351 Survivors
Northeast Airlines

CPSC 362

Summer 2018

Danny Pham

Brian Trinh

Jayro Alvarez

Royce Nguyen

Chapter 1: Software Requirements (SR)

1.1 Introduction

---

a) Purpose

To outline the services that the software provides and provide a description of the components and specifications needed to effectively produce the software.

CPSC 362 - Introduction to Software Engineering
Summer Session 2018
Royce Nguyen, Jayro Alvarez, Danny Pham, Brian Trinh

b) Scope of the Problem

- Northeast Airlines
    - Find current flights available to and from locations
    - Select specific seats according to graphical representation of plane cabin
    - Book/Reserve seats for flights to and from desired locations
    - Create new account
    - Login to user account
    - Review booked flight info.
    - Modify reservation
    - Cancel reservation
    - Print flight ticket

c) Definitions, Acronyms, or Abbreviations

No terms, acronyms, or abbreviations used in this document that need to be defined prior to reading.

d) References

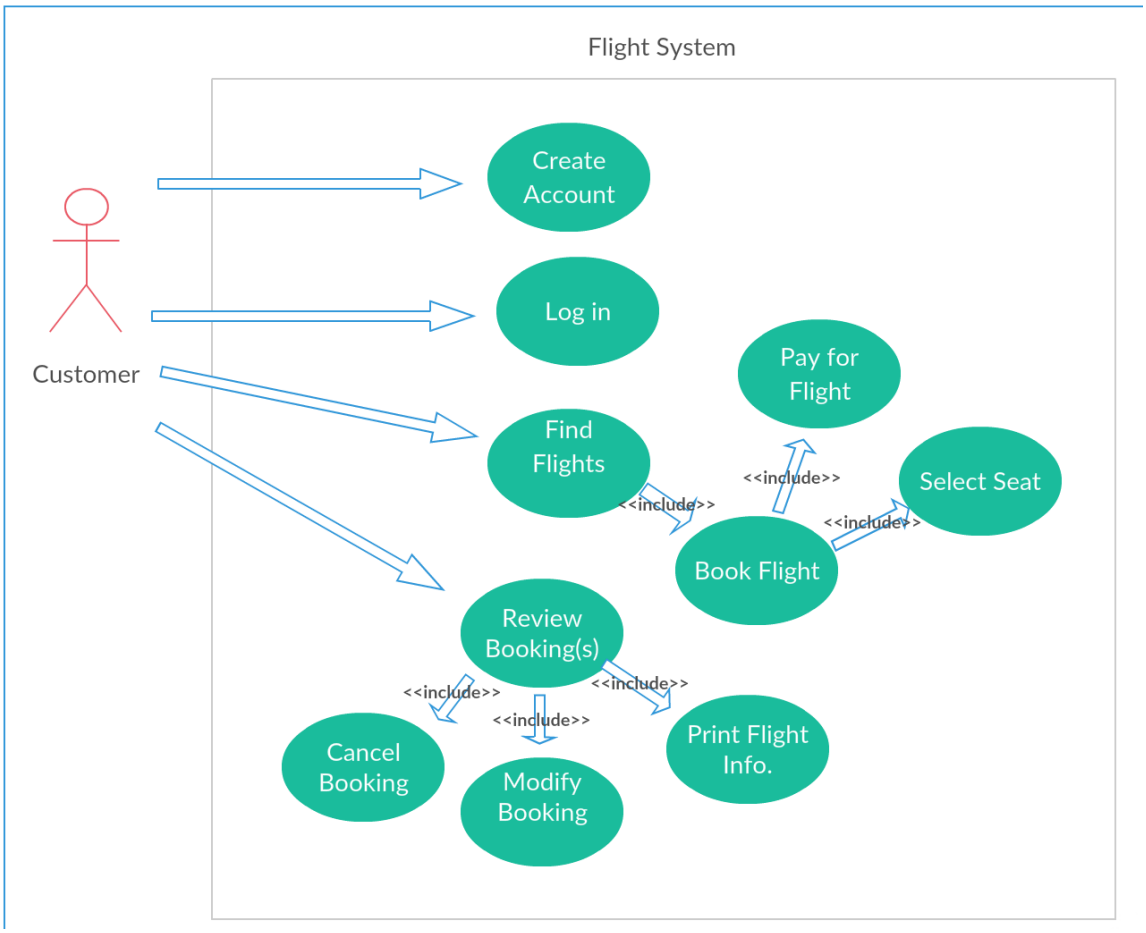
No outside documents referenced by this document.

e) Overview

The rest of this Software Requirements section contains the flow of the use-cases the software will encounter, these are necessary to implement the features mentioned in section 1.1b. There are also use-cases descriptions followed by the necessary functional requirements and descriptions needed to implement the use-cases.

1.2 Use-Case Modeling

a) Use-case diagrams



b) **Use-case descriptions**

Use Case ID:	UC-01		
Use Case Name:	Find Flights		
Created By:	Brian Trinh	**Last Updated By:**	Danny Pham
Date Created:	Jul 10, 2018	**Last Revision Date:**	Aug 2, 2018
Actors:	Customer		
Description:	The user can find a list of available flights to and from selected destinations.		
Trigger:	Customer selects the option to find available flights		
Preconditions:	1. Customer inputs valid to and from flight locations. 2. Airline has flights going from the to and from locations.		
Postconditions:	1. Website outputs a list of flights to and from the desired locations. 2. Website outputs a message stating flight info is invalid.		
Normal Flow:	1. Customer selects desired starting flight location. 2. Customer selects desired ending flight location. 3. Customer confirms their choices applied 4. Website outputs a list of flights to and from the desired locations.		
Alternative Flows:	1a. In step 1 of the normal flow, if the customer does not select a valid starting flight location 1. System will prompt customer to choose another starting flight location. 2. Customer recognizes the notification. 3. Use Case returns back to step 1 of the normal flow 2a. In step 2 of the normal flow, if the customer does not select a valid ending flight location 1. System will prompt customer to choose another ending flight location. 2. Customer recognizes the notification.		

	3. Use Case returns back to step 2 of the normal flow.
Exceptions:	N/A
Includes:	“Select Seats” Use Case
Frequency of Use:	On demand
Special Requirements:	N/A
Assumptions:	1. The customer knows flight info in order to correctly search for a desired flight.
Notes and Issues:	1. Depending on the search criteria, error handling for user input will need to be addressed.

Use Case ID:	UC-02		
Use Case Name:	Create Account		
Created By:	Brian Trinh	**Last Updated By:**	Danny Pham
Date Created:	Jul 10, 2018	**Last Revision Date:**	Aug 2, 2018
Actors:	Customer		
Description:	Allows user to create an account		
Trigger:	Select option to create a new user account		
Preconditions:	1. Customer must have a valid email address		
Postconditions:	1. Customer has a valid account 2. Customer email is sent a confirmation mail regarding the creation of their account		
Normal Flow:	1. Customer selects the option to create a new account 2. Customer enters required information when creating an account (Full name, account username, password, address, etc.) 3. Customer confirms submission of new account info 4. System outputs a message stating the account has successfully been created 5. TBD		
Alternative Flows:	N/A		
Exceptions:	4a. In step 4 of the normal flow, if the customer has inputted invalid information 1. System will output a message stating the account has not been created and state reasons why 2. Customer will be sent back to the account creation page 3. Repeat Steps 2 and 3 until the customer inputs valid information or customer cancels account creation		
Includes:	“Login” Use Case		

Frequency of Use:	On demand
Special Requirements:	1. The customer must know the appropriate information to create an account
Assumptions:	2. The customer understands English to progress through the interface
Notes and Issues:	1. Need to include error handling for various inputs such as a. Username b. Password c. Account Holder Name d. Credit Card Number e. Credit Card Expiration Date f. Address g. Email h. Date of Birth i. Home Phone Number j. Cell Phone Number k. Gender

Use Case ID:	UC-03		
Use Case Name:	Log In		
Created By:	Royce Nguyen	**Last Updated By:**	Brian Trinh
Date Created:	Jul 10, 2018	**Last Revision Date:**	Aug 2, 2018
Actors:	Customer		
Description:	Customer logs into a previously-created account in order to access the options to book flight and review bookings		
Trigger:	Customer opts to log into his/her account. Login is triggered after customer selects seats and proceeds to book the flight. (Customer must be logged into an account in order to book a flight)		
Preconditions:	1. Customer has active existing account		
Postconditions:	1. Customer is able to book flights or review bookings		
Normal Flow:	1. System prompts user login screen 2. Customer enters username and password 3. System enables customer to proceed to flight booking if login was prompted after selecting seat 4. System enables user to review bookings if any		
Alternative Flows:	2a. In step 2 of the normal flow, if the customer's login does not match any in the system: 1. System will prompt customer to re-enter login		
Exceptions:	2a. In step 2 of the normal flow, if the customer enters and invalid login 1. Login is disapproved 2. Message to customer to re-enter login information 3. Customer enters correct login 4. Use Case resumes on step 3 of normal flow		

Includes:	1. Review bookings 2. Find flights
Frequency of Use:	On demand
Special Requirements:	1. The customer must know their account username and password
Assumptions:	N/A
Notes and Issues:	N/A

Use Case ID:	UC-04		
Use Case Name:	Select Seat(s)		
Created By:	Danny Pham	**Last Updated By:**	Danny Pham
Date Created:	Jul 10, 2018	**Last Revision Date:**	Aug 2, 2018
Actors:	Customer		
Description:	Allows customer to select a seat from a visual representation of the available seats in a given flight		
Trigger:	Customer selects to book a flight from the menu after login		
Preconditions:	1. Customer must be logged in 2. Fight must have available seats		
Postconditions:	1. Customer reserves the selected seat 2. Selected seat is removed from available list		
Normal Flow:	1. Customer opts to Select Seat 2. System displays a visual representation of available seats 3. Customer selects seat(s) 4. System validates if the seat(s) is still available 5. System removes the seat(s) from available list 6. Seat(s) is assigned to the customer's account 7. System displays a confirmation message 8. System returns to home menu		
Alternative Flows:	3a. In step 3 of the normal flow, customer selects unavailable seats 1. Accept valid selections 2. Output message with invalid selections 3. Return to step 3 of normal flow		
Exceptions:	N/A		
Includes:	Book Flight		
Frequency of Use:	Executed whenever a customer books a flight		

Special Requirements:	N/A
Assumptions:	1. Customer intends to purchase seats
Notes and Issues:	N/A

Use Case ID:	UC-05		
Use Case Name:	Book Flight		
Created By:	Jayro Alvarez	**Last Updated By:**	Danny Pham
Date Created:	Jul 10, 2018	**Last Revision Date:**	Aug 2, 2018
Actors:	Customer		
Description:	Allows user to confirm and pay for selected seats on the flight chosen for the trip.		
Trigger:	User confirmation of flight after prompted with their selection of flight, seat, cost, flight number, departure/arrival time, etc.		
Preconditions:	1. Customer has valid flight selected. 2. Customer has valid seat selected.		
Postconditions:	1. Customer has flight successfully booked on account. 2. Customer has access to modify/cancel flight(s).		
Normal Flow:	1. Customer logs in or creates a new account. 2. Customer selects flight and seat. 3. Customer inputs payment information. 4. Customer confirms chosen flight information. 5. Flight is booked.		
Alternative Flows:	No alternative flows, just exception catching.		
Exceptions:	1a. In step 1 of the normal flow, if the customer tries to login with an incorrect username 1. Prompt to try again 2. Customer enters a correct username 3. Use case resumes on step 2 3a. In step 3 of of the normal flow, if the customer inputs incorrect payment information 1. Prompt to try again 2. Customer enters correct payment method 3. Use case resumes on step 5		
Includes:	Select Flight.		

Frequency of Use:	On demand, everytime a customer wants to book a flight.
Special Requirements:	N/A
Assumptions:	1. The customer knows where they want to go, when they want to travel, and how much they are willing to spend.
Notes and Issues:	1. How to confirm customer payment info.

Use Case ID:	UC-06		
Use Case Name:	Review Booking(s)		
Created By:	Jayro Alvarez	**Last Updated By:**	Royce Nguyen
Date Created:	Jul 10, 2018	**Last Revision Date:**	Jul 15, 2018
Actors:	Customer		
Description:	Allows the user to check on their current reservation.		
Trigger:	'Review Booking(s)' selection from menu.		
Preconditions:	1. Customer is logged in.		
Postconditions:	N/A		
Normal Flow:	1. Customer logs in or creates a new account. 2. Customer selects 'Review Booking(s)' menu option. 3. All booked flights on customer's account will be shown.		
Alternative Flows:	N/A		
Exceptions:	3a. In step 3 of the normal flow, if the customer has no preexisting flights 1. Popup such as "No Flights have been booked" will appear		
Includes:	1. Modify booking 2. Print flight info		
Frequency of Use:	On demand, everytime a customer wants to access any information relating to a previous flight they have booked.		
Special Requirements:	N/A		
Assumptions:	1. The customer knows to choose this menu option to access their booked flights.		
Notes and Issues:	N/A		

Use Case ID:	UC-07		
Use Case Name:	Modify Booking		
Created By:	Danny Pham	**Last Updated By:**	Danny Pham
Date Created:	Jul 12, 2018	**Last Revision Date:**	Aug 2, 2018
Actors:	Customer		
Description:	Allows the customer to modify their booking.		
Trigger:	User selects the Modify Booking option from the menu.		
Preconditions:	1. User is logged in 2. User has a booking		
Postconditions:	1. Booking information is updated 2. User is given a confirmation message		
Normal Flow:	1. Customer selects Modify Booking 2. Customer selects flight to modify 3. Customer updates flight information 4. Confirmation message is displayed 5. Customer returned to main menu		
Alternative Flows:	2a. In step 2 of the normal flow, if the customer does not have a booked flight 1. Output error message 2. Return to main menu 3a. In step 3 of the normal flow, the customer may opt to cancel the flight 1. Remove the booking 2. Return to step 4 of normal flow		
Exceptions:	3a. In step 3 of the normal flow, if the customer enters invalid information 1. Output error message specifying incorrect fields 2. Allow user to retry input		
Includes:	Cancel Booking		

Frequency of Use:	On demand
Special Requirements:	N/A
Assumptions:	1. Customer has a booked flight
Notes and Issues:	N/A

Use Case ID:	UC-08		
Use Case Name:	Cancel Booking		
Created By:	Royce Nguyen	**Last Updated By:**	Royce Nguyen
Date Created:	Jul 15, 2018	**Last Revision Date:**	Jul 15, 2018
Actors:	Customer		
Description:	Allows the customer to cancel their booking(s).		
Trigger:	User selects the Cancel Booking option from the menu.		
Preconditions:	1. User is logged in 2. User has a booking		
Postconditions:	1. Booking is canceled 2. User is given a confirmation message		
Normal Flow:	1. Customer selects Modify Booking 2. Customer selects flight to cancel 3. Customer updates flight information 4. Customer confirms cancelation 5. Customer enters password for account to verify 6. Confirmation message is displayed 7. Customer returned to main menu		
Alternative Flows:	4a. In step 4 if the customer decides to not cancel 1. Customer opts to not cancel 2. Customer returned to main menu		
Exceptions:	5a. In step 5 of the normal flow, if the customer enters invalid information 1. Output error message specifying incorrect fields 2. Allow user to retry input		
Includes:	N/A		
Frequency of Use:	On demand		

Special Requirements:	N/A
Assumptions:	1. Customer has a booked flight 2. Customer has an account
Notes and Issues:	N/A

Use Case ID:	UC-09		
Use Case Name:	Print Flight Information		
Created By:	Royce Nguyen	**Last Updated By:**	Royce Nguyen
Date Created:	Jul 15, 2018	**Last Revision Date:**	Jul 15, 2018
Actors:	Customer		
Description:	Allows the customer to print flight information from booking		
Trigger:	User selects the print option from reviewing bookings		
Preconditions:	1. User is logged in 2. User has a booking 3. User has a printer		
Postconditions:	1. Flight information is printed		
Normal Flow:	1. Customer selects Review Booking 2. Customer selects booked flight 3. Customer selects print option 4. Customer receives printed flight information		
Alternative Flows:	N/A		
Exceptions:	N/A		
Includes:	N/A		
Frequency of Use:	On demand		
Special Requirements:	N/A		
Assumptions:	1. Customer has a booked flight 2. Customer has an account 3. Customer has a printer		
Notes and Issues:	N/A		

Use Case ID:	UC-10		
Use Case Name:	Pay for Flight		
Created By:	Royce Nguyen	**Last Updated By:**	Brian Trinh
Date Created:	Jul 15, 2018	**Last Revision Date:**	Aug 3, 2018
Actors:	Customer		
Description:	Allows user to pay for booking		
Trigger:	User has chosen a flight to book		
Preconditions:	1. User is logged in 2. User has a flight selected to pay for		
Postconditions:	1. Booking is paid for 2. User is given a confirmation message		
Normal Flow:	1. Customer selects flight 2. Customer selects seats 3. Customer is prompted to input payment information 4. Customer confirms information 5. Customer receives a confirmation message		
Alternative Flows:	N/A		
Exceptions:	3a. In step 3 of the normal flow, if the customer enters invalid payment information 1. Output error message specifying incorrect fields 2. Allow user to retry input		
Includes:	Book flight		
Frequency of Use:	On demand		
Special Requirements:	N/A		
Assumptions:	1. Customer has a flight selected 2. Customer has an account		
Notes and Issues:	1. Payment will eventually deduct from a credit card.		

