数据结构大作业报告

高铁订票系统

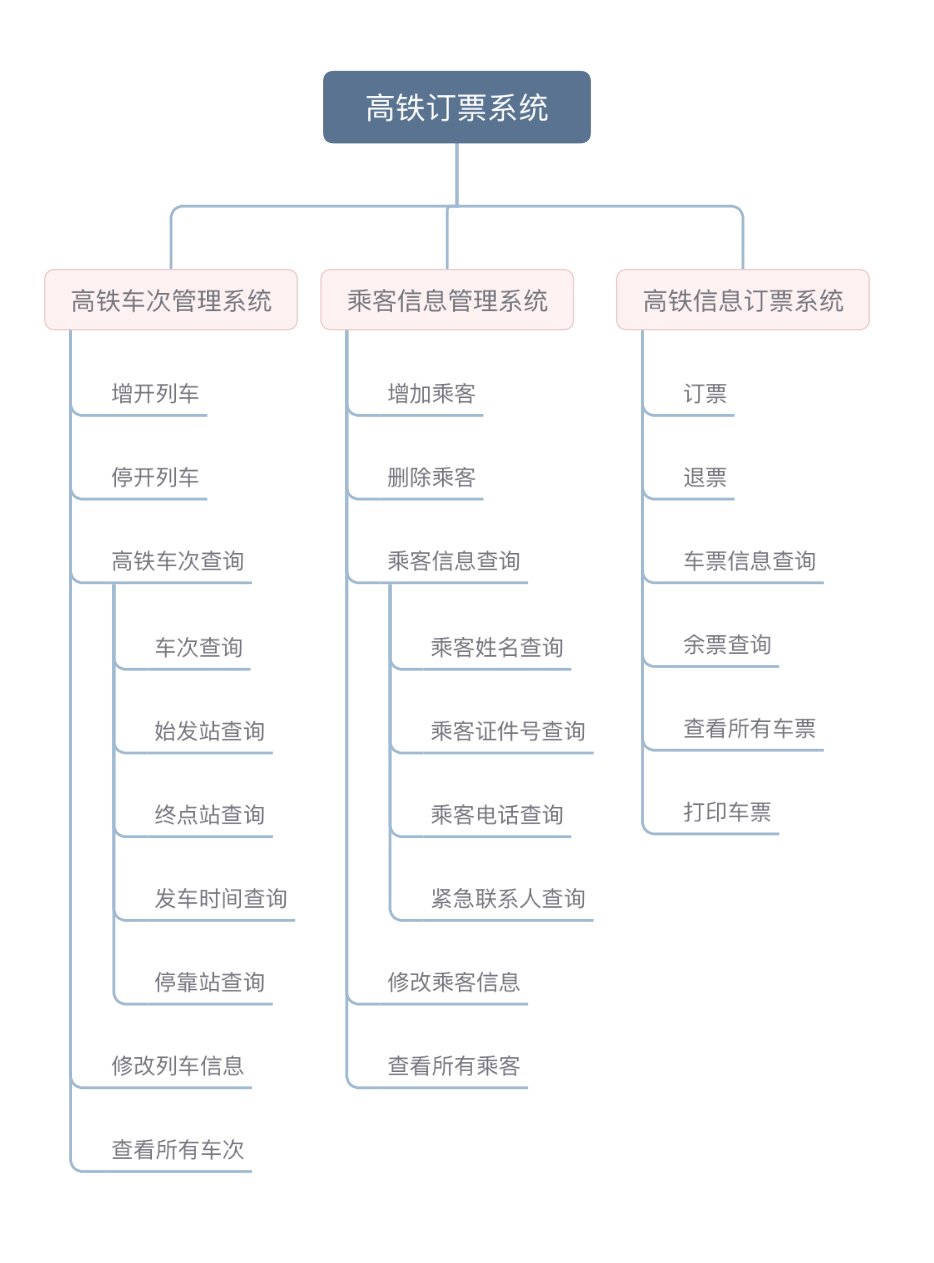
刘家彤 1201003023

2022

## 物理结构

本系统所有数据均采用链式存储结构，创建单链表实现信息的存储

## 程序框架



## 设计思路

程序的设计围绕文件的读取和链表操作，大致思路是程序运行开始就对文件进行读取并把数据存到链表中，用户利用switch case选择系统功能对链表进行操作，在结束操作后将链表全部存回文件中。

