

**数据结构实验报告**

**--飞机票管理系统**

**学院： 计算机科学学院**

**班级：计算机科学与技术1802班**

**姓名： 段星雨41809010218**

**王怡萱41809010225**

**张籽程41809010202**

**史 强41809010204**

**肖 媛41809010220**

**时间： 2019年1月6日**

**目录**

[一、 需求分析 1](#_Toc29295970)

[二、 概要设计 1](#_Toc29295971)

[三、 详细设计 2](#_Toc29295972)

[四、 调试分析 4](#_Toc29295973)

[五、 使用说明 5](#_Toc29295974)

[六、 测试结果 6](#_Toc29295975)

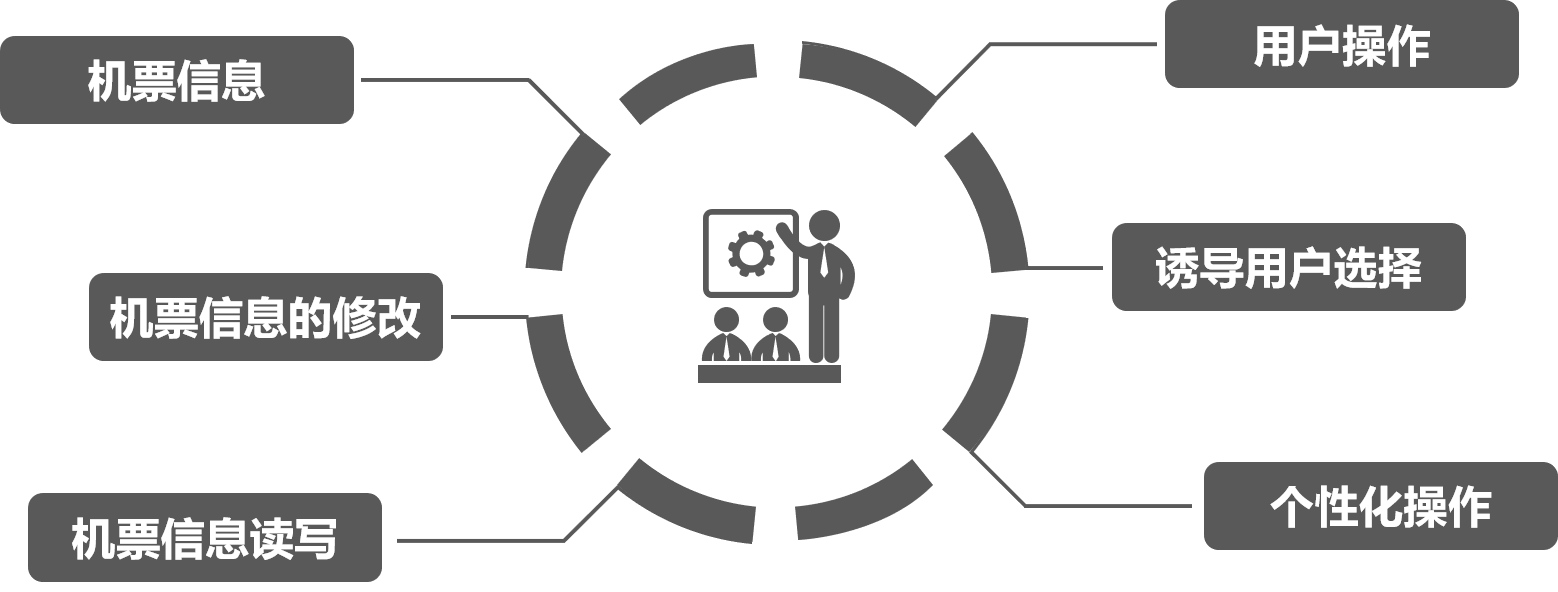
[七、 设计心得 9](#_Toc29295976)

[八、 小组分工 10](#_Toc29295977)

[九、 附录 10](#_Toc29295978)