1.3 Functional Requirements

---

a) Find Flights

- Functionality:
  - The user can find a list of available flights to and from selective destinations.
- Inputs/Outputs:
  - Inputs:
    - Flight Number
    - A departure airport
    - An arrival airport
  - Outputs:
    - A list of available flights best matching search criteria
- Constraints in Functions and Data
  - Flight Number: User must input a valid flight number
  - Location: User must have valid start and end locations
  - Airline must have flights available
- Error Handling
  - If the user inputs an invalid start and end location, system outputs a message asking for valid locations

b) Create Account

- Functionality:
  - Allows user to create an account
- Inputs/Outputs:
  - Inputs:
    - String of account name, password, full name, email, billing address, phone numbers, date of birth, sex, payment info
  - Outputs:
    - Confirmation of account create
- Constraints in Functions and Data:
  - Account username must be unique
  - Password must reach a certain security level
  - Email account must be valid
- Error Handling
  - N/A

c) Log into Account

- Functionality:
  - Allows user to log into user account
- Inputs/Outputs:
  - Inputs:
    - Username and password strings
  - Outputs:

        - Correct Combination: Allowance into system and “Correct Login” prompt
        - Incorrect Combination: Reiteration of login screen
- Constraints in Functions and Data:
    - User does not have an account
    - Username and password do not match
- Error Handling
    - If account and/or password do not match, return an error message

d) Validate ID

- Functionality:
    - Checks the user’s ID with the database to validate and authenticate
- Inputs/Outputs
    - Input:
        - Customer username and password
    - Output:
        - Successful account creation or log in
- Constraints in Functions and Data:
    - N/A
- Error Handling:
    - Whitespace/unexpected symbols handled by comparison to existing IDs

e) Select Seat

- Functionality:
    - Allows user to visually determine where they would like to be seated in plane.
- Inputs/Outputs:
    - Input:
        - Desired flight
    - Output:
        - Graphic representation of seating chart of plane
        - Number of available and unavailable seats
- Constraints in Functions and Data:
    - Number of available seats > 0
- Error Handling:
    - Selection of unavailable seat results in error message prompting user to try again.

f) Book Flight

- Functionality:
    - Allows user to confirm and pay for selected seats on the flight chosen for the trip
- Inputs/Outputs:
    - Inputs:
        - User-selected flight

        - User-selected seat(s)
    - Outputs:
        - Total cost of trip
        - Flight information (flight number, departure/arrival time, etc.)
        - Page stating a confirmation email has been sent to the user account's email
- Constraints in Functions and Data:
    - Seat selection must be input using a string
- Error Handling
    - If no card found on account, error handled by allowing user to enter new credit card information. This information is saved to their account

g) Payment for Booking

- Functionality:
    - Allows user to input payment method to finalize flight booking
- Inputs/Outputs:
    - Inputs:
        - Card Number
        - Expiration Date
        - CVC Number
        - Billing Address
        - Select to pay using registered card info
    - Outputs:
        - Confirmation message of payment received
        - Confirmation message of booking stored into the account
- Constraints in Functions and Data:
    - Only integer data types for card number, expiration date, and CVC number
        - Card number stored as a string
        - Expiration date is a string of MM/YY
        - CVC is a 3 digit integer
        - Billing Address is stored as a string
- Error Handling
    - Not enough integers for card number prompts system to output invalid card error message
    - Invalid expiration date (Ex: past data) prompts system to output invalid data error message

h) Save Card Info

- Functionality:
    - Saves debit/credit card information to user account for future use
- Inputs/Outputs
    - Inputs:
        - Card Number
        - Expiration Date

            - CVC Number
            - Billing Address
        - Outputs:
            - True if successful, output message
            - False if unsuccessful, output message
- Constraints in Functions and Data:
    - Card number stored as string
    - Expiration date is a string of MM/YYYY
    - CVC is a 3 digit integer
    - Billing Address is stored as a string
- Error Handling:
    - Invalid inputs prompt an error message

i) Review Booking

- Functionality:
    - Allows the user to check on their current reservation.
- Inputs/Outputs:
    - Input(s):
        - "Review Booking" menu option selected
    - Output(s):
        - Booking ID
            - Corresponding Flight ID
            - Corresponding Seat Number
- Constraints in Functions and Data:
    - User must have a reserved flight ticket
- Error Handling:
    - Output error message when no bookings reserved

j) Modify Current Booking(s)

- Functionality: User has the option to modify their booked flights (i.e.: Change assigned seats)
- Inputs/Outputs:
    - Input:
        - "Modify Booking(s)" menu option selected
        - Select available seat(s) to switch seat assignment(s)
    - Output:
        - Current seating chart
        - Confirmation of seat change
- Constraints in Functions and Data:
    - Input for a seat change must be in string format (Ex: A10, B5, C6, etc.)
- Error Handling:
    - If user attempts to change to an already taken seat, output an error message and allow them to choose again

k) Cancel Current Booking(s)

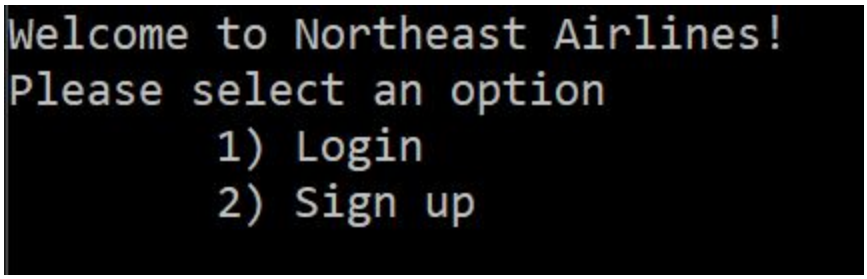
- Functionality: User has the option to cancel their booked flights
- Inputs/Outputs:
    - Input:
        - Select cancel option
    - Output:
        - Message to user confirming selection
        - Message outputting the refund amount due to cancellation
- Constraints in Functions and Data:
    - N/A
- Error Handling:
    - N/A

l) Print Flight Info

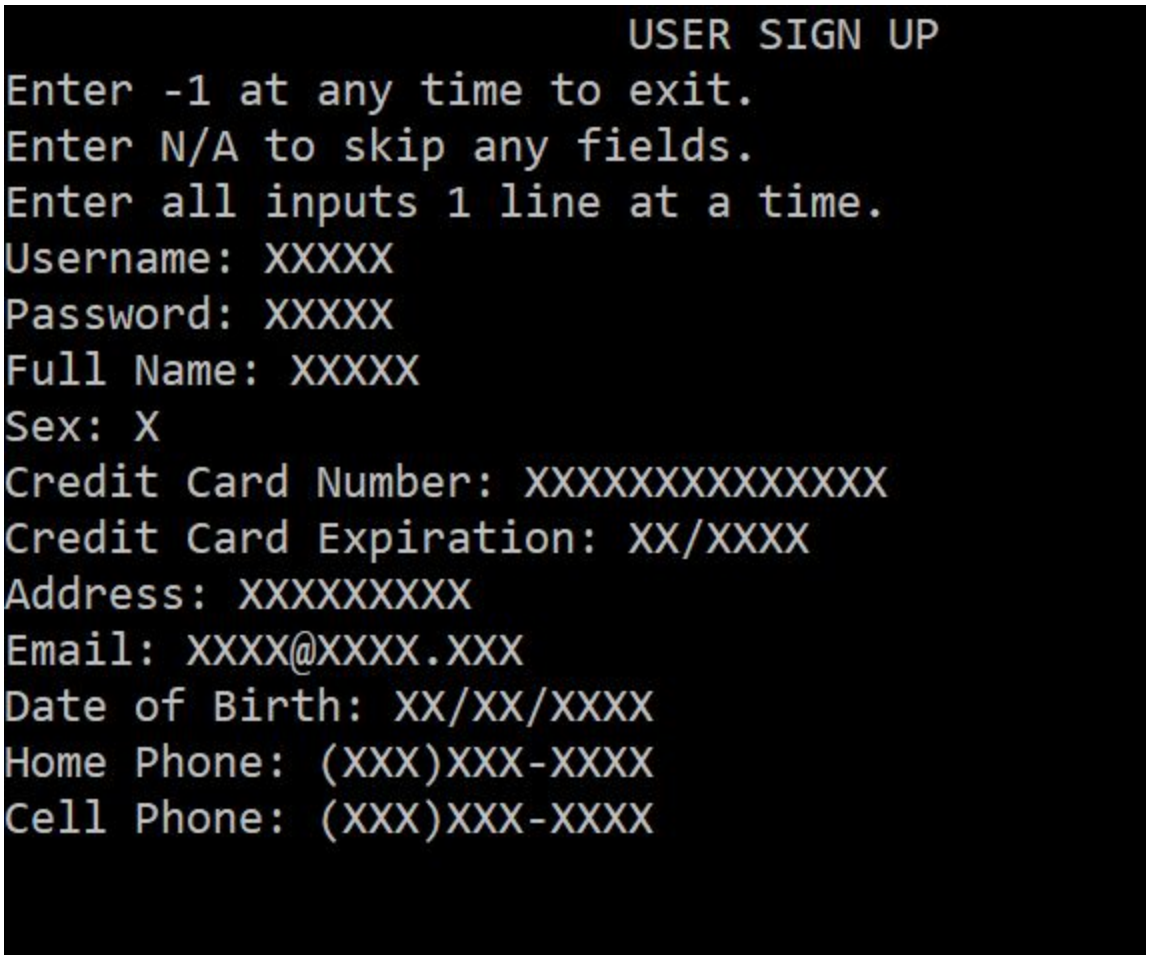
- Functionality: User can print out their boarding ticket for a booked flight
- Inputs/Outputs:
    - Input:
        - Select option to print booking
    - Output:
        - Confirmation of choice selected and display ticket with the following information:
            - Company name
            - Flight Number
            - Passenger Name
            - Account ID
            - Departure / Arrival Location
            - Seat
            - Departure / Arrival Time
- Constraints in Functions and Data:
    - User must have access to a printer
    - Printer must have ink available in order to print the ticket
- Error Handling:
    - N/A

1.4 External Interfaces

a) User Interfaces



1.4.1 Entry Point



1.4.2 Sign Up

```
                WELCOME
What would you like to do today?
        1) Search flights
        2) Review booked flights
        3) View Account Information
        4) Sign out
```

1.4.3 Main Menu

```
How would you like to search flights?
        1) Flight number
        2) Departure location
        3) Arrival location
        4) View all flights
        5) Return to main menu
```

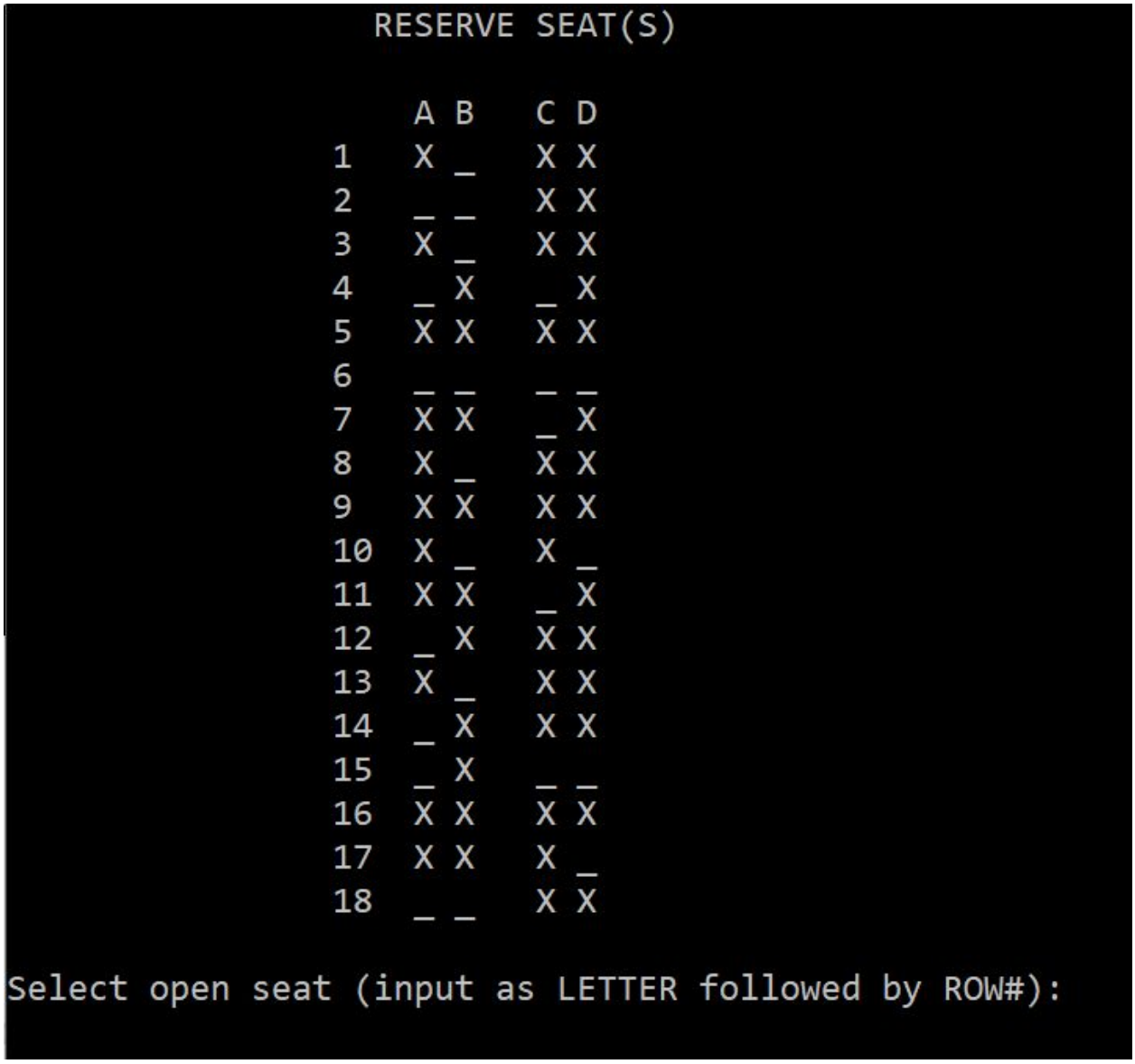
1.4.4 Searching for flights

```
                          ALL AVAILABLE FLIGHTS

FLIGHT NUMBER   TO      FROM    DEPARTURE DATE  ARRIVAL DATE    AVAILABLE SEATS
-------------------------------------------------------------------------------
NE123           CLT     BOS     8/1/2018 0800   8/1/2018 1015   24
NE435           BDL     JFK     8/1/2018 1200   8/1/2018 1500   35
NE234           LGA     PHL     8/1/2018 1130   8/1/2018 1300   35
NE724           PIT     JFK     8/1/2018 0400   8/1/2018 0600   30
NE525           BUF     BDL     8/1/2018 1600   8/1/2018 1930   24
NE770           BOS     PIT     8/1/2018 1900   8/1/2018 2130   35
NE563           JFK     PHL     8/1/2018 2200   8/1/2018 2330   35
NE912           LGA     BOS     8/1/2018 0700   8/1/2018 0850   30
NE266           CLT     BUF     8/1/2018 1230   8/1/2018 1420   24
NE397           PHL     CLT     8/1/2018 1500   8/1/2018 1645   35


Select flight to view available seats (Enter flight number):
```

1.4.5 Viewing flights



1.4.6 Seat selection

b) Hardware Interfaces

The major hardware characteristics revolve around only using a keyboard to control the whole system. The system itself runs on any piece of hardware that can handle C++ and since the software runs strictly on a terminal, the hardware does not have to rely on any U.I. hardware requirements such as minimum screen resolution or graphics cards.

c) Software Interface

This product has very specific connections to the software it is being run on. This is because the operating system being used, must be able to run C++ and work in a terminal.

d) Communications Interfaces

Initially, this product will not have any communication functions that are required. That is because of the lack of connections to external services, but in future versions there will be connections to email to send users confirmations and communications with network servers to update customer's information online.

1.5 Performance Requirements

There are a few major performance requirements that must be met when implementing this software:

1. When searching for flights under any parameter, the software should be able to access the stored flights in, at most, linear time.
2. When confirming a user's login information, the software should be able to access all of the system's user information in, at most, linear time.
3. When looking for **and** displaying all bookings under an account, the software should be able to access an account's bookings in, at most, linear time.

a) Reliability

This software should always be reliable; this can be attributed to how all major functions of the software run in linear time, in the worst-case scenario.

b) Response Time

Response time for this software is very fast since there are less flights than many bigger flight companies. This allocates far less time towards looking for objects such as accounts and flights.

