航空客运订票系统

杜金鸿,15338039

2017 年 6 月 27 日

目录

1 课题内容

1.1 问题描述

航空客运订票的业务活动包括：查询航线、客票预定和办理退票等。

每条航线所涉及的信息有：起点站名、终点站名、航班号、起降时间、飞机号、飞行周日（星期几）、票价、票价折扣、乘员定额、余票量、已订票的客户名单（包括姓名、证件类型、证件号码、订票数量、航班情况、舱位等级或订单要有编号）以及等候替补的客户名单（包括姓名、所需票量）。

1.2 功能要求

1. 录入：可以录入航班情况（数据可以存储在一个数据文件中，数据结构、具体数据自定。

2. 对飞机航班信息进行排序和查询：可按航班的航班号、起点站、到达站、起飞时间以及到达时间等信息查询某个航线的情况。(可以输入一个或多个信息进行查询。设计中，要求采用基数排序方法对一组具有结构特征的飞机航班号进行排序，利用二分查找的方法对排好序的航班记录按照航班号实现快速查找，而其他次关键字的查找则采用最简单的顺序查找方法进行。当然也可以使用其它的排序方法）假设每个航班记录包括 8 项，分别是：航班号，起点站，到达战，班期，起飞时间，到达时间，飞机型号以及票价等; 其航班号表示如下，

K0	K1	K2	K3	K4	K5
C	Z	3	8	6	9

   k0 和 k1 的输入值是航空公司的名称（CA 国航，CZ 南航），用 2 个大写字母表示，后 4 位为航班编号，这种航班号关键字可未成 2 段，即字母和数字。其余 7 项输入内容因不涉及本设计的核心，因此除了票价为数值型外，均定义为字符型即可。

3. 订票：可以订票，如果该航班已经无票，可以提供相关可选择航班（订票情况可以存在一个数据文件中，结构自己设定。相关可选择航班可包括进行中转的航班，比如输入西安－杭州，相关可选择航班包括西安－重庆－杭州)。

4. 退票：可退票，退票后修改相关数据文件；(若有等候替补的客户名单，则要及时通知这些客户并修改已订票客户名单)。

5. 修改航班信息：当航班信息改变可以修改航班数据文件。

6. 当客户不清楚航班的信息情况时，能通过输入起点站和终点站，输出费用最小的航线。(若无直达航线，可输出中转航线。若有直达航线，但费用并不是最小，则同时输出费用最小的直达航线（在直达航线中费用最小的航线）和费用最小的中转航线（在所有的航线中费用最小的航线)。

2 问题分析

2.1 存储结构

2.1.1 航班信息

由于航班记录涉及两个城市之间的次序关系，即每一个航班都起始于某一城市并到达另一城市，每两个城市之间可能有多个航班。因此考虑以非线性结构的图来存储航班信息，若以各城市为结点，则每个航班代表了两个结点的一条有向边。

由于每两个城市之间可能存在多个航班，即该图为有重边的非简单图，因此不能采用邻接矩阵存储。若用邻接表存储，则不利于对航班信息的索引（从某个城市出发的航班往往众多，而对这其中的某个航班进行查找需要时间复杂度 $O(n)$），而查找操作又是问题设计的主要问题，因此邻接表并不完全适合。

为了解决上述问题，本文采用邻接矩阵、长度可变的向量相结合的方式存储航班信息，并使用索引表提高索引效率。

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \qquad a_{ij}:$$

0	航班1
1	航班2
⋮	⋮
m	航班m+1

邻接矩阵描述了各城市之间的航班关系，其每个矩阵元素 a_{ij} 是一个向量，存储了从城市 i 到城市 j 的所有航班。因此每一个航班可用三元组表 $\{(m,n,l)\}$ 的形式索引。

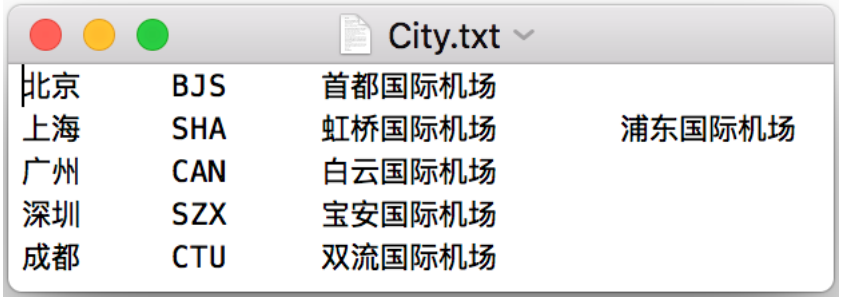
因为每个存储航班信息的单元相对较大，若对航班信息直接进行排序操作，需要对存储单元各数据项进行复制移动操作，显然时间成本太高。因此本文将航班信息以静态数据形式存储，使用索引表表示其次序关系，对航班信息进行的修改操作通过索引表索引对源信息进行修改，对航班信息进行的排序操作只在索引表层面进行。

2.1.2 客户信息

由于客户信息是与特定航班联系在一起的，因此本文将相应的客户信息分别以 *Booked*（已预订客户类）、*Alternative*（候补客户类）的数据形式保存在每个邻接矩阵元素中。订票、退票操作直接针对某一特定的航班进行，更加合理、方便。在 *Booked*、*Alternative* 类中，每条客户信息存储在长度可变的向量中，既方便索引，又避免了普通线性表存储空间固定的缺陷。

2.2 文件读写

由于在航班信息中，各城市及其机场的信息应该不经常变动，因此我们将这部分信息保存为 "*City.txt*" 文件。本文选取了国内飞机流量前五大城市及其机场，每个城市可以有多个机场信息，各个城市的信息之间以换行符 '\n' 分割，每个城市的信息包括城市名称、英文代码、机场名称，相应信息之间以制表符 '\t' 分割。详细内容如下：



在航班记录中，以城市英文代码以及数字来记录其起始、到达的机场，例如以 “*BJS0*” 代表北京的首都国际机场。航班记录存储在 “*plane.txt*” 文件中，详细内容如下：

plane.txt

```
2017    05      29      星期一
CA0001  BJS0    SHA1    一      06:25   08:40   A250    100%    经济舱  530     2       2       公务舱  1240    0       600
小明    身份证  12345678910     1       准点    经济舱  CA000120170529-1,小红   身份证  12345678911     1       准点    经济舱  CA000120170529-2,
小李    身份证  22222222222     1       经济舱,#
MU5341  SHA1    CAN0    一      09:25   12:40   A250    100%    经济舱  530     2       2       公务舱  1240    0       600
NA
NA#/
2017    05      30      星期二
CZ0001  BJS0    SHA1    二三四五 06:25  08:40   A320    100%    经济舱  530     0       160     公务舱  1240    0       60
John    护照    EE110   1       准点    经济舱  CZ000120170530-1,
NA#/
2017    05      31      星期三
CZ0001  BJS0    SHA1    二三四五 06:25  08:40   A320    100%    经济舱  530     0       160     公务舱  1240    0       60
NA
NA#
ZH0001  BJS0    SHA1    三      06:25   08:40   A320    100%    经济舱  530     0       160     公务舱  1240    0       60
Mary    护照    DA110   1       准点    经济舱  ZH000120170531-1,
NA#
CZ0002  SHA1    BJS0    三四五六 10:25  12:40   A320    100%    经济舱  500     0       160     公务舱  1040    0       60
NA
NA#/
2017    06      01      星期四
CZ0001  BJS0    SHA1    二三四五 06:25  08:40   A320    100%    经济舱  530     0       160     公务舱  1240    0       60
John    护照    EE110   1       准点    经济舱  CZ000120170601-1,
NA#
CZ0002  SHA1    BJS0    三四五六 10:25  12:40   A320    100%    经济舱  500     0       160     公务舱  1040    0       60
NA
NA#/
2017    06      02      星期五
CZ0001  BJS0    SHA1    二三四五 06:25  08:40   A320    100%    经济舱  530     0       160     公务舱  1240    0       60
John    护照    EE110   1       准点    经济舱  CZ000120170602-1,
NA#
CZ0002  SHA1    BJS0    三四五六 10:25  12:40   A320    100%    经济舱  500     0       160     公务舱  1040    0       60
John    护照    EE110   1       准点    经济舱  CZ000220170602-1,
NA#
ZH0002  SHA1    BJS0    五      06:25   08:40   A320    100%    经济舱  530     1       160     公务舱  1240    0       60      头等舱  2000    0
        5
小明    身份证  333333333       1       准点    经济舱  ZH000220170602-1,
NA#/
2017    06      03      星期六
CZ0002  SHA1    BJS0    三四五六 10:25  12:40   A320    100%    经济舱  500     0       160     公务舱  1040    0       60
NA
NA#/
2017    06      04      星期日
/
```

可以看到，航班记录以日期分割，每一天的航班记录以分隔符 ‘/’ 为结束字符。每一天的各个航班记录之间以分隔符 ‘#’ 为结束字符。每一个航班记录包括三行，第一行为航班信息，包括：航班编号、起始城市及机场、到达城市及机场、班期、起飞时间、到达时间、机型、折扣、舱位信息，其中舱位信息包括舱位类型、价格、已购人数、最多人数，且其类别数不固定；第二行为已预订客户信息，包括：姓名、证件类型、证件号码、购买票数、航班状态、舱位类型、机票号码；第三行为候补客户信息，包括：姓名、证件类型、证件号码、购买票数、舱位类型。若无已预订客户或候补客户，则以符号 “*NA*” 代替。

以日期区分航班记录，在后续推荐相关航班的时候将简化检索步骤。

对该文件的读取以及对文件的写入、信息的保存，利用了 C++ 的 *fstream* 类；对字符串的存储，利用了

C++ 内建的 *string* 类；对字符串和整型数据的转换利用了 C++ 的 *sstream* 类、*cstdlib* 类。

2.3 排序与查找

2.3.1 基数排序

由于对航班记录的查询更多的是基于航班编号，因此对其按照航班编号进行基数排序。假设航班记录的航班编号序列为 $\{p_0, p_1, \cdots, p_{n-1}\}$，每个航班记录 p_i 的航班编号有 6 个关键码 $c_i^0, c_i^1, \cdots, c_i^5$，包括 2 个表示航班公司的英文大写字母以及 4 个航班数字编号。假设 6 个关键码 $c^0, \cdots, c^5$ 的取值数分别为 $d_0, \cdots, d_5$，则 $d_0 = d_1 = 26, d_2 = \cdots = d_5 = 10$。

Algorithm 1 *LSD* 基数排序算法

```
j = 5;
while j ⩾ 0 do
    初始化 d_j 个临时存储单元;
    i = 0;
    for i < n do
        按键值 c_i^j 分配至临时存储单元;
        i++;
    end for
    依次收集各临时存储单元中的数据;
    j--;
end while
```

采用最次位优先 *LSD*(*least significant digit first*) 的策略对其进行基数排序，由于本文采用向量存储的三元组表存储航班记录索引，其长度可动态变化，因此基数排序使用的临时存储单元也利用同样的三元组表向量。分配与收集操作只需逐一添加向量的元素。最终可以得到升序排列的航班编号记录。

2.3.2 区间的二分查找

Algorithm 2 二分查找算法

```
设置初始查找区间: low = 0, high = n − 1;
while low <= high do
    mid = (low + high) / 2;
    if q < p_mid then
        high = mid − 1;
    else if q > p_mid then
        low = mid + 1;
    else if q == p_mid then
        return mid;
    end if
end while
```

对基数排序后的航班记录按照航班编号使用二分查找的方式进行查找。

设待查找的航班编号为 q，普通的二分查找算法如 *Algorithm* 2 所示，若查找成功，它仅会返回查找到的某一个记录。对于航班编号记录，往往同一个航班编号会对应多个时间不同的航班记录，因此若直接使用普通二分查找算法，无法保证查找到所有满足条件的记录。为此，本文对二分查找算法进行改进，使它能查找到满足条件的区间，称为二分查找区间算法。

由于航班编号记录已按照升序排列，因此对相应区间的查找，等价于对区间两个端点的查找。只需稍微修改普通的二分算法即可。

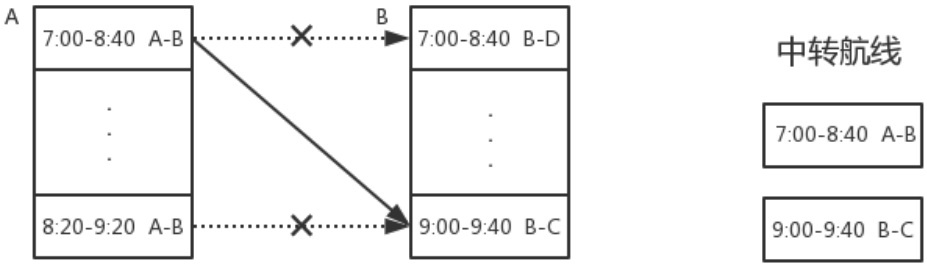
Algorithm 3 二分查找右端点算法

设置初始查找区间：$low = 0, high = n - 1$;
while $low < high$ do
  $mid = \dfrac{low + high}{2}$;
  if $q < p_{mid}$ then
    $high = mid$;
  else if $q \geqslant p_{mid}$ then
    $low = mid + 1$;
  end if
end while
if $low == n$ *or* $low == 0$ then
  查找失败;
else
  区间右端端点位置为 $low - 1$;
end if

同理可对于左端点进行二分查找。

2.4 航班推荐

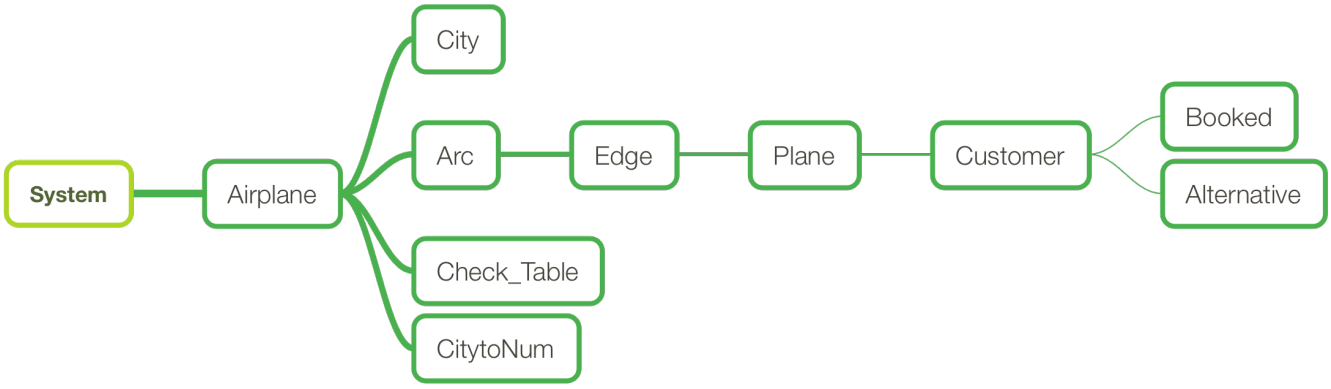
若所查询得航班已无票，或者客户仅输入起始城市、到达城市进行查询，则检索中转航班，并推荐费用最小者。由于航空中转点一般仅为一站，故只需对从该起始城市出发的航线中检索是否有到达目标城市的航线。在检索过程中，一方面需要判断航班的可达性，另一方面仍需判断航班的时间先后顺序。由实际情况，国内中转航班一般当天内到达，因此本文仅考虑同一天内的两个航班。



