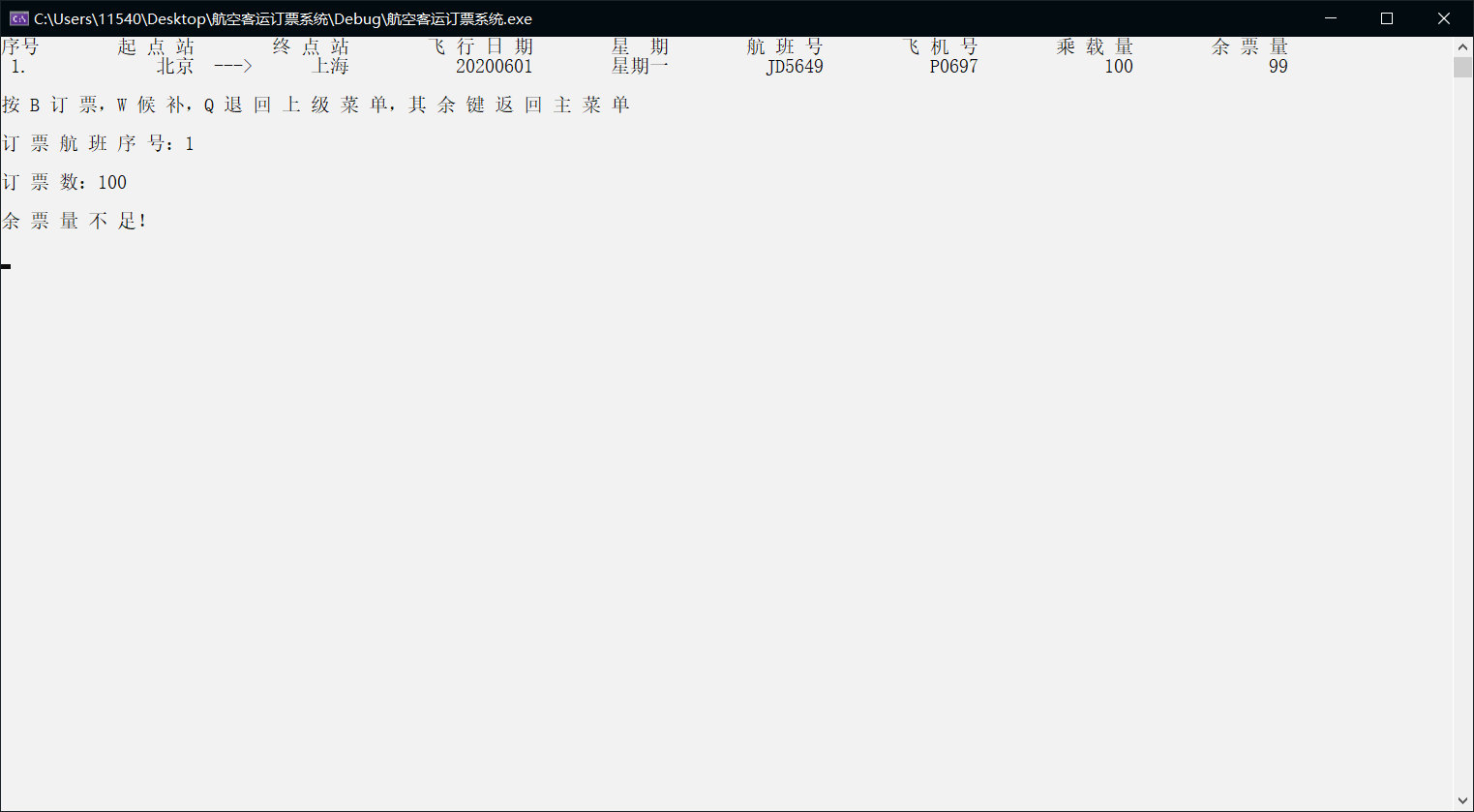


图4.1-15 订票超量提示

测试数据：

航班号：JD5649

候补航班序号：1

候补者姓名：WU

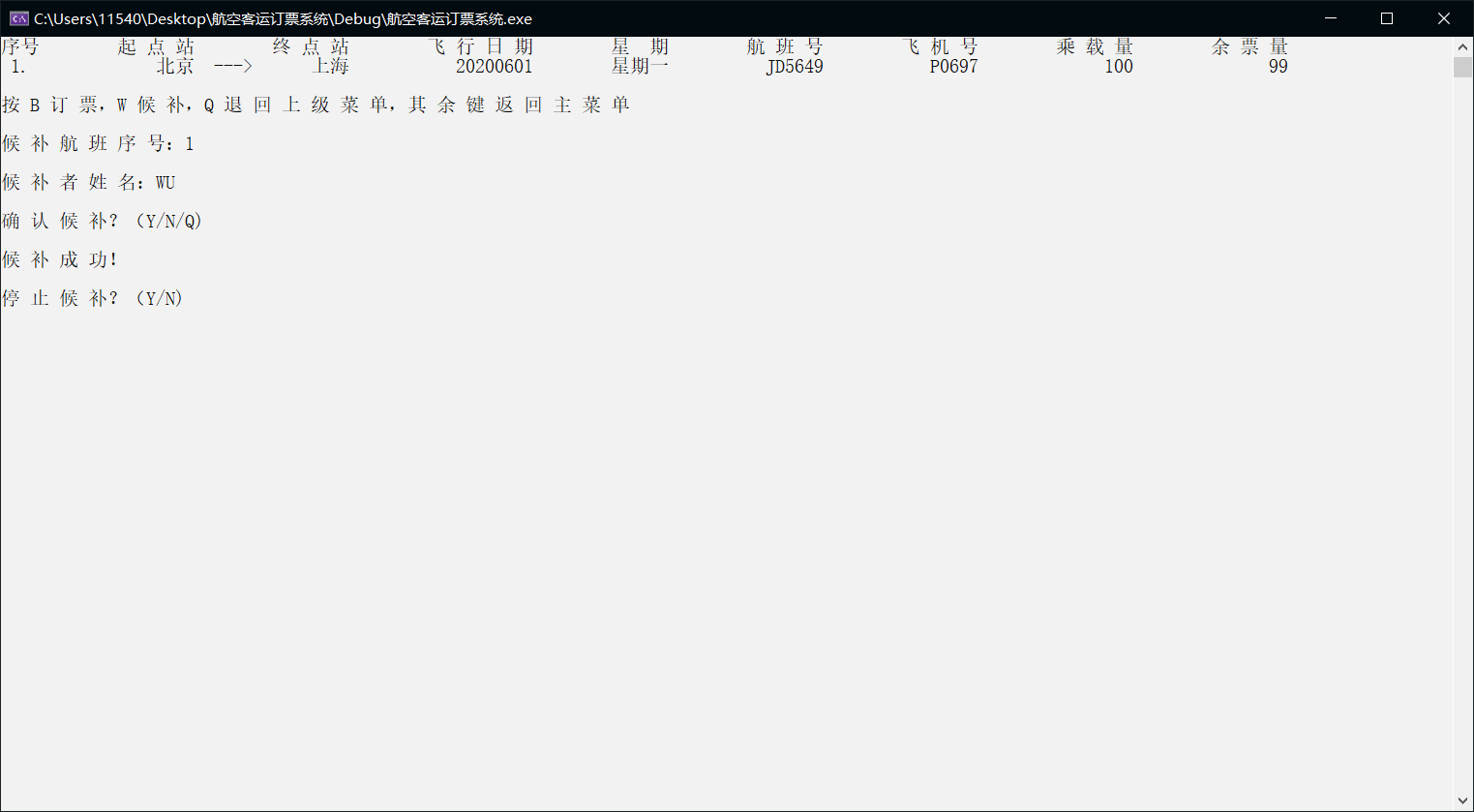


图4.1-16 候补业务功能测试

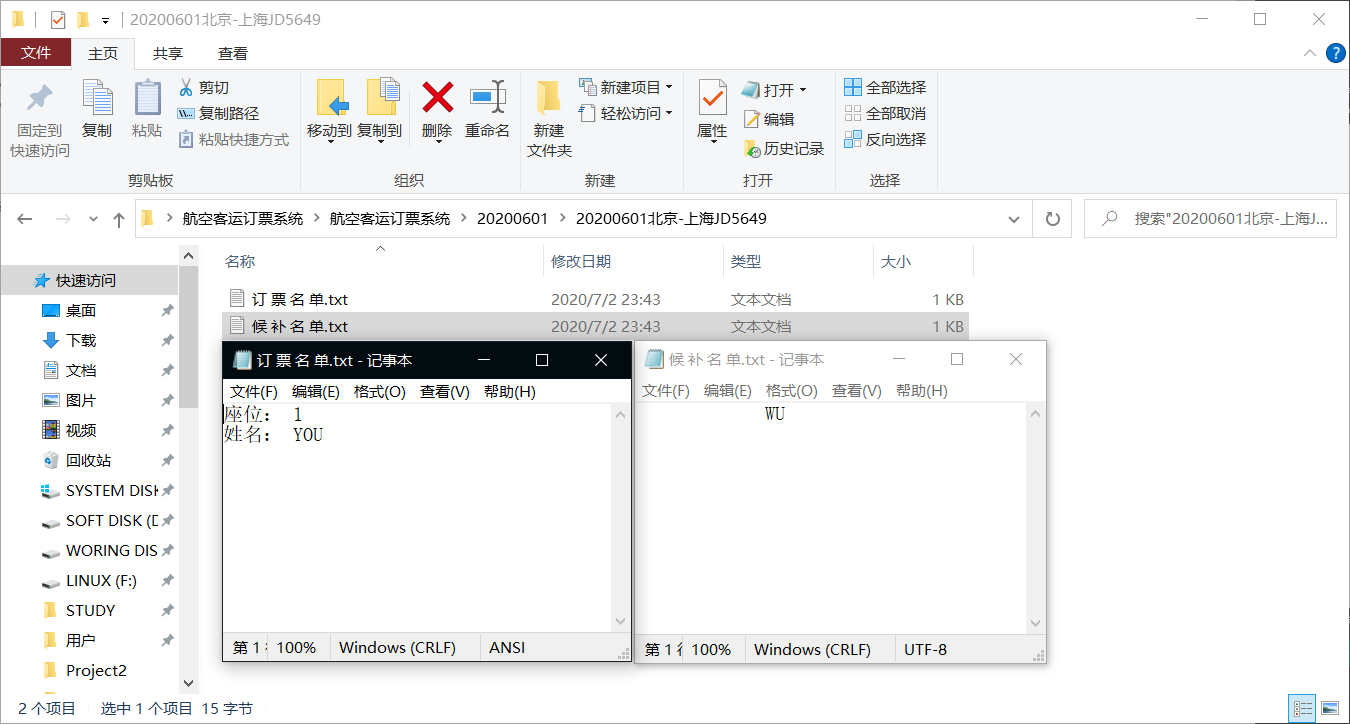