1.6 Other Requirements

---

a) Time
This software's main features should be able to run in, at most, linear time.

b) Cost
The cost for maintaining this software should not be high. A few admin members should be able to update the system with up-to-date information for customers to continue regular usage.

c) Site Adaptation
This software should be easily adapted on new sites. As long as it's inputted with the necessary flights information, customers should be able to start making accounts and using the system to book flights.

d) Security
The security requirements for this software are very high priority. User accounts are stored with credit card information and this means that this software should be secure from hackers.

Chapter 2: Specification/Analysis Modeling

2.1 Introduction

---

a) Purpose

To outline the software's precise specifications and any constraints.

CPSC 362 - Introduction to Software Engineering
Summer Session 2018
Royce Nguyen, Jayro Alvarez, Danny Pham, Brian Trinh

b) Definitions, Acronyms, or Abbreviations

No terms, acronyms, or abbreviations used in this document that need to be defined prior to reading.

c) References

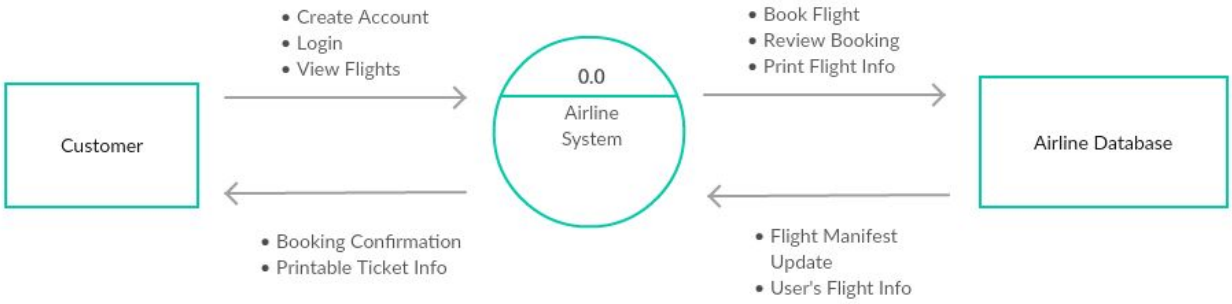
No outside documents referenced by this document.

d) Overview

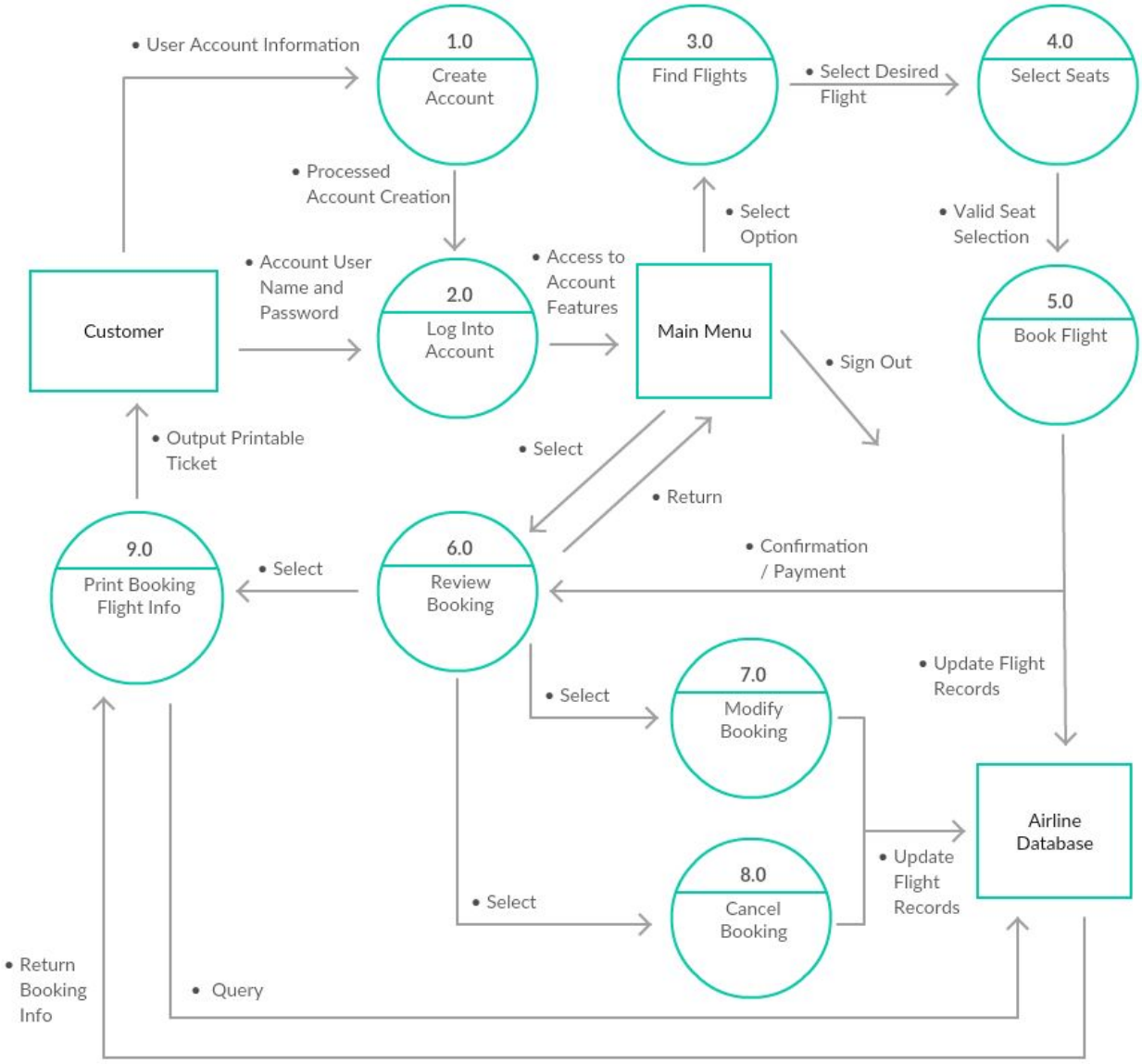
This section contains the software's Context and Class Diagrams.

2.2 Performance Requirements

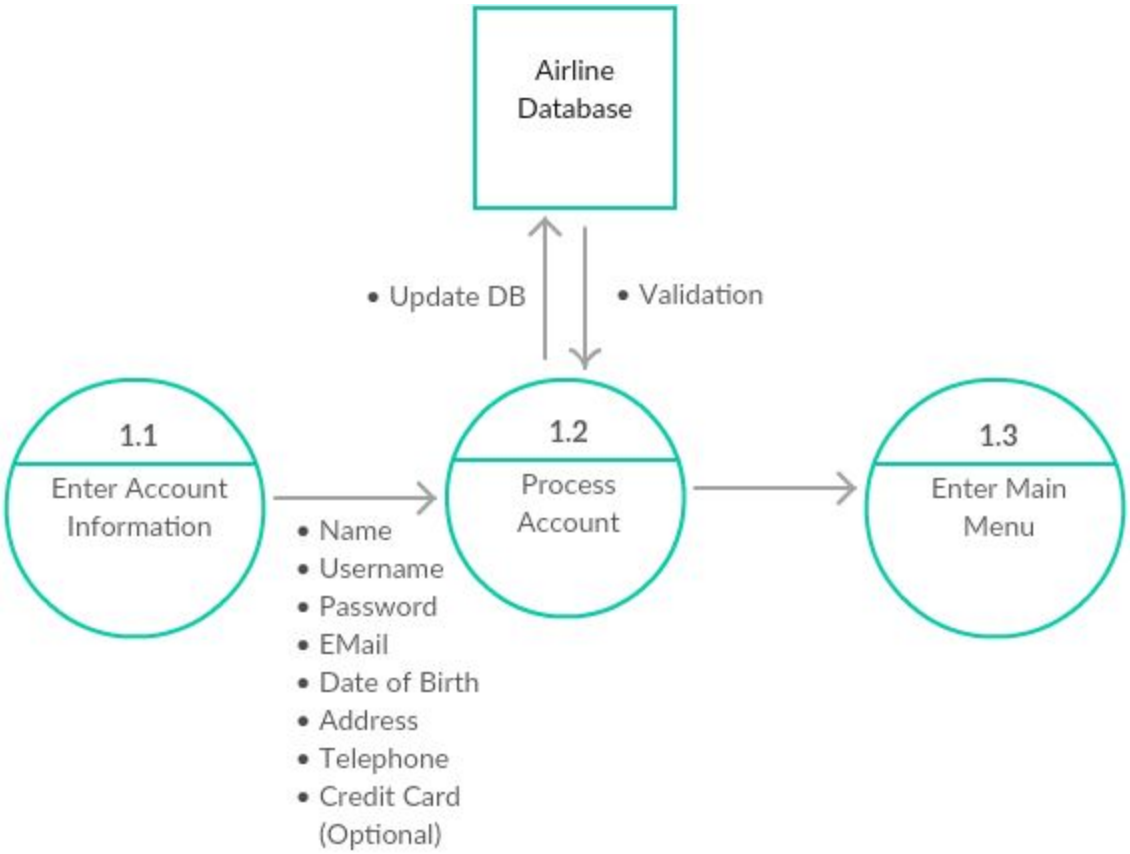
a) Level 0 Diagram

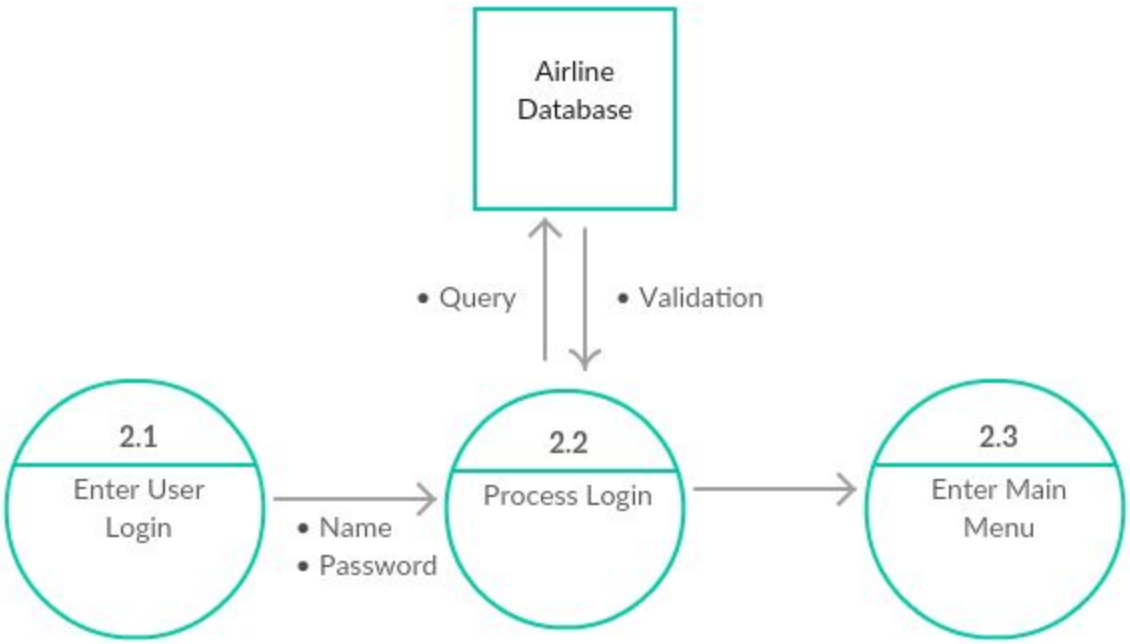


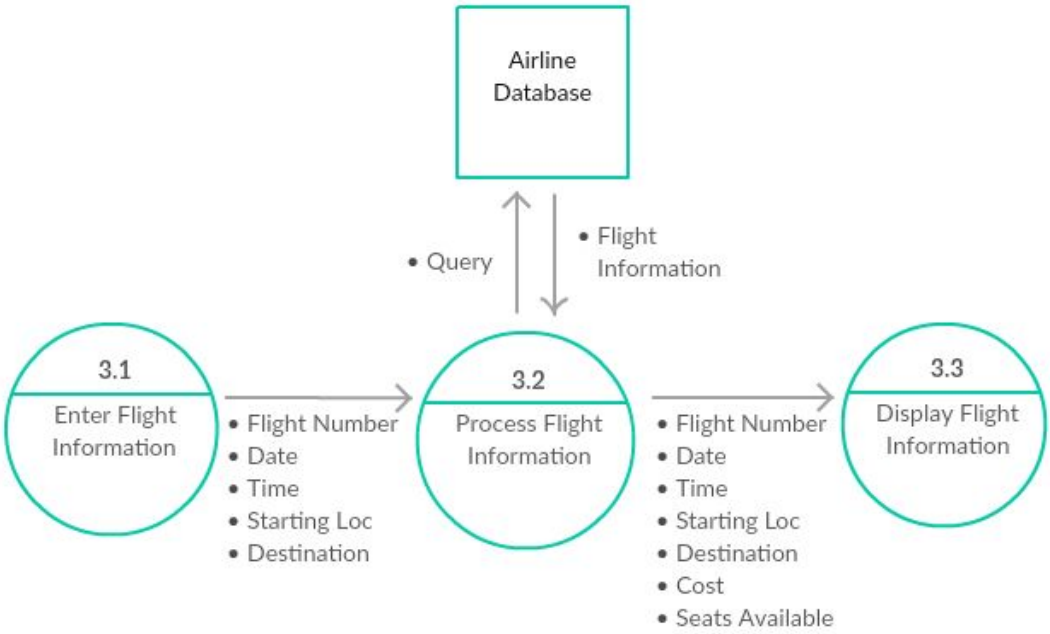
b) Level 1 Diagram

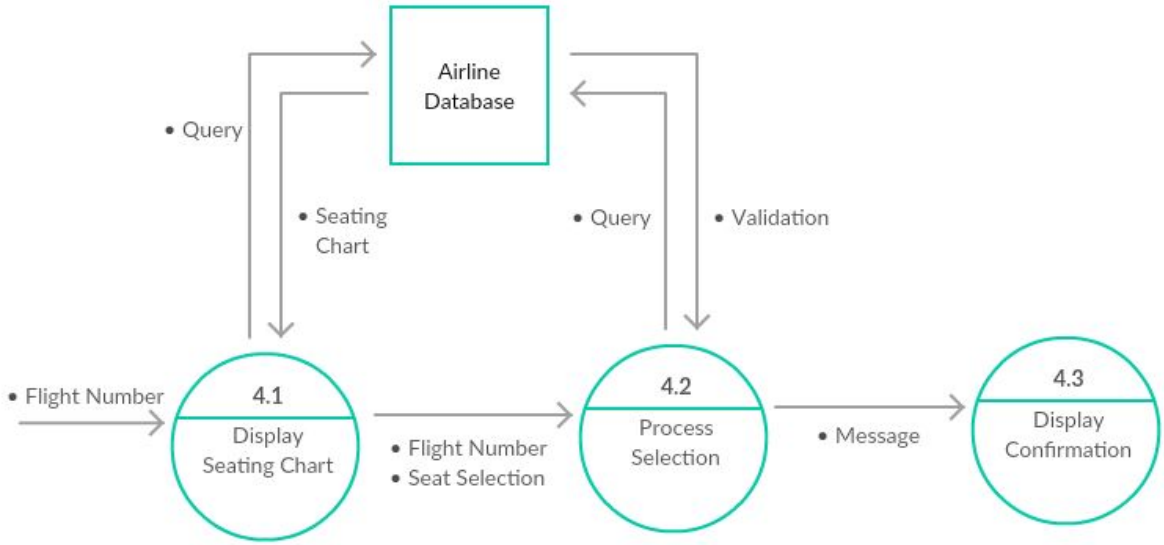


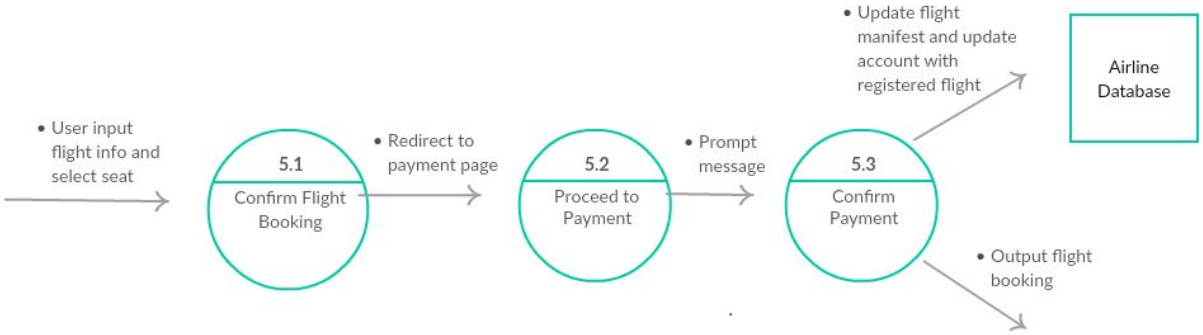
c) Level 2 Diagrams

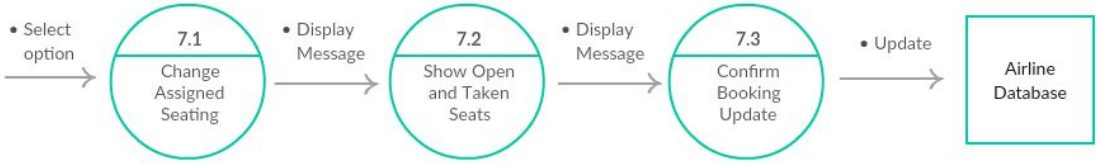


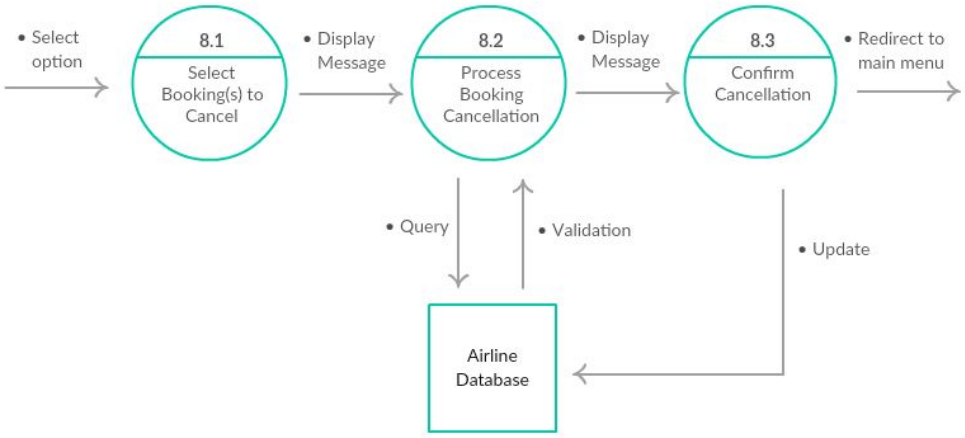



Airline Database
• Query
• Flight Information
3.1
Enter Flight Information
• Flight Number
• Date
• Time
• Starting Loc
• Destination
3.2
Process Flight Information
• Flight Number
• Date
• Time
• Starting Loc
• Destination
• Cost
• Seats Available
3.3
Display Flight Information