3 设计与编码

3.1 类的关系与定义

3.1.1 类的关系



各种类的包含关系如图所示，类的定义保存在头文件 “*airplane.hpp*” 中。

3.1.2 *System* 类

System 类负责航空订票系统的初始化工作，提供面向用户的函数接口。对航班信息的查询、排序等操作，均由该类向其类成员进行操作。

```
class System{
public:
    System();                          // 默认构造函数
    ~System();                         // 析构函数
private:
    /*---------------------------------- 初始化 ----------------------------------*/
    void Init();                       // 系统初始化
    void Input();                      // 录入航班情况
    /*---------------------------------- 普通用户 ----------------------------------*/
    void Sort();                       // 对航班进行排序
    void Check();                      // 查询航线
    void Search(string ID,string year,string month,string day,string start,string arrive,string
    ↪  start_city,string arrive_city);// 查找航班
    int Book(int i, int j, int k);     // 订票
    /*---------------------------------- 管理员 ----------------------------------*/
    void AddPlane();                   // 添加航班
    void Change();                     // 修改航班信息
    void Save();                       // 保存航班信息
    void AmdinLogin();                 // 管理员登录
```

```
    void AmdinLogout();          // 管理员注销
    void GetKey();               // 读取密码文件
    void ChangeKey();            // 修改管理员密码
    void Log();                  // 输出日志信息
    Airplane* airplane;          // Airplane类
    int prior;                   // 权限级别
    string key;                  // 密码
};
```

3.1.3 *Airplane* 类

Airplane 类是航空客运订票系统的主体部分。

```
class Airplane{
public:
    Airplane(int num_city=5);                                    // 默认构造函数
    ~Airplane();                                                 // 析构函数
    /*--------------------------------------  初始化  --------------------------------------*/
    void Init();                                                 // 航班初始化
    void addCity(const string &s, int i);                        // 添加城市结点
    void addArc(string &year,string &month,string &day,string &date,const string &s,int city0,int city1,int
    ↪  plat0,int plat1);                                         // 添加有向边
    void addPlane(string &year,string &month,string &day,string &date,const string &s);// 添加航班记录
    /*--------------------------------------排序与查找--------------------------------------*/
    void Sort();                                                 // 基数排序
    void Distribute(Check_Table q[],int m,int j);                // 分配
    void Collect(Check_Table q[],int m);                         // 收集
    void Search(Check_Table &c,string ID,string year,string month,string day,string start,string
    ↪  arrive,string start_city,string arrive_city);             // 查找航班
    void Transfer(Check_Table &c,string year,string month,string day,string start_city,string arrive_city);
    ↪                                                            // 筛选航班
    int SearchTransfer(Check_Table c[],string start_city,string arrive_city);// 查找中转航班
    /*--------------------------------------订票与退票--------------------------------------*/
    int addBooked(int m,int n,int l,string name,string kindID,string ID,string book_num,string state,string
    ↪  kind,string ticket);                                      // 添加订票客户
    int addAlternative(int m,int n,int l,string name,string kindID,string ID,string book_num,string kind);
    ↪                                                            // 添加候补客户
    void showBooked(int m,int n,int l);                          // 输出某航班已订票客户
    void showAlternative(int m,int n,int l);                     // 输出某航班候补客户
    string deleteBooked(int m,int n,int l,string name,string ID);// 退票
```

```
    void informAlternative(int m,int n,int l,string kind);        // 通知候补客户
    /*-------------------------------------修改与保存-------------------------------------*/
    void Change(int m,int n,int l);                               // 修改航班信息
    void DeletePlane(int m,int n,int l);                          // 删除航班
    void Save();                                                  // 保存航班信息
    void show(int i,int j,int k);                                 // 输出航班基本信息
    CitytoNum citytonum;                                          // 城市名称-城市序号索引表
  private:
    City* city;                                                   // 城市结点
    Arc* arc;                                                     // 有向边
    int num_city;                                                 // 城市个数
    string cityname,planename;                                    // 存储文件名称
    Check_Table check_table,check_table_time;                     // 索引表
  };
```

3.1.4 *City* 类

City 类存储城市结点信息。

```
  class City{                                 // 城市结点类
  public:
      City();                                 // 默认构造函数
      ~City();                                // 析构函数
      void addAirport(string airport);        // 添加机场
      string name,ID;                         // 城市名称、城市英文代码
      string* airport;                        // 机场
      int num_airport,max_num;                // 机场个数、最大容量
  };
```

3.1.5 *Check_Table* 类

Check_Table 类存储各个航班号的索引地址及日期信息。

```
  class Check_Table{                                      // 航班索引表
  public:
      Check_Table();                                      // 默认构造函数
      ~Check_Table();                                     // 析构函数
      Check_Table& operator=(Check_Table &c);             // 重载 = 运算符
      void addPlane(int m,int n,int o,string year="",string month="",string day="",string date="");
                                                          // 添加航班索引
```

```
    void Clear();                                   // 清除索引表
    int** plane;                                    // 三元组
    string** time;                                  // 日期四元组
    int num_plane,max_num;                          // 航班数、最大容量
};
```

3.1.6 *CitytoNum* 类

CitytoNum 类是将城市名称或英文代码转换为数字序号的词典。重载 [ ] 运算符之后，可以通过城市中文名称或者城市英文代码来索引得到该城市在 *Airplane* 类中的序号，例如：*CitytoNum*[“ 北京 ”] = 0。

```
class CitytoNum{                        // 城市名称-城市序号索引表类
public:
    CitytoNum();                        // 默认构造函数
    ~CitytoNum();                       // 析构函数
    int operator[](string cityname);    // 重载[ ]运算符
    void addcity(City city[],int num);  // 添加城市索引
    string* name,*ID;                   // 城市名称、城市英文代码
    int num;                            // 城市个数
};
```

3.1.7 *Arc* 类

Arc 类是城市结点形成的邻接矩阵。

```
class Arc{                              // 邻接矩阵类
public:
    Arc(){};                            // 默认构造函数
    ~Arc();                             // 析构函数
    void Init(int size);                // 初始化
    int size;                           // 大小
    Edge* arc;                          // 有向边
};
```

3.1.8 *Edge* 类

Edge 类是城市结点之间的有向边。每一个 *Edge*，即每一边存在着重边，即多个航班 *Plane*。

```
class Edge{                             // 有向边类
public:
```

```
    Edge();                         // 默认构造函数
    ~Edge();                        // 析构函数
    void Init();                    // 初始化
    void addPlane(Plane &p);        // 添加航班
    Plane* plane;                   // 航班
    int num_plane,max_num;          // 航班数量、最大容量
};
```

3.1.9 *Plane* 类

Plane 类记录航班的详细信息，它包括了客户类成员 *Customer*。

```
class Plane{                                                    // 航班结点类
public:
    Plane(){kind=nullptr;price=num_now=num_max=nullptr;};   // 默认构造函数
    Plane(string name,string year,string month,string day,string date,string schedule,string start,string
    ↪ arrive,string shape,string discount,int num,string* kind,int* price,int* num_now,int* num_max,int
    ↪ plat_start,int plat_arrive);
                                                                // 构造函数
    ~Plane();                                                   // 析构函数
    Plane& operator=(Plane &p);                                 // 重载 = 运算符
    string Save(string start_city, string arrive_city);         // 保存该航班
    int Minprice();                                             // 返回最低票价
    string name;                                                // 航班编号
    string year,month,day,date;                                 // 日期
    string start;                                               // 出发时间
    string arrive;                                              // 到达时间
    int plat_start,plat_arrive;                                 // 出发机场、到达机场序号
    string schedule;                                            // 班期
    string shape,discount;                                      // 机型、折扣
    int num;                                                    // 舱位类型数
    string* kind;                                               // 舱位类型名称
    int* price;                                                 // 价格
    int* num_now;                                               // 已订乘客数
    int* num_max;                                               // 最大乘客数
    Customer customer;                                          // 客户类
};
```

3.1.10 *Customer* 类

Customer 类包含了对应航班的已订票客户以及候补订票客户信息。

```
class Customer{                                  // 客户类
public:
    Customer();                                  // 默认构造函数
    ~Customer();                                 // 析构函数
    Customer& operator=(Customer &p);            // 重载 = 运算符
    void addBooked(string name,string kindID,string ID,string book_num,string state,string kind,string
    ↪  ticket);                                  // 添加已订票客户
    void addAlternative(string name,string kindID,string ID,string book_num,string kind);// 添加替补客户
    void showBooked();                           // 输出已订票客户
    void showAlternative();                      // 输出替补客户
    void informAlternative(string kind);         // 通知替补客户
    string deleteBooked(int de);                 // 退票
    string Save();                               // 保存客户信息
    Booked* booked;                              // 已订票客户
    Alternative* alternative;                    // 候补客户
    int num_booked,num_alternative;              // 已订票客户数、候补客户数
    int max_booked,max_alternative;              // 最大已订票客户数、最大候补客户数
};
```

3.1.11 *Booked* 类

Booked 类包含对应航班的已订票客户信息。

```
class Booked{                                    // 已订票客户类
public:
    Booked():name(" "),kindID(" "),ID(" "),num(" "),state(" "),kind(" "),ticket(" "){};
                                                 // 默认构造函数
    Booked(string name,string kindID,string ID,string book_num,string state,string kind,string ticket);
                                                 // 构造函数
    ~Booked(){};                                 // 析构函数
    Booked& operator=(Booked &b);                // 重载 = 运算符
    void show();                                 // 输出已订票客户信息
    string Save();                               // 保存
    string name,kindID,ID,num,state,kind,ticket; // 已订票客户信息
};
```

3.1.12 *Alternative* 类

Alternative 类包含对应航班的候补客户信息。

```
class Alternative{                                  // 候补客户类
public:
    Alternative():name(" "),kindID(" "),ID(" "),num(" "),kind(" "){};
                                                    // 默认构造函数
    Alternative(string name,string kindID,string ID,string book_num,string kind);
                                                    // 构造函数
    ~Alternative(){};                               // 析构函数
    Alternative& operator=(Alternative &a);         // 重载 = 运算符
    void show();                                    // 输出候补客户信息
    string inform();                                // 输出候补客户信息
    string Save();                                  // 保存
    string name,kindID,ID,num,kind;                 // 候补客户信息
};
```

3.2 类的主要实现

由于文章篇幅有限，本文仅对实验的重要部分的代码实现进行分析，所涉及到的所有代码均在附录给出。

3.2.1 订票系统

定义 *System* 类变量，并调用 *Init*() 方法后，该对象首先会读取数据文件，将存储在 "*City.txt*"，"*plane.txt*" 中的非结构化数据提取、转换、存储到已定义的类数据成员中；然后系统会显示操作提示菜单。

根据用户的输入，系统会作出相应的反馈，普通用户仅能进行航班搜索 *Search*(), 预订航班 *Book*()，退票操作，而管理员可以修改航班信息 *Change*()，保存航班信息 *Save*()，查看已订票客户信息、候补客户信息等。

Airplane 类是航空客运订票系统的主体部分。

Airplane 的初始化部分，通过分隔符切分读取保存在文本文件中的数据信息，并保存到相应的数据成员中。读取文件的过程利用了 *stringstream* 类，通过格式化输入、输出获取字符串数据。某些数据需要转换为整型数据，则使用 *atoi*(∗*char*) 函数进行转换。保存文件使用同样的分隔符分隔方式，以便重复读写数据。

3.2.2 基数排序

以下是 *Airplane* 的基数排序部分，由于航班号有两个字段，前两个字符为大写字母，后四个字符为阿拉伯数字，因此进行基数排序应分为两个阶段分别进行。基数排序从航班号末位开始，逐渐前移。

在分配阶段，由于字母位的类数为 26、数字位的类数为 10，因此对这两个字段分别定义临时存储结构 *Check_Table* 向量。若为字母位，则获取其字符的 *ASCLL* 码值后，减去 65，得到应分配到的序号；若为数字位，则直接将字符转化为数字用来索引。

在收集阶段，依次收集预定义的 *Check_Table* 向量，并赋值给类的 *check_table* 数据成员即可。

```
1         void Airplane::Sort(){
2         check_table = check_table_time;
3         for (int j=5; j>1; j--) {      // 数字位基数排序
4             Check_Table q[10];
5             Distribute(q,10,j);
6             Collect(q,10);
7         }
8         for (int j=1; j>=0; j--) {     // 字母位基数排序
9             Check_Table q[26];
10            Distribute(q,26,j);
11            Collect(q,26);
12        }
13    }
14
15    void Airplane::Distribute(Check_Table q[], int m, int j){
16        int k;
17        for (int i=0; i<check_table.num_plane; i++) {
18            char
    ↪ c=arc[check_table.plane[i][0]].arc[check_table.plane[i][1]].plane[check_table.plane[i][2]-1].name[j];
19            if (j<2) {                  // 字母位
20                k = c;
21                k = k-65;
22            }
23            else{                       // 数字位
24                k=atoi(&c);
25            }
26            q[k].addPlane(check_table.plane[i][0], check_table.plane[i][1], check_table.plane[i][2]);
27        }
28    }
29
30    void Airplane::Collect(Check_Table q[], int m){
31        Check_Table c;
32        for (int i=0; i<m; i++) {
33            for (int j=0; j<q[i].num_plane; j++) {
```

```
34                 c.addPlane(q[i].plane[j][0], q[i].plane[j][1], q[i].plane[j][2]);
35             }
36         }
37         check_table = c;
38     }
```

3.2.3 查找与推荐

在每次搜索航班时，*System* 首先会调用 *Sort*() 函数对航班号进行排序，因为可能管理员进行航班的添加、删除之后会影响航班的次序。

以下是 *Airplane* 的查找功能。用户输入通过 *System* 类传递到其数据成员 *airplane*，并调用其 *Search*() 函数，分别对各项查询信息进行判断，判断用户是否输入有效查询信息，若是，则根据该项信息进行查找，特别地，若用户输入航班号信息，则会对已按照航班号排序的航班记录采用二分查找，分别查找符合条件航班记录的左端点与右端点，将该区间的航班索引赋值给 *Check_Table* 类型的临时变量 *c*。用户不需要选择按哪一项进行查找，只需按意愿输入航班信息，在其余项输入 “NA” 即可。

若查找失败，则可通过 *c* 的航班记录数为 0 来判断。在进行完直达航班的查找之后，输出查询到的航班信息以及最低费用。查找失败，即 *c* 的长度为 0 时，输出相应信息。

若查找不到航班号，或者用户仅输入出发城市、到达城市信息，则进行中转航班的查找，执行函数 *Trasnfer*()。由于国内航班飞行时间较短，故本文考虑的中转航班仅为当天中转。这时，可利用读取数据时存储的依日期顺序排序索引表 *check_table_time* 来进行查找，将每一天的航班记录传入函数 *SearchTransfer*() 进行当日中转航班的查找。查找完成后输出相应信息。