在数据存储方面，除直接读取文件创建的train，passenger，ticket三个链表外，另外创建station链表存储停靠站信息、trainseat链表存储列车座位状态、booking链表存储新建订票的临时数据。特别的，在train和trainseat两个链表中进行了链表的嵌套，实现存储多层信息。

在功能实现方面，运用基本的链表创建，删除，添加，修改，配以多层不同链表的遍历实现。

在输出上，使用统一的格式形式，保证程序运行简洁。

在编写程序时，将结构体和函数的声明分别打包在不同的头文件中，方便修改。

## 主要函数

* 1. 文件操作（以车次管理为例）
     1. TrainData\* read\_train\_File();

从train.txt里读取列车信息并返回train链表的头指针

* + 1. void save\_train\_File(TrainData\* L);

将train链表保存在train.txt里

* 1. 链表操作（以车次管理为例）
     1. TrainData\* Insert\_Train(TrainData\* head);

输入一辆新列车的信息并插入train链表的末尾，返回train链表的头指针

* + 1. TrainData\* Revise\_Train(TrainData\* head);

遍历train链表寻找待修改的车次并修改这节点的信息，返回train链表的头指针

* 1. 订票系统
     1. TrainSeat\* crate\_seat(TrainData\* train,TicketData\* head3);

根据train链表创建每辆列车的seat链表，返回seat链表的头指针

* + 1. TrainSeat\* set\_seat(TrainSeat\* head1,TrainData\* head2,TicketData\* head3);

根据train链表和ticket链表补充seat链表中的seatstation链表的状态，返回seat链表的头指针

* + 1. Booking\* seek\_remain\_ticket(TrainSeat\* head,TrainData\* train);

输入时间、起始站、终点站查询余票，给booking链表各元素赋值，返回booking链表头指针

* + 1. Booking\* judge\_ticket(char date[10],Booking\* bhead,TrainSeat\* thead);

根据传入booking链表判断传入车次每个座位状态是否矛盾，并将booking链表中可选座位赋值，返回booking链表头指针

* + 1. void book\_ticket(TrainData\* head1,TicketData\* head2,TrainSeat\* head3,PassengerData\* head4);

订票功能：嵌套余票查询功能，利用booking链表和用户选择的车次更新ticket链表

## 订票程序的算法思路说明

* 1. 文件操作
     1. 读取文件
        1. 创建指向ticket链表的指针p，q并开辟空间，创建文件指针
        2. 循环开辟新空间，读入数据，p的next指针指向q，p指向q，循环至文件指针指到结尾
        3. ticket链表尾节点next指针置空
        4. 返回ticket链表头结点指针
     2. 保存文件
        1. 传入ticket链表，指针指向头节点
        2. 创建booking.txt文件指针，进行覆盖写
        3. 遍历ticket链表，将信息赋回文件
        4. 关闭文件指针，置空
  2. 车票信息查询
     + 1. 通过switch case选择查询条件，指针指向ticket链表头节点，count记为0
       2. 输入查询条件，利用指针进行遍历
       3. if判断，判断条件：指针所指节点对应信息与输入信息是否相同
       4. 满足条件输出该节点所有信息，并将count改为1，case结束
       5. 不满足条件继续遍历，至ticket链表尾节点
       6. if判断，判断条件：count为0，满足条件输出无匹配信息提示
  3. 查看所有车票
     + 1. 指针指向传入的ticket链表
       2. 循环遍历，输出遍历节信息
       3. 遍历至指针指向空为止
  4. 余票查询
     1. 创建seat链表【crate\_seat】
     2. 填充seat链表中的状态【set\_seat】
     3. 将（char）station转为（int）station【set\_code】
     4. 查询余票
        1. 获取系统时间（年、月、日、时、分）
        2. 输入查询条件：乘车时间、起始站、终点站
        3. 将输入的乘车时间拆分为年、月、日并转化为int类型，判断订票时间是否在两个月以内，不满足return出函数
        4. 满足时间条件，查询车次
        5. 遍历传入的train链表至空，遍历train链表中的station链表至空，flag置零
        6. if判断，如果当前station节点的站名与查询的起始站相同，指针后移一位，继续便利；不满足判断条件回到5
        7. if判断，如果当前station节点的站名与查询的终点站相同， flag=tag=1;break出station链表的本次遍历 不满足条件回到5
        8. if判断，条件为flag为1；满足条件创建booking链表新节点并赋值，break出train链表本次遍历；不满足break出train链表本次遍历
        9. if判断，条件tag为0；满足输出无可达列车提示，return出函数；不满足忽略
        10. 将booking链表头指针传入judge\_ticket函数并返回更新后的头指针
     5. 判断空位【judge\_ticket】
     6. 遍历booking链表至空并输出每个节点里的车次和可选座位数
  5. 订票
     + 1. 通过查询余票函数创建booking链表
       2. 四个指针分别指向train链表、ticket链表、seat链表、passenger链表
       3. 创建ticket类型新节点np
       4. 输入所定车次
       5. 遍历booking链表至该节点所含车次与输入车次相同，并将该节点的信息赋值给np
       6. 遍历train链表至同一车次，遍历该节点station链表起始站相同站，记录里程和时间，继续遍历至终点站相同站，记录里程和时间，按照二等座规格计算价格，退出遍历
       7. 输入证件号，遍历passenger链表查找是否有相同证件号，相同将passenger节点的信息赋值给np相关元素
       8. 证件号为新号，转至新增乘客函数【Insert\_Passenger】回到7
       9. 按位生成订单号（char转int）存入np
       10. 将np存入ticket链表结尾并置空
       11. 保存ticket链表至booking.txt
  6. 退票
     + 1. 输入退票的订单号
       2. 遍历传入的ticket链表至该节点订单号与输入订单号相同，输出该节点信息
       3. 获取系统时间
       4. 将节点中的开车时间拆分为年、月、日、时、分并转化为int类型，判断距离当前系统时间是否在2h内，是则需要获取节点中的票价并提示5%手续费
       5. 判断该节点是否为头节点，是进行head2=q->next，否进行p->next=q->next操作
       6. 保存ticket链表至booking.txt文件
  7. 打印车票
     + 1. 输入打印车票的订单号
       2. 遍历传入的ticket至订单号相同的节点，按照格式输出节点中的元素