Airline Database
• Query
• Seating Chart
• Query
• Validation
• Flight Number
4.1
Display Seating Chart
• Flight Number
• Seat Selection
4.2
Process Selection
• Message
4.3
Display Confirmation

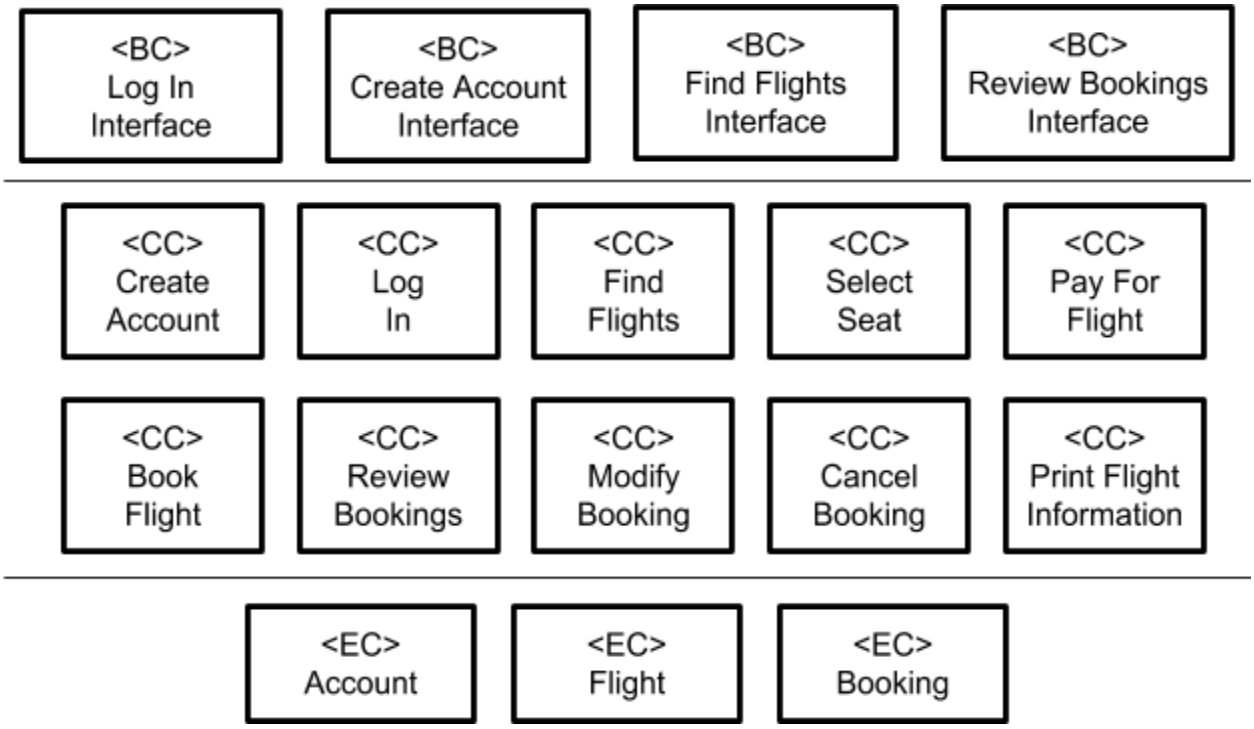

• User input flight info and select seat
5.1
Confirm Flight Booking
• Redirect to payment page
5.2
Proceed to Payment
• Prompt message
5.3
Confirm Payment
• Update flight manifest and update account with registered flight
Airline Database
• Output flight booking


• Select option
7.1
Change Assigned Seating
• Display Message
7.2
Show Open and Taken Seats
• Display Message
7.3
Confirm Booking Update
• Update
Airline Database

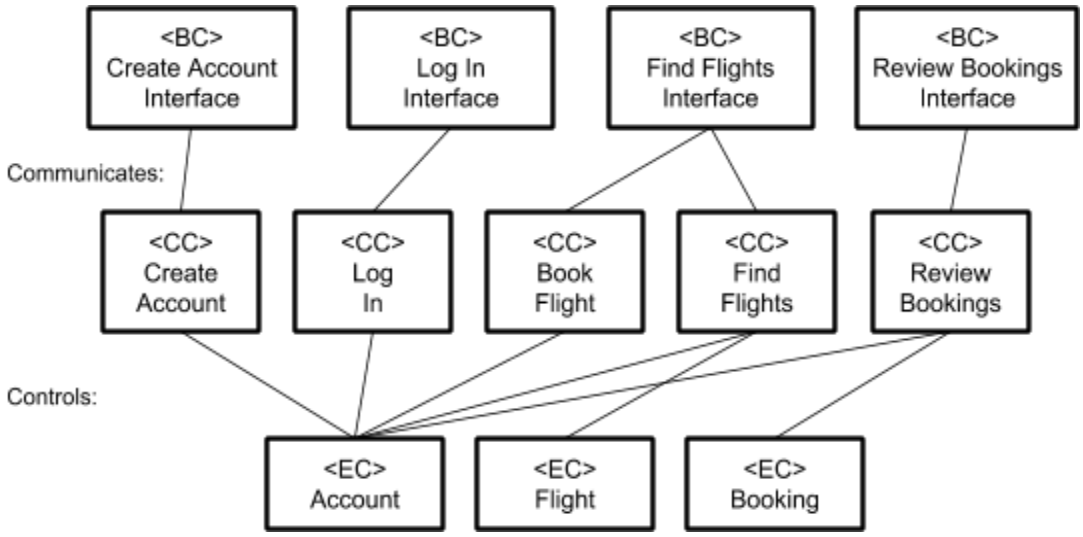

• Select option
8.1
Select Booking(s) to Cancel
• Display Message
8.2
Process Booking Cancellation
• Display Message
8.3
Confirm Cancellation
• Redirect to main menu
• Query
• Validation
• Update
Airline Database

2.3 Class Modeling

a) Initial Class Diagram



b) Simplified/Modified Class Diagram



Chapter 3: Design Modeling

3.1 Introduction

---

a) Purpose

To outline Functional and Object Oriented models in the designing of this software.

CPSC 362 - Introduction to Software Engineering
Summer Session 2018
Royce Nguyen, Jayro Alvarez, Danny Pham, Brian Trinh

b) Definitions, Acronyms or Abbreviations

No terms, acronyms, or abbreviations used in this document that need to be defined prior to reading.

c) References

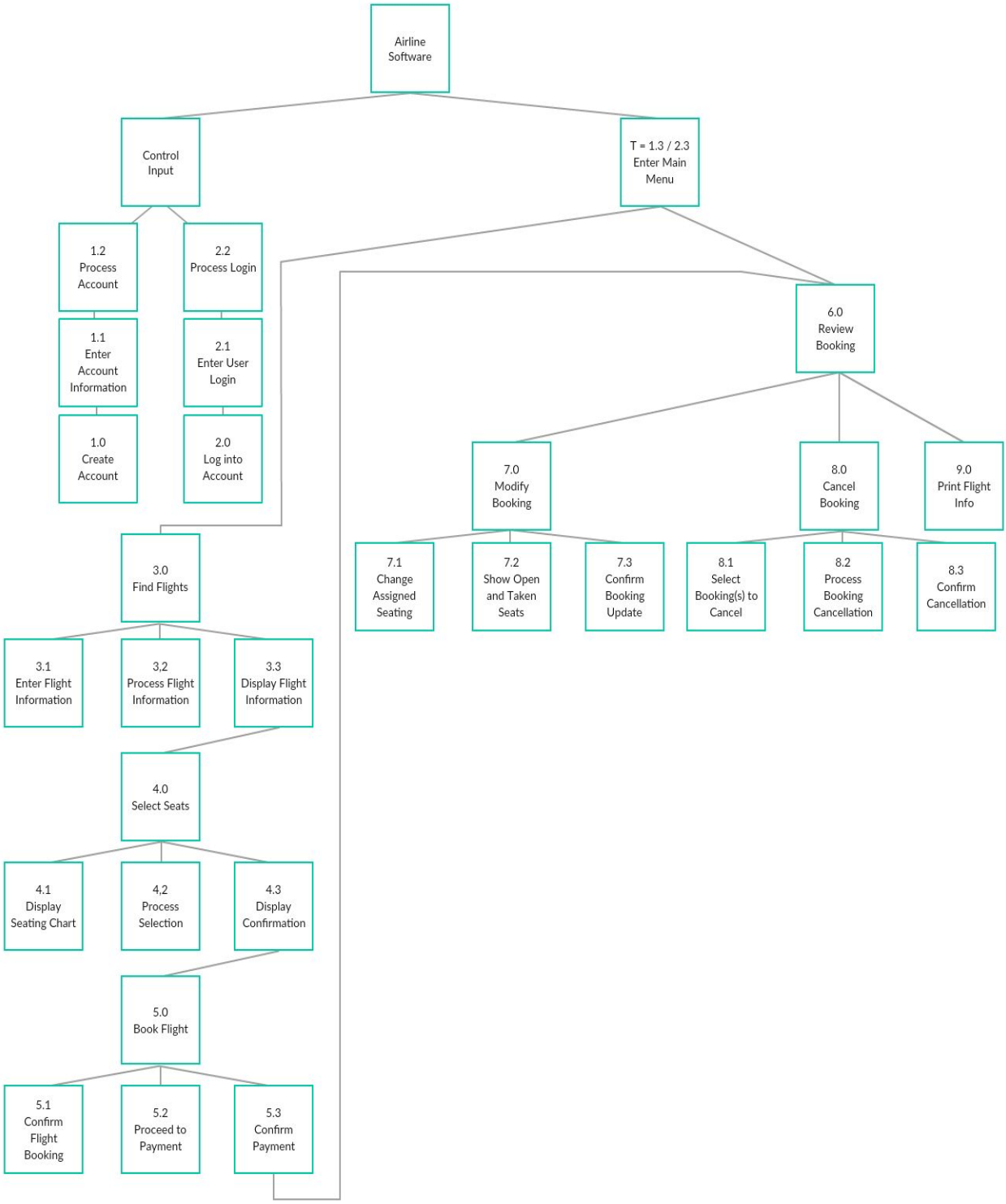
No outside documents referenced by this document.

d) Overview

This section contains the Program Structure Diagrams, Sequence Diagrams, and Detailed Class Diagrams that compose the software.