(部分 Unicode 符号在所展示代码中不可见，具体请查看源程序。)

```
1  void Airplane::Search(Check_Table &c,string ID,string year,string month,string day,string start,string
   ↪ arrive,string start_city,string arrive_city){
2      cout<<"*****************************  直达航班信息  *************************"<<endl;
3      c = check_table;
4      if (ID!="NA") {            // 二分查找航班号
5          int i=0,j= c.num_plane-1;
6          int low = 0,high = c.num_plane-1;
7          while (low<high) {
8              int mid = (low+high)>>1;
9              (ID<arc[c.plane[mid][0]].arc[c.plane[mid][1]].plane[c.plane[mid][2]-1].name)?high=mid:low=mid+1;
10         }
11         while (i<j) {
```

```
            int mid=(i+j)>>1;
            (ID>arc[c.plane[mid][0]].arc[c.plane[mid][1]].plane[c.plane[mid][2]-1].name)?i=mid+1:j=mid;
        }
        Check_Table ct;
        for(int k=i;k<low;k++)
            ct.addPlane(c.plane[k][0], c.plane[k][1], c.plane[k][2]);
        c = ct;
    }

    if (year!="NA"&&c.num_plane>0) {// 查找年份
        Check_Table ct;
        for (int i=0; i<c.num_plane; i++) {
            if(year == arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].year){
                ct.addPlane(c.plane[i][0], c.plane[i][1], c.plane[i][2]);
            }
        }
        c = ct;
    }
    if (month!="NA"&&c.num_plane>0) {// 查找月份
        Check_Table ct;
        for (int i=0; i<c.num_plane; i++) {
            if(month == arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].month){
                ct.addPlane(c.plane[i][0], c.plane[i][1], c.plane[i][2]);
            }
        }
        c = ct;
    }
    if (day!="NA"&&c.num_plane>0) {  // 查找日期
        Check_Table ct;
        for (int i=0; i<c.num_plane; i++) {
            if(day == arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].day){
                ct.addPlane(c.plane[i][0], c.plane[i][1], c.plane[i][2]);
            }
        }
        c = ct;
    }
    if (start!="NA"&&c.num_plane>0) {//查找出发时间
        Check_Table ct;
        for (int i=0; i<c.num_plane; i++) {
```

```
51              if(start == arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].start){
52                  ct.addPlane(c.plane[i][0], c.plane[i][1], c.plane[i][2]);
53              }
54          }
55          c = ct;
56      }
57      if (arrive!="NA"&&c.num_plane>0) {// 查找到达时间
58          Check_Table ct;
59          for (int i=0; i<c.num_plane; i++) {
60              if(arrive == arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].arrive){
61                  ct.addPlane(c.plane[i][0], c.plane[i][1], c.plane[i][2]);
62              }
63          }
64          c = ct;
65      }
66      if (start_city!="NA"&&c.num_plane>0) {// 查找出发城市
67          Check_Table ct;
68          for (int i=0; i<c.num_plane; i++) {
69              if(start_city == city[c.plane[i][0]].name || start_city == city[c.plane[i][0]].ID){
70                  ct.addPlane(c.plane[i][0], c.plane[i][1], c.plane[i][2]);
71              }
72          }
73          c = ct;
74      }
75      if (arrive_city!="NA"&&c.num_plane>0) {// 查找到达城市
76          Check_Table ct;
77          for (int i=0; i<c.num_plane; i++) {
78              if(arrive_city == city[c.plane[i][1]].name || arrive_city == city[c.plane[i][1]].ID){
79                  ct.addPlane(c.plane[i][0], c.plane[i][1], c.plane[i][2]);
80              }
81          }
82          c = ct;
83      }
84      if (c.num_plane==0) {
85          cout<<"无相应直达航线！"<<endl;
86      }
87      else{                           // 若有符合条件的航班，则查找最小费用航班
88          int price=99999;
89          for (int i=0; i<c.num_plane; i++) {
```

```
            cout<<'('<<i+1<<')'<<endl;
            cout<<city[c.plane[i][0]].name+city[c.plane[i][0]].airport[arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].plat_start]+"
 "+city[c.plane[i][1]].name+city[c.plane[i][1]].airport[arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].plat_arrive]<<endl;
            cout<<"\t航班号\t\t日期\t\t星期\t\t出发时间\t到达时间\t班期\t机型\t\t折扣\n\t";
            arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].show();
            price = min(price,arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].Minprice());
        }
        cout<<"最小费用航班为:";
        for (int i=0; i<c.num_plane; i++) {
            if(price ==arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].Minprice()){
                cout<<'('<<i+1<<')'<<'\t';
            }
        }
        cout<<endl;
    }
    bool is = c.num_plane;
    for (int i=0; i<c.num_plane; i++) {
        for (int j=0; j<arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].num; j++) {
            if (arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].num_max[j]>arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].num_now[j])
            {
                is = false;
                break;
            }
        }
    }
    if (is || (ID=="NA" && start_city!="NA" && arrive_city!="NA")) {
        cout<<"**************************  中转航班信息  **************************"<<endl;
        Transfer(c,year,month,day,start_city,arrive_city);
    }
}

void Airplane::Transfer(Check_Table &c0, string year, string month, string day, string start_city, string arrive_city){
    Check_Table c;
    c=check_table_time;
    int n = c0.num_plane;
```

```
123     if (year!="NA" && c.num_plane>0) {// 查找年份
124         Check_Table ct;
125         for (int i=0; i<c.num_plane; i++) {
126             if(year == arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].year){
127               ct.addPlane(c.plane[i][0], c.plane[i][1], c.plane[i][2], c.time[i][0],c.time[i][1],c.time[i][2],c.time[i][3]);
128             }
129         }
130         c = ct;
131     }
132     if (month!="NA" && c.num_plane>0) {// 查找月份
133         Check_Table ct;
134         for (int i=0; i<c.num_plane; i++) {
135             if(month == arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].month){
136               ct.addPlane(c.plane[i][0], c.plane[i][1], c.plane[i][2], c.time[i][0],c.time[i][1],c.time[i][2],c.time[i][3]);
137             }
138         }
139         c = ct;
140     }
141     if (day!="NA" && c.num_plane>0) {// 查找日期
142         Check_Table ct;
143         for (int i=0; i<c.num_plane; i++) {
144             if(day == arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].day){
145               ct.addPlane(c.plane[i][0], c.plane[i][1], c.plane[i][2], c.time[i][0],c.time[i][1],c.time[i][2],c.time[i][3]);
146             }
147         }
148         c = ct;
149     }
150     string year1=c.time[0][0],month1=c.time[0][1],day1=c.time[0][2];
151     Check_Table ct[num_city+1];
152     ct[num_city] = c0;
153     int num = c0.num_plane;
154     int price;
155     for (int i=0; i< c.num_plane; i++) {
156         string year2=check_table_time.time[i][0],month2=check_table_time.time[i][1],day2=check_table
    ↪ _time.time[i][2];
157         if (year1+month1+day1!=year2+month2+day2) {
158             price = SearchTransfer(ct,start_city,arrive_city);
159             if (price<9999) {
160                 cout<<year1<<'.'<<month1<<'.'<<day1<<endl;
```

```
161                 for (int j=num; j<ct[num_city].num_plane; j++) {
162                     cout<<'('<<num+j+1<<')'<<endl;
163                     cout<<city[citytonum[start_city]].name+city[citytonum[start_city]].airport[arc[citytonum
    ↪ [start_city]].arc[ct[num_city].plane[j][1]].plane[ct[num_city].plane[j][0]-1].plat_start]+"
    ↪ "+city[ct[num_city].plane[j][1]].name+city[ct[num_city].plane[j][1]].airport[arc[citytonum[start_city]].
    ↪ arc[ct[num_city].plane[j][1]].plane[ct[num_city].plane[j][0]-1].plat_arrive]+"
    ↪ "+city[citytonum[arrive_city]].name+city[citytonum[arrive_city]].airport[arc[ct[num_city].plane[j][1]].
    ↪ arc[citytonum[arrive_city]].plane[ct[num_city].plane[j][2]-1].plat_arrive]<<endl;
164                     cout<<"\t航班号\t\t日期\t\t星期\t\t出发时间\t到达时间\t班期\t\t机型\t\t折扣\n\t";
165 
    ↪ arc[citytonum[start_city]].arc[ct[num_city].plane[j][1]].plane[ct[num_city].plane[j][0]-1].show();
166                     cout<<"\t航班号\t\t日期\t\t星期\t\t出发时间\t到达时间\t班期\t机型\t\t折扣\n\t";
167 
    ↪ arc[ct[num_city].plane[j][1]].arc[citytonum[arrive_city]].plane[ct[num_city].plane[j][2]-1].show();
168                     cout<<endl;
169                 }
170                 cout<<"该日最低费用为:"<<price<<endl;
171                 year1=year2;month1=month2;day1=day2;
172             }
173             num = ct[num_city].num_plane;
174             for (int i=0; i<num_city; i++) {
175                 ct[i].Clear();
176             }
177         }
178         ct[c.plane[i][0]].addPlane(c.plane[i][0], c.plane[i][1], c.plane[i][2]);
179     }
180     price = SearchTransfer(ct,start_city,arrive_city);
181     if (price<9999) {
182         cout<<year1<<'.'<<month1<<'.'<<day1<<endl;
183         for (int i=num; i<ct[num_city].num_plane; i++) {
184             cout<<'('<<num+i+1<<')'<<endl;
185             cout<<city[citytonum[start_city]].name+city[citytonum[start_city]].airport[arc[citytonum[start
    ↪ _city]].arc[ct[num_city].plane[i][1]].plane[ct[num_city].plane[i][0]-1].plat_start]+"
    ↪ "+city[ct[num_city].plane[i][1]].name+city[ct[num_city].plane[i][1]].airport[arc[citytonum[start_city]].
    ↪ arc[ct[num_city].plane[i][1]].plane[ct[num_city].plane[i][0]-1].plat_arrive]+"
    ↪ "+city[citytonum[arrive_city]].name+city[citytonum[arrive_city]].airport[arc[ct[num_city].plane[i][1]].
    ↪ arc[citytonum[arrive_city]].plane[ct[num_city].plane[i][2]-1].plat_arrive]<<endl;
186             cout<<"\t航班号\t\t日期\t\t星期\t\t出发时间\t到达时间\t班期\t\t机型\t\t折扣\n\t";
187             arc[citytonum[start_city]].arc[ct[num_city].plane[i][1]].plane[ct[num_city].plane[i][0]-1].show();
```

```
188             cout<<"\t航班号\t\t日期\t\t星期\t\t出发时间\t到达时间\t班期\t机型\t折扣\n\t";
189             arc[ct[num_city].plane[i][1]].arc[citytonum[arrive_city]].plane[ct[num_city].plane[i][2]-1].show();
190             cout<<endl;
191         }
192         cout<<"该日最低费用为:"<<price<<endl;
193     }
194     num = ct[num_city].num_plane;
195     c0 = ct[num_city];
196     if (n==c0.num_plane) {
197         cout<<"无相应中转航班!"<<endl;
198     }
199 }
200
201 int Airplane::SearchTransfer(Check_Table c[], string start_city, string arrive_city){
202     Check_Table ct;
203     int price=9999;
204     int start=citytonum[start_city],arrive=citytonum[arrive_city];
205     for (int i=0; i<c[start].num_plane; i++) {
206         int mid = c[start].plane[i][1];
207         for (int j=0; j<c[mid].num_plane; j++) {
208             if (c[mid].plane[j][1]==arrive && arc[start].arc[mid].plane[c[start].plane[i][2]-1].arrive<=arc[mid]
    .arc[arrive].plane[c[mid].plane[j][2]-1].start)
    {
209                 if (arc[start].arc[mid].plane[c[start].plane[i][2]-1].Minprice()+arc[mid].arc[arrive].plane[c[mid].p
    lane[j][2]-1].Minprice()<price)
    {
210                     price=arc[start].arc[mid].plane[c[start].plane[i][2]-1].Minprice()+arc[mid].arc[arrive].plane[c[
    mid].plane[j][2]-1].Minprice();
211                     ct.Clear();
212                     ct.addPlane(c[start].plane[i][2], mid, c[mid].plane[j][2]);
213                 }
214                 else if (arc[start].arc[mid].plane[c[start].plane[i][2]-1].Minprice()+arc[mid].arc[arrive].plane[c[m
    id].plane[j][2]-1].Minprice()==price){
215                     ct.addPlane(c[start].plane[i][2], mid, c[mid].plane[j][2]);
216                 }
217             }
218         }
219     }
220     for (int i=0; i<ct.num_plane; i++) {
```

```
221             c[num_city].addPlane(ct.plane[i][0], ct.plane[i][1], ct.plane[i][2]);
222         }
223         return price;
224     }
```

3.2.4 订票与退票

由于订票与退票总是与特定航班紧密联系，通常订票或退票之前，总需要对该航班进行查询。若用户决定订票，则肯定已知将购票的航班；若用户决定退票，则可根据航班号以及日期来查找到对应的航班进行退票。因此本文将订票与退票功能绑定在航班查询之后，用户查询到航班并选择某一航班之后，可以对其进行订票或退票操作。

执行订票操作时，*System* 类的函数 *Book*() 首先会获取用户输入，*Airplane* 类的函数 *addBooked*() 会判断用户输入的舱位类型是否有足够多空位，如果没有则反馈 “订票失败” 信息，并将该客户加入该航班的替补客户中；如果有足够多空位，则执行对应航班的 *Customer* 类的 *addBooked*() 函数，向其 *Booked* 成员添加新元素，并返回 “订票成功” 信息。

执行退票操作时，*System* 类首先获取用户输入的姓名与证件号码，然后利用 *Airplane* 类的 *deleteBooked*() 函数查找该航班的已订票客户名单，若查找成功则删去该记录，调用 *Airplane* 类的 *informAlternative*() 函数将余票情况及候补客户名单写入系统日志，待管理员登陆后可以查看并通知候补客户订票，最后返回订票成功的信息；若查找失败，返回相应提示。

4 运行与测试

4.1 运行结果

4.1.1 普通用户

程序在 Xcode 8.3.2,C++11 标准下编译通过，运行程序后，首先可选择 “查询航班”” 以及 “管理员登录”。选择 “查询航班” 之后，输入各项信息（输入 “*NA*” 可跳过），当输入错误或不想继续查询时，可以输入 “-1” 返回，其中出发城市、到达城市可以输入中文名称，也可输入英文代码。如下图所示，当不存在直达航班时，或者查询到的直达航班均无票时，将自动查找中转航班以及最低费用并输出。

```
                    航空客运订票系统

                Designed By Jinhong Du

--------------------------------------------------------------------
请选择对应操作:
1.查询航班
2.管理员登录
任意键退出
1
--------------------------------------------------------------------
请输入相应查询信息（'NA'跳过,'-1'返回）
1.航班号:NA
2.年:NA
3.月:NA
4.日:NA
5.出发城市:北京
6.到达城市:广州
7.出发时间:NA
8.到达时间:NA

Searching...
*****************************    直达航班信息    *************************
无相应直达航班!
*****************************    中转航班信息    *************************
(1)
北京首都国际机场  ✈️ ✈️ ✈️  上海虹桥国际机场  ✈️ ✈️ ✈️  广州白云国际机场
    航班号        日期      星期      出发时间  到达时间  班期      机型      折扣
    CA0001   2017.05.29  星期一    06:25    08:40    一    A250     100%
        经济舱    530      1/2
        公务舱    1240     0/600
    航班号        日期      星期      出发时间  到达时间  班期      机型      折扣
    MU5341   2017.05.29  星期一    09:25    12:40    一    A250     100%
        经济舱    530      2/2
        公务舱    1240     0/600

该日最低费用为:1060
--------------------------------------------------------------------
请选择对应操作:
1.购票
2.退票
```

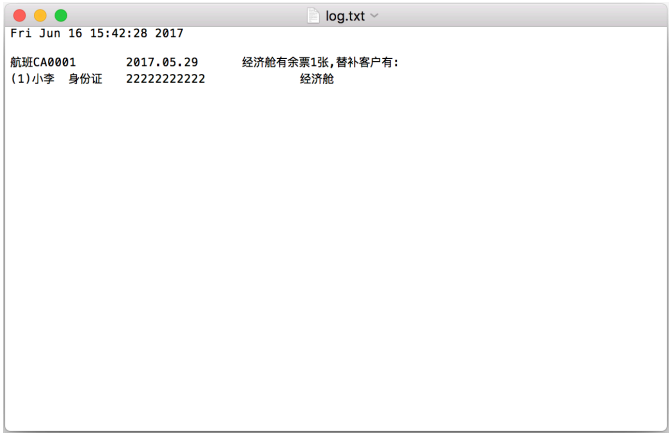
查询到航班信息之后，若有多个航班，则可以按照序号选择其中某一个航班或某一个中转航班组；若只有一个航班则不需进行选择。对选择的航班可以进行 “购票”、“退票” 操作，若选择的是对中转航班购票，则会依次对两个航班进行购票，如下左图所示；对直达航班购票如下右图所示；若要进行退票操作，则只能对直达