## 程序难点/亮点

* 1. 在记录列车的停靠站以及座位状态等数据的存储上使用了链表嵌套，在调用数据的时候数据之间的结构比较清晰统一，链表的使用相较于顺序存储结构更自由
  2. 在查询余票时可能会存在多辆车同时满足条件的情况，而每辆车每个座位在不同时刻会有不同的乘客，需要判断每个座位在当前条件下是否满足可乘坐的条件。因此我创建了seat链表来表示每辆车每个座位的状态；把不同车次下相同的起始站和终点站转化为不同的编码，这样只需要进行简单的数字比较就可以知道座位是否可坐，把这些座位排号，可以选择的座位存入booking数组中，系统自动选择数组中第一个元素进行座位的安排
  3. ticket链表信息的录入：在ticket链表中大多数信息是重复的，因此我采用遍历其他链表的方式对新订票的信息进行写入，避免了人工输入的复杂和易出错，在订票时只需要输入证件号，如果是系统中存储过的乘客就可以进行订票；对于新乘客需要信息补录，直接保存在passenger.exe中。其中订单号的保存是一大难点，虽然都是由已有信息组合起来的，但组合起来是一大问题，因此我用按位存储的方法对订单号的每一位单独赋值
  4. 不同函数之间的数据传递：对于函数来说返回值是唯一的，但在订票系统中需要大量的数据在不同的函数中传递使用，因此我创建了Booking结构体，并形成booking链表进行短暂存储
  5. 时间：在我的结构体中时间都是以8位字符串格式进行储存的，而获取系统时间所得都是int类型数字，因此比较起来也相当复杂，所以我是用字符转化拆分的方式，加以多重判断，实现了2h退票扣除手续费以及两个月订票期限的判断

