



# 《数据结构》课程设计

题 目 航空客运系统

学 院 计算机学院

专 业 软件工程

班 别 15级3班

学 号 3115005311

姓 名 邓智成

2016年12月10日

1. **需求分析**

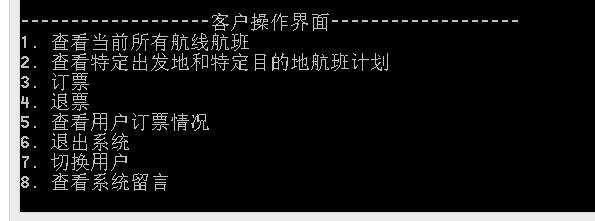
本程序是航空客运系统，它所设计的业务活动包括：查询航线、办理订票、办理退票、办理退票、查询用户历史订票、查询系统消息。

下面从头到尾来讲述一下这些功能的实现。

主页面

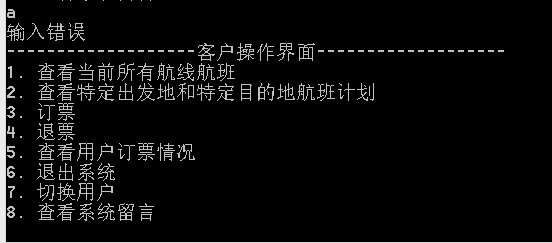
输入：页面提示的操作数字

输出：各功能模块操作界面



错误输入：数字不对或者输入字符

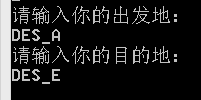
应对措施：提示输入错误并回到主页面



查询航线

输入：出发地的名字、目的地的名字

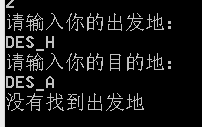
输出：出发地到目的地的航线以及航线相关的航班信息。航线信息包括航线号、出发地、目的地、一般出发日。航线信息包括班次号、日期、余票量、飞机号。





错误输入：输入出发地或目的地不存在

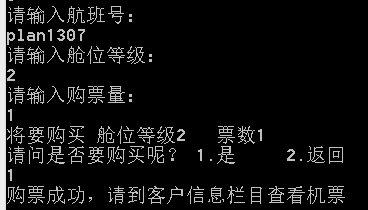
应对措施：输出错误信息



订票

输入：航班号、舱位等级、购买票数

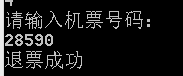
输出：成功购买信息或购买失败原因(航班号不存在、余票不足）



退票

输入：机票号码

输出：退票成功信息或者失败原因（机票号码不存在、机票不属于当前用户）



查看用户订票情况

输出：用户当前机票和历史机票



查看系统留言

输出：系统给用户的信息

![C:\Users\bcdzc\AppData\Roaming\Tencent\Users\394894672\QQ\WinTemp\RichOle\0TV_ZG@SJ[U4L](~VL1MA83.png](data:image/png;base64,)

1. **概要设计**

本程序中使用到的数据结构有

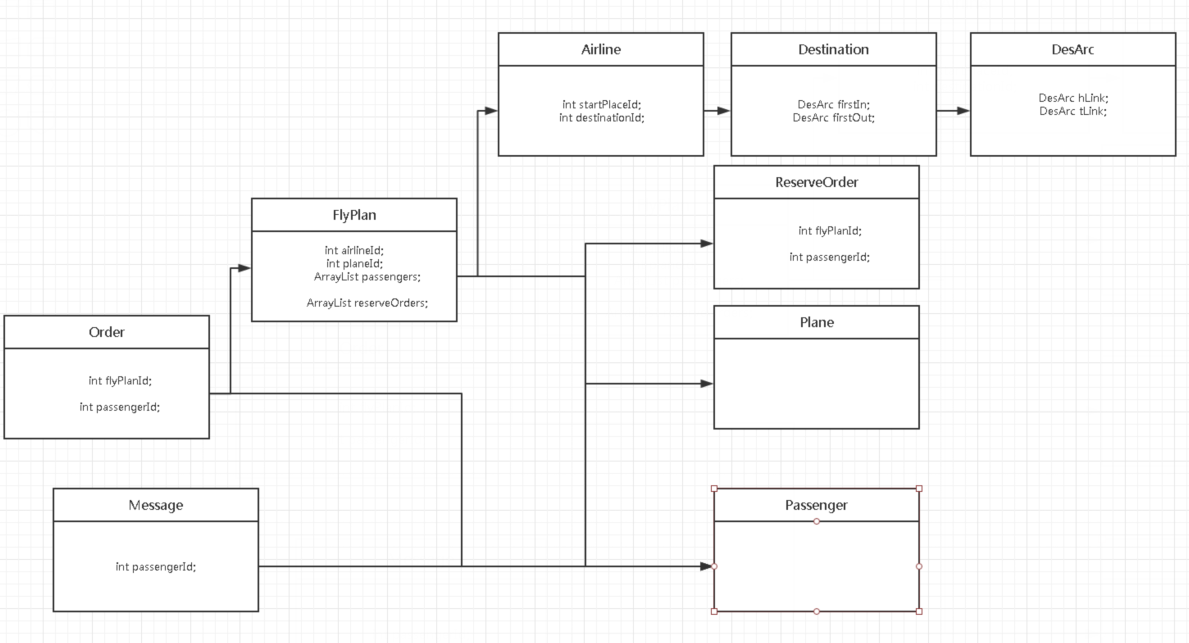
功能实体文件

Airline——航线，Destination——地点，FlyPlan——航班，Plane——飞机，Order——机票，Passenger——乘客，ReserveOrder——机票订单，Message——系统留言

