Please update with your name and comments for each section if you make changes. If you disagree with any parts of the design, put your proposed changes, pseudo code, etc.

Notes:

Does the input file of flights attach reservation numbers to the flight already? Or do we go through the list of passenger data to sort passengers onto flights?

Taking Input from File:

1. Passenger Input File - A map contains all passengers. We do not duplicate, copy, or have any other containers with passenger objects.

Sort each line for name, first name, membership, reservation number, flight

PassengerObject ( first, last, membership, reservation#, flight num );

map.insert( reservation #, undefined reference to PO );

2. Flight Input - FlightObject ( flight number, departCity, arrivalCity, depart time, arrive time, mileage)

This means that the mileage is calculated in the client file.

Reasons why FlightObject should not calculate mileage itself:

Passing the graph file to the flight object to then calculate the mileage sounds weird.

Somewhere in the project, a graph must be made and the data must be entered. If we put this in the constructor of a FlightObject, that’s expensive.

The minimum requirements:

1. Display a list of all passengers, their flights, and their From and To cities.

All passengers should be stored in a container. Each passenger object has an integer/string containing their flight number (Passengers should be stored in a map as well. The Reservation number is the key. Iterators can traverse a map from start to finish).

A flight container, best option is a map, with the flight number as the map’s key.

Also, each flight object should have a container of the reservation numbers in that flight.

Pseudo code: loop entire passenger container

print name and last name

flight container [ passenger.flightnumber ] . display

2. Display a list of all passengers going to a city

A map container called destinations. Key is the name of a city, a vector containing the list of flight numbers which have this destination. Whenever a flight is loaded. Access it’s destination, pull the vector from ‘destinations’ and push\_back it’s flight number.

Pseudo code: vector = destinations [ city name ]

// now we have a list of flight numbers

for all elements in vector

container = flights [ flight number ] . reservation numbers

for all elements in container

passengers [ specific rn ] . print details

3. Display a list of all passengers leaving from a city.

Same concept as #2 except another container called departure cities.

4. Display a list of all the passengers on a flight

Pseudo code: container = flights [ flight number ] . reservation numbers

for all elements in container

passengers [ specific rn ] . print details

5. Display a list of passengers who are not able to be on flights due to space limitations

This assumes that that input file will have more passengers attached to a flight number than “project specification” possible.

A flight object should have a container (priority queue?) of passengers on the flight. A second container ‘standby’ of passengers wishing to attend.

The problem with a priority queue is that an iterator does not exist for this container. Expensive and stupid to remove all values from a container to put them back in for the sake of accessing. Secondly, the container is not guaranteed to store items based on the order they are placed into the queue. Thus, it will be possible for the last economy passenger that “bought” their ticket to be given a seat in place of someone who “bought” their ticket before them.

Proposal: A deque in the class for each type of passenger.

deque pilotclub

deque firstclass … etc.

Super easy to count to 40 and figure out who doesn’t belong on the flight.

Pseudo code: loop entire flight container

flightcontainer[ i ] . getStandbys ( )

loop entire standby list

passengers [ specific rn ] . print

vector getStandbys () {

int startIndex = pilotclub.size() + firstclass.size() + etc.

iterator of economy vector

iterator + start index

for economy class loop with current iterator til end

store reservation number in temp vector

return temp vector

}

6. Display a list of passengers waiting for a flight

I don’t understand the difference between this and #5

7. Display a list of all flights

Pseudo code: for all elements in flight container

flight . print

8. Display a list of all flights with their passengers

Pseudo code: for all elements in flight container

flight . print

vector = flight . getPassengers ( )

for all elements of vector

passenger list [ specific rn ] . print

vector getPassengers () // inside flight object {

count = 0

vector passengerRN

go through all vectors and store passengerRN in order

count++

return passengerRN

}

9. Search for a passenger

Because our passenger list is a map, we can access all elements with iterator traversal.

Pseudo code: This function should be made by the client. Do we want to search for the passenger by name? reservation number? last name only?

vector findPassengerByName (searchName) {

vector passenger rn

for entire container of passengers ( name )

if passenger name == searchName

store passenger rn in vector

print vector

}

PassengerObject findPassengerByRN ( rn number ) {

return passengers [ specific rn ] // check if it exists of course

// if it doesn’t exist, don’t forget to delete the object it creates

// read std::map documentation if this doesn’t make sense

}

10. Search for a flight

Depends on how the user wants to search for the flight?

Possibilities: Search all flights with departure city

Search all flights with arrival city

Search all flights by flight number

Search all flights by arrival time // Anything else?

11. Delete a passenger from a flight.

Pseudo code: deletePassenger ( flight number, passenger name? reservation number ?){

// there’s a lot of possibilities to delete passengers

flight list [ flight number ] . delete ( name? rn? )

}

in flight class

delete ( // so many overloads for this method ) {

super simple to search all vectors for this data.

RN is easiest as only one RN will exist.

Multiple passengers with same name may exist.

}

one possibility:

vector delete (passenger Name) {

vector list

store all passengers with that name

return vector

}

then back in client function deletePassenger {

if vector.size () > 1

print out list of passengers

ask user to delete by reservation number instead

}