航班进行。

```
------------------------------------------------------------------------
请选择对应操作:
1.购票
2.退票
任意键返回
1
正在订购的航班为CA0001,2017.05.29,星期一,06:25-08:40
请输入订票乘客信息(返回请输入'-1'):
姓名:小林
证件类型:身份证
证件号码:440583199202122222
订票数量:1
航班情况:准点
舱位等级:公务舱
订单编号:201706132-1
订票成功!
正在订购的航班为MU5341,2017.05.29,星期一,09:25-12:40
请输入订票乘客信息(返回请输入'-1'):
姓名:小林
证件类型:身份证
证件号码:440583199202122222
订票数量:1
航班情况:准点
舱位等级:公务舱
订单编号:201706132-1
订票成功!
是否继续订票?
1.是
任意键返回
```

```
**************************    直达航班信息    **************************
(1)
上海虹桥国际机场                                          北京首都国际机场
    航班号        日期      星期     出发时间  到达时间  班期 机型      折扣
    CZ0002  2017.05.31  星期三   10:25   12:40   三四五六  A320    100%
        经济舱    500     0/160
        公务舱    1040    0/60
(2)
上海虹桥国际机场                                          北京首都国际机场
    航班号        日期      星期     出发时间  到达时间  班期 机型      折扣
    CZ0002  2017.06.01  星期四   10:25   12:40   三四五六  A320    100%
        经济舱    500     0/160
        公务舱    1040    0/60
(3)
上海虹桥国际机场                                          北京首都国际机场
    航班号        日期      星期     出发时间  到达时间  班期 机型      折扣
    CZ0002  2017.06.02  星期五   10:25   12:40   三四五六  A320    100%
        经济舱    500     0/160
        公务舱    1040    0/60
(4)
上海虹桥国际机场                                          北京首都国际机场
    航班号        日期      星期     出发时间  到达时间  班期 机型      折扣
    CZ0002  2017.06.03  星期六   10:25   12:40   三四五六  A320    100%
        经济舱    500     0/160
        公务舱    1040    0/60
最小费用航班为:(1) (2) (3) (4)
------------------------------------------------------------------------
请选择相应航班(任意键返回):1
------------------------------------------------------------------------
请选择对应操作:
1.购票
2.退票
```

客户退票之后，系统会自动把退票情况写入日志文件“log.txt”：



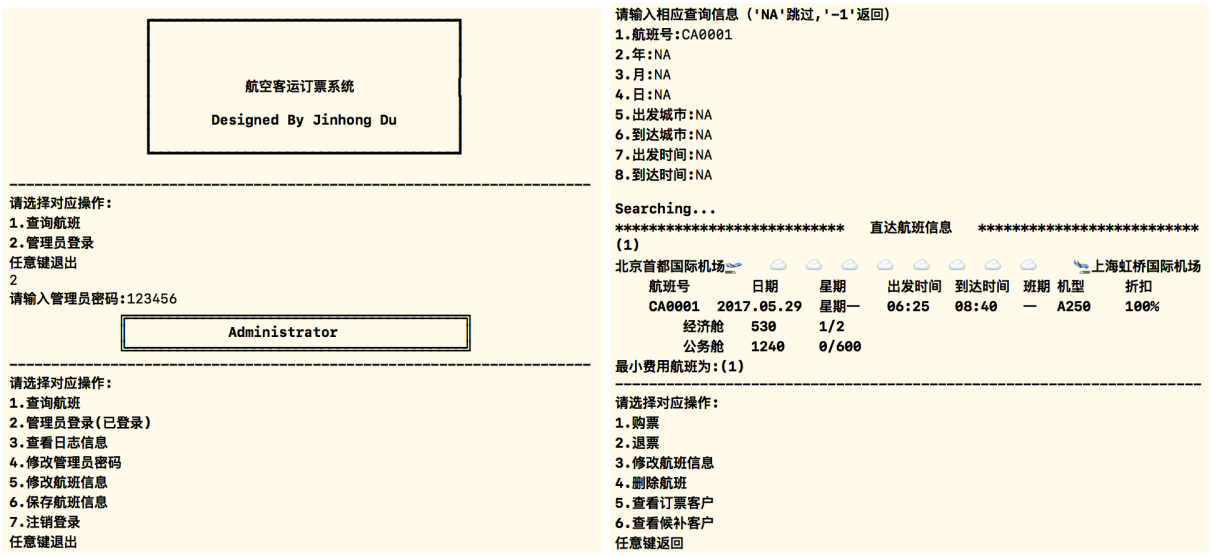
待管理员登录后系统可以使用 *Log*() 函数输出日志详情，日志记录了系统每一次登录时间以及退票明细、相应的替补客户名单。

4.1.2 管理员

在开始菜单中，选择管理员登录并输入密码后，即可进入管理员模式。管理员的初始密码为 123456，并保存在“*key*”文件中，登录后，也可以修改密码。管理员模式下，有更多功能可以使用，包括“查看日志信息”，“修改管理员密码”，“修改航班信息”，“保存航班信息”，“注销登录”。并且查询航班之后也有更多选项，包括“修改航班信息”，“删除航班”，“查看订票客户”，“查看候补客户”。

“修改航班信息”包括“添加航班”，“删除航班”，“修改航班信息”操作，其中后两者需要先查询到对应的航班才能进行。

“保存航班信息”可以及时存储航班信息，避免系统错误导致航班信息及客户信息丢失。当退出系统时，也会自动保存航班信息。



4.2 算法分析

4.2.1 向量自动扩充

实际上，数组本身并不具备可扩充性能，究其原因是它采用的是静态空间管理，在其内部开辟了数组并使用一段地址连续的物理空间。若采用静态空间管理策略，最大容量不变，则有明显的不足，一方面可能发生上溢，数组不足以存放所有元素；另一方面可能发生下溢，数组的装填因子远小于 50%。在实际问题中，我们很难准确预测空间的需求量，因此考虑可以采用动态空间管理，在即将发生上溢的时候，适当地扩大内部数组的容量，进行扩容。相比于直接对数组进行扩容、缩容，使用封装的向量，可以避免出现野指针的情形，更加安全。因为下溢并不是必须解决的问题，在空间资源足够的情况下，本文仅考虑扩容操作。

容易想到的是采用容量递增策略，也就是在当前容量等于最大容量的时候，追加固定的长度。在最坏的情况下，假设我们向初始容量为 0 的向量连续插入 $n=\dfrac{m\times I}{2}$ 个元素，其中 I 为每次扩容的长度，则各次扩充需要复制原向量的时间成本为分别 $0,I,2I,\cdots,(m-1)I$，总体耗时为

$$0+I+2I+\cdots+(m-1)I=I\cdot\frac{m(m-1)}{2}=(n^2),$$

分摊耗时为 $O(n)$。

考虑容量加倍策略，即在当前容量等于最大容量的时候，扩容使长度加倍。在最坏的情况下，在初始容量为 1 的已满向量中，连续插入 $n=2^{m-1}$ 个元素，则只需在第 $1,2,4,8,\cdots,2^{m-1}$ 次插入时需要扩容，各次扩容复制原向量的时间成本是 $1,2,4,8,\cdots,2^{m-1}$，则总体耗时为

$$1+2+4+8+\cdots+2^{m-1}=2n-1=O(n),$$

分摊耗时为 $O(1)$。

尽管在装填因子方面，递增策略接近于 100%，而加倍策略只能保证大于或等于 50%，但是在扩容时间开销上，后者要明显优于前者，在需要进行订票、退票等频繁改动操作的情况下，显然使用加倍策略更佳。

4.2.2 基数排序

本文基数排序利用的临时存储结构为向量，设航班记录为 n 个。因为分配过程需要遍历航班记录，故时间复杂度为 $O(n)$；收集过程需要遍历 m 个临时存储单元以及航班记录，故时间复杂度为 $O(m+n)$。基本操作的时间复杂度与使用链表作为临时存储单元是同数量级。

4.2.3 区间的二分查找

设航班记录共有 n 个，由于区间的二分查找主要拆分为两个端点的二分查找，而每个端点的二分查找至多不超过 $\log_2(n)+1$，每次迭代仅需常数时间，所以最坏情况下总体的时间复杂度为 $O(\log n)$。

4.2.4 中转航班查找

设共有 m 个城市，航班记录总数为 n 个，中转航班查找的出发城市有 n_0 个从该城市出发的航班，其中到达另外 $m-1$ 个城市的航班分别有 $j_1,\cdots,j_{m-1}$ 个，设从另外 $m-1$ 个城市出发的航班有 $n_1,\cdots,n_{m-1}$ 个，若依次查找，查找次数为

$$\sum_{i=1}^{m-1} j_i n_i,$$

其中

$$\sum_{i=1}^{m-1} j_i = n_0.$$

则最坏情况下，时间复杂度为 $O(n_0 \times n)$。之后还需对时间先后进行判断，更需花费大量时间。

如果利用按日期排列的航班记录，按每一天进行查找，则会减少大量不必要的比对工作. 假设航班记录对日期均匀分布，共有 d 天，则时间复杂度为 $O\left(\dfrac{n_0 \times n}{d^2}\right)$，大大减少了时间花销。

5 总结

航空客运订票系统设计的关键是选取适当的存储结构以及排序、搜索算法。在频繁添加、删除的场合，动态空间分配的策略显然更佳。因为系统的大部分功能都依赖于搜索查找功能，而对有序变量的搜索查找往往比无序变量要更高效，因此，排序算法与有序搜索算法对于搜索查找功能是非常重要的。相比于顺序查找，这种方式会节约大量时间开销。关键是结合实际问题，根据数据的特点，选取合适的排序算法、搜索算法，才能达到较好的结果。

在整个系统的设计过程中，主要遇到的问题是动态空间分配所产生的索引错误越界、浅复制析构错误等，分别的解决方法是进行索引位置范围的判断、重载 “=” 运算符进行深复制。

诚然，本文系统仍有许多不足，有许多可以改进之处。可以将某些类以基类派生的形式写出，例如 *Booked* 类与 *Alternative* 类，以节省资源、减少多余功能对系统资源的占用；系统的功能仍可以丰富，例如系统日志的写入可以包括客户订票的明细等。

6 附录

1. *airplane.hpp*

```
#ifndef airplane_hpp
#define airplane_hpp

#include<iostream>
#include<fstream>
#include<sstream>
using namespace std;
#include<string>
#include <cstdlib>

class Booked{                                          // 已订票客户类
public:
    Booked():name(" "),kindID(" "),ID(" "),num(" "),state(" "),kind(" "),ticket(" "){};
                                                       // 默认构造函数
    Booked(string name,string kindID,string ID,string book_num,string state,string kind,string
    ↪ ticket);
                                                       // 构造函数
    ~Booked(){};                                       // 析构函数
    Booked& operator=(Booked &b);                      // 重载 = 运算符
    void show();                                       // 输出已订票客户信息
    string Save();                                     // 保存
    string name,kindID,ID,num,state,kind,ticket;       // 已订票客户信息
};

class Alternative{                                     // 候补客户类
public:
    Alternative():name(" "),kindID(" "),ID(" "),num(" "),kind(" "){};
                                                       // 默认构造函数
    Alternative(string name,string kindID,string ID,string book_num,string kind);
                                                       // 构造函数
    ~Alternative(){};                                  // 析构函数
    Alternative& operator=(Alternative &a);            // 重载 = 运算符
    void show();                                       // 输出候补客户信息
    string inform();                                   // 输出候补客户信息
    string Save();                                     // 保存
```

```
35      string name,kindID,ID,num,kind;                 // 候补客户信息
36  };
37
38  class Customer{                                     // 客户类
39  public:
40      Customer();                                     // 默认构造函数
41      ~Customer();                                    // 析构函数
42      Customer& operator=(Customer &p);   // 重载 = 运算符
43      void addBooked(string name,string kindID,string ID,string book_num,string state,string
    ↪   kind,string ticket);                                    // 添加已订票客户
44      void addAlternative(string name,string kindID,string ID,string book_num,string kind);//
    ↪   添加替补客户
45      void showBooked();                              // 输出已订票客户
46      void showAlternative();                         // 输出替补客户
47      void informAlternative(string kind);            // 通知替补客户
48      string deleteBooked(int de);                    // 退票
49      string Save();                                  // 保存客户信息
50      Booked* booked;                                 // 已订票客户
51      Alternative* alternative;                       // 候补客户
52      int num_booked,num_alternative;                 // 已订票客户数、候补客户数
53      int max_booked,max_alternative;                 // 最大已订票客户数、最大候补客户数
54  };
55
56  class Check_Table{                                          // 航班索引表
57  public:
58      Check_Table();                                          // 默认构造函数
59      ~Check_Table();                                         // 析构函数
60      Check_Table& operator=(Check_Table &c);                 // 重载 = 运算符
61      void addPlane(int m,int n,int o,string year="",string month="",string day="",string date="");
62                                                              // 添加航班索引
63      void Clear();                                           // 清除索引表
64      int** plane;                                            // 三元组
65      string** time;                                          // 日期四元组
66      int num_plane,max_num;                                  // 航班数、最大容量
67  };
68
69  class Plane{                                                        // 航班结点类
70  public:
71      Plane(){kind=nullptr;price=num_now=num_max=nullptr;};   // 默认构造函数
```

```
72      Plane(string name,string year,string month,string day,string date,string schedule,string
    ↪   start,string arrive,string shape,string discount,int num,string* kind,int* price,int* num_now,int*
    ↪   num_max,int plat_start,int plat_arrive);
73                                                          // 构造函数
74      ~Plane();                                           // 析构函数
75      Plane& operator=(Plane &p);                         // 重载 = 运算符
76      string Save(string start_city, string arrive_city); // 保存该航班
77      int Minprice();                                     // 返回最低票价
78      string name;                                        // 航班编号
79      string year,month,day,date;                         // 日期
80      string start;                                       // 出发时间
81      string arrive;                                      // 到达时间
82      int plat_start,plat_arrive;                         // 出发机场、到达机场序号
83      string schedule;                                    // 班期
84      string shape,discount;                              // 机型、折扣
85      int num;                                            // 舱位类型数
86      string* kind;                                       // 舱位类型名称
87      int* price;                                         // 价格
88      int* num_now;                                       // 已订乘客数
89      int* num_max;                                       // 最大乘客数
90      Customer customer;                                  // 客户类
91  };
92
93  class Edge{                         // 有向边类
94  public:
95      Edge();                         // 默认构造函数
96      ~Edge();                        // 析构函数
97      void Init();                    // 初始化
98      void addPlane(Plane &p);        // 添加航班
99      Plane* plane;                   // 航班
100     int num_plane,max_num;          // 航班数量、最大容量
101 };
102
103 class Arc{                          // 邻接矩阵类
104 public:
105     Arc(){};                        // 默认构造函数
106     ~Arc();                         // 析构函数
107     void Init(int size);            // 初始化
108     int size;                       // 大小
```

```
109     Edge* arc;                          // 有向边
110 };
111
112 // 城市结点
113 class City{                             // 城市结点类
114 public:
115     City();                             // 默认构造函数
116     ~City();                            // 析构函数
117     void addAirport(string airport);    // 添加机场
118     string name,ID;                     // 城市名称、城市英文代码
119     string* airport;                    // 机场
120     int num_airport,max_num;            // 机场个数、最大容量
121 };
122
123 class CitytoNum{                        // 城市名称-城市序号索引表类
124 public:
125     CitytoNum();                        // 默认构造函数
126     ~CitytoNum();                       // 析构函数
127     int operator[](string cityname);    // 重载[]运算符
128     void addcity(City city[],int num);  // 添加城市索引
129     string* name,*ID;                   // 城市名称、城市英文代码
130     int num;                            // 城市个数
131 };
132
133 class Airplane{
134 public:
135     Airplane(int num_city=5);                                   // 默认构造函数
136     ~Airplane();                                                // 析构函数
137     /*----------------------------------  初始化  ----------------------------------*/
138     void Init();                                                // 航班初始化
139     void addCity(const string &s, int i);                       // 添加城市结点
140     void addArc(string &year,string &month,string &day,string &date,const string &s,int city0,int
    ↪ city1,int plat0,int plat1);                                   // 添加有向边
141     void addPlane(string &year,string &month,string &day,string &date,const string &s);//
    ↪ 添加航班记录
142     /*----------------------------------排序与查找----------------------------------*/
143     void Sort();                                                // 基数排序
144     void Distribute(Check_Table q[],int m,int j);               // 分配
145     void Collect(Check_Table q[],int m);                        // 收集
```

```
146     void Search(Check_Table &c,string ID,string year,string month,string day,string start,string
    ↪   arrive,string start_city,string arrive_city);              // 查找航班
147     void Transfer(Check_Table &c,string year,string month,string day,string start_city,string
    ↪   arrive_city);                                                  // 筛选航班
148     int SearchTransfer(Check_Table c[],string start_city,string arrive_city);// 查找中转航班
149     /*--------------------------------------订票与退票--------------------------------------*/
150     int addBooked(int m,int n,int l,string name,string kindID,string ID,string book_num,string
    ↪   state,string kind,string ticket);                              // 添加订票客户
151     int addAlternative(int m,int n,int l,string name,string kindID,string ID,string book_num,string
    ↪   kind);                                                         // 添加候补客户
152     void showBooked(int m,int n,int l);                          // 输出某航班已订票客户
153     void showAlternative(int m,int n,int l);                     // 输出某航班候补客户
154     string deleteBooked(int m,int n,int l,string name,string ID);// 退票
155     void informAlternative(int m,int n,int l,string kind);       // 通知候补客户
156     /*--------------------------------------修改与保存--------------------------------------*/
157     void Change(int m,int n,int l);                              // 修改航班信息
158     void DeletePlane(int m,int n,int l);                         // 删除航班
159     void Save();                                                 // 保存航班信息
160     void show(int i,int j,int k);                                // 输出航班基本信息
161     CitytoNum citytonum;                                         // 城市名称-城市序号索引表
162   private:
163     City* city;                                                  // 城市结点
164     Arc* arc;                                                    // 有向边
165     int num_city;                                                // 城市个数
166     string cityname,planename;                                   // 存储文件名称
167     Check_Table check_table,check_table_time;                    // 索引表
168   };
169
170   class System{
171   public:
172     System();                          // 默认构造函数
173     ~System();                         // 析构函数
174   private:
175     /*-------------------------------------- 初始化 --------------------------------------*/
176     void Init();                       // 系统初始化
177     void Input();                      // 录入航班情况
178     /*-------------------------------------- 普通用户 --------------------------------------*/
179     void Sort();                       // 对航班进行排序
180     void Check();                      // 查询航线
```

```
181     void Search(string ID,string year,string month,string day,string start,string arrive,string
    ↪  start_city,string arrive_city);// 查找航班
182     int Book(int i, int j, int k);      // 订票
183     /*---------------------------------- 管理员 ----------------------------------*/
184     void AddPlane();                    // 添加航班
185     void Change();                      // 修改航班信息
186     void Save();                        // 保存航班信息
187     void AmdinLogin();                  // 管理员登录
188     void AmdinLogout();                 // 管理员注销
189     void GetKey();                      // 读取密码文件
190     void ChangeKey();                   // 修改管理员密码
191     void Log();                         // 输出日志信息
192     Airplane* airplane;                 // Airplane类
193     int prior;                          // 权限级别
194     string key;                         // 密码
195  };
196  #endif /* airplane_hpp */
```