图4.1-17 订票候补业务功能保存后文件显示

6．退票业务

测试数据：

航班号：JD5649

退票航班序号：1

退票者姓名：YOU

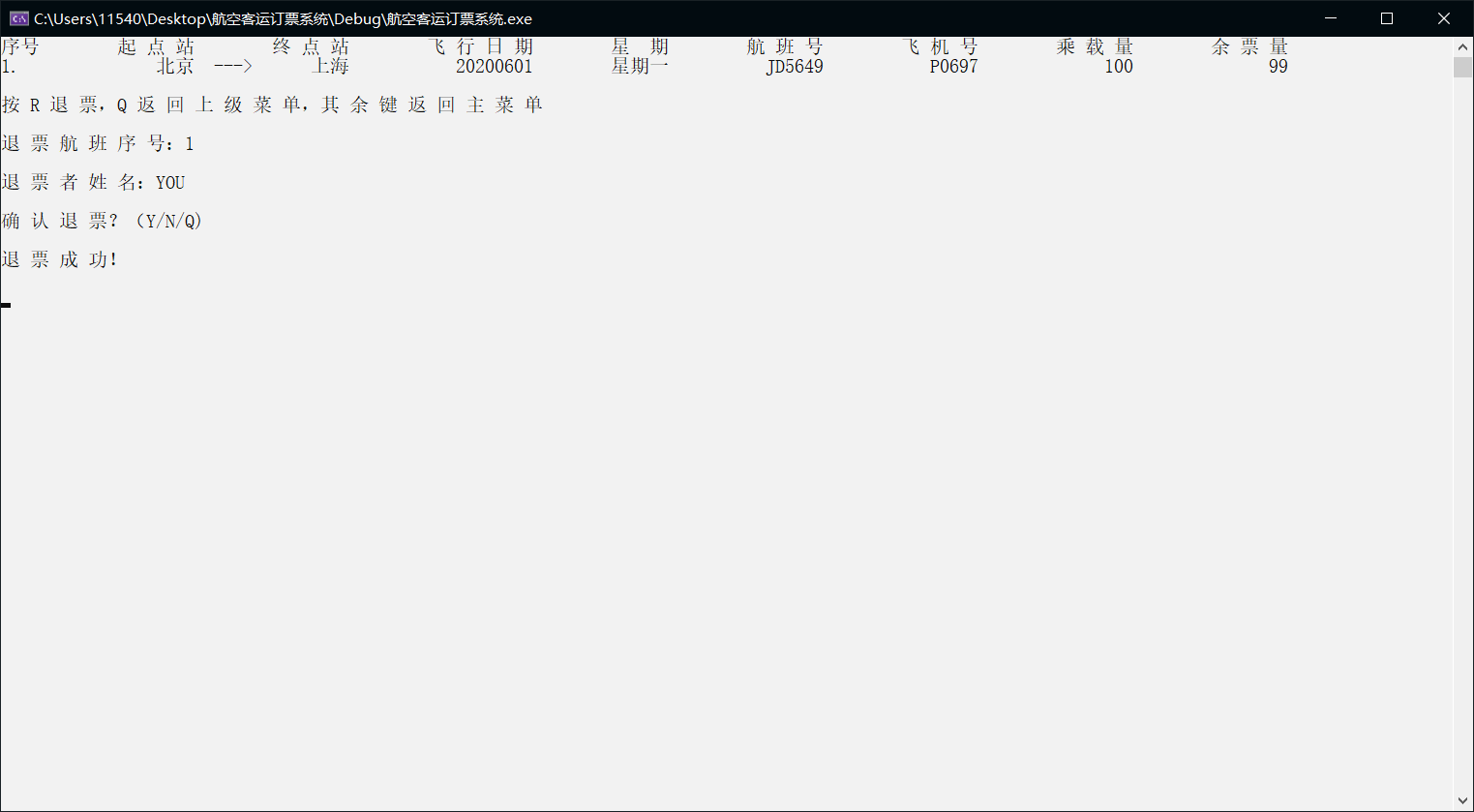


图4.1-18 退票业务功能测试

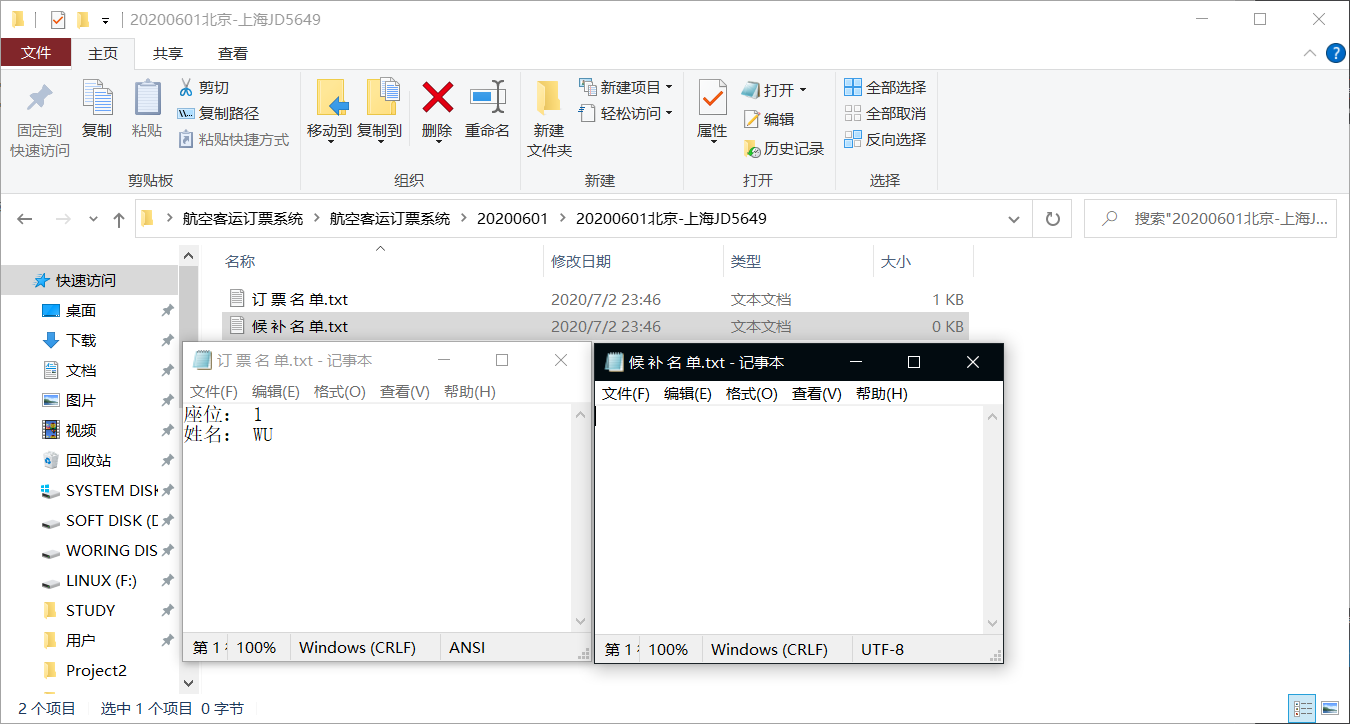


图4.1-19 退票业务功能保存后文件显示，候补者顺位订票

