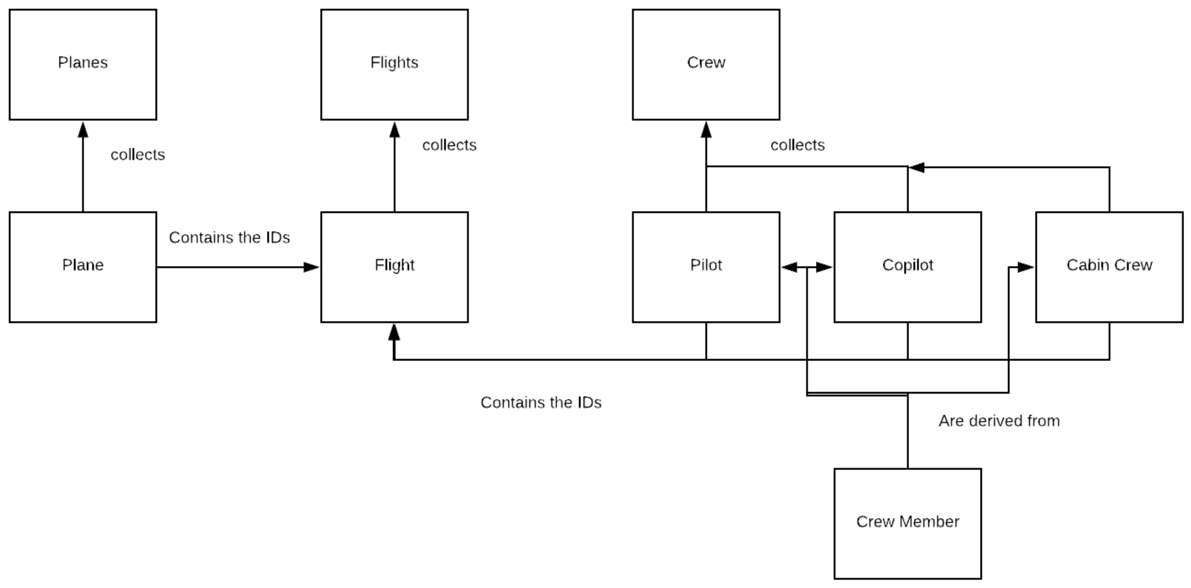
Homework 5

David Zhao

CSCE 1040

Design Doc


Planes
Flights
Crew
collects
collects
collects
Plane
Contains the IDs
Flight
Pilot
Copilot
Cabin Crew
Contains the IDs
Are derived from
Crew Member

Planes

Functions

- Add plane
- Edit plane
- Delete plane
- Search for planes
- Print all planes
- Find plane and print all details of that plane

Data

- Vector of planes

Plane

Functions

- Get/Set Make
- Get/Set Model
- Get/Set Tail number
- Get/Set Seats
- Get/Set Range
- Get/Set status
- default constructor

Data

- Make
- Model
- Tail Number
- Number of Seats
- Range
- Status

Pilot

Functions

- Get/Set Rating
- Get/Set Hours

Data

- Rating
- Flight hours

Copilot

Functions

- Get/Set Rating
- Get/Set Hours

Data

- Rating
- Flight hours

Flights

Functions

- Add flight
- Edit flight
- Delete flight
- Search for flights
- Print all flights
- Find flight and print all details of that flight
- Print specific flight schedule
- print all crew member's assignments
- delete all cancelled or completed
- update flights

Data

- Vector of flights

Crew Member

Functions

- Get/Set Name
- Get/set ID
- get/set status
- default constructor
- virtual PrintInfo

Data

- Name
- ID
- status

Cabin Crew

Functions

- Get/Set Position

Data

- Position

Crew

Functions

- Add crew member
- Edit crew member
- Delete crew member
- Search for crew member
- Print all crew
- Find crew member and print all details of that crew member

Data

- vector of type crew member pointers for virtual capability

Flight

Functions

- Get/set plane id
- get/set pilot id
- get/set copilot id
- get/set cabin crew
- get/set start time
- get/set end time
- get/set starting airport code
- get/set ending airport code
- get/set number of passengers
- get/set status
- default constructor

Data

- plane id
- pilot id
- copilot id
- cabin crew id
- start time
- end time
- starting airport
- ending airport
- number of passengers
- status

<u>Plane:</u>

String make, tailNum, status

Int model, seatsNum, range

Accessors and mutators for all of the above data

Default constructor sets all strings to “ “ and all integers to 0

<u>Planes:</u>

Vector <plane> planeList

```
Int searchPlane(string planeNum)
    iterates through the vector until it finds the plane that returns planeNum with
    getPlane(), then returns the iterating counter in the for loop
    if plane is not found, returns -1
bool addPlane(plane newPlane)
    use searchPlane with the given newPlane to see if any planes match. If not,
        push_back newPlane to vector of planes and return true and store data (storing
        data happens in menu function)
    else return false
bool editPlane(plane edited)
    call searchPlane using the plane number from edited, then use that as planeList[iterator]
    to find the plane
        if plane not found, then return false
    then check to see if get functions return default values from edited; if so, don’t replace
    for that value. If functions return non-default values, use set functions on
    planeList[iterator]. Then return true and store data
bool deletePlane(string planeNum)
    call searchPlane using planeNum
        if searchPlane returns -1, return false
    else, planeList.erase(planeList.begin() + number from searchPlane – 1) and store data
void printPlanes()
```

iterates through the vector (probably with .at() and a for loop) and prints out the get function for each one

bool printEntry(string planeID)

uses searchPlane(planeID) to find the plane, returns false if plane not found

if plane found, print out all the accessors and return true

<u>Crew Member:</u>

String name, status

Int id (or long, depends on how long expected ids are)