2. *System.cpp*

```
1  #include "airplane.hpp"
2  #include<ctime>
3  #include <cstdio>
4  System::System():prior(0){
5     ofstream file("/Users/dujinhong/Documents/study/数据结构/数据结构实验/大作业/airplane/airp
   ↪   lane/log.txt",ios::app);
6     time_t t;
7     t=time(&t);
8     file<<ctime(&t)<<endl;
9     file.close();
10    Init();
11 }
12
13 System::~System(){
14    Save();
15    delete airplane;
16 }
17
```

```
18  void System::Init(){
19      airplane = new Airplane;
20      airplane->Init();   // 航班信息初始化
21      GetKey();
22      int flag=1;
23      cout      <<"                                                    "<<endl;
24      cout      <<"                                                    "<<endl;
25      cout      <<"                                                    "<<endl;
26      cout      <<"                                                    "<<endl;
27      cout      <<"                    航空客运订票系统                "<<endl;
28      cout      <<"                                                    "<<endl;
29      cout      <<"                 Designed By Jinhong Du             "<<endl;
30      cout      <<"                                                    "<<endl;
31      cout      <<"                                                  "<<endl<<endl;
32      while (flag) {
33          int n;
34          cout<<"----------------------------------------------------------------------"<<endl;
35          cout<<"请选择对应操作:"<<endl;
36          cout<<"1.查询航班"<<endl;
37          cout<<"2.管理员登录";
38          if (prior) {
39              cout<<"(已登录)";
40          }
41          cout<<endl;
42          if (prior) {
43              cout<<"3.查看日志信息"<<endl;
44              cout<<"4.修改管理员密码"<<endl;
45              cout<<"5.修改航班信息"<<endl;
46              cout<<"6.保存航班信息"<<endl;
47              cout<<"7.注销登录"<<endl;
48          }
49          cout<<"任意键退出"<<endl;
50          cin>>n;
51          if (n<1 || n>2+5*prior) {
52              return;
53          }
54          switch (n) {
55              case 1:
56                  Check();break;
```

```
57              case 2:
58                  AmdinLogin();break;
59              case 3:
60                  Log();break;
61              case 4:
62                  ChangeKey();break;
63              case 5:
64                  Change();break;
65              case 6:
66                  Save();break;
67              case 7:
68                  AmdinLogout();break;
69              default:
70                  break;
71          }
72      }
73  }
74
75  void System::Sort(){
76      airplane->Sort();
77  }
78
79  void System::Check(){
80      Sort();
81      string ID,year,month,day,start,arrive,start_city,arrive_city;
82      cout<<"--------------------------------------------------------------------------------"<<endl;
83      cout<<"请输入相应查询信息（'NA'跳过,'-1'返回）"<<endl;
84      cout<<"1.航班号:";cin>>ID;if (ID=="-1"){return;}
85      cout<<"2.年:";cin>>year;if (year=="-1"){return;}
86      cout<<"3.月:";cin>>month;if (month=="-1"){return;}
87      cout<<"4.日:";cin>>day;if (day=="-1"){return;}
88      cout<<"5.出发城市:";cin>>start_city;if (start_city=="-1"){return;}
89      cout<<"6.到达城市:";cin>>arrive_city;if (arrive_city=="-1"){return;}
90      cout<<"7.出发时间:";cin>>start;if (start=="-1"){return;}
91      cout<<"8.到达时间:";cin>>arrive;if (arrive=="-1"){return;}
92      cout<<endl<<"Searching..."<<endl;
93      Search(ID,year,month,day,start,arrive,start_city,arrive_city);
94  }
95
```

```
96  void System::Search(string ID,string year,string month,string day,string start,string arrive,string
    ↪  start_city,string arrive_city){
97      Check_Table c;
98      (*airplane).Search(c,ID,year,month,day,start,arrive,start_city,arrive_city);
99      if (c.num_plane>0) {
100         int i=0;
101         if (c.num_plane>1) {
102             cout<<"------------------------------------------------------------------"<<endl;
103             cout<<"请选择相应航班(任意键返回):";cin>>i;
104             if (i<1 || i>c.num_plane) {
105                 return;
106             }
107             i--;
108         }
109         int n=1;
110         while (n) {
111             cout<<"------------------------------------------------------------------"<<endl;
112             cout<<"请选择对应操作:"<<endl;
113             cout<<"1.购票"<<endl;
114             cout<<"2.退票"<<endl;
115             if (prior==1) {
116                 cout<<"3.修改航班信息"<<endl;
117                 cout<<"4.删除航班"<<endl;
118                 cout<<"5.查看订票客户"<<endl;
119                 cout<<"6.查看候补客户"<<endl;
120             }
121             cout<<"任意键返回"<<endl;
122             cin>>n;
123             if (n<1 || n>2+3*prior) {
124                 return;
125             }
126             int flag=1;
127             switch (n) {
128                 case 1:{
129                     while (flag==1) {
130                         int d;
131                         if (arrive_city!="NA" && c.plane[i][1]!=airplane->citytonum[arrive_city]) {
132                             d = Book(airplane->citytonum[start_city],c.plane[i][1],c.plane[i][0]);
133                             if(d>=0){
```

```
134                     cout<<"订票成功!"<<endl;
135                 }
136                 else if(d!=-1){
137                     cout<<"订票失败!已加入替补客户名单。"<<endl;
138 
139                 }
140                 d = Book(c.plane[i][1],airplane->citytonum[arrive_city],c.plane[i][2]);
141                 if (d>=0) {
142                     cout<<"订票成功!"<<endl;
143                 }
144                 else if(d!=-1){
145                     cout<<"订票失败!已加入替补客户名单。"<<endl;
146                 }
147             }
148             else{
149                 d = Book(c.plane[i][0], c.plane[i][1], c.plane[i][2]);
150                 if(d>=0){
151                     cout<<"订票成功!"<<endl;
152                 }
153                 else if(d!=-1){
154                     cout<<"订票失败!已加入替补客户名单。"<<endl;
155                 }
156             }
157             cout<<"是否继续订票？"<<endl<<"1.是"<<endl<<"任意键返回"<<endl;
158             cin>>flag;
159         }
160     }
161         break;
162     case 2:{
163         while (flag==1) {
164             if (arrive_city!="NA" && c.plane[i][1]!=airplane->citytonum[arrive_city]) {
165                 cout<<"请分别退票!"<<endl;
166                 break;
167             }
168             string name,ID;
169             cout<<"请输入退票客户信息，退出请输入'-1'"<<endl;
170             cout<<"请输入退票客户姓名:";cin>>name;
171             if (name=="-1") {
172                 break;
```

```
173                     }
174                     cout<<"请输入退票客户证件号码:";cin>>ID;
175                     if (ID=="-1") {
176                         break;
177                     }
178                     string kind = airplane->deleteBooked(c.plane[i][0],c.plane[i][1],c.plane[i][2],name,ID);
179                     if (kind=="") {
180                         cout<<"输入错误或该用户未订票!"<<endl;
181                     }
182                     airplane->informAlternative(c.plane[i][0],c.plane[i][1],c.plane[i][2],kind);
183                     cout<<"退票成功!"<<endl<<"是否继续退票?
    ↪ "<<endl<<"1.是"<<endl<<"任意键返回"<<endl;
184                     cin>>flag;
185                 }
186             }
187                 break;
188             case 3:
189                 while (flag==1) {
190                     if (arrive_city!="NA" && c.plane[i][1]!=airplane->citytonum[arrive_city]) {
191                         cout<<"请分别修改航班信息!"<<endl;
192                         break;
193                     }
194                     airplane->Change(c.plane[i][0],c.plane[i][1],c.plane[i][2]);
195                     cout<<"修改成功!"<<endl;
196                     cout<<"是否继续修改? "<<endl<<"1.是"<<endl<<"任意键返回"<<endl;
197                     cin>>flag;
198                 }
199                 break;
200             case 4:
201                 if (arrive_city!="NA" && c.plane[i][1]!=airplane->citytonum[arrive_city]) {
202                     cout<<"请选择直达航线!"<<endl;
203                     break;
204                 }
205                 airplane->DeletePlane(c.plane[i][0],c.plane[i][1],c.plane[i][2]);
206                 break;
207             case 5:
208                 if (arrive_city!="NA" && c.plane[i][1]!=airplane->citytonum[arrive_city]) {
209                     cout<<"请选择直达航线!"<<endl;
210                     break;
```

```
211                 }
212                 airplane->showBooked(c.plane[i][0],c.plane[i][1],c.plane[i][2]);
213                 break;
214             case 6:
215                 if (arrive_city!="NA" && c.plane[i][1]!=airplane->citytonum[arrive_city]) {
216                     cout<<"请选择直达航线!"<<endl;
217                     break;
218                 }
219                 airplane->showAlternative(c.plane[i][0],c.plane[i][1],c.plane[i][2]);
220                 break;
221             default:
222                 break;
223         }
224     }
225 }
226 }
227
228 int System::Book(int i, int j, int k){
229     string name, kindId, ID, num, state, kind, ticket;
230     cout<<"正在订购的航班为";airplane->show(i,j,k);
231     cout<<"请输入订票乘客信息(返回请输入'-1'):"<<endl;
232     cout<<"姓名:";cin>>name;if (atoi(name.c_str())==-1) {return -1;}
233     cout<<"证件类型:";cin>>kindId;
234     if (atoi(kindId.c_str())==-1) {return -1;}
235     cout<<"证件号码:";cin>>ID;
236     if (atoi(ID.c_str())==-1) {return -1;}
237     cout<<"订票数量:";cin>>num;
238     if (atoi(name.c_str())==-1) {return -1;}
239     cout<<"航班情况:";cin>>state;
240     if (atoi(state.c_str())==-1) {return -1;}
241     cout<<"舱位等级:";cin>>kind;
242     if (atoi(kind.c_str())==-1) {return -1;}
243     cout<<"订单编号:";cin>>ticket;if (atoi(ticket.c_str())==-1){return -1;}
244     return airplane->addBooked(i,j,k,name, kindId, ID, num, state, kind, ticket);
245 }
246
247 void System::Change(){
248     int flag = 1;
249     while (flag) {
```

```
250         int n;
251         cout<<"-----------------------------------------------------------------"<<endl;
252         cout<<"请输入相应操作序号:"<<endl;
253         cout<<"1.添加航班"<<endl;
254         cout<<"2.删除航班"<<endl;
255         cout<<"3.修改航班"<<endl;
256         cout<<"任意键返回"<<endl;
257         cin>>n;
258         if (n<1 || n>3) {
259             return;
260         }
261         int i=1;
262         switch (n) {
263             case 1:{
264                 while (i==1) {
265                     AddPlane();
266
    ↪ cout<<"添加成功!是否继续添加航班?"<<endl<<"1.是"<<endl<<"任意键返回"<<endl;
267                     cin>>i;
268                 }
269             }
270                 break;
271             case 2:{
272                 cout<<"请搜索相应的航班进行删除:"<<endl;
273                 while (i==1) {
274                     Check();
275                     cout<<"是否继续修改？"<<endl<<"1.是"<<endl<<"任意键返回"<<endl;
276                     cin>>i;
277                 }
278             }
279                 break;
280             case 3:{
281                 cout<<"请搜索相应的航班进行修改:"<<endl;
282                 while (i==1) {
283                     Check();
284                     cout<<"是否继续修改？"<<endl<<"1.是"<<endl<<"任意键返回"<<endl;
285                     cin>>i;
286                 }
287             }
```

```
288             default:
289                 break;
290         }
291     }
292 }
293
294 void System::Save(){
295     airplane->Save();
296     cout<<"保存成功!"<<endl;
297 }
298
299
300 void System::AmdinLogin(){
301     string key2="";
302     cout<<"请输入管理员密码:";cin>>key2;
303     while (key2!=key) {
304         if (key2=="-1") {
305             return;
306         }
307         cout<<"密码错误！请重新输入(返回请输入-1):";cin>>key2;
308     }
309     prior = 1;
310     cout<<"                                                    "<<endl;
311     cout<<"                    Administrator                   "<<endl;
312     cout<<"                                                    "<<endl;
313 }
314
315 void System::AmdinLogout(){
316     prior = 0;
317     cout<<"注销成功!"<<endl;
318 }
319
320 void System::GetKey(){
321     ifstream fileIn("/Users/dujinhong/Documents/study/数据结构/数据结构实验/大作业/airplane/ai
    ↪  rplane/Key");
322     if (fileIn) {
323         getline(fileIn,key);
324         fileIn.close();
325     }
```

```
326 }
327
328 void System::ChangeKey(){
329     string key2;
330     cout<<"请输入原密码:";cin>>key2;
331     while (key2!=key) {
332         if (key2=="-1") {
333             return;
334         }
335         cout<<"密码错误！请重新输入(返回请输入-1):";cin>>key2;
336     }
337     cout<<"请输入新密码:";cin>>key;
338     ofstream fileOut("/Users/dujinhong/Documents/study/数据结构/数据结构实验/大作业/airplane/
    ↪  airplane/Key");
339     if (fileOut.is_open()) {
340         fileOut<<key;
341         fileOut.close();
342     }
343     cout<<"管理员密码修改成功!"<<endl;
344 }
345
346 void System::Log(){
347     ifstream fileIn("/Users/dujinhong/Documents/study/数据结构/数据结构实验/大作业/airplane/ai
    ↪  rplane/Log.txt");
348     if (fileIn) {
349         string s;
350         while (getline(fileIn,s)) {
351             cout<<s<<endl;
352         }
353         fileIn.close();
354     }
355 }
356
357 void System::AddPlane(){
358     string ss,name,start_city,arrive_city,year,month,day,date,shape,discount,kind,price,num_now,nu
    ↪  m_max,start,arrive,schedule;
359     int n;
360     cout<<"请输入新航班信息('-1'返回):"<<endl;
361     cout<<"航班编号:";cin>>name;if(name=="-1"){return;}
```

```
362     cout<<"出发城市及机场英文代码:";cin>>start_city;if(start_city=="-1"){return;}
363     cout<<"到达城市及机场英文代码:";cin>>arrive_city;if(arrive_city=="-1"){return;}
364     cout<<"年:";cin>>year;if(year=="-1"){return;}
365     cout<<"月:";cin>>month;if(month=="-1"){return;}
366     cout<<"日:";cin>>day;if(day=="-1"){return;}
367     cout<<"星期:";cin>>date;if(date=="-1"){return;}
368     cout<<"出发时间:";cin>>start;if(start=="-1"){return;}
369     cout<<"到达时间:";cin>>arrive;if(arrive=="-1"){return;}
370     cout<<"班期:";cin>>schedule;if(schedule=="-1"){return;}
371     cout<<"机型:";cin>>shape;if(shape=="-1"){return;}
372     cout<<"折扣:";cin>>discount;if(discount=="-1"){return;}
373     cout<<"舱位种类:";cin>>n;if(n==-1){return;}
374     ss=name+'\t'+start_city+'\t'+arrive_city+'\t'+schedule+'\t'+start+'\t'+arrive+'\t'+shape+'
    ↪  \t'+discount;
375     for (int i=0; i<n; i++) {
376         cout<<"舱位("<<i+1<<")名称:";cin>>kind;
377         cout<<"舱位("<<i+1<<")价格:";cin>>price;
378         ss=ss+'\t'+kind+'\t'+price+'\t'+"0"+'\t'+"0";
379     }
380     airplane->addPlane(year, month, day, date, ss+"\nNA\nNA");
381 }
```