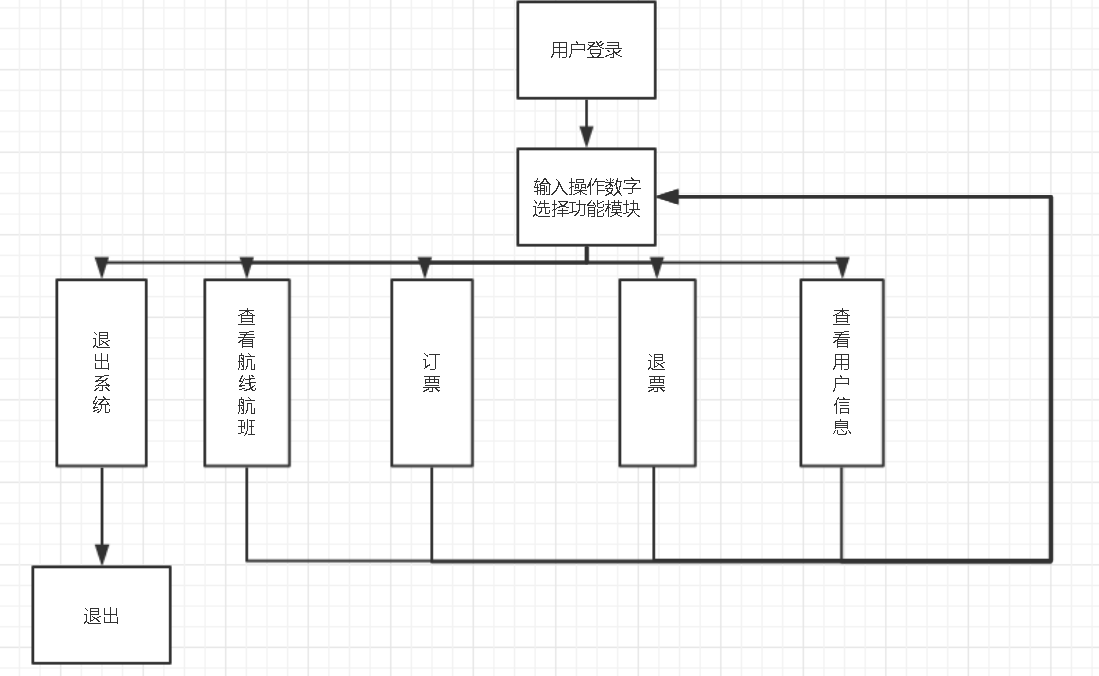
工具文件

Date——日期，String——字符串，ArrayList——链表实现的队列，DesArc——有向图中的弧

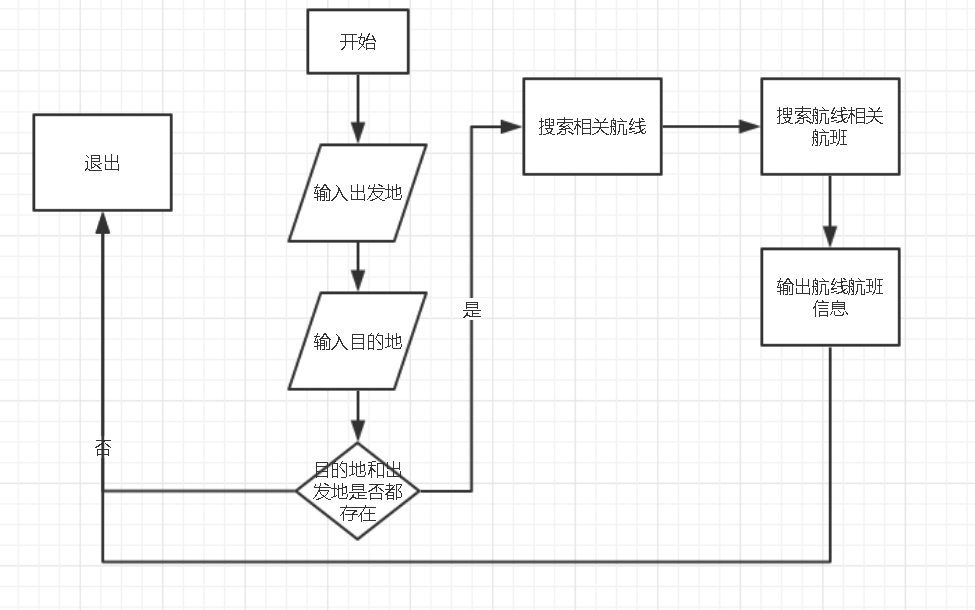
功能实体类之间的关联



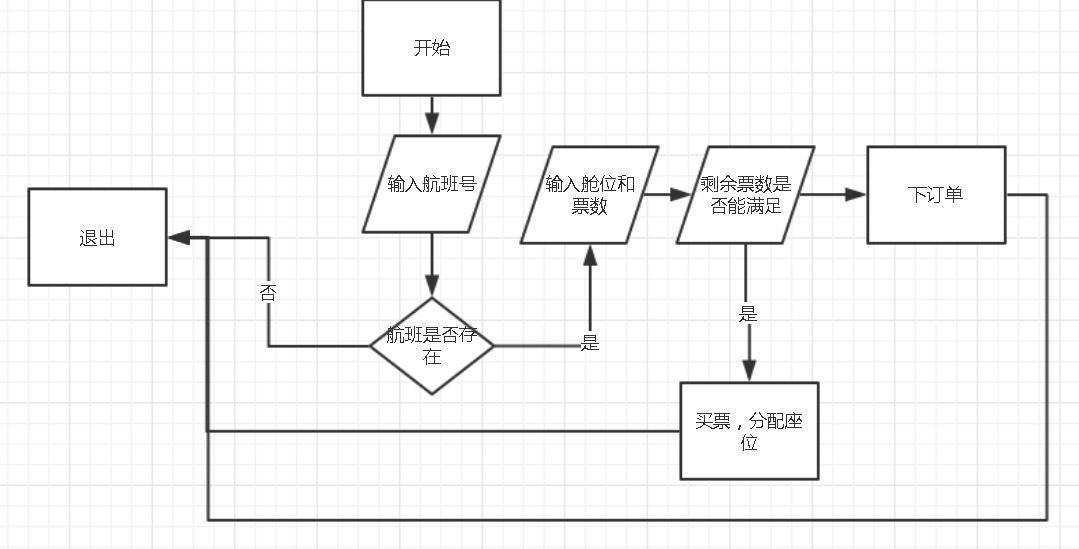
主程序流程



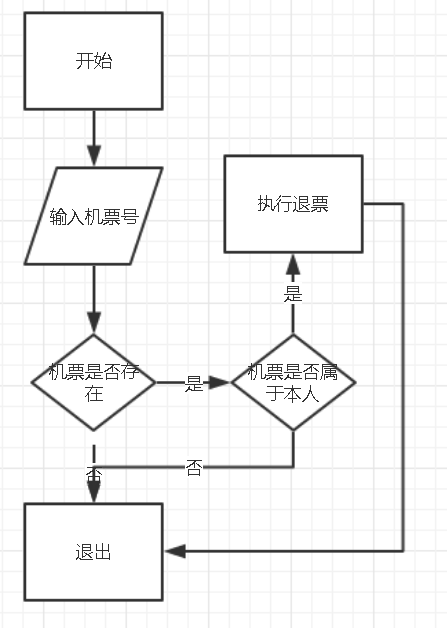
查询航线航班流程



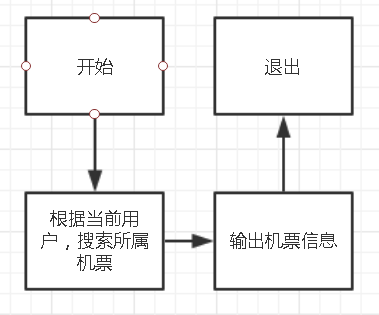
订票流程



退票流程

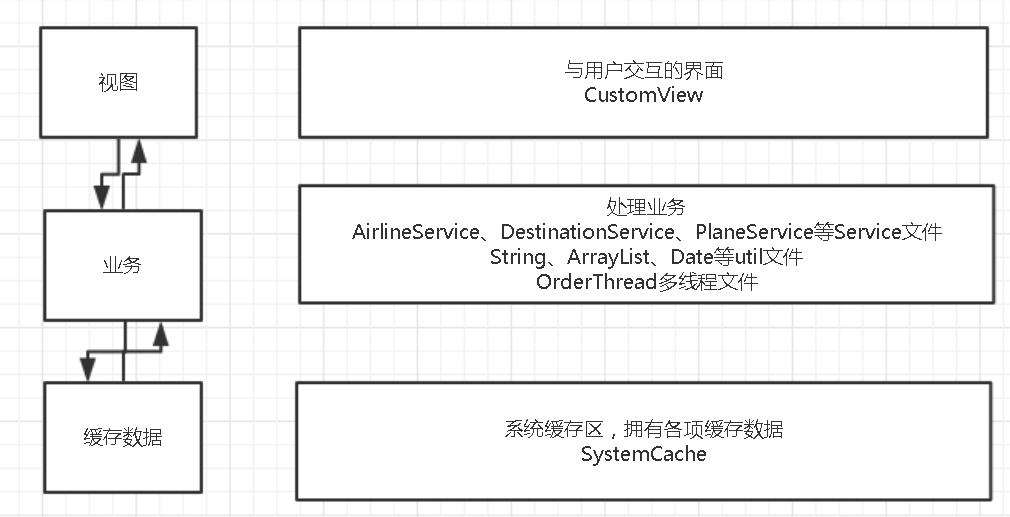


查看用户机票信息流程



软件架构

该系统分为视图层、业务层和缓存层，各层次之间相互调用



1. **详细设计**

这里会列出系统三个层次的主要设计逻辑和伪代码

上述数据结构中，如

Airline——航线，Destination——地点，FlyPlan——航班，Plane——飞机，Order——机票，Passenger——乘客，ReserveOrder——机票订单，Message——系统留言这些功能实体类的逻辑都大同小异，而数组队列和十字链表有向图是较复杂，需要用伪代码来说明。

数组队列（ArrayList）

typedef struct arraylist {

int E; //决定元素类型是int、String还是Object

void \*\*pItemList; //存储元素的二维指针

int capacity; //当前最大容量

int factor; //扩增因子

int size; //当前容量

} \*ArrayList;

enum ArrayList\_E{E\_STRING,E\_INT, E\_OBJECT};

//初始化

Void arraylist\_init(int E) {

为ArrayList items分配空间

items->E = E;

设定当前最大容量为10;

设定扩增因子为3；

设定当前容量为0；

为items->pItemList分配空间

return items;

}

//扩增

Void arraylist\_resize\_capacity(ArrayList items) {

If (items当前容量达到最大) {

Items的最大容量增加扩增因子的数量

开辟二维指针数组，空间数为当前最大容量

将原来二维指针数组的内容复制到新的空间上

释放原来的空间

Items的二维数组指向新的空间

}

}

//添加元素

Void arraylist\_add(ArrayList items,void \*pItem) {

扩增

将元素添加到末尾

Items的容量增加

}

//搜索特定元素的位置

void array\_indexOf(ArrayList items, void \*pItem) {

if (itmes的类型是E\_STRING) {

for (i = 0; i 小于 items的容量；i++) {

if(当前指针指向的items里的元素是pItem

或者 items里的元素化为char数组后跟pItem一样) {

返回i；

}

指针移到下一位

}

}

Else if (items的类型是E\_INT) {

for (i = 0; i 小于 items的容量；i++) {

if(当前指针指向的items里的元素是pItem

或者 items里的元素化为int后跟pItem一样) {

返回i；

}

指针移到下一位

}

}

Else if (itmes的类型是E\_OBJECT) {

for (i = 0; i 小于 items的容量；i++) {

if(当前指针指向的items里的元素是pItem) {

返回i；

}

指针移到下一位

}

}

返回-1；

}

//移除特定元素

int arraylist\_remove(ArrayList items, void \*pItem) {

for (i = 0; i小于items的容量;i++) {

if (items的类型是E\_STRING) {

if (当前指针指向的items里的元素是pItem

或者 items里的元素化为char数组后跟pItem一样) {

For (m = i; m < items的容量; m++) {

m后面的元素向前推进一位

}

将items最后一个元素设置为空

Itms的容量减1

返回i；

}

}