Accessors and mutators for all of the above

Default constructor sets strings to “ “ and int/float to 0

<u>Pilot:</u>

String rating

Int hours

Accessors and mutators for the two

Default constructor

<u>Copilot:</u>

String rating

Int hours

Accessors and mutators for the two

Default constructor

<u>Cabin Crew:</u>

String position

Accessor and mutator

Default constructor

<u>Crew:</u>

Vector <crewmember*> crewList

```
Int searchCrew(int/long crewID)
    iterates through the vector until it finds the crew member that returns crewID with getCrew(), then returns the iterating counter in the for loop
    if plane is not found, returns -1
bool addCrew(crewmember newCrew)
    use searchCrew with the given newCrew to see if any crew match (probably just gonna use getID and use that). If not,
        push_back newCrew to vector of crew members and return true and store data
    else return false
bool editCrew(crew edited)
    call searchCrew using the crew ID from edited, then use that as crewList[iterator] to find the crew member
        if crew member not found, then return false
    then check to see if get functions return default values from edited; if so, don't replace for that value. If functions return non-default values, use set functions on crewList[iterator]. Then return true and store data
bool deleteCrew(int/long crewID)
    call searchCrew using crewID
        if searchCrew returns -1, return false
    else, crewList.erase(crewList.begin() + number from searchFlights – 1 (not sure if -1 is necessary, clarify later)) and store data
void printCrew()
    iterates through the vector (probably with .at() and a for loop) and prints out the get function for each one
bool printEntry(int/long crewID)
    uses searchCrew(crewID) to find the crew member, returns false if crew member not found
    if crew member found, print out all the accessors and return true
```

Flight:

String planeID, startPort, endPort, status, startTime, endTime (start time and end time are strings entered by user. Only converted to time when actually needed to compare.)

Int pilotID, copilotID, cabin1, cabin2, cabin3, numPass, flightID

Accessors and mutators for all of the above

Default constructor sets strings to “ “ and ints to 0

<u>Flights:</u>

Vector <flight> flightList

Int searchFlights(int/long flightID)

iterates through the vector until it finds the flight that returns the flightID with getFlightID(), then returns the iterating counter in the for loop

if plane is not found, returns -1

void addFlight (flight newFlight) (function that returns int so that I can clarify which thing went wrong)

first, use updateFlights() to make sure all planes that are inactive are set to inactive. (addendum: placing most of this stuff within the main function so that I can search the other vectors) Then, use searchflight with the given newFlight to see if any flights match (probably just gonna use getID and use that) IDs, if the plane is busy, if the passengers of newFlight are less than the plane’s seats, and that the crew is not already assigned. Also check to see if the crew members are in valid positions. If not,

push_back newFlight to vector of flights and return true and store data

epiphany for later: use difftime to check differences between start and end time of all flights and new flight, then vice versa, then start and start, and end and end

make vector of integers to save indices, loop through list of flights and mark down any and all indices where there’s an overlap in crew or plane, then plug those indices into checking the times and difftime thing

bool editFlight(flight edited)

call searchFlight using the flight ID from edited, then use that as flightList[iterator] to find the flight

```
        if flight not found, then return false
    then check to see if get functions return default values from edited; if so, don’t replace
    for that value. If functions return non-default values, use set functions on
    flightList[iterator]. Then return true and store data
bool deleteFlight(int flightID)
    call searchFlights using flightID
        if searchFlight returns -1, return false
    else, flightList.erase(flightList.begin() + number from searchFlights – 1 (not sure if -1 is
    necessary, clarify later)) and store data
void printFlights()
    iterates through the vector (probably with .at() and a for loop) and prints out the get
    function for each one
bool printEntry(int flightID)
    uses searchFlights(flightID) to find the flight, returns false if flight not found
    if flight found, print out all the accessors and return true
int printFlightsAssignment(string planeID)
    has integer check for checking how many assignments were printed
    loop through vector and find all flights with planeID as the plane id
        if those flights are active, print those flights’ start and end times, iterate check
    return check at end
int printCrewAssignment(int/long crewID)
    has integer check
    loop through vector and find all flights with crewID as one of the crew members’ id
        if those flights are active, print those flights’ start and end times, iterate check
    return check
int printFlightStatus(string compareStatus)
    has integer check
    loop through vector and find all flights with status == compareStatus
```

```
        print out flightIDs, iterate check
    return check
int deleteFlightStatus(string conStatus)
    has integer check
    loop through vector and find all flights with status == conStatus
        delete those with erase(flightList.begin() + iterator - 1), restart iterator, check++
    return check
void updateFlights()
    check current time
    loop through flightsList to check if any of their end times are before current time
        if so, update those statuses
void storeFlights()
    prints out the size of the vector
    use printentry on every single thing in the vector into an output stream
void loadflights()
    uses getlines and cins on an input stream, plugs those values into a flight, then uses
    .push_back on that flight
```

Menu:

Options: Add a crew member, add a plane, add a flight, delete a crew member, delete a plane, delete a flight, edit a crew member, edit a plane, edit a flight, find a crew member, find a plane, find a flight, print all crew members, print all planes, print all flights, print all details of a crew member, print all details of a plane, print all details of a flight, print assignment schedule for a flight, print assignment schedule for a crew member, print flights from status, delete cancelled flights, delete removed flights, update flights

Each of the above is just a combination of the methods and functions of the classes

Loading and Storing:

For storing data:

Use the functions for each collection class

For loading data:

Use the functions for each collection class