3. *airplane.cpp*

```
1  #include "airplane.hpp"
2
3  Airplane::Airplane(int
   ↪  num_city/*=5*/):num_city(num_city),cityname("City.txt"),planename("plane.txt"){
4      city = new City[num_city];
5      arc = new Arc[num_city];
6      for(int i=0; i<num_city; i++){
7          arc[i].Init(num_city);
8      }
9  }
10
11 Airplane::~Airplane(){
12     delete[] city;
13     delete[] arc;
```

```
14  }
15
16  void Airplane::Init(){
17      ifstream fileIn("/Users/dujinhong/Documents/study/数据结构/数据结构实验/大作业/airplane/ai
    ↪   rplane/City.txt");
18      string s;
19      if (fileIn) {
20          int i=0;
21          while (getline(fileIn,s)) {
22              addCity(s,i++);
23          }
24          fileIn.close();
25      }
26      citytonum.addcity(city,num_city);
27      fileIn.open("/Users/dujinhong/Documents/study/数据结构/数据结构实验/大作业/airplane/airpla
    ↪   ne/plane.txt");
28      if (fileIn) {
29          string year,month,day,date;
30          while (getline(fileIn,s,'\n')) {
31              istringstream str(s);
32              getline(str,year,'\t');
33              getline(str,month,'\t');
34              getline(str,day,'\t');
35              getline(str,date,'\t');
36              if (getline(fileIn,s,'/') && s!="" ) {
37                  istringstream str2(s);
38                  while(getline(str2,s,'#')){
39                      addPlane(year,month,day,date,s);
40                      getline(str2,s,'\n');
41                  }
42                  getline(fileIn,s,'\n');
43              }
44          }
45          fileIn.close();
46      }
47  }
48
49  void Airplane::addCity(const string &s, int i){
50      istringstream str(s);
```

```
51      getline(str,city[i].name,'\t');
52      getline(str,city[i].ID,'\t');
53      string airport;
54      while(getline(str,airport,'\t')){
55          city[i].addAirport(airport);
56      }
57  }
58
59  void Airplane::addPlane(string &year,string &month,string &day,string &date,const string &s){
60      istringstream str(s);
61      string list;
62      string name,start,arrive;
63
64      getline(str,name,'\t');
65      getline(str,start,'\t');
66      getline(str,arrive,'\t');
67      int city0=-1,city1=-1,plat0,plat1;
68      for (int i=0; i<num_city; i++) {
69          if (city[i].ID==start.substr(0,3)) {
70              city0 = i;
71          }
72          else if (city[i].ID==arrive.substr(0,3)){
73              city1 = i;
74          }
75          if (city0>=0 && city1>=0) {
76              break;
77          }
78      }
79      plat0=atoi(start.substr(2,1).c_str());
80      plat1=atoi(arrive.substr(2,1).c_str());
81      addArc(year,month,day,date,s,city0,city1,plat0,plat1);
82      int i=0;
83      for (i=check_table_time.num_plane-1; i>=0; i--) {
84          if (year+month+day>=check_table_time.time[i][0]+check_table_time.time[i][1]+check_tabl
    ↪  e_time.time[i][2])
    ↪  {
85              break;
86          }
87      }
```

```
88      if (i==check_table_time.num_plane-1) {
89          check_table_time.addPlane(city0,city1,arc[city0].arc[city1].num_plane,year,month,day,date);
90      }
91      else{
92          check_table_time.addPlane(city0,city1,arc[city0].arc[city1].num_plane,year,month,day,date);
93          for (int j=check_table_time.num_plane-2; j>i; j--) {
94              for (int k=0; k<3; k++) {
95                  check_table_time.plane[j+1][k] = check_table_time.plane[j][k];
96              }
97              for (int k=0; k<4; k++) {
98                  check_table_time.time[j+1][k] = check_table_time.time[j][k];
99              }
100         }
101         check_table_time.plane[i+1][0] = city0;
102         check_table_time.plane[i+1][1] = city1;
103         check_table_time.plane[i+1][2] = arc[city0].arc[city1].num_plane;
104         check_table_time.time[i+1][0] = year;
105         check_table_time.time[i+1][1] = month;
106         check_table_time.time[i+1][2] = day;
107         check_table_time.time[i+1][3] = date;
108     }
109 }
110
111 void Airplane::addArc(string &year,string &month,string &day,string &date,const string &s,int
    ↪  city0,int city1,int plat0,int plat1){
112     istringstream strings(s);
113     string ss;
114     getline(strings,ss,'\n');
115     istringstream str(ss);
116     getline(str,ss,'\t');
117     string name=ss;
118     getline(str,ss,'\t');getline(str,ss,'\t');
119     getline(str,ss,'\t');
120     string schedule=ss;
121     getline(str,ss,'\t');
122     string start=ss;
123     getline(str,ss,'\t');
124     string arrive=ss;
125     getline(str,ss,'\t');
```

```
126     string shape=ss;
127     getline(str,ss,'\t');
128     string discount=ss;
129     string kind[3];
130     int price[3],now[3],max[3];
131     int num=0;
132     while (getline(str,ss,'\t')) {
133         kind[num] = ss;
134         getline(str,ss,'\t');
135         price[num]=atoi(ss.c_str());
136         getline(str,ss,'\t');
137         now[num]=atoi(ss.c_str());
138         getline(str,ss,'\t');
139         max[num]=atoi(ss.c_str());
140         num++;
141     }
142     Plane p(name,year,month,day,date,schedule,start,arrive,shape,discount,num,kind,price,now,max,p
    ↪  lat0,plat1);
143     arc[city0].arc[city1].addPlane(p);
144
145     getline(strings,ss,'\n');
146     if (ss!="NA") {
147         istringstream str(ss);
148         string sss;
149         while (getline(str,sss,',')) {
150             istringstream s4(sss);
151             getline(s4,sss,'\t');
152             string name2=sss;
153             getline(s4,sss,'\t');
154             string kindID=sss;
155             getline(s4,sss,'\t');
156             string ID=sss;
157             getline(s4,sss,'\t');
158             string num2=sss;
159             getline(s4,sss,'\t');
160             string state=sss;
161             getline(s4,sss,'\t');
162             string kind2=sss;
163             getline(s4,sss,'\t');
```

```
164             string ticket=sss;
165             arc[city0].arc[city1].plane[arc[city0].arc[city1].num_plane-1].customer.addBooked(name2,
    ↪  kindID, ID, num2, state, kind2, ticket);
166         }
167     }
168     getline(strings,ss);
169     if (ss!="NA") {
170         stringstream str(ss);
171         string sss;
172         while (getline(str,sss,',')){
173             istringstream s4(sss);
174             getline(s4,sss,'\t');
175             string name2=sss;
176             getline(s4,sss,'\t');
177             string kindID=sss;
178             getline(s4,sss,'\t');
179             string ID=sss;
180             getline(s4,sss,'\t');
181             string num2=sss;
182             getline(s4,sss,'\t');
183             string kind2=sss;
184             arc[city0].arc[city1].plane[arc[city0].arc[city1].num_plane-1].customer.addAlternative(name2,
    ↪  kindID, ID, num2, kind2);
185         }
186     }
187 }
188
189 void Airplane::Sort(){
190     check_table = check_table_time;
191     for (int j=5; j>1; j--) {
192         Check_Table q[10];
193         Distribute(q,10,j);
194         Collect(q,10);
195     }
196     for (int j=1; j>=0; j--) {
197         Check_Table q[26];
198         Distribute(q,26,j);
199         Collect(q,26);
200     }
```

```
201 }
202
203 void Airplane::Distribute(Check_Table q[], int m, int j){
204     int k;
205     for (int i=0; i<check_table.num_plane; i++) {
206         char
    ↪  c=arc[check_table.plane[i][0]].arc[check_table.plane[i][1]].plane[check_table.plane[i][2]-1].name[j];
207         if (j<2) {
208             k = c;
209             k = k-65;
210         }
211         else{
212             k=atoi(&c);
213         }
214         q[k].addPlane(check_table.plane[i][0], check_table.plane[i][1], check_table.plane[i][2]);
215     }
216 }
217
218 void Airplane::Collect(Check_Table q[], int m){
219     Check_Table c;
220     for (int i=0; i<m; i++) {
221         for (int j=0; j<q[i].num_plane; j++) {
222             c.addPlane(q[i].plane[j][0], q[i].plane[j][1], q[i].plane[j][2]);
223         }
224     }
225     check_table = c;
226 }
227
228 void Airplane::Search(Check_Table &c,string ID,string year,string month,string day,string
    ↪  start,string arrive,string start_city,string arrive_city){
229     cout<<"**************************  直达航班信息  **************************"<<endl;
230     c = check_table;
231     if (ID!="NA") {
232         int i=0,j= c.num_plane-1;
233         int low = 0,high = c.num_plane-1;
234         while (low<high) {
235             int mid = (low+high)>>1;
236             (ID<arc[c.plane[mid][0]].arc[c.plane[mid][1]].plane[c.plane[mid][2]-1].name)?high=mid:low=m
    ↪  id+1;
```

```
237         }
238         while (i<j) {
239             int mid=(i+j)>>1;
240             (ID>arc[c.plane[mid][0]].arc[c.plane[mid][1]].plane[c.plane[mid][2]-1].name)?i=mid+1:j=mid;
241         }
242         Check_Table ct;
243         for(int k=i;k<low;k++)
244             ct.addPlane(c.plane[k][0], c.plane[k][1], c.plane[k][2]);
245         c = ct;
246     }
247     if (year!="NA"&&c.num_plane>0) {
248         Check_Table ct;
249         for (int i=0; i<c.num_plane; i++) {
250             if(year == arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].year){
251                 ct.addPlane(c.plane[i][0], c.plane[i][1], c.plane[i][2]);
252             }
253         }
254         c = ct;
255     }
256     if (month!="NA"&&c.num_plane>0) {
257         Check_Table ct;
258         for (int i=0; i<c.num_plane; i++) {
259             if(month == arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].month){
260                 ct.addPlane(c.plane[i][0], c.plane[i][1], c.plane[i][2]);
261             }
262         }
263         c = ct;
264     }
265     if (day!="NA"&&c.num_plane>0) {
266         Check_Table ct;
267         for (int i=0; i<c.num_plane; i++) {
268             if(day == arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].day){
269                 ct.addPlane(c.plane[i][0], c.plane[i][1], c.plane[i][2]);
270             }
271         }
272         c = ct;
273     }
274     if (start!="NA"&&c.num_plane>0) {
275         Check_Table ct;
```

```
276         for (int i=0; i<c.num_plane; i++) {
277             if(start == arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].start){
278                 ct.addPlane(c.plane[i][0], c.plane[i][1], c.plane[i][2]);
279             }
280         }
281         c = ct;
282     }
283     if (arrive!="NA"&&c.num_plane>0) {
284         Check_Table ct;
285         for (int i=0; i<c.num_plane; i++) {
286             if(arrive == arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].arrive){
287                 ct.addPlane(c.plane[i][0], c.plane[i][1], c.plane[i][2]);
288             }
289         }
290         c = ct;
291     }
292     if (start_city!="NA"&&c.num_plane>0) {
293         Check_Table ct;
294         for (int i=0; i<c.num_plane; i++) {
295             if(start_city == city[c.plane[i][0]].name || start_city == city[c.plane[i][0]].ID){
296                 ct.addPlane(c.plane[i][0], c.plane[i][1], c.plane[i][2]);
297             }
298         }
299         c = ct;
300     }
301     if (arrive_city!="NA"&&c.num_plane>0) {
302         Check_Table ct;
303         for (int i=0; i<c.num_plane; i++) {
304             if(arrive_city == city[c.plane[i][1]].name || arrive_city == city[c.plane[i][1]].ID){
305                 ct.addPlane(c.plane[i][0], c.plane[i][1], c.plane[i][2]);
306             }
307         }
308         c = ct;
309     }
310     if (c.num_plane==0) {
311         cout<<"无相应直达航班！"<<endl;
312     }
313     else{
314         int price=99999;
```

```
315         for (int i=0; i<c.num_plane; i++) {
316             cout<<'('<<i+1<<')'<<endl;
317             cout<<city[c.plane[i][0]].name+city[c.plane[i][0]].airport[arc[c.plane[i][0]].arc[c.plane[i][1]].pla」
    ↪ ne[c.plane[i][2]-1].plat_start]+"
    ↪  "+city[c.plane[i][1]].name+city[c.plane[i][1]].airport[arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plan」
    ↪ e[i][2]-1].plat_arrive]<<endl;
318             cout<<"\t航班号\t\t日期\t\t星期\t\t出发时间\t到达时间\t班期\t机型\t\t折扣\n\t";
319             arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].show();
320             price = min(price,arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].Minprice());
321         }
322         cout<<"最小费用航班为:";
323         for (int i=0; i<c.num_plane; i++) {
324             if(price ==arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].Minprice()){
325                 cout<<'('<<i+1<<')'<<'\t';
326             }
327         }
328         cout<<endl;
329     }
330     bool is = c.num_plane;
331     for (int i=0; i<c.num_plane; i++) {
332         for (int j=0; j<arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].num; j++) {
333             if (arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].num_max[j]>arc[c.plane[i][0]].arc[c.」
    ↪  plane[i][1]].plane[c.plane[i][2]-1].num_now[j])
    ↪  {
334                 is = false;
335                 break;
336             }
337         }
338     }
339     if (is || (ID=="NA" && start_city!="NA" && arrive_city!="NA")) {
340         cout<<"**************************  中转航班信息  **************************"<<endl;
341         Transfer(c,year,month,day,start_city,arrive_city);
342     }
343 }
344
345 void Airplane::Transfer(Check_Table &c0, string year, string month, string day, string start_city,
    ↪  string arrive_city){
346     Check_Table c;
347     c=check_table_time;
```

```
348     int n = c0.num_plane;
349     if (year!="NA" && c.num_plane>0) {
350         Check_Table ct;
351         for (int i=0; i<c.num_plane; i++) {