## 运行截屏



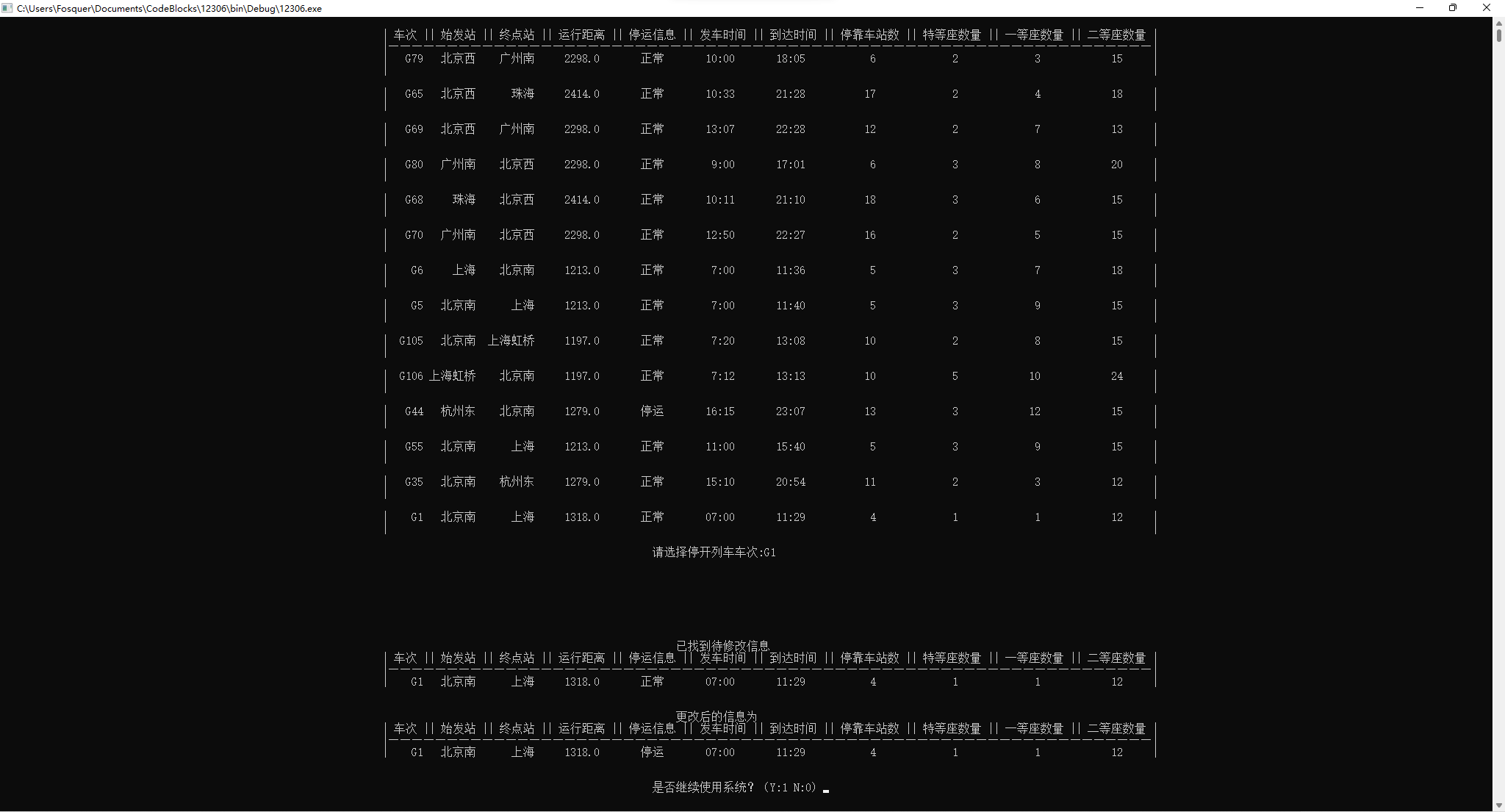
图表 1 高铁管理系统主页面



图表 2 车次管理系统主页面



图表 3 增加列车（乘客系统类似，以下省略）