# 需求分析

飞机票管理系统是一个集成的环境，可以记录机票信息，而且可以修改信息，也可以读取文件中的机票信息，同时也可以将信息写入文件保存。

小组成员通过商讨，将整个系统分为两个大类：机票信息和用户操作

1. 机票信息：对机票的操作，分为储存，管理，修改，读写等
2. 用户操作：使用人性化方式引导用户来进行选择，并转到对应接口

# 概要设计

本程序采用层次编程，并对相关类进行封装，保证程序的精简、易懂、方便调试。

整个程序分为四个层次，层层相套。最底层为airplane，用来储存航班信息，并对其进行相关操作，然后是list，用来构造链表，并对其进行操作，再然后是global，集成了list的方法，然后再次封装，集成外层操作，最后是main，用于程序的开始，导入外层包global，让程序可以编译运行。

# 详细设计

**底层：**

1. **Airplane**

**（1）结构体**

typedef struct airplane {

……

} airplane\_t;

机票信息结构体：储存机票的相关信息。

**（2）set/get方法**

void set\_id(airplane\_t\* p\_air, unsigned id);

void set\_route(airplane\_t\* p\_air, char route[]);

void set\_nub(airplane\_t\* p\_air, unsigned nub);

void set\_time(airplane\_t\* p\_air, char time[]);

void set\_price(airplane\_t\* p\_air, unsigned price);

封装航班类，对航班信息修改，函数参数为操作的结构体和要修改的数据值。

void get\_info(airplane\_t\* p\_air);

获取键盘输入，将相关信息传入结构体，参数为操作的结构体。

（3）其他方法

void show(airplane\_t\* p\_air);

对结构体信息进行输出，方法参数为操作的结构体

1. **List**

**（1）结构体**

typedef struct node {

airplane\_t air;

struct node\* p\_next;

} node\_t;

储存航班信息的链表，data为航班信息结构体，next为指向下个节点的指针。

typedef struct list {

node\_t\* head;

unsigned count;

} list\_t;

因为链表的头结点不为空，data域储存数据，所以head指向第一个存数据的节点，无

数据则为NULL，count为计数单元，是节点总个数。

**（2）方法**

void init\_list(list\_t\* p\_list);

初始化链表操作，方法参数为list结构体。

int insert\_node(list\_t\* p\_list, airplane\_t\* p\_air);

插入节点操作，将数据插入链表，方法参数为list结构体和航班信息结构体。

void traverse\_list(list\_t\* p\_list);

遍历链表操作，方法参数为list结构体。

void destroy\_list(list\_t\* p\_list);

销毁链表操作，将链表销毁，释放内存，方法参数为list结构体。

int remove\_node(list\_t\* p\_list, unsigned id);

删除节点（以id方式），对id进行查找，然后删除该节点，返回操作的结果，方法参

数为list结构体和无符号数id。

node\_t\* find\_node(list\_t\* p\_list, unsigned id);

查询节点（以id方式），通过id查找节点，然后返回该节点的地址，方法参数为list结

构体和无符号数id。

**外部：**

**1.globa**

**（1）结构体**

typedef struct all\_data {

unsigned choice;// 保存菜单选项编号

char file\_name[20];//文件名

list\_t air\_list;// 存储航线信息用的链表

} all\_data\_t;

总数据结构体，储存当前要使用的文件名，存储航线信息用的链表，还有外部输入菜单

选项。

**（2）方法**

void start(all\_data\_t\*);

开始方法，用于程序的开始，函数参数为总数据结构体。

void make\_choice(all\_data\_t\*);

菜单方法，输出菜单的方法并记录操作选项，方法参数为总数据结构体。

void do\_trans(all\_data\_t\*);

根据选择调用相应的方法，方法参数为总数据结构体。

void show\_list(all\_data\_t \*);

输出链表内的数据，方法参数为总数据结构体。

void add\_record(all\_data\_t\*);

添加数据方法，方法参数为总数据结构体。

void delete\_record(all\_data\_t\*);

删除数据方法，方法参数为总数据结构体。

void find\_record\_by\_id(all\_data\_t\*);

寻找节点方法，方法参数为总数据结构体。

void modify\_item(all\_data\_t\*);

修改数据方法，方法参数为总数据结构体。

void quit(all\_data\_t\*);

退出程序方法，方法参数为总数据结构体。

void read\_db(all\_data\_t\*);