if (items的类型是E\_INT) {

if (当前指针指向的items里的元素是pItem

或者 items里的元素化为int后跟pItem一样) {

For (m = i; m < items的容量; m++) {

m后面的元素向前推进一位

}

将items最后一个元素设置为空

Itms的容量减1

返回i；

}

}

if (items的类型是E\_STRING) {

if (当前指针指向的items里的元素是pItem{

For (m = i; m < items的容量; m++) {

m后面的元素向前推进一位

}

将items最后一个元素设置为空

Itms的容量减1

返回i；

}

}

}

}

//移除特定位置的元素

void \*arraylist\_removeIndex(ArrayList items,int index) {

if (index大于items的容量 或者 index小于0)

返回空值

Else {

For (m = index; m < items容量 – 1；m++) {

将数组中的元素从index开始向前移动一位

}

Itmes的最后一位元素设置为空

Items容量减一

返回删除元素的void指针

}

}

//设置特定位置的元素

void arraylist\_set(ArrayList items,int index,void \*pItem) {

if (index 小于items的容量 和 items的容量大于0) {

将items特定位置上的元素换为pItem

}

}

//获取指定位置上的元素

void \*arraylist\_get(ArrayList items,int index){

If (index 大于或等于0 和 index小于items容量)

返回items指定位置上的元素

返回空值

}

//移除此列表中的所有元素

void arraylist\_clear(ArrayList items) {

释放原有元素的内存

设定items的最大容量为0

设定items的扩增因子为3

设定items的容量为0

为items的元素指针开辟一个二维数组空间

}

用十字链表实现的有向图

该有向图中结点为Destination，弧为DesArc。

Destination中除了基本信息，有两个DesArc的指针firstIn和firstOut，分别对应第一个以该节点为尾部的弧和第一个以该节点为头部的弧。

DesArc有两个DesArc的指针tLink和hLink，分别对应弧尾节点的其他入度弧

和弧头节点的其他出度弧。DesArc还有tailVex和headVex来识别所连接地点在缓存数组中的位置。

//根据传入的Airline的队列来初始化十字链表

void initOrthogonalList(ArrayList airlines) {

for (i = 0; i < airlines的容量; i++) {

获取airlines的i下标的Airline

初始化DesArc

设置DesArc的tailVex为Airline的出发地在数组中的下标

设置DesArc的headVex为Airline的目的地在数组中的下标

If (headVex下标的Destination的入度弧不为空)

将DesArc的hLink设置为该地点的入度弧

Else

将DesArc的hLink设置为空

If(tailVex下标的Destiantion的出度弧不为空)

将DesArc的tLink设置为该地点的出度弧

Else

将DesArc的tLink设置为空

将headVex下标的Destination的入度弧设置为本弧

将tailVex下标的Destiantion的出度弧设置为本弧

添加本弧到缓存队列中

}

}

//清空十字链表

void cleanOrthogonalList() {

清空缓存中的弧队列

获取缓存中的地点队列destinations

For (i = 0; i < destinations的容量；i++) {

获取destiantions的i下标的Destiantion

将该Destiantion的入度弧和出度弧设置为空

}

}

//搜索特定地点之间可连通的弧，结果弧的id以”id1 id2 ... idn”字符串的形式存储在success队列中

void searchDestinationPath(Destination start,

Destination target, StringBuilder sb, ArrayList success, ArrayList searchedDes) {