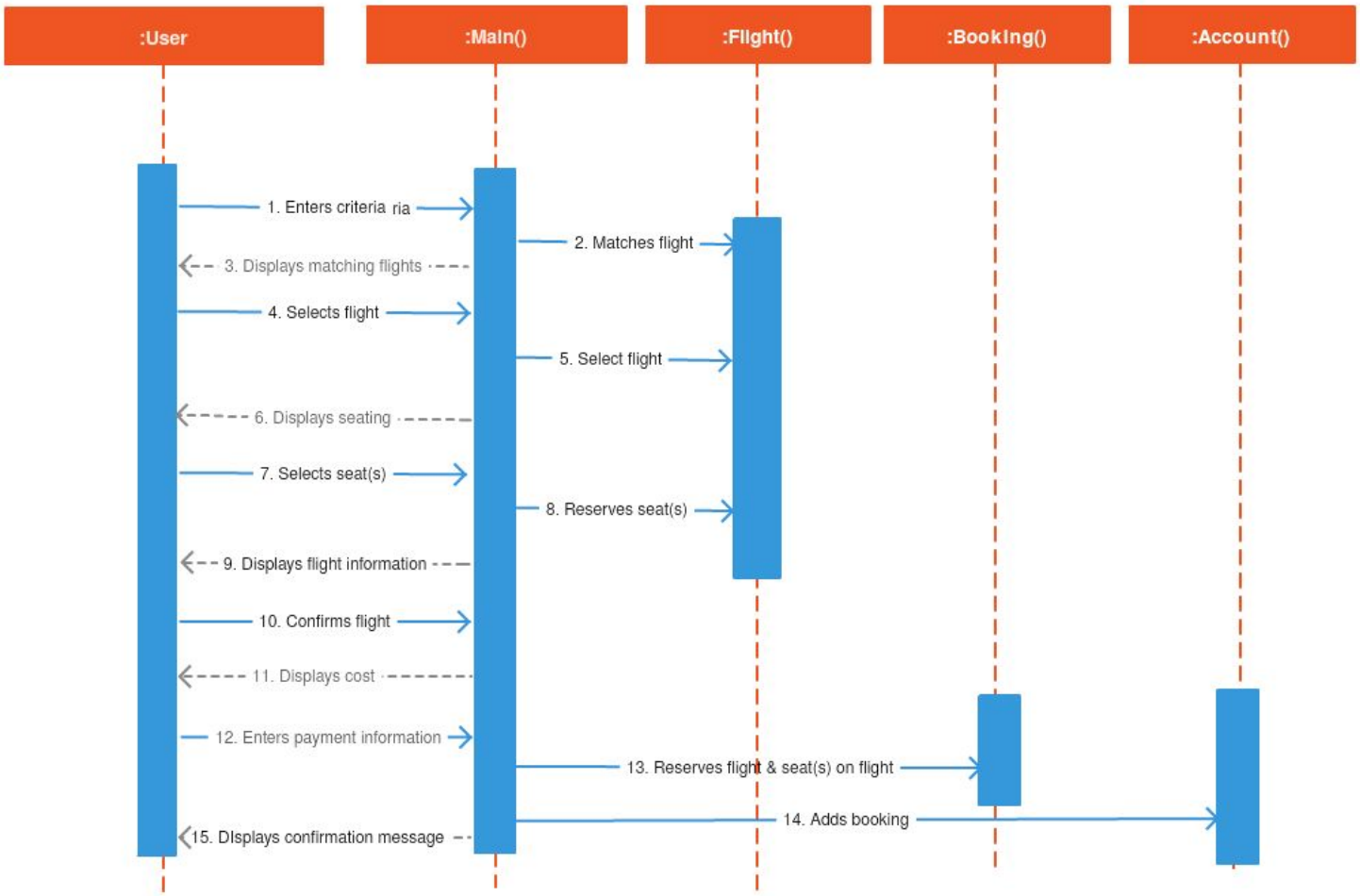
3.2 Functional Modeling

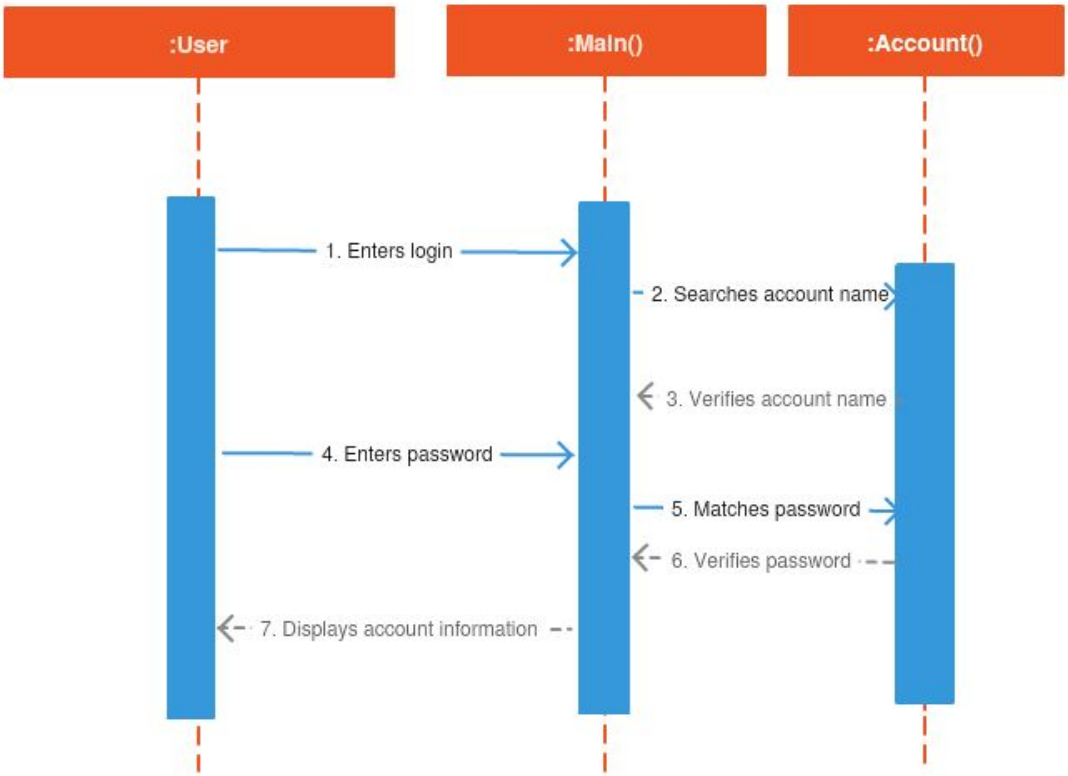
Program Structure Diagram

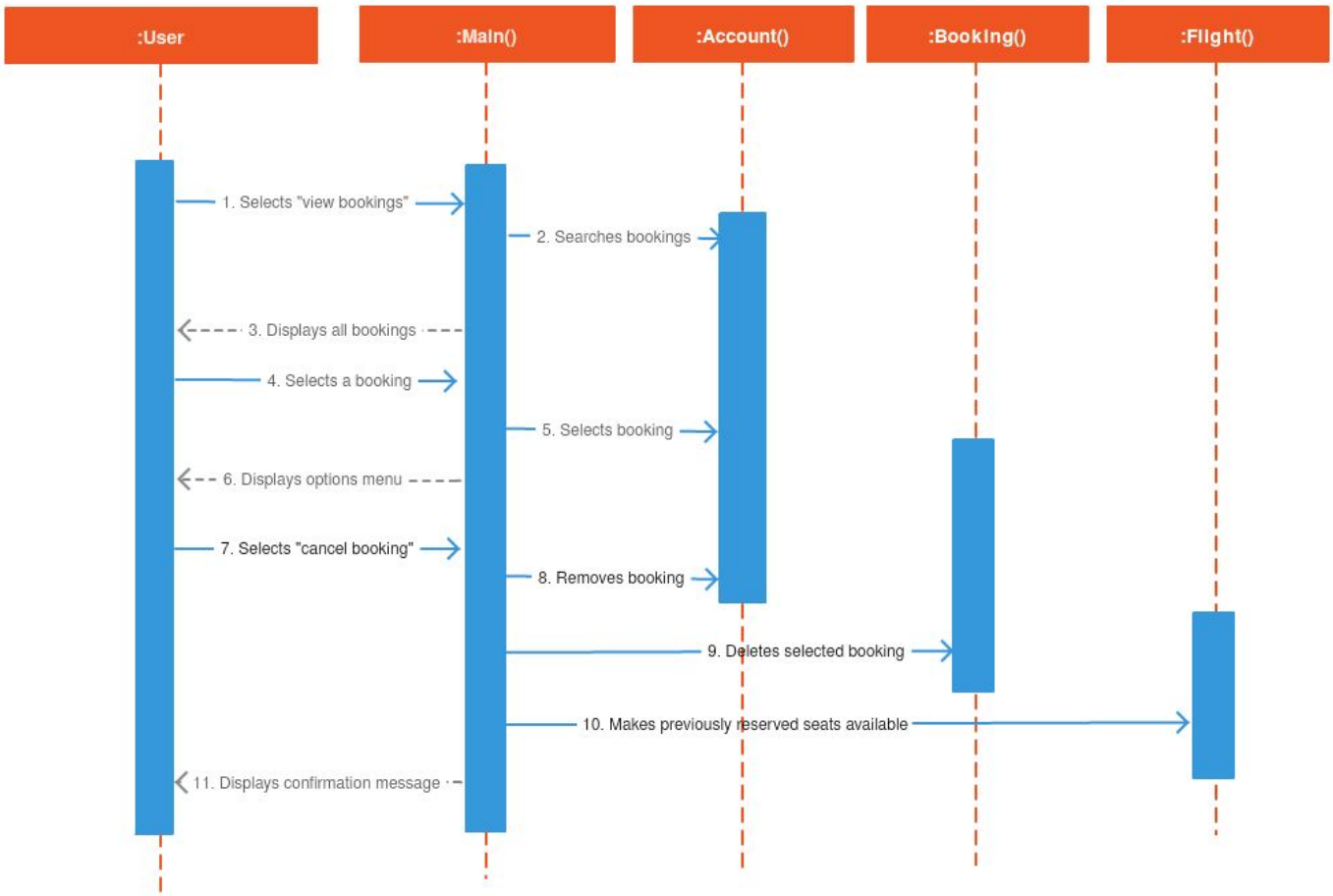


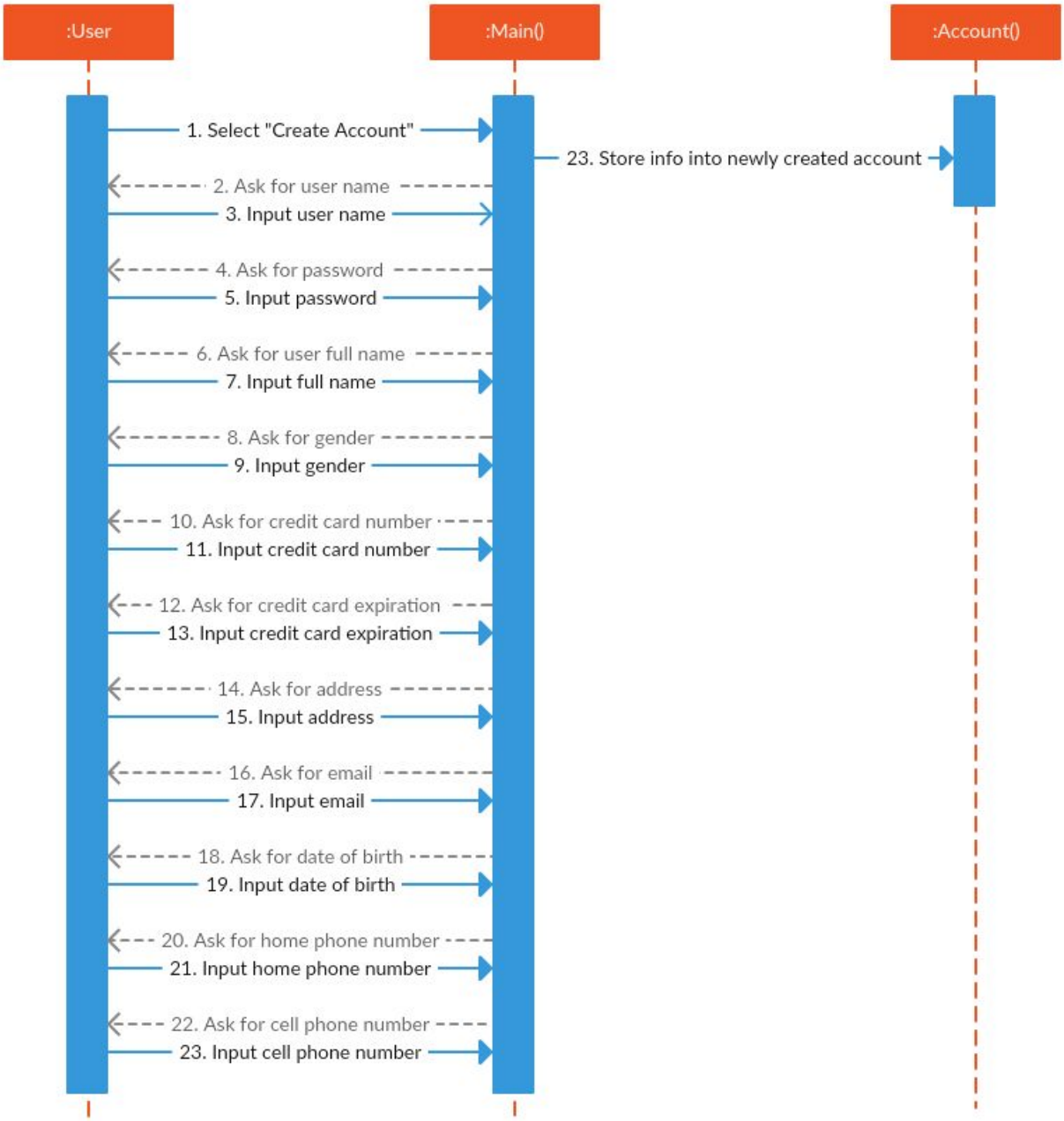
3.3 Object Oriented Modeling

a) Sequence Diagrams

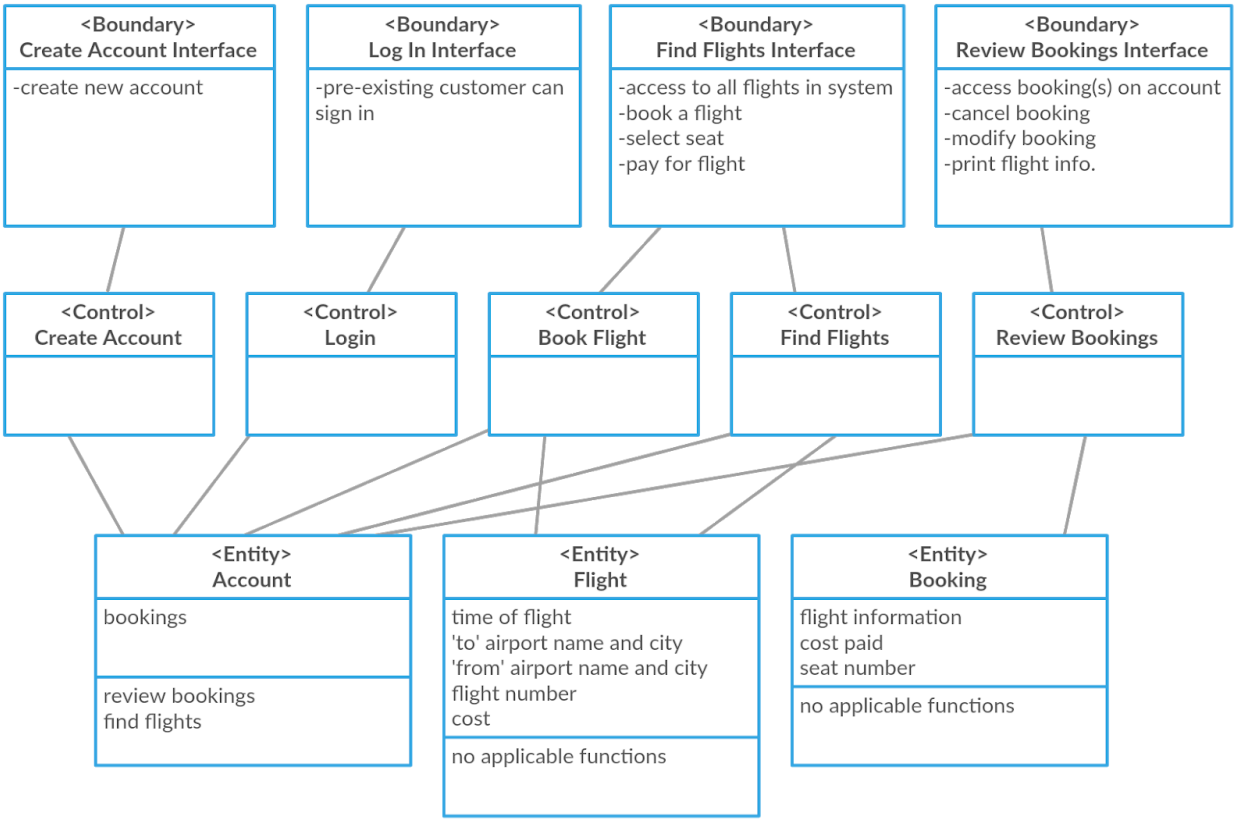



:User
:Main()
:Account()
1. Enters login
2. Searches account name
3. Verifies account name
4. Enters password
5. Matches password
6. Verifies password
7. Displays account information


:User
:Main()
:Account()
:Booking()
:Flight()
1. Selects "view bookings"
2. Searches bookings
3. Displays all bookings
4. Selects a booking
5. Selects booking
6. Displays options menu
7. Selects "cancel booking"
8. Removes booking
9. Deletes selected booking
10. Makes previously reserved seats available
11. Displays confirmation message


:User
:Main()
:Account()
1. Select "Create Account"
23. Store info into newly created account
2. Ask for user name
3. Input user name
4. Ask for password
5. Input password
6. Ask for user full name
7. Input full name
8. Ask for gender
9. Input gender
10. Ask for credit card number
11. Input credit card number
12. Ask for credit card expiration
13. Input credit card expiration
14. Ask for address
15. Input address
16. Ask for email
17. Input email
18. Ask for date of birth
19. Input date of birth
20. Ask for home phone number
21. Input home phone number
22. Ask for cell phone number
23. Input cell phone number

b) Detailed Class Diagram



Chapter 4: Implementation

4.1 Introduction

---

a) Purpose

This section covers the implementation environment and source code of the program.

CPSC 362 - Introduction to Software Engineering
Summer Session 2018
Royce Nguyen, Jayro Alvarez, Danny Pham, Brian Trinh

b) Definitions, Acronyms or Abbreviations

No terms, acronyms, or abbreviations used in this document that need to be defined prior to reading.

c) References

No outside documents referenced by this document.

d) Overview

This section describes the implementation environment of the program. Source code is documented in Appendix B.

4.2 Implementation Environment

The program was developed in C++ due to the scope of knowledge in the team utilizing Microsoft Visual Studio on Mac and Windows environments. The team had majority experience using C++ and taking into account the deadlines, would be unable to pick up a more proper front end language.

Chapter 5: Testing

5.1 Introduction

---

a) Purpose

To demonstrate testing of a particular feature to show error handling.

CPSC 362 - Introduction to Software Engineering
Summer Session 2018
Royce Nguyen, Jayro Alvarez, Danny Pham, Brian Trinh

b) Definitions, Acronyms or Abbreviations

No terms, acronyms, or abbreviations used in this document that need to be defined prior to reading.

c) References

No outside documents referenced by this document.

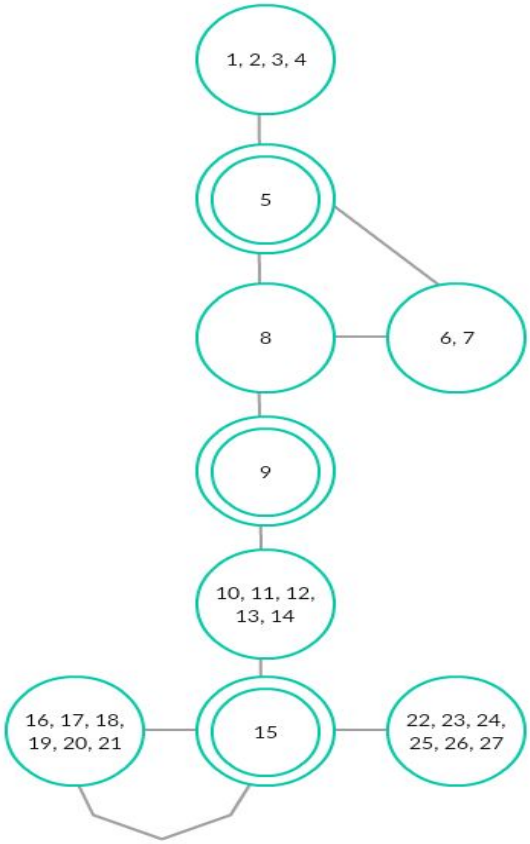
d) Overview

This section includes testing various modules using either the white box or black box testing methods. The white box testing method illustrates a module through a flow graph and creating test cases based on the cyclomatic complexity while the black box testing method use an Equivalence Partitioning approach to create test cases.