读取方法，从文件中读取数据，方法参数为总数据结构体。

void write\_db(all\_data\_t\*);

写入方法，将数据写入文件，方法参数为总数据结构体。

void init\_struct(all\_data\_t\*,char[]);

初始化函数，初始化总数据结构体，方法参数为总数据结构体和文件名字符数组。

# 调试分析

**1.问题发现**

**（1）**在将链表传入到方法是，方法结束后发现链表节点未被修改或录入数据，经多次检查程序代码和断点调试后发现，链表节点未被初始化，所以导致问题的出现。

**（2）**在进行程序调试时发现，程序多次输出菜单，导致cmd中输出过多影响美观，在查找资料后，发现system（“cls”）清屏函数，然后while（1）无限循环输出菜单满足了我们的所需。

**（3）**在c文件中对自定义头文件中接口进行实现的时候，编译提示头文件接口未实现，经多次查看程序后发现，在c文件中未导入头文件，之后修改问题解除。

**（4）**在执行选择操作是，发现程序会依次执行之后的选项，结合所学知识发现是switch中case未加break语句。

**2.复杂度分析**

**（1）时间复杂度**

大部分函数时间复杂度均为O(1)，牵扯到寻找节点的的方法时，最好时间复杂度为O(1),最坏时间复杂度为O(n)。

**（2）空间复杂度**

空间复杂度随着输入数据变多增加，是一个变化的过程。从最初从文件中录入的n个数据为O(n),然后录入m个数据，空间复杂度变为O(m+n)。

# 使用说明

打开程序后会有如下的选择界面，然后输入对应的选项。总共有以下五个选项，根据选择执行对应的程序来执行操作。

**0.显示所有航班信息**

**1.添加航班信息**

**2.删除航班信息**

**3.查找航班信息**

**4.修改航班信息**

**5.退出系统**

0.显示所有航班信息：

遍历输出航班的信息，若是开始程序未进行其他操作，则输出的为文件中的数据。

1.添加航班信息：

提示输入航班的五项数据，输入完成后会将数据存储。

2.删除航班信息。

提示要删除的航班信息，若输出正确序号则提示删除成功，若未找到航班，则提示删除失败。

3.查找航班信息

提示输入id进行查找，查找到输出航班信息，若未查到则提示未找到。

4.修改航班信息