7.保存航班信息功能



图4.1-20 保存航班信息功能测试

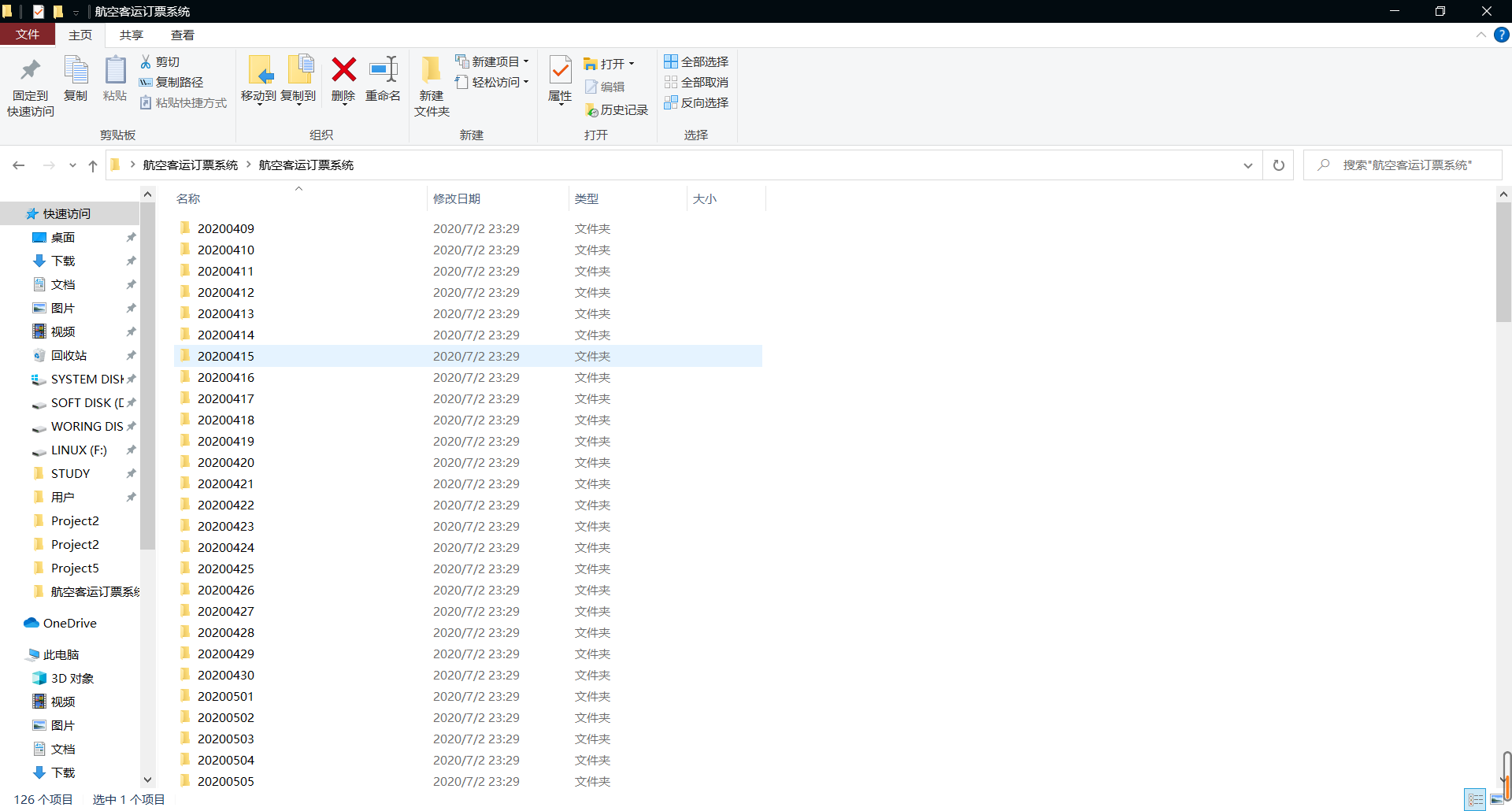


图4.1-21 保存的航班信息形式

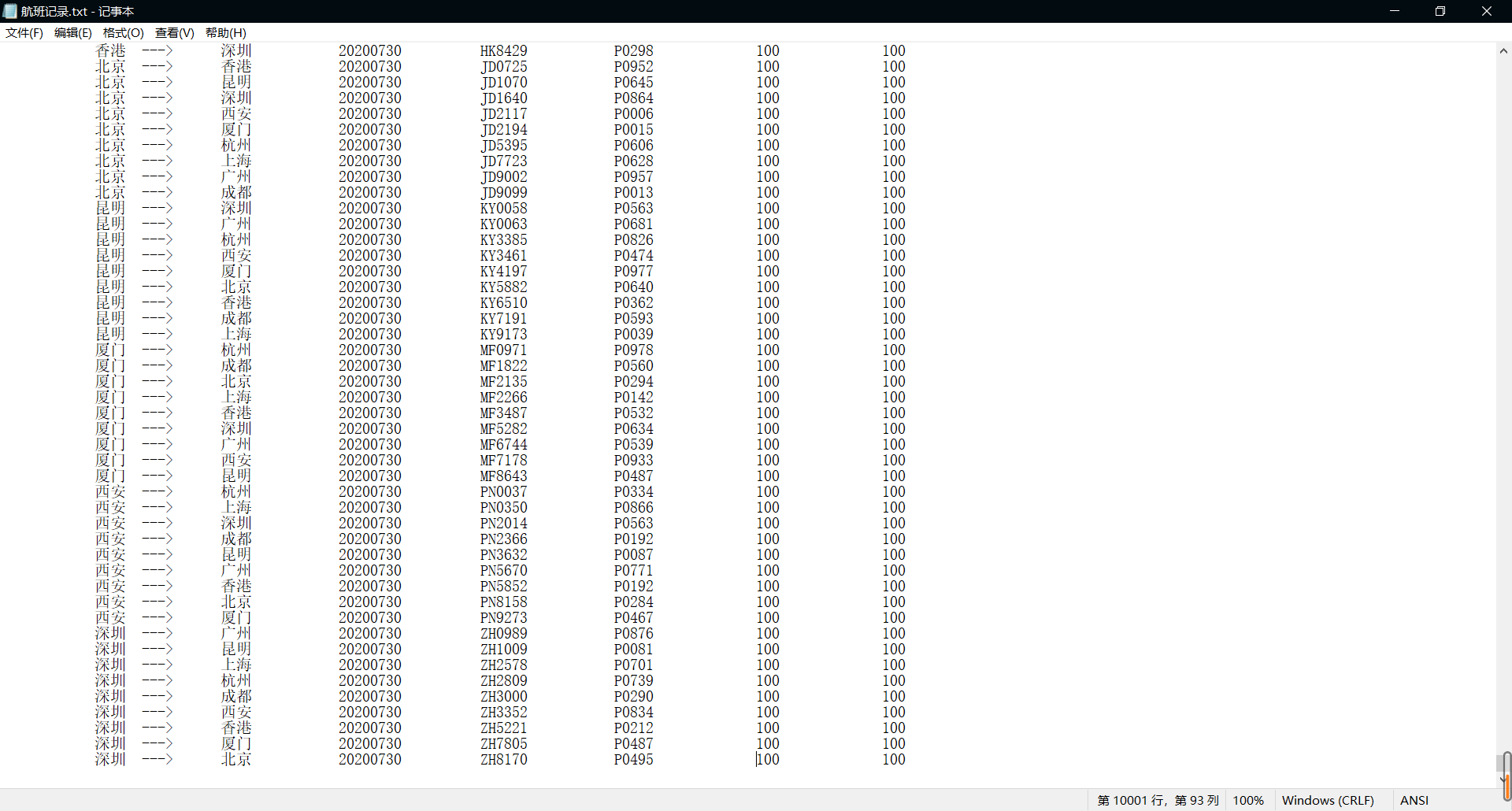


图4.1-22 保存近万条航班信息

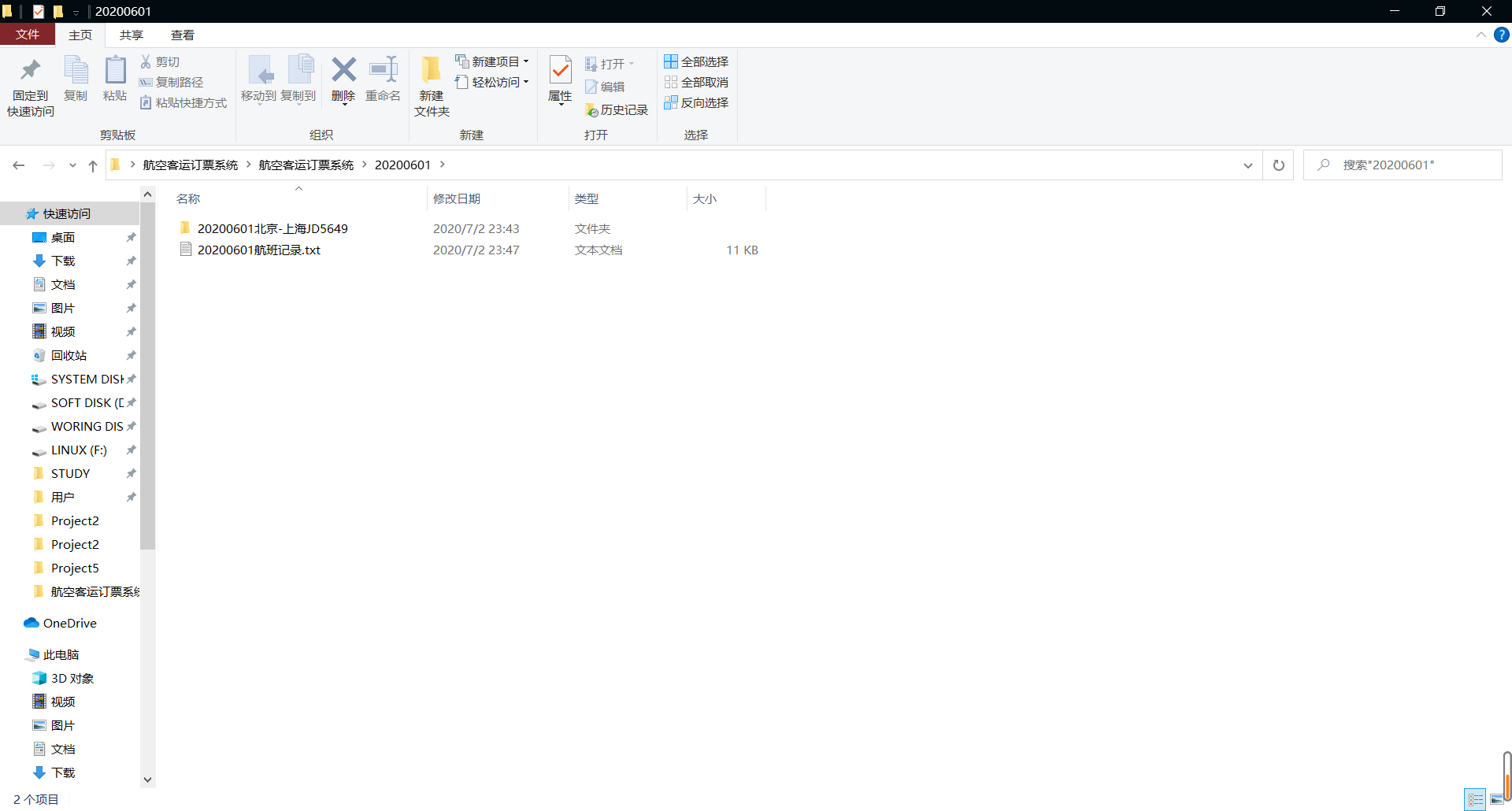