5.2 Module Testing

a) Module Name – Modify Booking

b) Independent Path Testing



Linearly Independent Paths:
1) 1 > 5 > 6 > 8 > 9 > 10 > > 15 > 22

2) 1 > 5 > 8 > 9 > 10 > 15 > 22

3) 1 > 5 > 8 > 9 > 10 > 15 > 16 > 15 > 22

i) Test Case #1
- Expected Value: Booking seat is not changed
- Test Result: Booking seat is not changed
- Conclusion: Passed

ii) Test Case #2
- Expected Value: Flight booking has been found
- Test Result: Flight booking has been found
- Conclusion: Passed

iii) Tese Case #3
- Expected Value: Booking seat has been changed
- Test Result: Booking seat has been changed
- Conclusion: Passed

c) Black Box Testing

Test cases based on Equivalence Classes, 1 Valid & 2 Invalid:

1. Select valid booking & Select Valid Seat
    - Expected Value: We expect a successfully modified booking
    - Test Result: Seat was changed on the booking
    - Conclusion: Passed
2. Select valid booking & Select Invalid Seat
    - Expected Value: We expect to not be able to modify a booking
    - Test Result: Seat remains unchanged
    - Conclusion: Not Passed
3. Select invalid booking
    - Expected Value: We expect to not be able to modify a booking
    - Test Result: Seat remains unchanged
    - Conclusion: Not Passed

5.3 Validation Testing

1. Module Name - Search Flights
2. Equivalence Class
    - Valid Flight Search (via Flight Number)
    - Invalid Flight Search (via Flight Number)
3. Test Cases and Test Results
    - Test Case #1
        i. Expected Value: Output flights based on user input
        ii. Test Result: List of flights displayed on the console
        iii. Conclusion: Passed
    - Test Case #2
        i. Expected Value: Invalid user input
        ii. Test Result: Invalid entry message
        iii. Conclusion: Passed

Chapter 6: User Manual

6.1 Introduction

---

a) Purpose

To describe the process of installation and run time of the project for the client to understand how it works.

CPSC 362 - Introduction to Software Engineering
Summer Session 2018
Royce Nguyen, Jayro Alvarez, Danny Pham, Brian Trinh

e) Definitions, Acronyms or Abbreviations

No terms, acronyms, or abbreviations used in this document that need to be defined prior to reading.

f) References

No outside documents referenced by this document.

b) Overview

The rest of this section would contain useful client-side information required for them to become fully capable of understanding how the program works, the limitations, how to operate it, and a demonstration of the working product.

6.2 Hardware Configuration

TBD

6.3 System Parameters

TBD

6.4 Operation Procedure

TBD

6.5 Demonstration

TBD

Appendix A

Team Meeting Reports

Group Name: 351 Survivors
Reporter: Brian Trinh
Date: 7/5/18 **Starting Time:** 10:14 AM **Ending Time:** 11:39 AM
Participants: Brian Trinh, Jayro Alvarez, Royce Nguyen

Missing: Danny Nguyen (Notified the team of absence ahead of time. Will forward group meeting report and update.)

Discussion Topic: Requirements Analysis (RA), Use Cases

Discussion Content:
1.1.a) Purpose section: Brief and straight-forward description the program's purpose.
1.1.b) Scope of the Problem: Name of program TBD, listed out possible functions.
1.1.c) Definitions, etc.: TBD
1.1.d) References: TBD
1.1.e) Overview: TBD

1.2.a) Use-Case Modeling: Created general blueprint of the model's setup through the list of functions.
1.2.b) Use-Case Description: TBD

1.3.a) Find Flights: Allows the user to find a list of available flights to and from selected locations
1.3.b) Seat Selection: Allows user to visually determine where they would like to be seated in plane.
1.3.c) Book/Reserve Flight: Allows user to confirm and pay for selected seats on the flight chosen for the trip
1.3.d Create Account: TBD
1.3.e) User Login: Allows user to log onto user account
1.3.f) Review Booking: Allows the user to check on their current reservation
1.3.g) Modify/Cancel Current Booking(s): User has the option to cancel or modify their booked flights (1.e.: booked flights, reserved a flight, etc.)
1.3.h) Print Flight Ticket: User can print out their boarding ticket, required to board their plane at the airport.

Reviews Conducted:
Finish Requirements Analysis (RA) by next Tuesday (7/10/18).

Problems Identified/Proposed Solutions:
No problems identified so far.
Any other comments:
Established Google Docs Folder for project documents.

Group Name: 351 Survivors
Reporter: Jayro Alvarez
Date: 7/10/18 **Starting Time:** 10:00 **Ending Time:** 11:40

Participants: Brian Trinh, Jayro Alvarez, Royce Nguyen, Danny Nguyen

Missing: N/A

Discussion Topic: Requirements Analysis (RA), Use Cases (Continued from Last Meeting Period)

Discussion Content:
1.2a) Use-case diagrams made
1.2b) Worked on use-case descriptions for:

1. Find Flights
2. Create Account
3. User Login
4. Select Seat
5. Book Flight

Reviews Conducted:
Last meeting we had concluded that we were going to complete the Requirements Analysis this session. We have not finished but have been making progress. Our goal for next session is to complete all remaining use-case descriptions and fill in the remaining sections of the Requirements Analysis doc.

Problems identified/Proposed Solutions:
Some issues with the layout and wording of our shared document for the Requirements Analysis, but solution was to make the order of each section match to eliminate any confusion.

Any other comments:
N/A

Group Name: 351 Survivors
Reporter: Danny Pham
Date: 7/12/2018 **Starting Time:** 10:37am **Ending Time:** 11:55am
Participants: Danny Pham, Brian Trinh, Jayro Alvarez

Missing: Royce Nguyen

Discussion Topic:

Discussion Content:
Allocation and planning of remainder of Phase 1 and 2
- Additional use case descriptions
- External interfaces
- Performance requirements

Project name – Northeast Airlines

Reviews Conducted:
Revision of Use Case Diagram
Planned sections 1 and 2 for deadline

Problems identified/Proposed Solutions:
Use case diagram does not follow proper flow – adjusted use case data flow

Any other comments:
N/A

Group Name: 351 Survivors
Reporter: Royce Nguyen
Date: 7/17/18 **Starting Time:** 9:35 AM **Ending Time:** 10:31
Participants: Brian Trinh, Jayro Alvarez, Royce Nguyen, Danny Pham

Missing:

Discussion Topic:
Phase 3, updating Phases 1 and 2 based on feedback given back

Discussion Content:
We discussed adding more functional requirements and updating the use case diagram.

Reviews Conducted:
Finished adding functional requirements and updated the use case diagram from Requirements Analysis.
Will complete Phase 3 Design Modeling by Thursday (7/19/18)
Splitting up work for Phase 3:
3.1 - Danny
3.2 - Brian
3.3 - Royce & Jayro

Problems Identified/Proposed Solutions:
There were an insufficient number of functional requirements and there were also problems with the use case diagram which were pointed out in the feedback.

Any other comments:
N/A

Group Name: 351 Survivors
Reporter: Brian Trinh
Date: 7/19/18 **Starting Time:** 9:54 AM **Ending Time:** 11:45 AM
Participants: Brian Trinh, Danny Pham, Jayro Alvarez

Missing: Royce Nguyen (Met on Google Docs)

Discussion Topic: Design Modeling (Ch. 3), Implementation (Ch. 4)

Discussion Content:
- Level 2 DFD for Program Structure Diagram (PSD)
- Split the work for the Level 2 DFDs for the PSD
- Completed Sequence Diagram
- Completed Detailed Class Diagram

Reviews Conducted:
- Complete PSD
- Work on Implementation
- Create Github for Implementation portion
- Create Discord for easier communication

Problems Identified/Proposed Solutions:
- How to split up implementation work between group members

Any other comments:
N/A

Group Name: 351 Survivors
Reporter: Jayro Alvarez
Date: 7/24/18 **Starting Time:** 10:15AM **Ending Time:** 11:10AM
Participants: Brian Trinh, Jayro Alvarez, Royce Nguyen, Danny Pham

Missing: N/A

Discussion Topic:
Phase 4 Implementation

Discussion Content:
We began discussing the specifics of our implementation. Started the code on visual studio together to determine what data types and classes are used. This is essential for us to be able to begin implementations on our own.
Started a shared Github repository so we all have access to each other's work.

Reviews Conducted:
Phase 3 Design Models completed from last week.
Will each start with implementation of assigned functions from last lab session and compare work on Thursday lab session(7/26/18).

Problems Identified/Proposed Solutions:
Code execution failure, uploaded to github for further inspection individually after lab session.

Any other comments:
N/A

Group Name: 351 Survivors
Reporter: Danny Pham
Date: 7/26/2017 **Starting Time:** 9:45 **Ending Time:** 11:45
Participants: Danny Pham, Brian Trinh, Jayro Alvarez

Missing: Royce Nguyen

Discussion Topic: Chapter 4, 5, 6, Implementation

Discussion Content:
Further work breakdown
Project integration on github
- Account.h
- Booking.h
- Updated Main()

Phase 6 complete

Reviews Conducted:
Revision of PSD to include case names
Revision on Lv1 Lv2 DFD

Problems identified/Proposed Solutions:
Errors with 2D array implementation ➔ implement as 1D array, manually slice

Any other comments:
N/A

Appendix B:

Source Code

Source Code - put this in Appendix B

Put your implementation code here with proper comments

i.e. ***module/class header*** and also within the body of a module

Module/class header: include

a) Module name or Class name in the Design

b) Date of the code

c) Programmer's name

d) Brief description of the class/service/function

e) Any important data structure in class/functions

f) choice of a specific algorithm within service/function

e.g. choosing quick sort rather than bubble sort etc.