提示输入要修改航班的的id，若找到航班，则提示修改哪项数据，修改完成后输出新的航班信息，提示是否继续修改；若未找到航班，则提示未找到。

5.退出系统

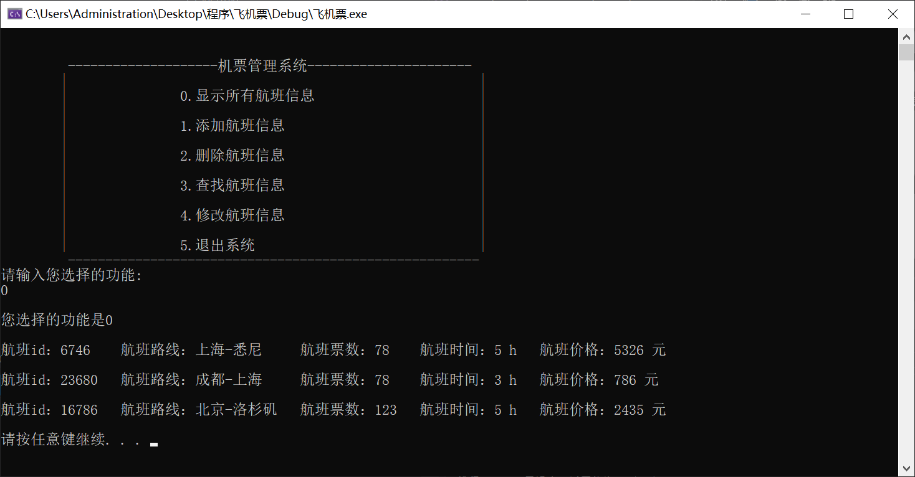
直接退出程序，执行完退出系统操作后，按任意键则可关闭窗口，并将数据写入文件中。

# 测试结果

**开始菜单**



**显示所有航班信息**

****

**添加航班信息**

****

**删除航班信息**

****

**查找航班信息**

****

****

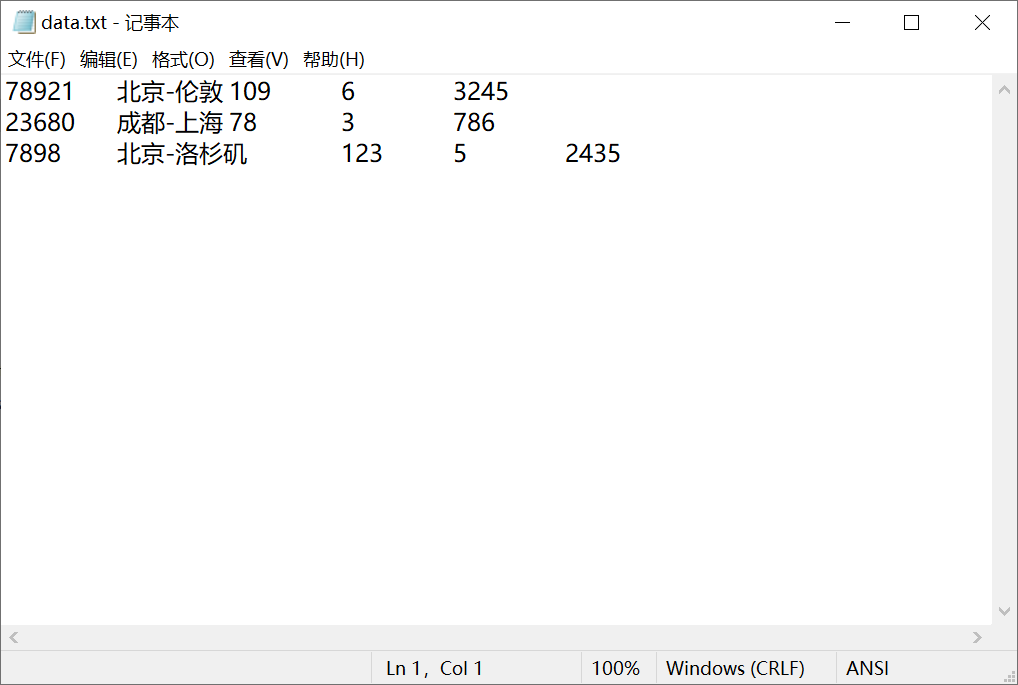
**修改航班信息**

****

**退出系统**



**最终文件内容**

****

# 设计心得

做了一周的课程设计，有很多的心得体会。本次的任务原则上是设计，其实就是一次将所学知识融会贯通的大作业，是对书本知识的熟悉和应用，更多的也是锻炼同学之间的合作精神，促进了同学之间的相互探讨和相互学习。

在我们小组有解决不了的问题时，我们会主动查阅相关的资料，或向其他同学询问，这

不仅丰富了我们的知识，还增进了我们同学之间的友谊。为了增大信息的安全性，需要用文件来存储信息，由于我们在.上课时不注重对文件的运用，所以在这方面有较大的困难。我先将书本认认真真地看了一遍，又做了一下课后习题来验证和增进自己的理解，终于，经过我们的不懈努力，我们小组的程序有了突破，成功地实现了用文件来保存并查看学生的信息。

这道课程设计花了我们一周的时间，通过几人合作才完成。在这次设计中，我遇到了很多困难，但我没有向困难低头，通过查询资料，虚心向同学请教，才一一得到了解决。我发现平时学的太少了，在今后我首先要把老师在课堂讲的弄懂，弄透，在此基础上利用课余时间多看课外书籍，拓宽知识面。在设计过程中，合作精神必不可少。

通过此次课程设计，加深了我们对知识的认识，也强化了我们对知识的应用，最重要的是对以往知识疏漏的补充，为我们以后的学习和工作打下了坚实的基础。此次课程设计让我明白了一个深刻的道理，团队精神固然重要，但也往往要靠个人的努力，只有自己亲身去实践和经历，才能学到更多的东西，心里才会更踏实。总之，这次课程设计，让我受益良多，感谢学校和老师给我们这次课程设计的机会，诚挚的感谢我们的指导老师，在设计的过程中老师的精心辅导和不厌其烦的答疑才使我们顺利完成的了此次设计，她无私的奉献精神也促使我们更加努力地学习，增加了我们对知识的追求和欲望。