352             if(year == arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].year){
353                 ct.addPlane(c.plane[i][0], c.plane[i][1], c.plane[i][2],
    ↪  c.time[i][0],c.time[i][1],c.time[i][2],c.time[i][3]);
354             }
355         }
356         c = ct;
357     }
358     if (month!="NA" && c.num_plane>0) {
359         Check_Table ct;
360         for (int i=0; i<c.num_plane; i++) {
361             if(month == arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].month){
362                 ct.addPlane(c.plane[i][0], c.plane[i][1], c.plane[i][2],
    ↪  c.time[i][0],c.time[i][1],c.time[i][2],c.time[i][3]);
363             }
364         }
365         c = ct;
366     }
367     if (day!="NA" && c.num_plane>0) {
368         Check_Table ct;
369         for (int i=0; i<c.num_plane; i++) {
370             if(day == arc[c.plane[i][0]].arc[c.plane[i][1]].plane[c.plane[i][2]-1].day){
371                 ct.addPlane(c.plane[i][0], c.plane[i][1], c.plane[i][2],
    ↪  c.time[i][0],c.time[i][1],c.time[i][2],c.time[i][3]);
372             }
373         }
374         c = ct;
375     }
376     string year1=c.time[0][0],month1=c.time[0][1],day1=c.time[0][2];
377     Check_Table ct[num_city+1];
378     ct[num_city] = c0;
379     int num = c0.num_plane;
380     int price;
381     for (int i=0; i< c.num_plane; i++) {
382         string year2=check_table_time.time[i][0],month2=check_table_time.time[i][1],day2=check_ta
    ↪  ble_time.time[i][2];
```

```
383         if (year1+month1+day1!=year2+month2+day2) {
384             price = SearchTransfer(ct,start_city,arrive_city);
385             if (price<9999) {
386                 for (int j=num; j<ct[num_city].num_plane; j++) {
387                     cout<<'('<<num+j+1<<')'<<endl;
388                     cout<<city[citytonum[start_city]].name+city[citytonum[start_city]].airport[arc[cityto」
    ↪ num[start_city]].arc[ct[num_city].plane[j][1]].plane[ct[num_city].plane[j][0]-1].plat_start]+"
    ↪  "+city[ct[num_city].plane[j][1]].name+city[ct[num_city].plane[j][1]].airport[arc[citytonum[start」
    ↪ _city]].arc[ct[num_city].plane[j][1]].plane[ct[num_city].plane[j][0]-1].plat_arrive]+"
    ↪ "+city[citytonum[arrive_city]].name+city[citytonum[arrive_city]].airport[arc[ct[num_city].plane」
    ↪ [j][1]].arc[citytonum[arrive_city]].plane[ct[num_city].plane[j][2]-1].plat_arrive]<<endl;
389
    ↪ cout<<"\t航班号\t\t日期\t\t星期\t\t出发时间\t到达时间\t班期\t\t机型\t\t折扣\n\t";
390
    ↪ arc[citytonum[start_city]].arc[ct[num_city].plane[j][1]].plane[ct[num_city].plane[j][0]-1].show();
391
    ↪ cout<<"\t航班号\t\t日期\t\t星期\t\t出发时间\t到达时间\t班期\t\t机型\t\t折扣\n\t";
392
    ↪ arc[ct[num_city].plane[j][1]].arc[citytonum[arrive_city]].plane[ct[num_city].plane[j][2]-1].show();
393                     cout<<endl;
394                 }
395                 cout<<"该日最低费用为:"<<price<<endl;
396                 year1=year2;month1=month2;day1=day2;
397             }
398             num = ct[num_city].num_plane;
399             for (int i=0; i<num_city; i++) {
400                 ct[i].Clear();
401             }
402         }
403         ct[c.plane[i][0]].addPlane(c.plane[i][0], c.plane[i][1], c.plane[i][2]);
404     }
405     price = SearchTransfer(ct,start_city,arrive_city);
406     if (price<9999) {
407         cout<<year1<<'.'<<month1<<'.'<<day1<<endl;
408         for (int i=num; i<ct[num_city].num_plane; i++) {
409             cout<<'('<<num+i+1<<')'<<endl;
```

```
410             cout<<city[citytonum[start_city]].name+city[citytonum[start_city]].airport[arc[citytonum[s
    tart_city]].arc[ct[num_city].plane[i][1]].plane[ct[num_city].plane[i][0]-1].plat_start]+"
    "+city[ct[num_city].plane[i][1]].name+city[ct[num_city].plane[i][1]].airport[arc[citytonum[start_
    city]].arc[ct[num_city].plane[i][1]].plane[ct[num_city].plane[i][0]-1].plat_arrive]+"
    "+city[citytonum[arrive_city]].name+city[citytonum[arrive_city]].airport[arc[ct[num_city].plane
    [i][1]].arc[citytonum[arrive_city]].plane[ct[num_city].plane[i][2]-1].plat_arrive]<<endl;
411             cout<<"\t航班号\t\t日期\t\t星期\t\t出发时间\t到达时间\t班期\t\t机型\t\t折扣\n\t";
412
    arc[citytonum[start_city]].arc[ct[num_city].plane[i][1]].plane[ct[num_city].plane[i][0]-1].show();
413             cout<<"\t航班号\t\t日期\t\t星期\t\t出发时间\t到达时间\t班期\t\t机型\t折扣\n\t";
414
    arc[ct[num_city].plane[i][1]].arc[citytonum[arrive_city]].plane[ct[num_city].plane[i][2]-1].show();
415             cout<<endl;
416         }
417         cout<<"该日最低费用为:"<<price<<endl;
418     }
419     num = ct[num_city].num_plane;
420     c0 = ct[num_city];
421     if (n==c0.num_plane) {
422         cout<<"无相应中转航班!"<<endl;
423     }
424 }
425
426 int Airplane::SearchTransfer(Check_Table c[], string start_city, string arrive_city){
427     Check_Table ct;
428     int price=9999;
429     int start=citytonum[start_city],arrive=citytonum[arrive_city];
430     for (int i=0; i<c[start].num_plane; i++) {
431         int mid = c[start].plane[i][1];
432         for (int j=0; j<c[mid].num_plane; j++) {
433             if (c[mid].plane[j][1]==arrive && arc[start].arc[mid].plane[c[start].plane[i][2]-1].arrive<=arc[
    mid].arc[arrive].plane[c[mid].plane[j][2]-1].start)
    {
434                 if (arc[start].arc[mid].plane[c[start].plane[i][2]-1].Minprice()+arc[mid].arc[arrive].plane[c[m
    id].plane[j][2]-1].Minprice()<price)
    {
435                     price=arc[start].arc[mid].plane[c[start].plane[i][2]-1].Minprice()+arc[mid].arc[arrive].pla
    ne[c[mid].plane[j][2]-1].Minprice();
436                     ct.Clear();
```

```
437                     ct.addPlane(c[start].plane[i][2], mid, c[mid].plane[j][2]);
438                 }
439                 else if (arc[start].arc[mid].plane[c[start].plane[i][2]-1].Minprice()+arc[mid].arc[arrive].plane
    ↪  [c[mid].plane[j][2]-1].Minprice()==price){
440                     ct.addPlane(c[start].plane[i][2], mid, c[mid].plane[j][2]);
441                 }
442             }
443         }
444     }
445     for (int i=0; i<ct.num_plane; i++) {
446         c[num_city].addPlane(ct.plane[i][0], ct.plane[i][1], ct.plane[i][2]);
447     }
448     return price;
449 }
450
451 int Airplane::addBooked(int m, int n, int l, string name, string kindID, string ID, string book_num,
    ↪  string state, string kind, string ticket){
452     int i=0;
453     for (; i<arc[m].arc[n].plane[l-1].num; i++) {
454         if (arc[m].arc[n].plane[l-1].kind[i]==kind) {
455             break;
456         }
457     }
458     int num=atoi(book_num.c_str());
459     if (arc[m].arc[n].plane[l-1].num_max[i]-arc[m].arc[n].plane[l-1].num_now[i]>=num) {
460         arc[m].arc[n].plane[l-1].customer.addBooked(name, kindID, ID, book_num, state, kind, ticket);
461         return num;
462     }
463     arc[m].arc[n].plane[l-1].customer.addAlternative(name, kindID, ID, book_num, kind);
464     return 0;
465 }
466
467 int Airplane::addAlternative(int m, int n, int l, string name,string kindID,string ID,string
    ↪  book_num,string kind){
468     arc[m].arc[n].plane[l-1].customer.addAlternative(name,kindID,ID,book_num,kind);
469     return atoi(book_num.c_str());
470 }
471
472 void Airplane::showBooked(int m, int n, int l){
```

```
473     arc[m].arc[n].plane[l-1].customer.showBooked();
474 }
475
476 void Airplane::showAlternative(int m, int n, int l){
477     arc[m].arc[n].plane[l-1].customer.showAlternative();
478 }
479
480 string Airplane::deleteBooked(int m, int n, int l, string name, string ID){
481     int i=0;
482     for (; i<arc[m].arc[n].plane[l-1].customer.num_booked; i++) {
483         if (arc[m].arc[n].plane[l-1].customer.booked[i].name==name &&
    ↪  arc[m].arc[n].plane[l-1].customer.booked[i].ID==ID) {
484             break;
485         }
486     }
487     if (i==arc[m].arc[n].plane[l-1].customer.num_booked) {
488         return "";
489     }
490     int num_booked = atoi(arc[m].arc[n].plane[l-1].customer.booked[i].num.c_str());
491     string kind = arc[m].arc[n].plane[l-1].customer.deleteBooked(i);
492     int j=0;
493     for (j=0; j<arc[m].arc[n].plane[l-1].num; j++) {
494         if (kind==arc[m].arc[n].plane[l-1].kind[j]) {
495             break;
496         }
497     }
498     arc[m].arc[n].plane[l-1].num_now[j] -= num_booked;
499     return kind;
500 }
501
502 void Airplane::informAlternative(int m, int n, int l,string kind){
503
504     int i;
505     for (i=0; i<arc[m].arc[n].plane[l-1].num; i++) {
506         if (arc[m].arc[n].plane[l-1].kind[i]==kind) {
507             break;
508         }
509     }
510     ofstream file("/Users/dujinhong/Documents/study/数据结构/数据结构实验/大作业/airplane/airp」
    ↪  lane/log.txt",ios::app);
```

```
511     file<<"航班"<<arc[m].arc[n].plane[l-1].name<<'\t'<<arc[m].arc[n].plane[l-1].year<<'.'<<arc[m].
    ↪  arc[n].plane[l-1].month<<'.'<<arc[m].arc[n].plane[l-1].day<<'\t'<<kind<<"有余票"<<arc[m].a
    ↪  rc[n].plane[l-1].num_max[i]-arc[m].arc[n].plane[l-1].num_now[i]<<"张,"<<"替补客户有:\n";
512     file.close();
513     arc[m].arc[n].plane[l-1].customer.informAlternative(kind);
514 }
515
516 void Airplane::Save(){
517     ofstream fileOut("/Users/dujinhong/Documents/study/数据结构/数据结构实验/大作业/airplane/
    ↪  airplane/plane.txt");
518     if (fileOut.is_open()) {
519         string year=" ",month=" ",day=" ",date=" ";
520         for (int i=0; i<check_table_time.num_plane; i++) {
521             string year2=check_table_time.time[i][0],month2=check_table_time.time[i][1],day2=check
    ↪  _table_time.time[i][2],date2=check_table_time.time[i][3];
522             if (year!=year2 || month!=month2 || day!=day2 || date !=date2) {
523                 if (i>0) {
524                     fileOut<<'/'<<'\n';
525                 }
526                 fileOut<<year2<<'\t'<<month2<<'\t'<<day2<<'\t'<<date2<<'\n';
527                 year = year2; month = month2; day = day2; date = date2;
528             }
529             else{
530                 fileOut<<'\n';
531             }
532             fileOut<<arc[check_table_time.plane[i][0]].arc[check_table_time.plane[i][1]].plane[check_t
    ↪  able_time.plane[i][2]-1].Save(city[check_table_time.plane[i][0]].ID,city[check_table_time.plane[i]
    ↪  [1]].ID);
533             fileOut<<'#';
534         }
535         fileOut<<'/';
536     }
537 }
538
539 void Airplane::Change(int m, int n, int l){
540     cout<<"-----------------------------------------------------------------"<<endl;
541     cout<<"请输入航班信息（'NA'跳过,'-1'返回）:"<<endl;
542     string in;
543     cout<<"起飞时间:";cin>>in;
```

```
544     if (in==”-1”) {
545         return;
546     }
547     else if (in!=”NA”) {
548         arc[m].arc[n].plane[l-1].start = in;
549     }
550     cout<<”到达时间:”;cin>>in;
551     if (in==”-1”) {
552         return;
553     }
554     else if (in!=”NA”) {
555         arc[m].arc[n].plane[l-1].arrive = in;
556     }
557     cout<<”折扣:”;cin>>in;
558     if (in==”-1”) {
559         return;
560     }
561     else if (in!=”NA”) {
562         arc[m].arc[n].plane[l-1].discount = in;
563     }
564     cout<<”舱位类型数:”;cin>>in;
565     if (in==”-1”) {
566         return;
567     }
568     else if (in!=”NA”) {
569         arc[m].arc[n].plane[l-1].num = atoi(in.c_str());
570         for (int i=0; i<arc[m].arc[n].plane[l-1].num; i++) {
571             cout<<”舱位类型”<<i+1<<”:”;
572             cin>>arc[m].arc[n].plane[l-1].kind[i];
573             cout<<”价格:”;
574             cin>>arc[m].arc[n].plane[l-1].price[i];
575             cout<<”已订人数:”;
576             cin>>arc[m].arc[n].plane[l-1].num_now[i];
577             cout<<”满载人数:”;
578             cin>>arc[m].arc[n].plane[l-1].num_max[i];
579         }
580     }
581 }
582
```

```
583 void Airplane::DeletePlane(int m, int n, int l){
584     if (arc[m].arc[n].num_plane==l) {
585         check_table.num_plane--;
586         check_table_time.num_plane--;
587     }
588     else{
589         arc[m].arc[n].plane[l] = arc[m].arc[n].plane[arc[m].arc[n].num_plane-1];
590         int i;
591         for (i=0; i<check_table.num_plane; i++) {
592             if (check_table.plane[i][0]==m && check_table.plane[i][1]==n &&
    ↪  check_table.plane[i][2]==l) {
593                 break;
594             }
595         }
596         for (int j=i; i<check_table.num_plane-1; j++) {
597             for (int k=0; k<3; k++) {
598                 check_table.plane[j][k] = check_table.plane[j+1][k];
599             }
600             for (int k=0; k<4; k++) {
601                 check_table_time.time[j][k] = check_table_time.time[j+1][k];
602             }
603         }
604         check_table.num_plane--;
605         for (i=0; i<check_table_time.num_plane; i++) {
606             if (check_table_time.plane[i][0]==m && check_table_time.plane[i][1]==n &&
    ↪  check_table_time.plane[i][2]==l) {
607                 break;
608             }
609         }
610         for (int j=i; i<check_table_time.num_plane-1; j++) {
611             for (int k=0; k<3; k++) {
612                 check_table_time.plane[j][k] = check_table_time.plane[j+1][k];
613             }
614             for (int k=0; k<4; k++) {
615                 check_table_time.time[j][k] = check_table_time.time[j+1][k];
616             }
617         }
618         check_table_time.num_plane--;
619     }
```

```
620 }
621
622 void Airplane::show(int i, int j, int k){
623    cout<<arc[i].arc[j].plane[k-1].name<<','<<arc[i].arc[j].plane[k-1].year<<'.'<<arc[i].arc[j].plane[k-
    ↪  1].month<<'.'<<arc[i].arc[j].plane[k-1].day<<','<<arc[i].arc[j].plane[k-1].date<<','<<arc[i].arc[j
    ↪  ].plane[k-1].start<<'-'<<arc[i].arc[j].plane[k-1].arrive<<endl;
624 }
```