=== Main CPP File ===

```
// Authors: Royce Nguyen, Jayro Alvarez, Brian Trinh, Danny Pham
// Course: CPSC 362
// Filename: FlightsUpdated.cpp
// Purpose: main
// Last updated: 7/31/2018

#include <iostream>
#include <string>
#include <vector>
#include <iomanip>
#include "Account.h"
#include "Booking.h"
#include "Flight.h"

using namespace std;

//=== FUNCTION PROTOTYPES ===

//print header for flights list
void printHeader();

//takes in seat# as int in seating array, returns seat# as string version
string seatIntToStr(int);

//takes in seat# as string, returns seat# as int index in seating array
int seatStrToInt(string);

int main()
{
	//--- Creation of different seat arrays ---
	//# of Rows: 18
	//# of Seats per Row: 4
	//Total Seats= 72
	char seating1[72] =
	{
		'X', 'O', 'X', 'X',
		'O', 'O', 'X', 'X',
		'X', 'O', 'X', 'X',
		'O', 'X', 'O', 'X',
		'X', 'X', 'X', 'X',
		'O', 'O', 'O', 'O',
		'X', 'X', 'O', 'X',
		'X', 'O', 'X', 'X',
		'X', 'X', 'X', 'X',
		'X', 'O', 'X', 'O',
		'X', 'X', 'O', 'X',
		'O', 'X', 'X', 'X',
		'X', 'O', 'X', 'X',
		'O', 'X', 'X', 'X',
		'O', 'X', 'O', 'O',
		'X', 'X', 'X', 'X',
		'X', 'X', 'X', 'O',
		'O', 'O', 'X', 'X'
	};
	char seating2[72] =
	{
		'O', 'O', 'X', 'X',
		'O', 'O', 'X', 'X',
		'X', 'O', 'X', 'X',
		'O', 'O', 'O', 'X',
```

```
        'X', 'X', 'X', 'O',
        'O', 'O', 'O', 'O',
        'X', 'X', 'O', 'X',
        'O', 'O', 'X', 'X',
        'X', 'O', 'X', 'X',
        'O', 'O', 'O', 'O',
        'X', 'X', 'O', 'X',
        'O', 'X', 'O', 'X',
        'X', 'O', 'X', 'O',
        'O', 'X', 'O', 'X',
        'X', 'O', 'O', 'O',
        'X', 'X', 'X', 'X',
        'X', 'X', 'X', 'O',
        'O', 'O', 'O', 'X'
};
char seating3[72] =
        {
        'O', 'O', 'X', 'O',
        'O', 'O', 'X', 'X',
        'X', 'O', 'X', 'X',
        'O', 'X', 'O', 'O',
        'X', 'X', 'O', 'X',
        'O', 'O', 'O', 'O',
        'X', 'X', 'O', 'O',
        'X', 'O', 'X', 'O',
        'O', 'X', 'X', 'X',
        'O', 'O', 'X', 'O',
        'X', 'X', 'O', 'X',
        'O', 'X', 'X', 'X',
        'X', 'O', 'X', 'X',
        'O', 'X', 'X', 'X',
        'O', 'X', 'O', 'O',
        'O', 'X', 'X', 'X',
        'X', 'X', 'O', 'O',
        'O', 'O', 'O', 'X'
};
char seating4[72] =
{
        'X', 'O', 'X', 'O',
        'O', 'O', 'O', 'X',
        'X', 'O', 'X', 'X',
        'O', 'X', 'O', 'X',
        'X', 'X', 'X', 'X',
        'O', 'O', 'O', 'O',
        'X', 'X', 'O', 'X',
        'O', 'O', 'X', 'X',
        'X', 'X', 'X', 'O',
        'X', 'O', 'X', 'O',
        'O', 'X', 'O', 'X',
        'O', 'X', 'X', 'X',
        'X', 'O', 'X', 'X',
        'O', 'X', 'X', 'X',
        'O', 'X', 'O', 'O',
        'X', 'X', 'X', 'X',
        'X', 'X', 'X', 'O',
        'O', 'O', 'O', 'X'
};


// === Hardcoded admin accounts ===
Account admin1 = Account("admin1", "admin", "Danny Pham", "N/A", "N/A", "10802 Stanford Ave, Garden Grove,
```

```
CA 92840", "dpham92@csu.fullerton.edu", "Sept 22, 1992", "N/A", "(949) 631 - 1166", "M");
	Account admin2 = Account("admin2", "admin", "Brian Trinh", "N/A", "N/A", "13221 Newhope St, Garden Grove, CA, 92843", "briantrinh@csu.fullerton.edu", "May 18, 1994", "N/A", "(714) 723 - 1637", "M");
	Account admin3 = Account("admin3", "admin", "Royce Nguyen", "N/A", "N/A", "N/A", "N/A", "N/A", "N/A", "N/A", "M");
	Account admin4 = Account("admin4", "admin", "Jayro Alvarez", "N/A", "N/A", "16064 Mount Lister Ct, Fountain Valley, CA 92708", "jayroalvarez@csu.fullerton.edu", "March 28, 1997", "N/A", "(949) 345 - 9831", "M");

	vector<Account> accountList;
	accountList.push_back(admin1);
	accountList.push_back(admin2);
	accountList.push_back(admin3);
	accountList.push_back(admin4);

	//=== Creation of Unique Flights ===
	Flight flight1 = Flight("NE123", "CLT", "BOS", 8, 1, 2018, 800, 8, 1, 2018, 1015, seating1, 449.99);
	Flight flight2 = Flight("NE435", "BDL", "JFK", 8, 1, 2018, 1200, 8, 1, 2018, 1500, seating2, 365.99);
	Flight flight3 = Flight("NE234", "LGA", "PHL", 8, 1, 2018, 1130, 8, 1, 2018, 1300, seating3, 299.99);
	Flight flight4 = Flight("NE724", "PIT", "JFK", 8, 1, 2018, 400, 8, 1, 2018, 600, seating4, 440.99);
	Flight flight5 = Flight("NE525", "BUF", "BDL", 8, 1, 2018, 1600, 8, 1, 2018, 1930, seating1, 349.50);
	Flight flight6 = Flight("NE770", "BOS", "PIT", 8, 1, 2018, 1900, 8, 1, 2018, 2130, seating2, 289.99);
	Flight flight7 = Flight("NE563", "JFK", "PHL", 8, 1, 2018, 2200, 8, 1, 2018, 2330, seating3, 449.50);
	Flight flight8 = Flight("NE912", "LGA", "BOS", 8, 1, 2018, 700, 8, 1, 2018, 850, seating4, 375.99);
	Flight flight9 = Flight("NE266", "CLT", "BUF", 8, 1, 2018, 1230, 8, 1, 2018, 1420, seating1, 450.00);
	Flight flight10 = Flight("NE397", "PHL", "CLT", 8, 1, 2018, 1500, 8, 1, 2018, 1645, seating2, 379.50);

	vector<Flight> flightList;
	flightList.push_back(flight1);
	flightList.push_back(flight2);
	flightList.push_back(flight3);
	flightList.push_back(flight4);
	flightList.push_back(flight5);
	flightList.push_back(flight6);
	flightList.push_back(flight7);
	flightList.push_back(flight8);
	flightList.push_back(flight9);
	flightList.push_back(flight10);

	// === Login Segment ===
	// Danny Pham 07/28/18
	// Module prompts the user to login or sign up
	// int accountListIter                    <- Holds index to signed in account
	cout << "Welcome to Northeast Airlines!\n";

	string username = "";
	bool userFound = false;
	bool userValidated = false;
	int accountListIter = 0;
	while ((username.compare("-1") != 0) && !userFound) {

		// ============== Login select ================
		string selection = "";
		cout << "Please select an option" << endl;
		cout << "\t1) Login" << endl;
		cout << "\t2) Sign up" << endl;
		cin >> selection;
		system("CLS");

		// ========================= Login ============================
		// User signs in with Username and Password
		if (selection.compare("1") == 0) {
```

```
		cout << "\t\t\tUSER LOGIN" << endl;
		cout << "Please enter your username (-1 to exit): ";
		cin >> username;

		accountListIter = 0;
		while ((accountListIter < accountList.size()) && !userFound) {
			if (accountList[accountListIter].getUsername() == username) {
				userFound = true;
			}
			else {
				accountListIter++;
			}
		}

		if (!userFound) {
			system("CLS");
			cout << flush;
			cout << "Username not found. Try again.\n";
		}
		else {
			cout << "Please enter your password: ";
			string pw = "";
			cin >> pw;
			if (accountList[accountListIter].validatePassword(pw)) {
				userValidated = true;
				system("CLS");
				cout << flush;
			}
			else {
				system("CLS");
				cout << flush;
				cout << "Incorrect Password!" << endl;
			}
		}
}

// ==== Sign up ====
// User inputs relevant fields
// Data stored into the accounts vector
else if (selection.compare("2") == 0) {
		cout << "\t\t\tUSER SIGN UP" << endl;
		string _userName;
		string _password;
		string _accountHolderName;
		string _creditCardNum;
		string _creditCardExpiration;
		string _billingAddress;
		string _email;
		string _dateOfBirth;
		string _homePhone;
		string _cellPhone;
		string _sex;

		cout << "Enter -1 at any time to exit." << endl;
		cout << "Enter N/A to skip any fields." << endl;
		cout << "Enter all inputs 1 line at a time." << endl;

		cout << "Username: ";
		cin >> _userName;

		cout << "Password: ";
```

```
                                cin >> _password;

                                cout << "Full Name: ";
                                cin >> ws;
                                getline(cin, _accountHolderName);

                                cout << "Sex: ";
                                cin >> ws;
                                getline(cin, _sex);

                                cout << "Credit Card Number: ";
                                cin >> ws;
                                getline(cin, _creditCardNum);

                                cout << "Credit Card Expiration: ";
                                cin >> ws;
                                getline(cin, _creditCardExpiration);

                                cout << "Address: ";
                                cin >> ws;
                                getline(cin, _billingAddress);

                                cout << "Email: ";
                                cin >> ws;
                                getline(cin, _email);

                                cout << "Date of Birth: ";
                                cin >> ws;
                                getline(cin, _dateOfBirth);

                                cout << "Home Phone: ";
                                cin >> ws;
                                getline(cin, _homePhone);

                                cout << "Cell Phone: ";
                                cin >> ws;
                                getline(cin, _cellPhone);

                                Account newAccount = Account(_userName, _password, _accountHolderName, _creditCardNum,
_creditCardExpiration, _billingAddress, _email, _dateOfBirth, _homePhone, _cellPhone, _sex);
                                accountList.push_back(newAccount);

                                accountListIter = accountList.size()-1;
                                userFound = true;
                                userValidated = true;
                                username = _userName;

                                system("CLS");
                                cout << flush;

                        }
                        else {
                                cout << "Invalid input. Please try again." << endl;
                        }
                }

                if (username.compare("-1") == 0) {
                        cout << "Exiting Northeast Airlines..." << endl;
                }

                // User is considered signed in if username and password are valid
```

```
if (userFound && userValidated) {

	cout << "\t\tWELCOME" << endl;
	string selection = "";
	while (selection.compare("4") != 0)
	{
		cout << "What would you like to do today?" << endl;
		cout << "\t1) Search flights" << endl;
		cout << "\t2) Review booked flights" << endl;
		cout << "\t3) View Account Information" << endl;
		cout << "\t4) Sign out" << endl;
		cin >> ws;
		getline(cin, selection);
		system("CLS");
		cout << flush;

		if (selection.compare("1") == 0)
		{
			int choice;
			//cout << "Searching flights" << endl;
			cout << "How would you like to search flights? " << endl;
			cout << "\t1) Flight number" << endl;
			cout << "\t2) Departure location" << endl;
			cout << "\t3) Arrival location" << endl;
			cout << "\t4) View all flights" << endl;
			cout << "\t5) Return to main menu" << endl;
			cin >> choice;
			system("CLS");
			cout << flush;

			string searchString;
			string flightChoice;
			string seatChoice;
			int flightsFound = 0;
			int counter = 0;
			if (choice == 1) // By flight #
			{
				cout << "\t\t\tSEARCH BY FLIGHT NUMBER" << endl;
				cout << "Enter flight number: ";
				cin >> searchString;

				bool searchFound = false;
				for (int i = 0; i < flightList.size(); i++)
				{
					if (searchString == flightList[i].getFlightNumber())
					{
						searchFound = true;
					}
				}
				while(searchFound == false)
				{
					system("CLS");
					cout << "\t\t\tSEARCH BY FLIGHT NUMBER" << endl;
					cout << "Invalid entry. Please enter valid flight number: ";
					cin >> searchString;
					for (int i = 0; i < flightList.size(); i++)
					{
						if (searchString == flightList[i].getFlightNumber())
						{
							searchFound = true;
						}
```

```
		}

	}
	system("CLS");
	cout << flush;
	cout << "\t\t\t\tSEARCH RESULTS" << endl << endl;
	printHeader();
	for (int i = 0; i < flightList.size(); i++)
	{
		if (searchString == flightList[i].getFlightNumber())
		{
			flightList[i].print();
			flightsFound++;
		}
	}

	cout << endl << endl;
	cout << "Select flight to view available seats (Enter flight number): ";
	cin >> flightChoice;
	bool flightChoiceGood = false;
	for (int i = 0; i < flightList.size(); i++)
	{
		if (flightChoice == flightList[i].getFlightNumber())
		{
			flightChoiceGood = true;
		}
	}
	while(flightChoiceGood == false)
	{
		system("CLS");
		cout << "\tNOTICE: INVALID FLIGHT SELECTED." << endl;
		cout << "Please select valid flight from list below." << endl << endl;
		printHeader();
		for (int i = 0; i < flightList.size(); i++)
		{
			if (searchString == flightList[i].getFlightNumber())
			{
				flightList[i].print();
				flightsFound++;
			}
		}
		cout << endl << endl;
		cout << "Select flight to view available seats (Enter flight number): ";
		cin >> flightChoice;
		for (int i = 0; i < flightList.size(); i++)
		{
			if (flightChoice == flightList[i].getFlightNumber())
			{
				flightChoiceGood = true;
			}
		}
	}
	int selectedFlight;   //assigned flight, initialized in loop below.

	system("CLS");
	cout << flush;
	for (int i = 0; i < flightList.size(); i++)
	{
		if (flightChoice == flightList[i].getFlightNumber())
		{
			selectedFlight = i;
```

```
                                                flightList[i].displaySeats();
                                        }
                                }
                                cout << "\nSelect open seat (input as LETTER followed by ROW#): ";
                                cin >> seatChoice;
                                while (flightList[selectedFlight].seats[seatStrToInt(seatChoice)] == 'X'){
                                        system("CLS");
                                        cout << flush;
                                        flightList[selectedFlight].displaySeats();
                                        cout << "\nINVALID, seat " << seatChoice<< " is taken, try again."
<< endl;
                                        cout << "Select open seat (input as LETTER followed by ROW#): ";
                                        cin >> seatChoice;
                                }
                                system("CLS");
                                cout << flush;
                                cout << "Seat, " << seatChoice << ", has been selected." << endl;
                                cout << "Would you like to:" << endl;
                                cout << "\t1) Pay for flight and add to bookings" << endl;
                                cout << "\t2) Return to Main Menu and lose progress" << endl;
                                cin >> choice;

                                while (choice != 1 && choice != 2)
                                {
                                        system("CLS");
                                        cout << flush;
                                        cout << "INVALID menu choice. Try again." << endl;
                                        cout << "Seat, " << seatChoice << ", has been selected." << endl;
                                        cout << "Would you like to:" << endl;
                                        cout << "\t1) Pay for flight and add to bookings" << endl;
                                        cout << "\t2) Return to Main Menu and lose progress" << endl;
                                        cin >> choice;
                                }
                                        //pay for flight
                                        if (choice == 1)
                                        {
                                                system("CLS");
                                                cout << flush;
                                                cout << "\t\tPAY FOR FLIGHT" << endl;
                                                cout << "\t\t==============" << endl;
                                                cout << "Flight Summary:  " << endl;
                                                cout << "Flight#: " << right << setw(41) <<
flightList[selectedFlight].getFlightNumber() << endl;
                                                cout << "Departing: " << right << setw(22) <<
flightList[selectedFlight].getDepartureLocation()
                                                << "- " << flightList[selectedFlight].getDepartureDate() <<
" @ " << right << setw(4) <<
                                                flightList[selectedFlight].getDepartureTime() << endl;
                                                cout << "Arriving: " << right << setw(23) <<
flightList[selectedFlight].getArrivalLocation()
                                                << "- " << flightList[selectedFlight].getArrivalDate() << "
@ " << right << setw(4) <<
                                                flightList[selectedFlight].getArrivalTime() << endl;
                                                cout <<
"================================================" << endl;
                                                cout << "Price (tax included): " << right << setw(28) <<
fixed << setprecision(2) << flightList[selectedFlight].getprice() << endl;
                                                cout <<
"================================================" << endl;
                                                cout << "1. Pay for flight with CC on account" << endl;
                                                cout << "2. Cancel and return to main menu" << endl;
```

```
                                                cin >> choice;
                                                if (choice == 1)
                                                {
                                                        if(accountList[accountListIter].checkCC() ==
false)
                                                        {
                                                                system("CLS");
                                                                string ccNum, ccExp;
                                                                cout << "\tERROR, NO CREDIT
CARD ON ACCOUNT." << endl;
                                                                cout << "Please enter following credit
card information: " << endl;
                                                                cout << "Enter 16 digit Credit Card: ";
                                                                cin >> ccNum;

accountList[accountListIter].setCreditCardNum(ccNum);
                                                                cout << "Enter expiration date in
format MM/YYYY: ";
                                                                cin >> ccExp;

accountList[accountListIter].setCreditCardExpiration(ccExp);
                                                                system("CLS");
                                                                cout << "Credit card: " <<
accountList[accountListIter].getCreditCardNum() << " has been saved." << endl;
                                                                cout <<
"=================================================" << endl;
                                                                cout << "Continue with charge of: " <<
flightList[selectedFlight].getprice() << "?" << endl;
                                                                cout << "1. Yes" << endl;
                                                                cout << "2. Cancel and return to main
menu" << endl;
                                                                cin >> choice;
                                                                if (choice == 1)
                                                                {
                                                                        system("CLS");
                                                                        //update seating array for
specified flight
flightList[selectedFlight].seats[seatStrToInt(seatChoice)] = 'X';
                                                                        //add booking to account
accountList[accountListIter].addBooking(flightList[selectedFlight].getFlightNumber(), accountList[accountListIter].getID(),
seatStrToInt(seatChoice));
                                                                        cout << "\t\tSUCCESS!" <<
endl;
                                                                        cout << "\tBOOKING
ADDED TO ACCOUNT." << endl;
                                                                        cout << "*(Review flights
from main menu to edit booking(s)*" << endl;
                                                                        cout <<
"=================================================" << endl;
                                                                }
                                                                else if (choice == 2)
                                                                {
                                                                        //do nothing, return to main
menu
                                                                        system("CLS");
                                                                }
                                                        }
                                                        else
                                                        {
```

```
                                                                                system("CLS");
                                                                                //update seating array for specified
flight

flightList[selectedFlight].seats[seatStrToInt(seatChoice)] = 'X';
                                                                                //add booking to account

accountList[accountListIter].addBooking(flightList[selectedFlight].getFlightNumber(), accountList[accountListIter].getID(),
seatStrToInt(seatChoice));
                                                                                cout << "\t\tSUCCESS!" << endl;
                                                                                cout << "\tBOOKING ADDED TO
ACCOUNT." << endl;
                                                                                cout << "Card Used: " <<
accountList[accountListIter].getCreditCardNum() << endl;
                                                                                cout << "*(Review flights from main
menu to edit booking(s)*" << endl;
                                                                                cout <<
"================================================" << endl;
                                                                                        }
                                                                        }
                                                                        else if(choice == 2)
                                                                        {
                                                                                //do nothing, return to main menu
                                                                                system("CLS");
                                                                        }
                                                                }
                                                                //return to main menu
                                                                else if (choice == 2)
                                                                {
                                                                        //do nothing, goes to main menu
                                                                        system("CLS");
                                                                        cout << flush;
                                                                }
                                                        }

                                                        else if (choice == 2) // By departure location
                                                        {
                                                                string depLoc;
                                                                cout << "\t\t\tSEARCH BY DEPARTURE LOCATION" << endl;
                                                                cout << "Enter departure airport: ";
                                                                cin >> searchString;
                                                                system("CLS");
                                                                cout << flush;

                                                                cout << "\t\t\t\tSEARCH RESULTS" << endl << endl;
                                                                printHeader();
                                                                for (int i = 0; i < flightList.size(); i++)
                                                                {
                                                                        if (searchString == flightList[i].getDepartureLocation())
                                                                        {
                                                                                flightList[i].print();
                                                                        }
                                                                }
                                                                cout << endl << endl;
                                                                cout << "Select flight to view available seats (Enter flight number): ";
                                                                cin >> flightChoice;
                                                                bool flightChoiceGood = false;
                                                                for (int i = 0; i < flightList.size(); i++)
                                                                {
                                                                        if (flightChoice == flightList[i].getFlightNumber())
                                                                        {
```

```
                                        flightChoiceGood = true;
                                }
                        }
                        while(flightChoiceGood == false)
                        {
                                system("CLS");
                                cout << "\tNOTICE: INVALID FLIGHT SELECTED." << endl;
                                cout << "Please select valid flight from list below." << endl << endl;
                                printHeader();
                                for (int i = 0; i < flightList.size(); i++)
                                {
                                        if (searchString == flightList[i].getDepartureLocation())
                                        {
                                                flightList[i].print();
                                                flightsFound++;
                                        }
                                }
                                cout << endl << endl;
                                cout << "Select flight to view available seats (Enter flight number): ";
                                cin >> flightChoice;
                                for (int i = 0; i < flightList.size(); i++)
                                {
                                        if (flightChoice == flightList[i].getFlightNumber())
                                        {
                                                flightChoiceGood = true;
                                        }
                                }
                        }
                        int selectedFlight;   //assigned flight, initialized in loop below.

                        system("CLS");
                        cout << flush;
                        for (int i = 0; i < flightList.size(); i++)
                        {
                                if (flightChoice == flightList[i].getFlightNumber())
                                {
                                        selectedFlight = i;
                                        flightList[i].displaySeats();
                                }
                        }
                        cout << "\nSelect open seat (input as LETTER followed by ROW#): ";
                        cin >> seatChoice;
                        while (flightList[selectedFlight].seats[seatStrToInt(seatChoice)] == 'X'){
                                system("CLS");
                                cout << flush;
                                flightList[selectedFlight].displaySeats();
                                cout << "\nINVALID, seat " << seatChoice<< " is taken, try again."
<< endl;
                                cout << "Select open seat (input as LETTER followed by ROW#): ";
                                cin >> seatChoice;
                        }
                        system("CLS");
                        cout << flush;
                        cout << "Seat, " << seatChoice << ", has been selected." << endl;
                        cout << "Would you like to:" << endl;
                        cout << "\t1) Pay for flight and add to bookings" << endl;
                        cout << "\t2) Return to Main Menu and lose progress" << endl;
                        cin >> choice;
                        while (choice != 1 && choice != 2){
                                system("CLS");
                                cout << flush;
```

```
                                        cout << "INVALID menu choice. Try again." << endl;
                                        cout << "Seat, " << seatChoice << ", has been selected." << endl;
                                        cout << "Would you like to:" << endl;
                                        cout << "\t1) Pay for flight and add to bookings" << endl;
                                        cout << "\t2) Return to Main Menu and lose progress" << endl;
                                        cin >> choice;
                                }
                                        //pay for flight