If (start 和 target相同) {

将StringBulider所记录的字符串添加到success中

}

Else {

获取target的入度弧DesArc arc

For (;arc不为空;arc变为弧头节点的其他入度弧) {

获取arc的弧尾节点为Destiantion \_d

If (searchedDes中含有\_d的名字)

Continue;

\_d变位arc的弧头节点

searchedDes添加\_d

添加arc弧的id到sb中

searchDestinationPath(start, arc的弧尾节点, sb, success, searchedDes);进行递归

}

}

}

主页面

主页面只有一个CustomView文件，它负责提供各项功能的视图给用户，并且和各项service文件交互，以进行操作或获取信息。

CustomView主要流程

Void enterCustomView() {

输入操作码为x

Switch(x) {

1：查看当前所有航班页面；break；

2: 查看特定地点航班页面；break；

3：订票页面；break；

4：退票页面；break；

5：查看用户订票情况；break；

6：退出系统：break；

7：切换用户：break；

8： 查看系统留言：break；

default：

}

}

//查看特定地点航班

void queryAirlineByDestination() {

输入出发地为start

输入目的地为destination

根据名字在缓存中寻找出发地和目的地

searchDestinationPath();根据出发地和目的地寻找可行的航线路径

把航线路径按长度排序

For (i = 0; i < 所有航线路径的数量; i++)

For (j = 0; j < 航线路径下拥有航线数量； j++) {

获取当前j下标的航线

获取与该航线相关的航班

打印

}

}

//买票

void doCreateOrder() {

输入航班号

根据航班号在内存中寻找相关航班

输入舱位等级和票数

If (剩余票数足够用户购买) {

生成机票

飞机剩余票数减一

}

Else {

询问用户是否生成预备订单

If(用户承诺下生成订单)

生成预备订单，一旦剩余票数足够就为用户购买

}

}

//退票

Void returnOrder() {

输入机票号

If (机票号存在)

If (机票属于当前用户)

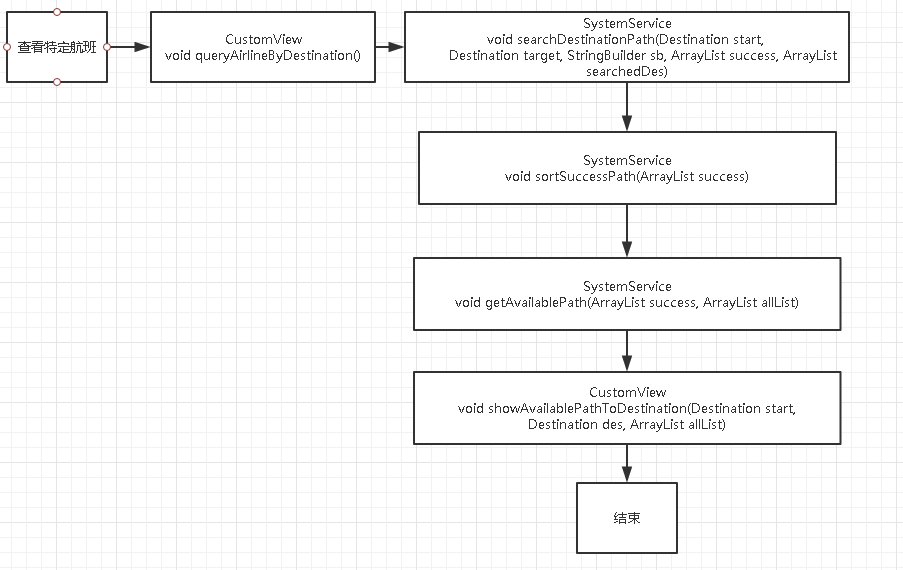
为用户添加过期机票并作废

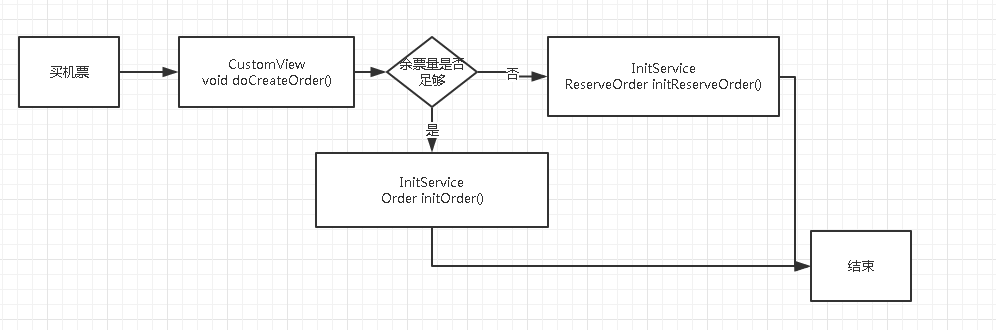
飞机剩余票数加一

}

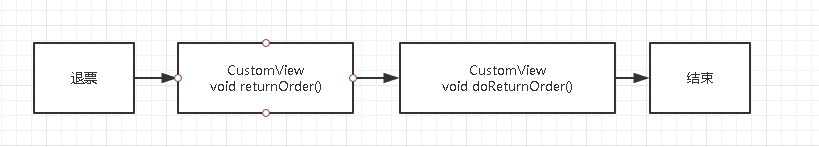
过程和调用函数图

查看航线

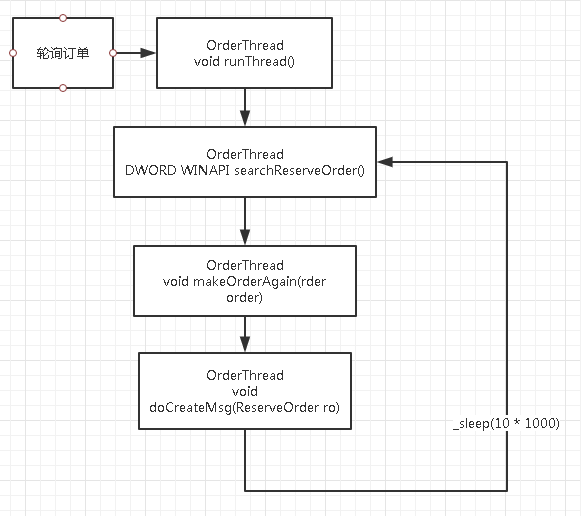


买票

退票



轮询预备订单队列



1. **调试分析**

时间复杂度

十字链表相关

十字链表初始化——O(n^2)

十字链表搜索——O(n^2)

十字链表删除——O(n)

数组队列相关

数组队列插入——O(1)

数组队列删除——O(n)

数组队列搜索下标——O(n)

查询航线——O(n^2)

购买机票——O(n)

退票——O(n)

查看用户机票信息——O(n)