4. *City.cpp*

```
1  #include "airplane.hpp"
2  City::City():num_airport(0),max_num(1){
3      airport = new string[max_num];
4  }
5
6  void City::addAirport(string s){
7      if(max_num==num_airport){
8          string *p;
9          max_num = 2*max_num;
10         p = new string[max_num];
11         for (int i=0; i<num_airport; i++) {
12             p[i] = airport[i];
13         }
14         delete[] airport;
15         airport = p;
16     }
17     airport[num_airport++] = s;
18 }
19
20 City::~City(){
21     delete[] airport;
22 }
```

5. *Check_Table.cpp*

```
#include "airplane.hpp"
Check_Table::Check_Table():num_plane(0),max_num(1){
    plane = new int*[max_num];
    time = new string*[max_num];
    for (int i=0; i<max_num; i++) {
        plane[i] = new int[3]();
        time[i] = new string[4];
    }
}

Check_Table::~Check_Table(){
    if (max_num>0) {
        for (int i=0; i<max_num; i++) {
            delete[] plane[i];
            delete[] time[i];
        }
        delete[] plane;
        delete[] time;
    }
}

Check_Table& Check_Table::operator=(Check_Table &c){
    if (max_num>0) {
        for (int i=0; i<max_num; i++) {
            delete[] plane[i];
            delete[] time[i];
        }
        delete[] plane;
        delete[] time;
    }
    num_plane = c.num_plane;max_num = c.max_num;
    plane = new int*[max_num];
    time = new string*[max_num];
    for (int i=0; i<max_num; i++) {
        plane[i] = new int[3];
        time[i] = new string[4];
    }
    for (int i=0; i<num_plane; i++) {
        for (int j=0; j<3; j++) {
```

```
40              plane[i][j]=c.plane[i][j];
41          }
42          for (int j=0; j<4; j++) {
43              time[i][j] = c.time[i][j];
44          }
45      }
46      return *this;
47  }
48
49  void Check_Table::addPlane(int m, int n, int o,string year/*=""*/,string month/*=""*/,string
    ↪ day/*=""*/,string date/*=""*/){
50      if (num_plane==max_num) {
51          int** p;
52          string** q;
53          max_num = 2*max_num;
54          p = new int*[max_num];
55          q = new string*[max_num];
56          for (int i=0; i<max_num; i++) {
57              p[i] = new int[3];
58              q[i] = new string[4];
59          }
60          for (int i=0; i<num_plane; i++) {
61              for (int j=0; j<3; j++) {
62                  p[i][j] = plane[i][j];
63              }
64              for (int j=0; j<4; j++) {
65                  q[i][j] = time[i][j];
66              }
67          }
68          for (int i=0; i<max_num/2; i++) {
69              delete[] plane[i];
70              delete[] time[i];
71          }
72          delete[] plane;
73          delete[] time;
74          plane = p;
75          time = q;
76      }
77      plane[num_plane][0] = m;
```

```
78      plane[num_plane][1] = n;
79      plane[num_plane][2] = o;
80      time[num_plane][0] = year;
81      time[num_plane][1] = month;
82      time[num_plane][2] = day;
83      time[num_plane++][3] = date;
84
85  }
86
87  void Check_Table::Clear(){
88      if (max_num>0) {
89          for (int i=0; i<max_num; i++) {
90              delete[] plane[i];
91              delete[] time[i];
92          }
93          delete[] plane;
94          delete[] time;
95      }
96      num_plane = 0;
97      max_num = 1;
98      plane = new int*[max_num];
99      time = new string*[max_num];
100     for (int i=0; i<max_num; i++) {
101         plane[i] = new int[3];
102         time[i] = new string[4];
103     }
104 }
```

6. *CitytoNum.cpp*

```
1   #include "airplane.hpp"
2
3   CitytoNum::CitytoNum():num(0){
4   }
5
6   CitytoNum::~CitytoNum(){
7       if (num>0) {
8           delete[] name;
```

```
        delete [] ID;
    }
}

int CitytoNum::operator[](string cityname){
    for (int i=0; i<num; i++) {
        if (cityname==name[i] || cityname==ID[i]) {
            return i;
        }
    }
    return -1;
}

void CitytoNum::addcity(City *city, int num){
    if (this->num>0) {
        delete[] name;
        delete [] ID;
    }
    this->num=num;
    name = new string[num];
    ID = new string[num];
    for (int i=0; i<num; i++) {
        name[i] = city[i].name;
        ID[i] = city[i].ID;
    }
}
```

7. *Arc.cpp*

```
#include "airplane.hpp"
void Arc::Init(int size){
    arc = new Edge[size];
    this->size = size;
    for (int i=0; i<size; i++) {
        arc->Init();
    }
}

```

```
Arc::~Arc(){
    if (size>0) {
        delete[] arc;
    }
}
```

8. *Edge.cpp*

```
#include "airplane.hpp"
Edge::Edge(){
    Init();
}

Edge::~Edge(){
    if (num_plane>0) {
        delete[] plane;
    }

}

void Edge::Init(){
    num_plane = 0;
    max_num = 1;
    plane = new Plane[max_num];
}

void Edge::addPlane(Plane &p){
    if (num_plane==max_num) {
        Plane *q;
        max_num = 2*max_num;
        q = new Plane[max_num];
        for (int i=0; i<num_plane; i++) {
            q[i] = plane[i];
        }
        if (max_num/2>0) {
            delete[] plane;
        }
        plane = q;
```

```
31      }
32      plane[num_plane++] = p;
33  }
```

9. *Plane.cpp*

```
1   #include "airplane.hpp"
2
3
4   Plane::Plane(string name,string year,string month,string day,string date,string schedule,string
    ↪  start,string arrive,string shape,string discount,int num, string* kind,int* price,int* num_now,int*
    ↪  num_max,int plat_start,int plat_arrive):name(name),year(year),month(month),day(day),date(
    ↪  date),schedule(schedule),start(start),arrive(arrive),shape(shape),discount(discount),num(num),pl
    ↪  at_start(plat_start),plat_arrive(plat_arrive){
5       this->kind = new string[num];
6       this->price = new int[num];
7       this->num_now = new int[num];
8       this->num_max = new int[num];
9       for (int i=0; i<num; i++) {
10          this->kind[i]=kind[i];
11          this->price[i]=price[i];
12          this->num_now[i]=num_now[i];
13          this->num_max[i]=num_max[i];
14      }
15  }
16
17  Plane::~Plane(){
18      if (num>0) {
19          delete[] kind;
20          delete[] price;
21          delete[] num_max;
22          delete[] num_now;
23      }
24  }
25
26  Plane& Plane::operator=(Plane &p){
27      name = p.name;
28      year = p.year;
```

```
29      month = p.month;
30      day = p.day;
31      date = p.date;
32      start = p.start;
33      arrive = p.arrive;
34      plat_start = p.plat_start;
35      plat_arrive = p.plat_arrive;
36      schedule =p.schedule;
37      shape = p.shape;
38      discount = p.discount;
39      if (num>0) {
40          delete[] kind;
41          delete[] price;
42          delete[] num_max;
43          delete[] num_now;
44      }
45      num = p.num;
46      kind = new string[num];
47      price = new int[num];
48      num_now = new int[num];
49      num_max = new int[num];
50      for (int i=0; i<num; i++) {
51          kind[i] = p.kind[i];
52          price[i] = p.price[i];
53          num_now[i]=p.num_now[i];
54          num_max[i]=p.num_max[i];
55      }
56      customer = p.customer;
57      return *this;
58  }
59
60  void Plane::show(){
61      cout<<name<<'\t'<<year<<'.'<<month<<'.'<<day<<'\t'<<date<<'\t'<<start<<'\t'<<ar
    ↪   rive<<'\t'<<schedule<<'\t'<<shape<<'\t'<<discount<<'\n';
62      for (int i=0; i<num; i++) {
63          cout<<"\t\t"<<kind[i]<<'\t'<<price[i]<<"
    ↪     "<<'\t'<<num_now[i]<<'/'<<num_max[i]<<endl;
64      }
65  }
```

```
66
67  int Plane::Minprice(){
68      int pri=99999;
69      for (int i=0; i<num; i++) {
70          if (pri>price[i]) {
71              pri=price[i];
72          }
73      }
74      return pri;
75  }
76
77  string Plane::Save(string start_city,string arrive_city){
78      ostringstream s;
79
80      s<<name+'\t'+start_city<<plat_start<<'\t'+arrive_city<<plat_arrive<<'\t'+schedule+'\t'
    ↪   +start+'\t'+arrive+'\t'+shape+'\t'+discount+'\t';
81      for (int i=0; i<num; i++) {
82          s<<kind[i]<<'\t'<<price[i]<<'\t'<<num_now[i]<<'\t'<<num_max[i]<<'\t';
83      }
84      s<<'\n';
85      s<<customer.Save();
86
87      return s.str();
88  }
```

10. *Customer.cpp*

```
1   #include "airplane.hpp"
2
3   Customer::Customer():num_booked(0),num_alternative(0),max_booked(1),max_alternative(1){
4       booked = new Booked[max_booked];
5       alternative = new Alternative[max_alternative];
6   }
7
8   Customer::~Customer(){
9       if(max_booked>0)
10          delete[] booked;
11      if(max_alternative>0)
```

```
12          delete[] alternative;
13  }
14
15  Customer& Customer::operator=(Customer &p){
16      if(max_booked>0)
17          delete[] booked;
18      if(max_alternative>0)
19          delete[] alternative;
20      num_booked = p.num_booked;
21      num_alternative = p.num_alternative;
22      max_booked = p.max_booked;
23      max_alternative = p.max_alternative;
24      booked = new Booked[max_booked];
25      alternative = new Alternative[max_alternative];
26      for (int i =0; i<num_booked; i++) {
27          booked[i] = p.booked[i];
28      }
29      for (int i=0; i<num_alternative; i++) {
30          alternative[i] = p.alternative[i];
31      }
32
33      return *this;
34  }
35
36  void Customer::addBooked(string name,string kindID,string ID,string num,string state,string
    ↪ kind,string ticket){
37      if(max_booked==num_booked){
38          Booked *p;
39          max_booked = 2*max_booked;
40          p = new Booked[max_booked];
41          for (int i=0; i<num_booked; i++) {
42              p[i] = booked[i];
43          }
44          delete[] booked;
45          booked = p;
46      }
47      Booked b(name,kindID,ID,num,state,kind,ticket);
48      booked[num_booked++] = b;
49  }
```

```
50
51  void Customer::addAlternative(string name,string kindID,string ID,string num,string kind){
52      if(max_alternative==num_alternative){
53          Alternative *p;
54          max_alternative = 2*max_alternative;
55          p = new Alternative[max_alternative];
56          for (int i=0; i<num_alternative; i++) {
57              p[i] = alternative[i];
58          }
59          delete[] alternative;
60          alternative = p;
61      }
62      Alternative a(name,kindID,ID,num,kind);
63      alternative[num_alternative++] = a;
64  }
65
66  void Customer::showBooked(){
67      for (int i=0; i<num_booked; i++) {
68          cout<<'('<<i+1<<')';
69          booked[i].show();
70      }
71  }
72
73  void Customer::showAlternative(){
74      for (int i=0; i<num_alternative; i++) {
75          cout<<'('<<i+1<<')';
76          alternative[i].show();
77      }
78  }
79
80  void Customer::informAlternative(string kind){
81      int n=1;
82      ofstream file("/Users/dujinhong/Documents/study/数据结构/数据结构实验/大作业/airplane/air
    ↪ plane/log.txt",ios::app);
83      for (int i=0; i<num_alternative; i++) {
84          if (alternative[i].kind==kind) {
85              file<<'('<<n++<<')'<<alternative[i].inform();
86          }
87      }
```

```
88      file.close();
89  }
90
91  string Customer::deleteBooked(int de){
92      string s=booked[de].kind;
93      if (num_booked>1) {
94          booked[de]=booked[num_booked-1];
95      }
96      num_booked--;
97      return s;
98  }
99
100 string Customer::Save(){
101     ostringstream s;
102     if (num_booked>0) {
103         for (int i=0; i<num_booked; i++) {
104             s<<booked[i].Save()<<',';
105         }
106     }
107     else{
108         s<<"NA";
109     }
110     s<<'\n';
111     if (num_alternative>0) {
112         for (int i=0; i<num_alternative; i++) {
113             s<<alternative[i].Save()<<',';
114         }
115     }
116     else{
117         s<<"NA";
118     }
119     return s.str();
120 }
```

11. *Booked.cpp*

```
1   #include "airplane.hpp"
2
```

```
3  Booked::Booked(string name,string kindID,string ID,string num,string state,string kind,string
   ↪  ticket):name(name),kindID(kindID),ID(ID),num(num),state(state),kind(kind),ticket(ticket){
4  }
5
6  Booked& Booked::operator=(Booked &b){
7     name=b.name;
8     kindID=b.kindID;
9     ID=b.ID;
10    num=b.num;
11    state=b.state;
12    kind=b.kind;
13    ticket=b.ticket;
14    return *this;
15 }
16
17 void Booked::show(){
18    cout<<name<<'\t'<<kindID<<'\t'<<ID<<'\t'<<num<<'\t'<<state<<'\t'<<kind<<'\t'<
   ↪  <ticket<<endl;
19 }
20
21 string Booked::Save(){
22    ostringstream s;
23    s<<name<<'\t'<<kindID<<'\t'<<ID<<'\t'<<num<<'\t'<<state<<'\t'<<kind<<'\t'<<tic
   ↪  ket;
24    return s.str();
25 }
```

12. *Alternative.cpp*

```
1  #include "airplane.hpp"
2
3  Alternative::Alternative(string name,string kindID,string ID,string num,string
   ↪  kind):name(name),kindID(kindID),ID(ID),num(num),kind(kind){}
4
5  Alternative& Alternative::operator=(Alternative &b){
6     name=b.name;
7     kindID=b.kindID;
8     ID=b.ID;
```

```
9     num=b.num;
10    kind=b.kind;
11    return *this;
12  }
13
14  void Alternative::show(){
15    cout<<inform();
16  }
17
18  string Alternative::inform(){
19    return name+'\t'+kindID+'\t'+ID+'\t'+'\t'+kind+'\n';
20  }
21
22  string Alternative::Save(){
23    ostringstream s;
24    s<<name<<'\t'<<kindID<<'\t'<<ID<<'\t'<<num<<'\t'<<kind;
25    return s.str();
26  }
```