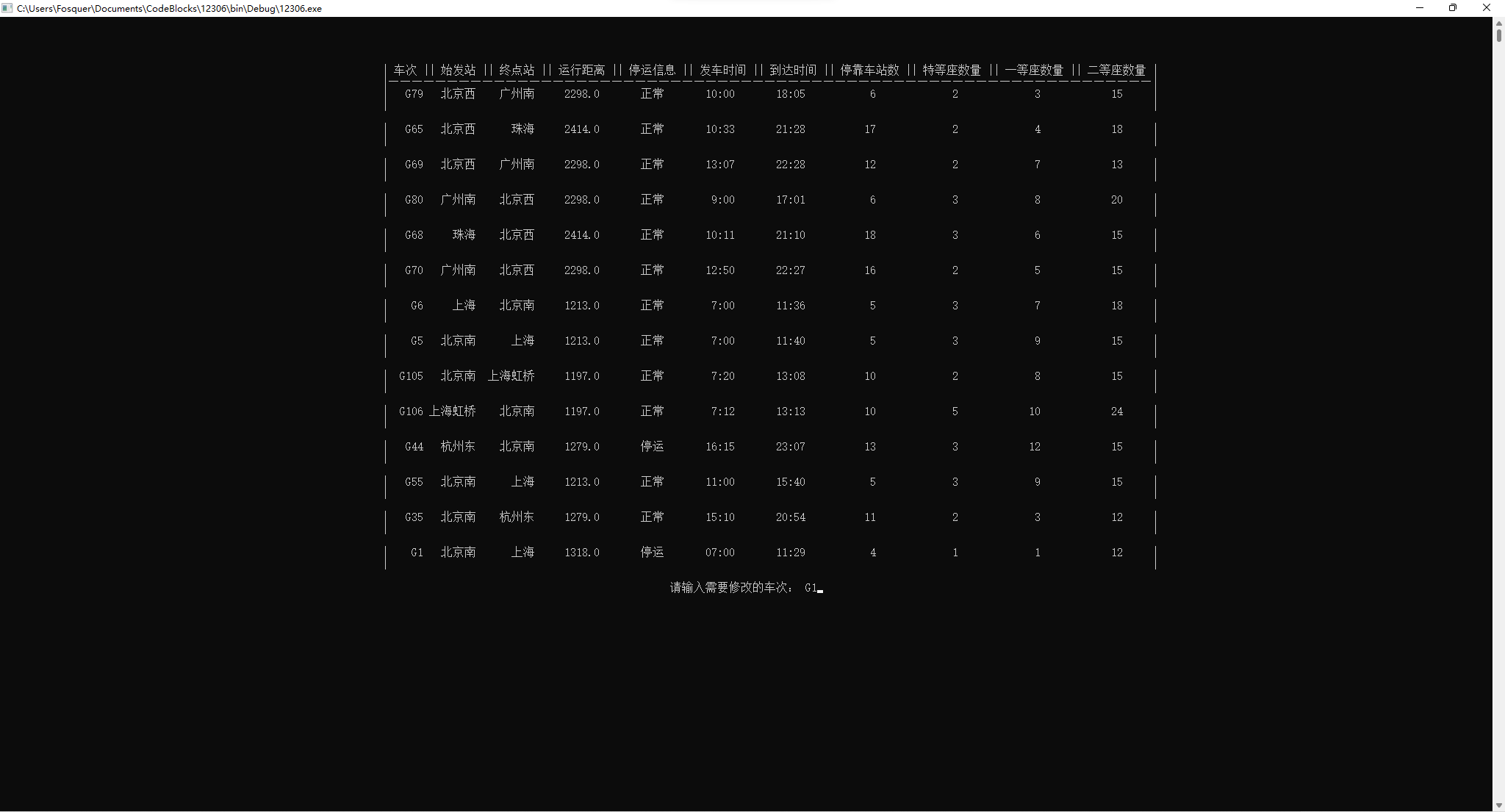
图表 4 停开列车



图表 5 车次查询主页面



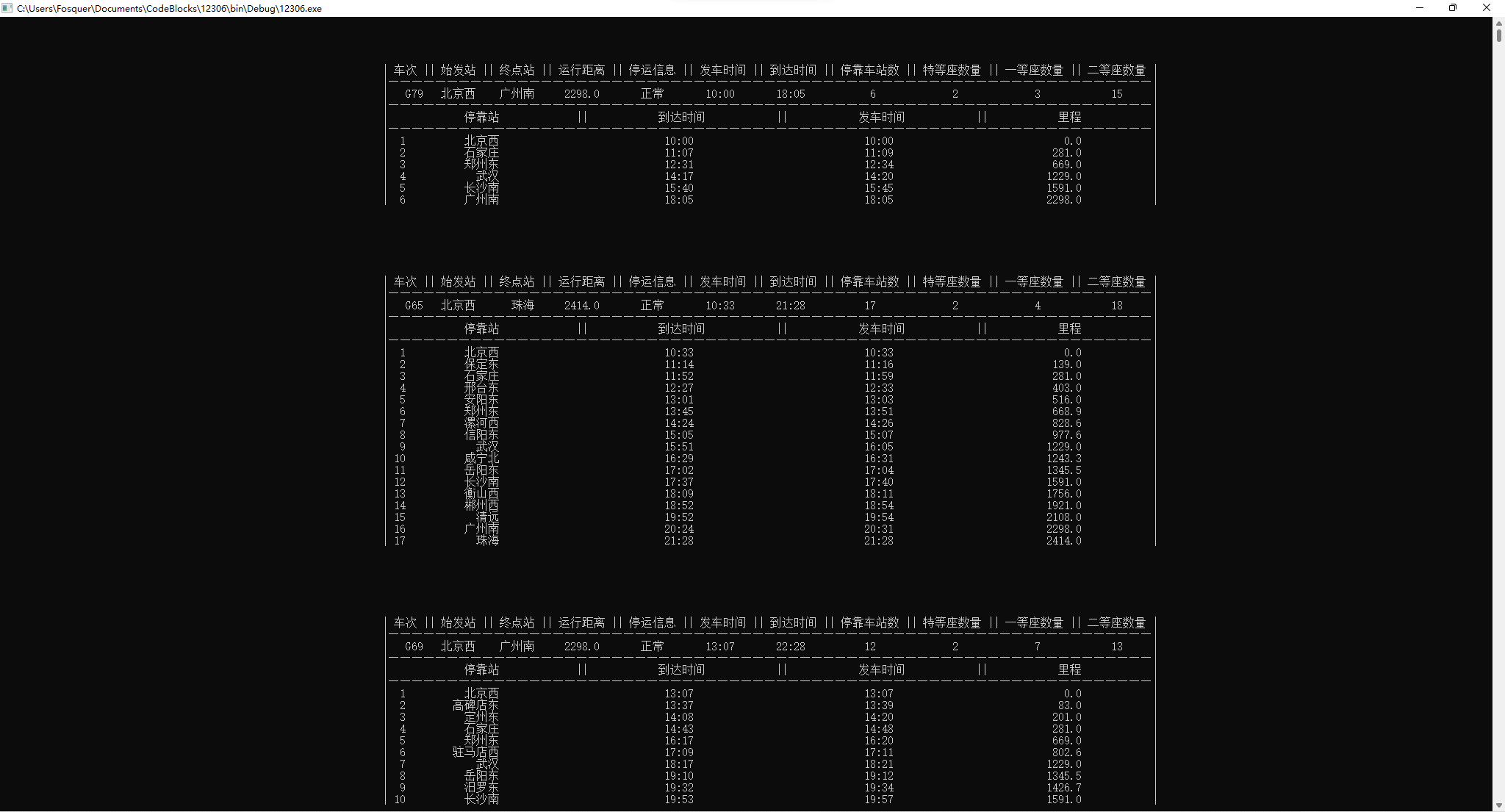