调试过程中出现以下问题：

1. 函数中生成的char数组在函数返回数组首地址后立马失效

解决：把char数组作为参数传到函数里面，或者在函数中为char数组开辟空间，使之不会被回收空间

1. 数互相调用有点杂乱

解决：我把系统分为三个层次，把直接操作的对象存到缓存区，把能够对缓存对象进行操作的函数放到业务区，再暴露函数接口到视图区，实现一定程度的函数解耦，有一定的层次感。

③找有向图之间的路径过程繁杂

解决：在这个过程中我是通过递归来实现的。我用一个数组队列来存储已经访问过的地点，用另一个数组队列来存储已经访问过且有效的弧，每当进行递归判断出发点和终止点相同的时候，就把有效的弧存到第三个数组队列，再按顺序清空地点队列和弧队列。最后得出了一个能有效连接两个地点的弧队列。

改进地方：

1. 用户管理模块尚未完成，现有用户都是内存里的数据。
2. 未提供查看特定航线和特定航班的功能，尚未提供查看任意时间任意航线任意班的功能。
3. 未针对飞机的舱位位置进行限制，这一部分可以扩充为飞机信息管理模块。
4. 尚未能够自由添加航线航班。
5. 下订单到的过程可以改进为提醒用户——购买机票这样的过程，以免用户下订单后反悔退票。

体会

用c语言来实现功能还是很考基本功的，c语言不想java那样有灵敏的垃圾回收机制，以致在开发过程中不得不时常注意开辟空间和回收空间，不过这也是锻炼严谨的编程习惯所需的。

在整个项目从设计到实施，再到最后的测试，实际上用的最多的时间是最后的测试部分，因为在这一阶段时常会发现之前的代码大多数都有些千奇百怪的漏洞，而不得不回过头来修正。有时碰到一个不得不返工的问题又会占用不少时间，拖延了进度。所以，我发现在设计项目的过程中所花的时间越多，在测试上所花的时间就会越少。良好严谨的设计能够带来顺畅的生产流程，相反，如果前期的设计没有花足够的心思去考虑各种情况，那么写出来的代码就会破绽百出，增加返工的时间。

在这次项目中我应用到的课堂知识主要是队列和有向图，在设计的过程中我参考了书本上的算法，网络上相关的博客文章介绍，在和同学们的讨论中慢慢形成。在这个项目中使用到的Arraylist我觉得具备了一点泛型的神韵，但是这只是用void\*指针来保存传进来的对象而已，实际上并没有对对象类别的判断。所以这个队列是需要改进的。而在有向图我是运用十字链表来存储，初始化方法可以参考课本写出来，但是基于两点的路径搜索算法只能一步步摸索出来，最后是可以实现出来了，为整个系统提供了搜索出发地和目的地所关联航线航班的功能。

总的来说，这次数据结构课设让我充分运用了从课堂上学到的知识，从设计环节中考虑各种各样的数据结构，到实现环节中碰到的算法问题，到最后测试环节中不断返工的体会，这都是我从这次项目中学到的东西。

1. **用户使用说明**

本程序是航空客运系统，支持用户搜索各个地点的航班，订票、退票、下预备订单以及

查看过往订票信息。

接下来按搜索航线、订票、下预备订单、退票和查看过往订票信息为例，对各个操作进行说明。

1. 搜索航线。您需要根据屏幕提示输入有效的出发点名字和有效的目的地名字，如果系统能够从内存中获取相关地点信息，就会在屏幕上打印出发点到目的地的所有航线，以及航线下现有的航班。针对每条航线，系统只会输出5个航班。视情况而定，可能从出发地到目的地会有转机的情况，也就是说会有中介点，由于航班出发时间各不相同，请用户对准时间购买机票，以免有时间和金钱的损失。
2. 购买机票。在搜索航线里，您应该看到了某条航线下的某个航班的航班号，把它记下来，然后进入订票界面，输入上述航班号，就能够进入购票环节。购票环节会让你输入购票数和舱位，如果剩余票数足够您购买，就会生成机票，购买成功。如果剩余票数不足，则会进入预备订单环节。
3. 预备订单。假如购买机票环节中，飞机的剩余票数不足以用户购买，则会根据用户的意愿，安排进等候队列。万一有其他乘客退票，使剩余票数达到等候用户的要求，就会自动为等候用户生成机票。需要注意的是，这一过程是先到先得的，剩余的票数会优先给等候队列前面的用户。当用户预备订单生效的时候，系统会发提醒信息到用户，用户需要查看系统信息栏以确定订单是否生效。
4. 退票。假如您当前不需要所购买的机票了，可以执行退票环节。系统会让你输入机票号，当确认机票号所关联的机票是本人购买，就会执行退票。
5. 查看过往订票信息。进入用户历史信息，可以看到现有机票和历史机票的详细信息。
6. **测试结果**