图4.1-23 有购票信息的航班生成的文件夹

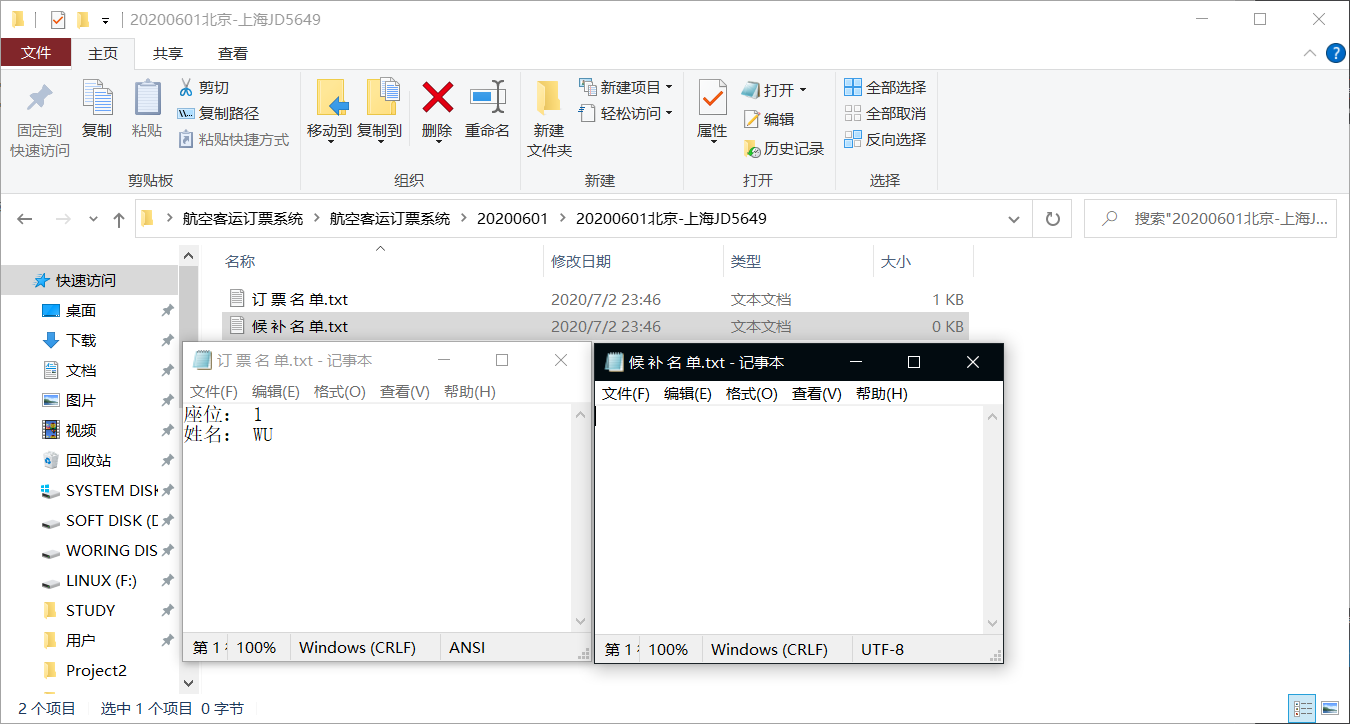


图4.1-24 订票名单和候补名单的保存

## 4.2 测试过程中遇到的问题解决与待解决的问题

1.录入功能显示出来的功能的格式有问题。

解决：反复调试寻找最佳的显示格式。

2.航班录入加载时间过长

解决：选择合适的AVL树结构进行数据的操作，提高算法的运行效率。