图表 6 按车次查询结果



图表 7 修改列车信息



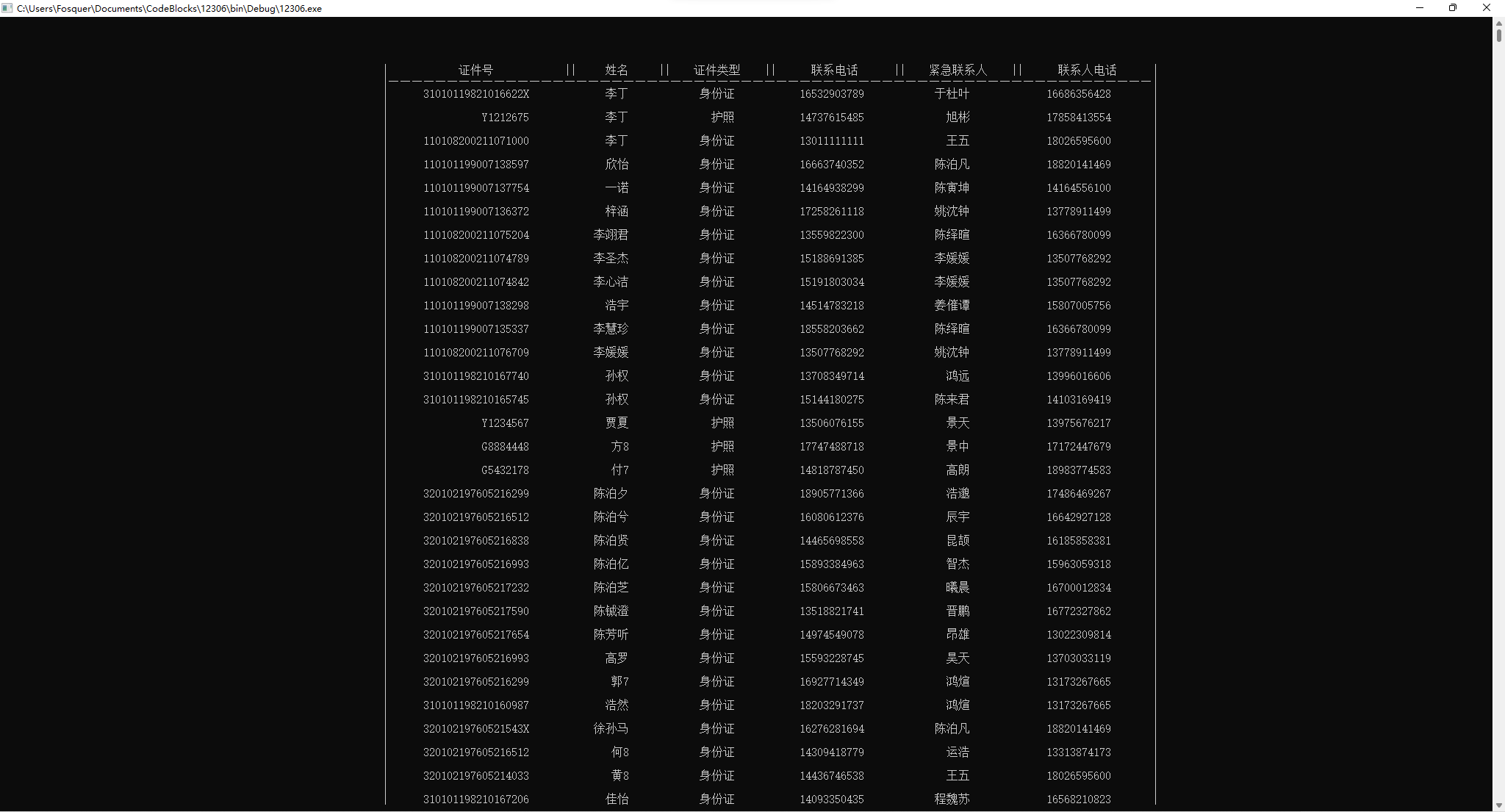
图表 8 修改列车信息结果