# 小组分工

段星雨 ：global编写，程序组合、修改、调试，实验报告撰写

王怡萱 ：list编写

张籽程 ：list，global编写

史 强 ：airplane编写

肖 媛 ：airplane编写

# 附录

**1.airplane.h**

#ifndef \_\_AIRPLANE\_H\_\_

#define \_\_AIRPLANE\_H\_\_

// 定义航班信息结构体

typedef struct airplane {

unsigned id;//编号

char route[15];//路线

unsigned nub;//票数

char time[8];//时间

unsigned price;//价格

} airplane\_t;

//修改航班数据函数

void set\_id(airplane\_t\* p\_air, unsigned id);

void set\_route(airplane\_t\* p\_air, char route[]);

void set\_nub(airplane\_t\* p\_air, unsigned nub);

void set\_time(airplane\_t\* p\_air, char time[]);

void set\_price(airplane\_t\* p\_air, unsigned price);

// 获取键盘录入的航班信息

void get\_info(airplane\_t\* p\_air);

// 输出显示航班结构体

void show(airplane\_t\* p\_air);

#endif // \_\_AIRPLANE\_H\_\_

**2.airplane.c**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "airplane.h"

void set\_id(airplane\_t\* p\_air, unsigned id) {

p\_air->id = id;

}

void set\_route(airplane\_t\* p\_air, char route[]) {

strcpy(p\_air->route, route);

}

void set\_nub(airplane\_t\* p\_air, unsigned nub) {

p\_air->nub = nub;

}

void set\_time(airplane\_t\* p\_air, char time[]) {

strcpy(p\_air->time, time);

}

void set\_price(airplane\_t\* p\_air, unsigned price) {

p\_air->price = price;

}

void get\_info(airplane\_t\* p\_air) {

printf("请输入航班信息，名称(id),路线,票数,时间(默认h）,价格(默认元):\n");

scanf("%u %s %u %s %u", &p\_air->id, p\_air->route, &p\_air->nub, p\_air->time, &p\_air->price);

}

void show(airplane\_t\* p\_air) {

printf("航班id：%u\t航班路线：%s\t航班票数：%u\t航班时间：%s h\t航班价格：%u 元\n\n", p\_air->id, p\_air->route,p\_air->nub, p\_air->time, p\_air->price);}

**3.list.h**

#ifndef \_\_LIST\_H\_\_

#define \_\_LIST\_H\_\_

#include "airplane.h"

// 结点是链表的元素

// 定义结点类型

typedef struct node {

airplane\_t air;

struct node\* p\_next;

} node\_t;

// 定义链表类型

// head:指向链表首结点的指针

// count:统计结点数量

typedef struct list {

node\_t\* head;

unsigned count;

} list\_t;

// 初始化链表

void init\_list(list\_t\* p\_list);

// 插入结点

int insert\_node(list\_t\* p\_list, airplane\_t\* p\_air);

// 遍历链表

void traverse\_list(list\_t\* p\_list);

// 销毁链表

void destroy\_list(list\_t\* p\_list);

// 移除结点(按照字段id删除)

int remove\_node(list\_t\* p\_list, unsigned id);

// 按照id进行查找

node\_t\* find\_node(list\_t\* p\_list, unsigned id);

#endif

**4.list.c**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include "list.h"

// 初始化列表实现

void init\_list(list\_t\* p\_list) {

// 头指针指向空

p\_list->head = NULL;

// 结点个数为0

p\_list->count = 0;

}

// 向链表中插入结点实现

int insert\_node(list\_t\* p\_list, airplane\_t\* p\_air) {

node\_t\* p\_node = (node\_t\*)malloc(sizeof(node\_t));

if (NULL == p\_node) {

return -1; // 分配结点内存失败，返回-1

}

p\_node->air = \*p\_air;//把航班结构体给list链表节点

p\_node->p\_next = p\_list->head;

// 新插入的结点成为新的头结点

p\_list->head = p\_node;

// 结点个数加1

++p\_list->count;

return 0;