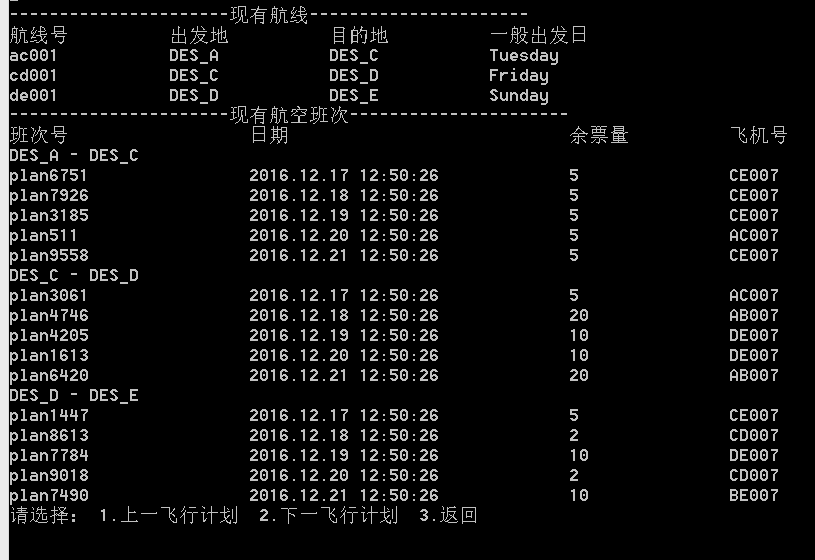
查看当前所有航线和航班，可以查看上一条航线和下一条航线



查看特定出发地和目的地的航空计划，航空计划包括能够直达航班和其下的航线，需要转机的航班组合和其下的航线。用户可以查看上一飞行计划和下一飞行计划。

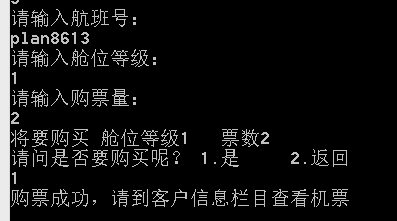


图示输入出发地DES\_A, 目的地DES\_E



图示输入出发地DES\_A, 目的地DES\_E

购买机票

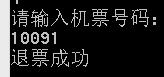


图示输入航班号plan8613，购买舱位2的机票2张



图示购买航班号为plan8613的机票两张

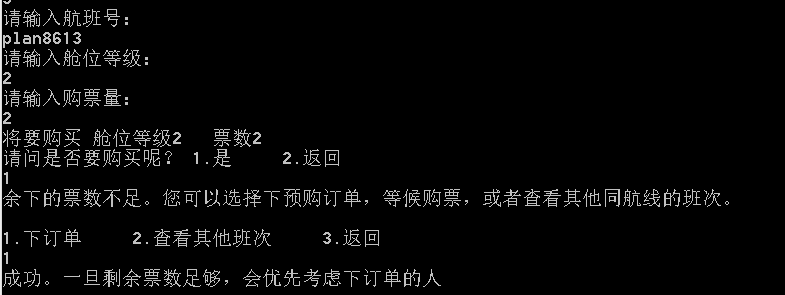
退票



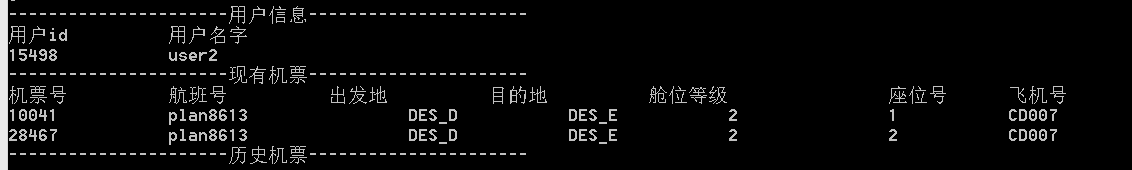


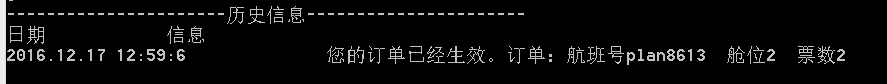
图为机票号为10091的机票退票后的情况

下预备订单



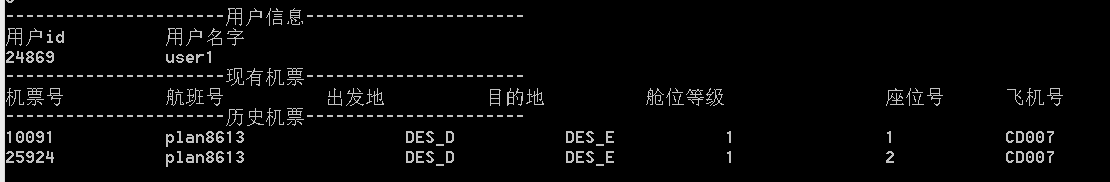
图示为购买航班号为8613的机票2张，余票不足生成订单



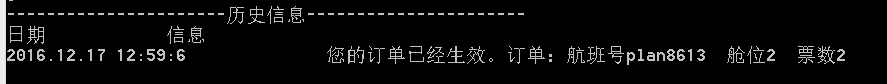


图为剩余票数足够后自动为用户购买机票，并以系统信息的形式通知用户

查看用户历史机票



查看系统信息



附录：部分重要代码（省略了视图层和业务层部分代码）

缓存区

SystemCache.c

#include <stdio.h>

#include "util.h"

#include "model.h"

#include "allService.h"

typedef struct {

//地点

ArrayList destinations;

//航线

ArrayList airlines;

//飞机

ArrayList planes;

//机票

ArrayList orders;

//过期机票

ArrayList overdueOrders;

//乘客

ArrayList passengers;

//航班

ArrayList plans;

//过期航班

ArrayList overduePlans;

//系统留言

ArrayList msgs;

//弧

ArrayList arcs;

}SystemCache;

//获得系统缓存单例

SystemCache\* getSystemCache();

//初始化十字链表

void initOrthogonalList(ArrayList airlines);

//寻找地点id对应的实体类在数组中的下标

int LocateDestination(int destinationId);

//清空十字链表

void cleanOrthogonalList();

//寻找出发地和目的地之间的航线航班

void searchDestinationPath(Destination start,

Destination target, StringBuilder sb, ArrayList success, ArrayList searchedDes);

//根据航线长度对航线排序

void sortSuccessPath(ArrayList nList);

//分割航线

ArrayList getAvailablePath(ArrayList success, ArrayList allList);

//获取航线下的航班

ArrayList getDetailFlyPlan(ArrayList lines);

#endif

SystemService.c

SystemCache\* getSystemCache() {

static SystemCache\* s = NULL;

if (s) {

return s;

}

//懒汉式初始化单例

s = (SystemCache\*)malloc(sizeof(SystemCache));

s->destinations = arraylist\_init(E\_OBJECT);

s->airlines = arraylist\_init(E\_OBJECT);

s->planes = arraylist\_init(E\_OBJECT);

s->orders = arraylist\_init(E\_OBJECT);

s->overdueOrders = arraylist\_init(E\_OBJECT);

s->passengers = arraylist\_init(E\_OBJECT);

s->plans = arraylist\_init(E\_OBJECT);

s->overduePlans = arraylist\_init(E\_OBJECT);

s->msgs = arraylist\_init(E\_OBJECT);

s->arcs = arraylist\_init(E\_OBJECT);

return s;

}

//int转char

char\* INT\_to\_CHAR(int number, char\* str) {

sprintf(str,"%d",number);

return str;

}

//char转int

int CHAR\_TO\_INT(char\* str) {

int size = 0;

while (str[size] != '\0' && str[size] != ' ')

size++;

int i = size - 1;

int ratio = 1;

int total = 0;

for (; i >= 0 ; i--) {

int num = str[i] - 48;

total += num \* ratio;

ratio = ratio \* 10;

}

return total;

}

//初始化十字链表

void initOrthogonalList(ArrayList airlines) {

ArrayList destinations = getSystemCache()->destinations;

int i = 0;

for (i = 0; i < airlines->size; i++) {

//获得出发地实体

int tailVex = LocateDestination(((Airline)arraylist\_get(airlines, i))->startPlaceId);

//获得目的地实体

int headVex = LocateDestination(((Airline)arraylist\_get(airlines, i))->destinationId);

DesArc dn = initDesArc();

dn->tailVex = tailVex;

dn->headVex = headVex;

dn->hLink = NULL;

dn->tLink = NULL;

dn->airlineId = ((Airline)arraylist\_get(airlines, i))->id;

if (((Destination)arraylist\_get(destinations, headVex))->firstIn != NULL) {

dn->hLink = ((Destination)arraylist\_get(destinations, headVex))->firstIn;// 关联弧头节点的其他入度弧

}

else

dn->hLink = NULL;

if (((Destination)arraylist\_get(destinations, tailVex))->firstOut != NULL) {

dn->tLink = ((Destination)arraylist\_get(destinations, tailVex))->firstOut;// 关联弧尾节点的其他出度弧

}

else

dn->tLink = NULL;

((Destination)arraylist\_get(destinations, headVex))->firstIn = dn; // 将弧头节点的入度弧设定为本弧

((Destination)arraylist\_get(destinations, tailVex))->firstOut = dn; // 将弧尾节点的出度弧设定为本弧

arraylist\_add(getSystemCache()->arcs, dn);