图表 9 查看所有列车



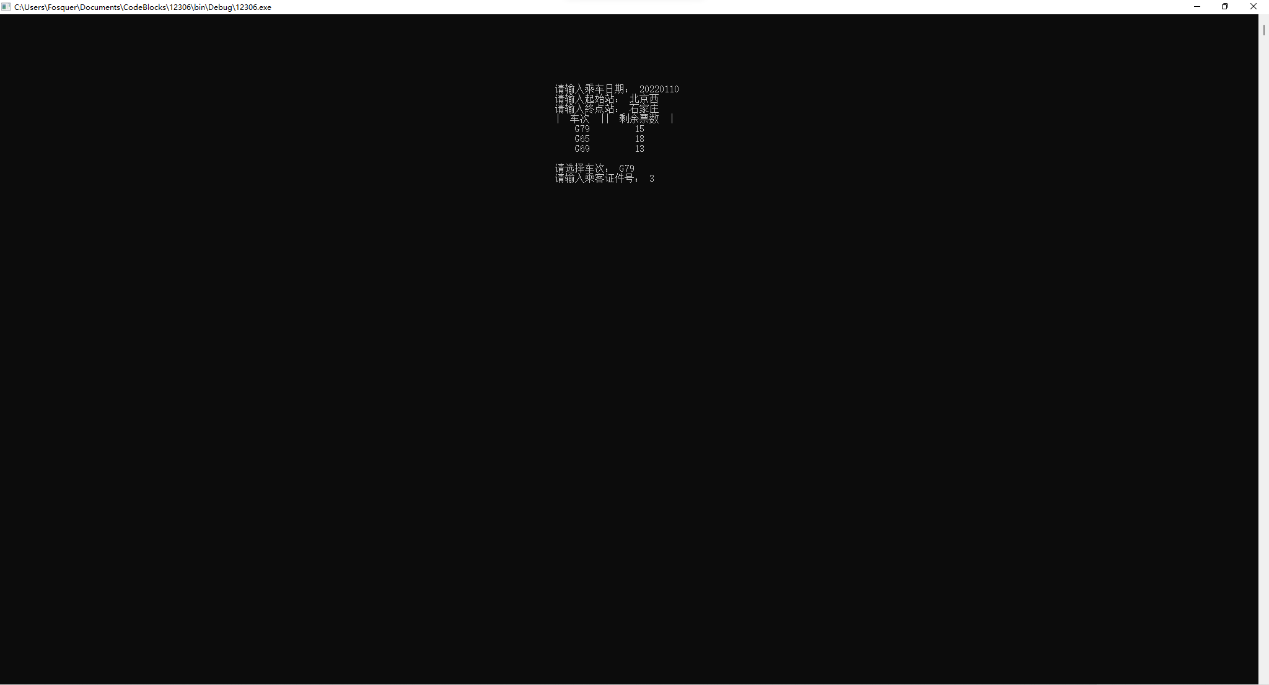
图表 10 乘客信息管理系统



图表 11 查看所有乘客



图表 12 订票系统主页面