}

// 遍历输出链表各结点

void traverse\_list(list\_t\* p\_list) {

node\_t\* p\_node = NULL;

// 如果p\_list指针为空，或结点个数个0，则无需遍历，立即退出该函数

if (!p\_list || !p\_list->count) {

printf("没有数据");

return;

}

// 遍历链表输出各结点的数据域的值

for (p\_node = p\_list->head; p\_node; p\_node = p\_node->p\_next) {

show(&p\_node->air);

}

}

// 销毁链表实现

void destroy\_list(list\_t\* p\_list) {

// 如果p\_list指针为空，或结点个数个0，则无需遍历，立即退出该函数

if (!p\_list || !p\_list->count) {

return;

}

// 删除结点时，每次删除头点结点确

while (p\_list->head) {

// 定义p\_node指向头结点

node\_t\* p\_node = p\_list->head;

// 头结点的下一个结点成为新的头结点

p\_list->head = p\_list->head->p\_next;

// 释放p\_node所记录的结点的内存

free(p\_node);

// 结点个数自减1

--p\_list->count;

}

}

// 从链表中移除指定id的结点实现

int remove\_node(list\_t\* p\_list, unsigned id) {

node\_t\* p\_now = NULL; //当前结点指针

node\_t\* p\_pre = NULL; //当前结点的前结点指针

// 如果p\_list指针为空，或结点个数个0，则无需移除，立即退出该函数

if (!p\_list || !p\_list->count) {

return -1;

}

// 前结点指针为头结点指针

p\_pre = p\_list->head;

// 当前结点为头结点的下一个节点

p\_now = p\_pre->p\_next;

// 若头结点是要删除的结点

if (p\_list->head->air.id == id) {

// 头结点的下一个结点成为新的头结点

p\_list->head = p\_list->head->p\_next;

// 前结点指针所指向的内存

free(p\_pre);

// 结点个数自减1

--p\_list->count;

return 0;

}

// 若头结点不是要删除的结点，则遍历查找

for (; p\_now; p\_now = p\_now->p\_next) {

// 找到要删除的结点

if (p\_now->air.id == id) {

// 前结点指针的指针域记录当前结点的下一个结点地址

p\_pre->p\_next = p\_now->p\_next;

// 释放当前结点

free(p\_now);

// 结点个数自减1

--p\_list->count;

return 0;

}

// 更新前结点指针，前结点记录当前结点指针的值

p\_pre = p\_now;

}

return -1;

}

// 查找结点(按航班编号)

node\_t\* find\_node(list\_t\* p\_list, unsigned id) {

// 定义循环指针变量指针

node\_t\* p\_now = NULL;

// 如果p\_list指针为空，或结点个数个0，立即NULL, 退出该函数

if (!p\_list || !p\_list->count) {

return NULL;

}

// p\_now记录头结点地址

p\_now = p\_list->head;

// 循环查找指定id的结点

while (p\_now) {

// 若找到结点，则退出循环

if (p\_now->air.id == id) {

break;

}

// 更新循环变量

p\_now = p\_now->p\_next;

}

// 返回查找到的结点指针 如果最后一个也不是要找的 则返回null

return p\_now;

}

**5.global.h**

#ifndef \_\_GLOBAL\_H\_\_

#define \_\_GLOBAL\_H\_\_

#include "list.h"

// 定义文件操作结构体

typedef struct all\_data {

unsigned choice;// 保存菜单选项编号

char file\_name[20];//文件名

list\_t air\_list;// 存储航线信息用的链表

} all\_data\_t;

// 显示菜单选择项

void make\_choice(all\_data\_t\*);

// 启动系统

void start(all\_data\_t\*);

//根据选项执行程序

void do\_trans(all\_data\_t\*);

// 输出文件中的记录

void show\_list(all\_data\_t \*);

// 添加记录

void add\_record(all\_data\_t\*);

// 删除某条记录

void delete\_record(all\_data\_t\*);

// 通过id查找到某条记录在链表中的结点

void find\_record\_by\_id(all\_data\_t\*);

// 修改记录中的某些项

void modify\_item(all\_data\_t\*);

// 退出系统

void quit(all\_data\_t\*);