}

}

//获得目标地点在缓存数组中的下标

int LocateDestination(int destinationId) {

ArrayList desList = getSystemCache()->destinations;

int i = 0;

int size = desList->size;

for (; i < size; i++) {

if (((Destination)arraylist\_get(desList, i))->id == destinationId) {

return i;

}

}

return -1;

}

//清空十字链表

void cleanOrthogonalList() {

//清空arcs

ArrayList arcs = getSystemCache()->arcs;

arraylist\_clear(arcs);

//设置firstIn和firstOut为空

ArrayList des = getSystemCache()->destinations;

int i = 0;

int size = des->size;

for (; i < size; i++) {

Destination d = (Destination)arraylist\_get(des, i);

d->firstIn = NULL;

d->firstOut = NULL;

}

}

String list\_to\_str(ArrayList list) {

char t[1000] = {};

int i = 0;

int count = 0;

int j = 0;

for (i;i<list->size;i++) {

String temp = (String)arraylist\_get(list, i);

while (temp[j] != '\0') {

t[count] = temp[j];

count++;

j++;

}

t[count++] = ' ';

j = 0;

}

return t;

}

//搜索出发地到目的地之间的可行航线

void searchDestinationPath(Destination start,

Destination target, StringBuilder sb, ArrayList success, ArrayList searchedDes) {

if (start == target || start->id == target->id) { //如果出发地等于目的地，把缓存的航线组成字符串存到success队列

arraylist\_removeIndex(sb, sb->size - 1); //消去空格

char (\*value)[200] = (char(\*)[200])malloc(sizeof(char) \* 200);

arraylist\_add(success, stringbuilder\_toString(sb, value));

} else {

//获得目的地的入度弧

DesArc arc = target->firstIn;

for (; arc != NULL; arc = arc->hLink) {

Destination \_d =

(Destination)arraylist\_get(getSystemCache()->destinations, arc->tailVex);

//如果弧尾结点被访问过，就转到下一条入度弧

if (arraylist\_indexOf(searchedDes, \_d->name) != -1) {

continue;

}

//添加弧头节点到已访问队列

\_d = (Destination)arraylist\_get(getSystemCache()->destinations, arc->headVex);

arraylist\_add(searchedDes, \_d->name);

char temp[10] = {};

INT\_to\_CHAR(arc->id, temp);

temp[5] = ' ';

stringbuilder\_append(sb, temp); //添加弧id到StringBuilder

searchDestinationPath(start, //递归查询

(Destination)arraylist\_get(getSystemCache()->destinations, arc->tailVex),

sb, success, searchedDes);

}

}

if (!(sb->size == 0)) { //取出最后一个弧id

stringbuilder\_delete(sb, sb->size - 6, sb->size);

}

if (searchedDes->size != 0) { //取出最后一个访问结点

arraylist\_removeIndex(searchedDes, searchedDes->size - 1);

}

}

//根据航线长度进行排序

void sortSuccessPath(ArrayList nList) {

int size = nList->size;

int i, j;

for (i = 0; i < size; i++) {

String temp;

for (j = 0; j < size - i - 1; j++) {

String sj = (String)arraylist\_get(nList, j);

String sj1 = (String)arraylist\_get(nList, j + 1);

if (string\_size(sj) > string\_size(sj1)) {

temp = sj;

arraylist\_set(nList, j, sj1);

arraylist\_set(nList, j + 1, temp);

}

}

}

}

//分割航线

ArrayList getAvailablePath(ArrayList success, ArrayList pathList) {

int i = 0;

int size = success->size;

for (; i < size; i++) {

ArrayList list = arraylist\_init(E\_OBJECT);

String suc = (String)arraylist\_get(success, i);

ArrayList successList = string\_split(suc, ' '); //航线id以空格为分隔符

int j = 0;

int size\_j = successList->size;

for (; j < size\_j; j++) {

String arcId = arraylist\_get(successList, j);

DesArc arc = getDesArcById(CHAR\_TO\_INT(arcId));

Airline airline = getAirlineById(arc->airlineId);

arraylist\_add(list, airline);

}

arraylist\_add(pathList, list);

}

return pathList;

}

//获得目标航线下的航班

ArrayList getDetailFlyPlan(ArrayList lines) {

ArrayList allPlan = arraylist\_init(E\_OBJECT);

int i = 0;

int size = lines->size;

for (; i < size; i++) {

Airline line = (Airline)arraylist\_get(lines, i);

ArrayList plans = getAvailableFlyPlanByAirlineId(line->id);

arraylist\_add(allPlan, plans);

}

return allPlan;

}

订单线程

OrderThread.c

#include <stdio.h>

#include <windows.h>

#include "SystemCache.h"

//生成系统信息提醒用户

void doCreateMsg(ReserveOrder ro) {

Message msg = initMessage();

msg->passengerId = ro->passengerId;

msg->date = getDate();

FlyPlan plan = getFlyPlanById(ro->flyPlanId);

String msg1 = "ÄúµÄ¶©µ¥ÒÑ¾­ÉúÐ§¡£¶©µ¥£ºº½°àºÅ";

String msg2 = " ²ÕÎ»";

String msg3 = " Æ±Êý";

String planId = plan->flyPlanId;

char cangwei[5] = {};

INT\_to\_CHAR(ro->classLevel, cangwei);

char piaoshu[5] = {};

INT\_to\_CHAR(ro->number, piaoshu);

char temp[1000] = {};

int i = 0;

int count = 0;

while (msg1[i] != '\0') {

temp[count++] = msg1[i++];

}

i = 0;

while (planId[i] != '\0') {

temp[count++] = planId[i++];

}

i = 0;

while (msg2[i] != '\0') {

temp[count++] = msg2[i++];

}

i = 0;

while (cangwei[i] != '\0') {

temp[count++] = cangwei[i++];

}

i = 0;

while (msg3[i] != '\0') {

temp[count++] = msg3[i++];

}

i = 0;

while (piaoshu[i] != '\0') {

temp[count++] = piaoshu[i++];

}

msg->msg = temp;

arraylist\_add(getSystemCache()->msgs, msg);

}

//购买机票，分配座位

void makeOrderAgain(Order order) {

FlyPlan plan = getFlyPlanById(order->flyPlanId);

Passenger p = getPassengerById(order->passengerId);

if (arraylist\_indexOf(plan->passengers, p) == -1) {

arraylist\_add(plan->passengers, p);

}

Plane plane = getPlaneById(plan->planeId);

plane->takenSeat = plane->takenSeat + 1;

order->seat = plane->takenSeat;

createOrder(order);

}

//生成机票

void doCreateOrderAgain(ReserveOrder ro) {

FlyPlan plan = getFlyPlanById(ro->flyPlanId);

int i;

for (i = 0; i < ro->number; i++) {

Order order = initOrder();

order->flyPlanId = plan->id;

order->classLevel = ro->classLevel;

order->passengerId = ro->passengerId;

makeOrderAgain(order);

}

}

//线程函数，搜索订单

DWORD WINAPI searchReserveOrder() {

while (1) {

ArrayList plans = getSystemCache()->plans;

int i;

//轮询航班下的等候人群

for (i = 0; i < plans->size; i++) {

FlyPlan plan = (FlyPlan)arraylist\_get(plans, i);