3.清除航班记录不完全

解决：算法漏洞，AVL树中的某些航班并未遍历，反复测试找出漏洞并修复。

4.数据保存至文件过于杂乱影响观感

解决：创建文件夹，以航班的日期作为文件夹名，每个文件夹里存放当天的航班记录，以及创建每个航班的文件夹以存放订票名单与候补名单。

5.保存数据不是保存近万条记录

解决：由随机数生成的航班记录仍有重复项，未对重复项做处理，导致航班某些航班消失，进而导致万条记录不全。修改航班记录重复处理机制，使得重复的航班不再录入，解决问题。

待解决的问题：

1. 没有图形化界面，用户操作还是不够友好
2. 保存功能创建文件夹调用dos命令，运行效率低，影响整体效率。

第 5 章 心得体会

头一次面对这样的工程，其工作量是前所未有的，各种问题也接踵而至，对于数据结构设计不合理导致结构推翻重做，对于算法使用不准确导致重复代码过多，对于题目需求不精准导致功能实现不够好，对于不清楚的功能实现束手无策，焦头烂额……

每一个问题看起来都那么的致命。也许是这份课设本身就没有那么好做，也许是做之前觉得没有先考虑清楚，也许是拖时间到最后一周才赶代码，也许是这一学期的网课导致自己基础掌握不扎实，以至于怠惰不前，又也许还是经验不足的问题呢？这份课设引人深思，似乎给我指明了问题所在。

还是需要努力的掌握好每一个知识点，融会贯通，这样才能想到更好的主意。当然还需要更多的练手，这样才能化主意为实际真正有用的东西。最重要的是——个人心态需要调整，不要再轻易轻视或者无视什么东西，存在并发挥效果的东西，一定有其存在的意义，不要因为懒而失去了什么。

上学期结尾我觉得课设很有意思，虽然很累。这学期我依然觉得课设很有意思，虽然更加力不从心。继续努力吧，为了之后能让自己的想法成为现实，路还很长。

第 6 章 参考文献

**[1]**《数据结构》陈越，何钦铭，徐镜春，魏宝刚，杨枨 高等教育出版社