// 从文件中读取信息到链表中

void read\_db(all\_data\_t\*);

// 将链表中的数据写入到文件

void write\_db(all\_data\_t\*);

// 初始化globa结构体操作

void init\_struct(all\_data\_t\*,char[]);

#endif

**6.global.c**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "global.h"

void make\_choice(all\_data\_t\* p\_data) {

// 显示菜单选项

printf("\n\n\t --------------------机票管理系统----------------------\n");

printf("\t| |\n");

printf("\t| 0.显示所有航班信息 |\n");

printf("\t| |\n");

printf("\t| 1.添加航班信息 |\n");

printf("\t| |\n");

printf("\t| 2.删除航班信息 |\n");

printf("\t| |\n");

printf("\t| 3.查找航班信息 |\n");

printf("\t| |\n");

printf("\t| 4.修改航班信息 |\n");

printf("\t| |\n");

printf("\t| 5.退出系统 |\n");

printf("\t -------------------------------------------------------\n");

printf("请输入您选择的功能:\n");

scanf("%u", &(p\_data->choice));

printf("\n您选择的功能是%u\n\n", p\_data->choice);

}

void start(all\_data\_t\* p\_data) {

read\_db(p\_data);//读取文件里的 文件

while (1) {

make\_choice(p\_data);

do\_trans(p\_data);//选项执行

system("cls");//清屏

}

}

// 根据选项选取对应的操作

void do\_trans(all\_data\_t\* p\_data) {

switch (p\_data->choice) {

case 0: {

show\_list(p\_data);//显示航班信息

break;

}

case 1: {

add\_record(p\_data);//添加航班信息

break;

}

case 2: {

delete\_record(p\_data);//删除航班信息

break;

}

case 3: {

find\_record\_by\_id(p\_data);//查找航班

break;

}

case 4: {

modify\_item(p\_data);//修改航班信息

break;

}

case 5: {

quit(p\_data);//退出程序

break;

}

default:

printf("没有选择任何操作选项...\n");

}

system("pause");//保持，暂停

}

void quit(all\_data\_t\* p\_data) {//退出程序

write\_db(p\_data); // 将链表中的数据写入到文件中

destroy\_list(&p\_data->air\_list); // 销毁链表

exit(0); // 退出程序

}

// 遍历显示链表中的数据

void show\_list(all\_data\_t\* p\_data) {

// 遍历链表

traverse\_list(&p\_data->air\_list);

}

// 初始化总结构体

void init\_struct(all\_data\_t\* p\_data, char file[]) {

// 默认初始化为0

p\_data->choice = 0;

// 初始化链表

init\_list(&p\_data->air\_list);

//初始化文件名

strcpy(p\_data->file\_name, file);

}

// 添加新数据

void add\_record(all\_data\_t\* p\_data)

{

// 定义 航班结构体

airplane\_t\* air;

air= (airplane\_t\*)malloc(sizeof(airplane\_t));

// 从键盘获取输入的航班信息

get\_info(air);

// 把结点插入到链表里

insert\_node(&p\_data->air\_list, air);

}

// 删除数据

void delete\_record(all\_data\_t\* p\_data) {

unsigned id; // 要删除id

int temp;//返回删除结果

printf("请输入您想要删除的航班编号:");

scanf("%u", &id);

// 按照id查找数据并删除

temp=remove\_node(&(p\_data->air\_list), id);

if (temp==0)

printf("\n删除成功");

else

printf("\n删除失败");

}

// 通过id查找某条数据

void find\_record\_by\_id(all\_data\_t\* p\_data) {

unsigned id;

// 循环变量 结点指针

node\_t\* p\_node = NULL;

printf("请输入您想要查找的航班编号:\n");

scanf("%u", &id);

// 根据id查找到结构体指针

p\_node = find\_node(&p\_data->air\_list, id);

// 找到即展示

if (p\_node) {

// 展示查找到的航班信息

printf("\n查找到以下信息：\n");

show(&p\_node->air);

}

else {

// 提示没有匹配数据

printf("对不起，查无此航班!!!\n");

}

}

// 修改航班信息

void modify\_item(all\_data\_t\* p\_data)