Plane plane = getPlaneById(plan->planeId);

int availableTicket = plane->maxSeat - plane->takenSeat;

//如果剩余机票足够订单上的机票数量，就执行购票

if (plan->reserveOrders->size != 0 && availableTicket != 0) {

ReserveOrder ro = (ReserveOrder)arraylist\_get(plan->reserveOrders, 0);

if (ro->number <= availableTicket) {

doCreateOrderAgain(ro);

arraylist\_remove(plan->reserveOrders, ro);

doCreateMsg(ro);

}

}

}

\_sleep(10 \* 1000);

}

}

//启动线程

void runThread() {

CreateThread(

NULL

,0

,searchReserveOrder

,NULL

,0

,NULL);

}

数组队列

#ifndef ARRAYLIST\_H\_INCLUDED

#define ARRAYLIST\_H\_INCLUDED

#include "struct.h"

#include "\_string.h"

enum ArrayList\_E{E\_STRING,E\_INT, E\_OBJECT}; //元素种类

ArrayList arraylist\_init(int E); //初始化队列

void arraylist\_add(ArrayList items,void \*pItem); //添加元素

int arraylist\_indexOf(ArrayList items,void \* pItem); //查询元素下标

int arraylist\_remove(ArrayList items, void \*pItem); //移除元素

void \*arraylist\_removeIndex(ArrayList items,int index); //移除下标元素

void arraylist\_set(ArrayList items,int index,void \*pItem); //设定特定下标特定元素

void \*arraylist\_get(ArrayList items,int index); //获得特定下标元素

void arraylist\_clear(); //队列清空

StringBuilder stringbuilder\_init();

#endif // ARRAYLIST\_H\_INCLUDED

ArrayList.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "arraylist.h"

//初始化

ArrayList arraylist\_init(int E) {

ArrayList items = malloc(sizeof(struct arraylist));

items->E = E;

items->capacity = 10;

items->factor = 3;

items->size = 0;

items->pItemList = malloc(sizeof(void \*) \* items->capacity);

return items;

}

//重置容量,当可用容量不够用时,自动扩大3个

static void arraylist\_resize\_capacity(ArrayList items) {

if(items->size == items->capacity)

{

//printf("空间用完了,扩大空间 -> %d\n",items->size);

items->capacity = items->size + items->factor; //扩大容量

void \*\*newItemList = malloc(sizeof(void \*) \* items->capacity);

//将原来的指针指向的内容复制到新的空间中

memcpy(newItemList,items->pItemList,sizeof(void \*) \* items->size);

free(items->pItemList); //释放原来指向的空间

items->pItemList = newItemList; //指向新空间

}

}

//将指定的元素追加到此列表的尾部。

void arraylist\_add(ArrayList items,void \*pItem) {

arraylist\_resize\_capacity(items);

\*(items->pItemList + items->size) = pItem;

items->size++; //size + 1

}

//搜索给定参数第一次出现的位置,找到返回index,找不到返回-1

int arraylist\_indexOf(ArrayList items, void \*pItem) {

int i=0;

int size = items->size;

void \*\*pItemList = items->pItemList;

if(items->E == E\_STRING)

{

for(; i<size; i++)

{

if(\*pItemList == pItem || strcmp(\*pItemList,pItem) == 0)

{

return i;

}

pItemList++;

}

}

else if(items->E == E\_INT)

{

//printf("------- %d,%d\n",\*(int \*)pItemList,\*(int \*)pItem);

for(; i<size; i++)

{

//如果指针的值或指针指向的最终变量的值是一样的

if(\*pItemList == pItem || \*(int \*)pItemList == \*(int \*)pItem)

{

return i;

}

pItemList++;

}

}

else

{

for(; i<size; i++)

{

//如果指针的值或指针指向的最终变量的值是一样的

if(\*pItemList == pItem)

{

return i;

}

pItemList++;

}

}

return -1;

}

//从此列表中移除指定元素的单个实例,如果存在就返回1,反之返回0

int arraylist\_remove(ArrayList items, void \*pItem)

{

int i=0;

int size = items->size;

void \*\*pItemList = items->pItemList;

for(; i<size; i++)

{

if(items->E == E\_STRING)

{

if(\*(pItemList+i) == pItem || strcmp(\*(pItemList+i),pItem) == 0)

{

//移动指针指向的地址,即将数组中的元素从index开始向前移动一位

int m;

for(m = i; m<size-1; m++)

{

\*(pItemList + m) = \*(pItemList + m + 1);

}

\*(pItemList + size - 1) = NULL; //将新的元素插入到这个位置

items->size--; //size - 1

return i;

}

}

else if(items->E == E\_INT)

{

if(\*(pItemList+i) == pItem || \*(int \*)(pItemList + i) == \*(int \*)pItem)

{

//移动指针指向的地址,即将数组中的元素从index开始向前移动一位

int m;

for(m = i; m<size-1; m++)

{

\*(pItemList + m) = \*(pItemList + m + 1);

}

\*(pItemList + size - 1) = NULL; //将新的元素插入到这个位置

items->size--; //size - 1

return i;

}

}

else

{

if(\*(pItemList+i) == pItem)

{

//移动指针指向的地址,即将数组中的元素从index开始向前移动一位

int m;

for(m = i; m<size-1; m++)

{

\*(pItemList + m) = \*(pItemList + m + 1);

}

\*(pItemList + size - 1) = NULL; //将新的元素插入到这个位置

items->size--; //size - 1

return i;

}

}

}

return -1;

}

//移除此列表中指定位置上的元素,返回被移除的元素

void \*arraylist\_removeIndex(ArrayList items,int index)

{

int size = items->size;

if(index >= size || index < 0)

{

return NULL;

}

else

{

void \*\*pItemList = items->pItemList;

void \*pItem = \*(pItemList + index); //取得index位置的指针的值

//移动指针指向的地址,即将数组中的元素从index开始向前移动一位

int m;

for(m = index; m<size-1; m++)

{

\*(pItemList + m) = \*(pItemList + m + 1);

}

\*(pItemList + size - 1) = NULL; //将尾部的指针的值设置为NULL

items->size--; //size - 1

return pItem;

}

}

//用指定的元素替代此列表中指定位置上的元素,返回: 以前位于该指定位置上的元素

void arraylist\_set(ArrayList items,int index,void \*pItem)

{

int size = items->size;

if(index <= size && size > 0)

{

\*(items->pItemList + index) = pItem;

}

}

//返回此列表中指定位置上的元素

void \*arraylist\_get(ArrayList items,int index)

{

int size = items->size;

if(index >= 0 && index <= size && size > 0)

{

return \*(items->pItemList+index);

}

else

{

return NULL;

}

}

//移除此列表中的所有元素

void arraylist\_clear(ArrayList items)

{

int i = items->size - 1;

for (; i >= 0; i--) {

arraylist\_removeIndex(items, i);

}

free(items->pItemList); //释放内存

//重新初始化

items->capacity = 10;

items->factor = 3;

items->size = 0;

items->pItemList = malloc(sizeof(void \*) \* items->capacity);

}