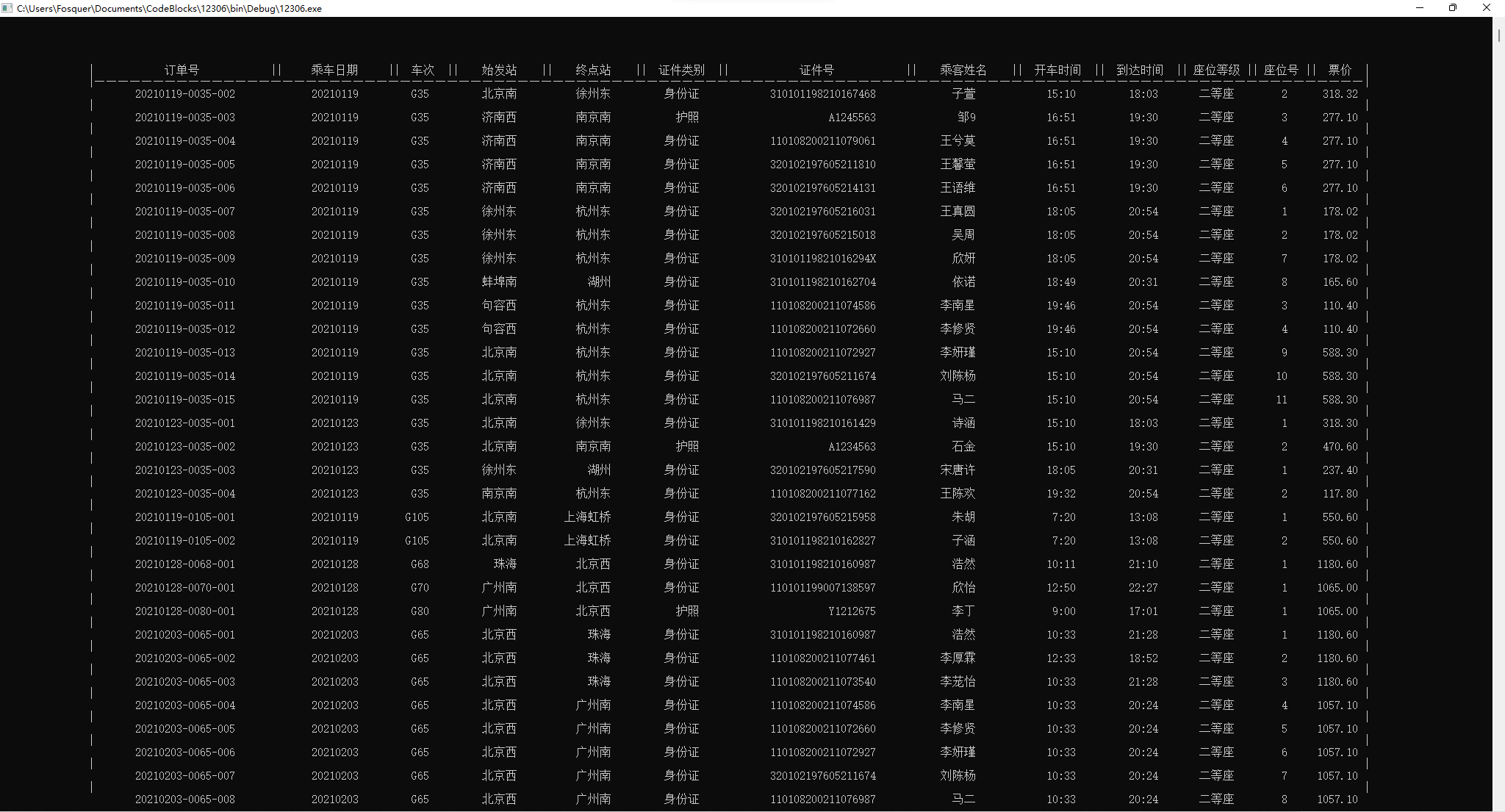
图表 13 订票



图表 14 订票结果



图表 15 退票



图表 16 查看所有订单



图表 17 打印一张车票