{

unsigned id;

unsigned sel\_item; // 修改选项

node\_t\* p\_node = NULL; //找到对应结点 记录对应结点的地址

printf("\n请输入您要修改的航班编号:");

scanf("%u", &id);

// 根据航班id序号查找

p\_node = find\_node(&p\_data->air\_list, id);

// 未找到

if (NULL == p\_node) {

printf("\n对不起，查无此航班!!!\n");

return;

}

// 找到输出菜单 选择对应修改项目

while (1)

{

printf("\n 1 修改航班编号\n\n");

printf(" 2 修改航班航线\n\n");

printf(" 3 修改航班票数\n\n");

printf(" 4 修改航班时间\n\n");

printf(" 5 不执行修改\n\n");

printf("请输入您想要执行的操作:");

scanf("%u", &sel\_item);

switch (sel\_item) {

case 1: {

printf("请输入新的航班编号:");

scanf("%u", &p\_node->air.id);// 通过指针找到对应数据 修改

show(&p\_node->air); // 展示

break;

}

case 2: {

printf("请输入新的航班航线:");

scanf("%s", p\_node->air.route);

show(&p\_node->air);

break;

}

case 3: {

printf("请输入新的航班票数:");

scanf("%u", &p\_node->air.nub);

show(&p\_node->air);

break;

}

case 4: {

printf("请输入新的航班时间:");

scanf("%s", p\_node->air.time);

show(&p\_node->air);

break;

}

case 5: {

printf("\n修改完成!!!\n");

return;

}

}

}

}

// 读取数据文件

void read\_db(all\_data\_t\* p\_data)

{

airplane\_t air; // 定义航班结构体 为了从文件中读取

//a+文件不存在，建新文件；文件存在，写入的数据追加到文件的末尾

FILE\* fp = fopen(p\_data->file\_name, "a+");

if (NULL == fp) { // 如果文件打开失败

printf("文本文件打开失败!!!\n");

return; // 退出函数

}

// 文件打开成功

// !feof(fp) 没有读到文件末尾 继续读

while (!feof(fp)) {

// memset 清空结构体（要清空的指针，赋值内容，内存大小）

memset(&air, 0, sizeof(air));

// fscanf从文件中读取航班信息，存到结构体变量中，返回参数列表中被成功赋值的参数个数。

if (fscanf(fp, "%u\t%s\t%u\t%s\t%u", &air.id, air.route, &air.nub, air.time, &air.price) == 5)

{

insert\_node(&p\_data->air\_list, &air); // 把结构体插入到链表中

}

}

fclose(fp); // 关闭文件

}

// 写文件

void write\_db(all\_data\_t\* p\_data) {

// 遍历链表用到 结点指针

node\_t\* p\_node = NULL;

// w+既可以读取也可以写入，文件不存在，创建新文件；文件存在，那么清空文件内容

FILE\* fp = fopen(p\_data->file\_name, "w+");

if (NULL == fp) { // 如果文件打开失败

printf("文本文件打开失败!!!\n");

return;// 退出函数

}

// 调整文件指针 写文件时从文件的起始位置开始写入

/\*

（fp，偏移量，起始位置）

起始位置：

文件开头 SEEK\_SET 0

当前位置 SEEK\_CUR 1

文件末尾 SEEK\_END 2

\*/

//只调节写入为开头，不具体位移，偏移量为0

fseek(fp, 0, SEEK\_SET);

// 获取链表头结点

p\_node = p\_data->air\_list.head; //循环变量 访问各个结点 p\_node

// 当结点不为空

while (p\_node) {

// 将航班信息按照格式写入到文件中去

fprintf(fp, "%u\t%s\t%u\t%s\t%u\n", p\_node->air.id, p\_node->air.route,

p\_node->air.nub, p\_node->air.time, p\_node->air.price);

// 更新结点 更新循环变量的值

p\_node = p\_node->p\_next;

}

// 关闭文件

fclose(fp);

}

**7.main.c**

#include <stdio.h>

#include "global.h"

int main(void) {

all\_data\_t\* p\_data;

p\_data = (all\_data\_t\*)malloc(sizeof(all\_data\_t));

init\_struct(p\_data,"data.txt");

start(p\_data);

return 0;

}