                                        if (choice == 1)
                                        {
                                                system("CLS");
                                                cout << flush;
                                                cout << "\t\tPAY FOR FLIGHT" << endl;
                                                cout << "\t\t==============" << endl;
                                                cout << "Flight Summary:  " << endl;
                                                cout << "Flight#: " << right << setw(41) <<
flightList[selectedFlight].getFlightNumber() << endl;
                                                cout << "Departing: " << right << setw(22) <<
flightList[selectedFlight].getDepartureLocation()
                                                << "- " << flightList[selectedFlight].getDepartureDate() <<
" @ " << right << setw(4) <<
                                                flightList[selectedFlight].getDepartureTime() << endl;
                                                cout << "Arriving: " << right << setw(23) <<
flightList[selectedFlight].getArrivalLocation()
                                                << "- " << flightList[selectedFlight].getArrivalDate() << "
@ " << right << setw(4) <<
                                                flightList[selectedFlight].getArrivalTime() << endl;
                                                cout <<
"================================================" << endl;
                                                cout << "Price (tax included): " << right << setw(28) <<
fixed << setprecision(2) << flightList[selectedFlight].getprice() << endl;
                                                cout <<
"================================================" << endl;
                                                cout << "1. Pay for flight with CC on account" << endl;
                                                cout << "2. Cancel and return to main menu" << endl;
                                                cin >> choice;
                                                if (choice == 1)
                                                {
                                                        if (accountList[accountListIter].checkCC() ==
false)
                                                        {
                                                                system("CLS");
                                                                string ccNum, ccExp;
                                                                cout << "\tERROR, NO CREDIT
CARD ON ACCOUNT." << endl;
                                                                cout << "Please enter following credit
card information: " << endl;
                                                                cout << "Enter 16 digit Credit Card: ";
                                                                cin >> ccNum;

accountList[accountListIter].setCreditCardNum(ccNum);
                                                                cout << "Enter expiration date in
format MM/YYYY: ";
                                                                cin >> ccExp;

accountList[accountListIter].setCreditCardExpiration(ccExp);
                                                                system("CLS");
                                                                cout << "Credit card: " <<
accountList[accountListIter].getCreditCardNum() << " has been saved." << endl;
                                                                cout <<
"================================================" << endl;
```

```
                                                                        cout << "Continue with charge of: " <<
flightList[selectedFlight].getprice() << "?" << endl;
                                                                        cout << "1. Yes" << endl;
                                                                        cout << "2. Cancel and return to main
menu" << endl;
                                                                        cin >> choice;
                                                                        if (choice == 1)
                                                                        {
                                                                                system("CLS");
                                                                                //update seating array for
specified flight

flightList[selectedFlight].seats[seatStrToInt(seatChoice)] = 'X';

                                                                                //add booking to account

accountList[accountListIter].addBooking(flightList[selectedFlight].getFlightNumber(), accountList[accountListIter].getID(),
seatStrToInt(seatChoice));
                                                                                cout << "\t\tSUCCESS!" <<
endl;
                                                                                cout << "\tBOOKING
ADDED TO ACCOUNT." << endl;
                                                                                cout << "*(Review flights
from main menu to edit booking(s)*" << endl;
                                                                                cout <<
"================================================" << endl;
                                                                        }
                                                                        else if (choice == 2)
                                                                        {
                                                                                //do nothing, return to main
menu
                                                                                system("CLS");
                                                                        }
                                                                }
                                                                else
                                                                {
                                                                        system("CLS");
                                                                        //update seating array for specified
flight

flightList[selectedFlight].seats[seatStrToInt(seatChoice)] = 'X';
                                                                        //add booking to account

accountList[accountListIter].addBooking(flightList[selectedFlight].getFlightNumber(), accountList[accountListIter].getID(),
seatStrToInt(seatChoice));
                                                                        cout << "\t\tSUCCESS!" << endl;
                                                                        cout << "\tBOOKING ADDED TO
ACCOUNT." << endl;
                                                                        cout << "Card Used: " <<
accountList[accountListIter].getCreditCardNum() << endl;
                                                                        cout << "*(Review flights from main
menu to edit booking(s)*" << endl;
                                                                        cout <<
"================================================" << endl;
                                                                }
                                                        }
                                                        else if (choice == 2)
                                                        {
                                                                //do nothing, return to main menu
                                                                system("CLS");
                                                        }
```

```
                        }
                        //return to main menu
                        else if (choice == 2)
                        {
                                //do nothing, goes to main menu
                                system("CLS");
                                cout << flush;
                        }
}

else if (choice == 3) // By arrival location
{
        string arrLoc;
        cout << "\t\t\tSEARCH BY ARRIVAL LOCATION" << endl;
        cout << "Enter arrival airport: ";
        cin >> searchString;
        system("CLS");
        cout << flush;


        cout << "\t\t\t\tSEARCH RESULTS" << endl << endl;
        printHeader();
        for (int i = 0; i < flightList.size(); i++)
        {
                if (searchString == flightList[i].getArrivalLocation())
                {
                        flightList[i].print();
                }
        }
        cout << endl << endl;
        cout << "Select flight to view available seats (Enter flight number): ";
        cin >> flightChoice;
        bool flightChoiceGood = false;
        for (int i = 0; i < flightList.size(); i++)
        {
                if (flightChoice == flightList[i].getFlightNumber())
                {
                        flightChoiceGood = true;
                }
        }
        while(flightChoiceGood == false)
        {
                system("CLS");
                cout << "\tNOTICE: INVALID FLIGHT SELECTED." << endl;
                cout << "Please select valid flight from list below." << endl << endl;
                printHeader();
                for (int i = 0; i < flightList.size(); i++)
                {
                        if (searchString == flightList[i].getArrivalLocation())
                        {
                                flightList[i].print();
                                flightsFound++;
                        }
                }
                cout << endl << endl;
                cout << "Select flight to view available seats (Enter flight number): ";
                cin >> flightChoice;
                for (int i = 0; i < flightList.size(); i++)
                {
                        if (flightChoice == flightList[i].getFlightNumber())
                        {
```

```
                                        flightChoiceGood = true;
                                }
                        }
                }
                int selectedFlight;   //assigned flight, initialized in loop below.

                system("CLS");
                cout << flush;
                for (int i = 0; i < flightList.size(); i++)
                {
                        if (flightChoice == flightList[i].getFlightNumber())
                        {
                                selectedFlight = i;
                                flightList[i].displaySeats();
                        }
                }
                cout << "\nSelect open seat (input as LETTER followed by ROW#): ";
                cin >> seatChoice;
                while (flightList[selectedFlight].seats[seatStrToInt(seatChoice)] == 'X'){
                        system("CLS");
                        cout << flush;
                        flightList[selectedFlight].displaySeats();
                        cout << "\nINVALID, seat " << seatChoice<< " is taken, try again."
<< endl;
                        cout << "Select open seat (input as LETTER followed by ROW#): ";
                        cin >> seatChoice;
                }
                system("CLS");
                cout << flush;
                cout << "Seat, " << seatChoice << ", has been selected." << endl;
                cout << "Would you like to:" << endl;
                cout << "\t1) Pay for flight and add to bookings" << endl;
                cout << "\t2) Return to Main Menu and lose progress" << endl;
                cin >> choice;
                while (choice != 1 && choice != 2){
                        system("CLS");
                        cout << flush;
                        cout << "INVALID menu choice. Try again." << endl;
                        cout << "Seat, " << seatChoice << ", has been selected." << endl;
                        cout << "Would you like to:" << endl;
                        cout << "\t1) Pay for flight and add to bookings" << endl;
                        cout << "\t2) Return to Main Menu and lose progress" << endl;
                        cin >> choice;
                }
                        //pay for flight
                        if (choice == 1)
                        {
                                system("CLS");
                                cout << flush;
                                cout << "\t\tPAY FOR FLIGHT" << endl;
                                cout << "\t\t==============" << endl;
                                cout << "Flight Summary:  " << endl;
                                cout << "Flight#: " << right << setw(41) <<
flightList[selectedFlight].getFlightNumber() << endl;
                                cout << "Departing: " << right << setw(22) <<
flightList[selectedFlight].getDepartureLocation()
                                << "- " << flightList[selectedFlight].getDepartureDate() <<
" @ " << right << setw(4) <<
                                flightList[selectedFlight].getDepartureTime() << endl;
                                cout << "Arriving: " << right << setw(23) <<
flightList[selectedFlight].getArrivalLocation()
```

```
                                                                << "- " << flightList[selectedFlight].getArrivalDate() << "
@ " << right << setw(4) <<
                                                                flightList[selectedFlight].getArrivalTime() << endl;
                                                                cout <<
"================================================" << endl;
                                                                cout << "Price (tax included): " << right << setw(28) <<
fixed << setprecision(2) << flightList[selectedFlight].getprice() << endl;
                                                                cout <<
"================================================" << endl;
                                                                cout << "1. Pay for flight with CC on account" << endl;
                                                                cout << "2. Cancel and return to main menu" << endl;
                                                                cin >> choice;
                                                                if (choice == 1)
                                                                {
                                                                        if (accountList[accountListIter].checkCC() ==
false)
                                                                        {
                                                                                system("CLS");
                                                                                string ccNum, ccExp;
                                                                                cout << "\tERROR, NO CREDIT
CARD ON ACCOUNT." << endl;
                                                                                cout << "Please enter following credit
card information: " << endl;
                                                                                cout << "Enter 16 digit Credit Card: ";
                                                                                cin >> ccNum;

accountList[accountListIter].setCreditCardNum(ccNum);
                                                                                cout << "Enter expiration date in
format MM/YYYY: ";
                                                                                cin >> ccExp;

accountList[accountListIter].setCreditCardExpiration(ccExp);
                                                                                system("CLS");
                                                                                cout << "Credit card: " <<
accountList[accountListIter].getCreditCardNum() << " has been saved." << endl;
                                                                                cout <<
"================================================" << endl;
                                                                                cout << "Continue with charge of: " <<
flightList[selectedFlight].getprice() << "?" << endl;
                                                                                cout << "1. Yes" << endl;
                                                                                cout << "2. Cancel and return to main
menu" << endl;
                                                                                cin >> choice;
                                                                                if (choice == 1)
                                                                                {
                                                                                        system("CLS");
                                                                                        //update seating array for
specified flight
flightList[selectedFlight].seats[seatStrToInt(seatChoice)] = 'X';
                                                                                        //add booking to account
accountList[accountListIter].addBooking(flightList[selectedFlight].getFlightNumber(), accountList[accountListIter].getID(),
seatStrToInt(seatChoice));
                                                                                        cout << "\t\tSUCCESS!" <<
endl;
                                                                                        cout << "\tBOOKING
ADDED TO ACCOUNT." << endl;
                                                                                        cout << "*(Review flights
from main menu to edit booking(s)*" << endl;
                                                                                        cout <<
```

```
"================================================" << endl;
                                                                                }
                                                                                else if (choice == 2)
                                                                                {
                                                                                        //do nothing, return to main
menu
                                                                                        system("CLS");
                                                                                }
                                                                        }
                                                                        else
                                                                        {
                                                                                system("CLS");
                                                                                //update seating array for specified
flight

flightList[selectedFlight].seats[seatStrToInt(seatChoice)] = 'X';
                                                                                //add booking to account

accountList[accountListIter].addBooking(flightList[selectedFlight].getFlightNumber(), accountList[accountListIter].getID(),
seatStrToInt(seatChoice));
                                                                                cout << "\t\tSUCCESS!" << endl;
                                                                                cout << "\tBOOKING ADDED TO
ACCOUNT." << endl;
                                                                                cout << "Card Used: " <<
accountList[accountListIter].getCreditCardNum() << endl;
                                                                                cout << "*(Review flights from main
menu to edit booking(s)*" << endl;
                                                                                cout <<
"================================================" << endl;
                                                                        }
                                                                }
                                                                else if (choice == 2)
                                                                {
                                                                        //do nothing, return to main menu
                                                                        system("CLS");
                                                                }
                                                        }
                                                        //return to main menu
                                                        else if (choice == 2)
                                                        {
                                                                //do nothing, goes to main menu
                                                                system("CLS");
                                                                cout << flush;
                                                        }
                                        }

                                        else if (choice == 4) // View all flights
                                        {
                                                cout << "\t\t\t\tALL AVAILABLE FLIGHTS" << endl << endl;
                                                printHeader();
                                                for (int i = 0; i < flightList.size(); i++)
                                                {
                                                        flightList[i].print();
                                                }
                                                cout << endl << endl;
                                                cout << "Select flight to view available seats (Enter flight number): ";
                                                cin >> flightChoice;
                                                bool flightChoiceGood = false;
                                                for (int i = 0; i < flightList.size(); i++)
                                                {
                                                        if (flightChoice == flightList[i].getFlightNumber())
```

```
                {
                        flightChoiceGood = true;
                }
        }
        while(flightChoiceGood == false)
        {
                system("CLS");
                cout << "\tNOTICE: INVALID FLIGHT SELECTED." << endl;
                cout << "Please select valid flight from list below." << endl << endl;
                printHeader();
                for (int i = 0; i < flightList.size(); i++)
                {
                        flightList[i].print();
                }
                cout << endl << endl;
                cout << "Select flight to view available seats (Enter flight number): ";
                cin >> flightChoice;
                for (int i = 0; i < flightList.size(); i++)
                {
                        if (flightChoice == flightList[i].getFlightNumber())
                        {
                                flightChoiceGood = true;
                        }
                }
        }
        int selectedFlight;   //assigned flight, initialized in loop below.

        system("CLS");
        cout << flush;
        for (int i = 0; i < flightList.size(); i++)
        {
                if (flightChoice == flightList[i].getFlightNumber())
                {
                        selectedFlight = i;
                        flightList[i].displaySeats();
                }
        }
        cout << "\nSelect open seat (input as LETTER followed by ROW#): ";
        cin >> seatChoice;
        while (flightList[selectedFlight].seats[seatStrToInt(seatChoice)] == 'X'){
                system("CLS");
                cout << flush;
                flightList[selectedFlight].displaySeats();
                cout << "\nINVALID, seat " << seatChoice<< " is taken, try again."
<< endl;
                cout << "Select open seat (input as LETTER followed by ROW#): ";
                cin >> seatChoice;
        }
        system("CLS");
        cout << flush;
        cout << "Seat, " << seatChoice << ", has been selected." << endl;
        cout << "Would you like to:" << endl;
        cout << "\t1) Pay for flight and add to bookings" << endl;
        cout << "\t2) Return to Main Menu and lose progress" << endl;
        cin >> choice;
        while (choice != 1 && choice != 2){
                system("CLS");
                cout << flush;
                cout << "INVALID menu choice. Try again." << endl;
                cout << "Seat, " << seatChoice << ", has been selected." << endl;
                cout << "Would you like to:" << endl;
```

```
                                        cout << "\t1) Pay for flight and add to bookings" << endl;
                                        cout << "\t2) Return to Main Menu and lose progress" << endl;
                                        cin >> choice;
                                }
                                        //pay for flight
                                        if (choice == 1)
                                        {
                                                system("CLS");
                                                cout << flush;
                                                cout << "\t\tPAY FOR FLIGHT" << endl;
                                                cout << "\t\t==============" << endl;
                                                cout << "Flight Summary:  " << endl;
                                                cout << "Flight#: " << right << setw(41) <<
flightList[selectedFlight].getFlightNumber() << endl;
                                                cout << "Departing: " << right << setw(22) <<
flightList[selectedFlight].getDepartureLocation()
                                                << "- " << flightList[selectedFlight].getDepartureDate() <<
" @ " << right << setw(4) <<
                                                flightList[selectedFlight].getDepartureTime() << endl;
                                                cout << "Arriving: " << right << setw(23) <<
flightList[selectedFlight].getArrivalLocation()
                                                << "- " << flightList[selectedFlight].getArrivalDate() << "
@ " << right << setw(4) <<
                                                flightList[selectedFlight].getArrivalTime() << endl;
                                                cout <<
"================================================" << endl;
                                                cout << "Price (tax included): " << right << setw(28) <<
fixed << setprecision(2) << flightList[selectedFlight].getprice() << endl;
                                                cout <<
"================================================" << endl;
                                                cout << "1. Pay for flight with CC on account" << endl;
                                                cout << "2. Cancel and return to main menu" << endl;
                                                cin >> choice;
                                                if (choice == 1)
                                                {
                                                        if (accountList[accountListIter].checkCC() ==
false)
                                                        {
                                                                system("CLS");
                                                                string ccNum, ccExp;
                                                                cout << "\tERROR, NO CREDIT
CARD ON ACCOUNT." << endl;
                                                                cout << "Please enter following credit
card information: " << endl;
                                                                cout << "Enter 16 digit Credit Card: ";
                                                                cin >> ccNum;

accountList[accountListIter].setCreditCardNum(ccNum);
                                                                cout << "Enter expiration date in
format MM/YYYY: ";
                                                                cin >> ccExp;

accountList[accountListIter].setCreditCardExpiration(ccExp);
                                                                system("CLS");
                                                                cout << "Credit card: " <<
accountList[accountListIter].getCreditCardNum() << " has been saved." << endl;
                                                                cout <<
"================================================" << endl;
                                                                cout << "Continue with charge of: " <<
flightList[selectedFlight].getprice() << "?" << endl;
                                                                cout << "1. Yes" << endl;
```

```
                                                                        cout << "2. Cancel and return to main
menu" << endl;
                                                                        cin >> choice;
                                                                        if (choice == 1)
                                                                        {
                                                                                system("CLS");
                                                                                //update seating array for
specified flight

flightList[selectedFlight].seats[seatStrToInt(seatChoice)] = 'X';
                                                                                //add booking to account

accountList[accountListIter].addBooking(flightList[selectedFlight].getFlightNumber(), accountList[accountListIter].getID(),
seatStrToInt(seatChoice));
                                                                                cout << "\t\tSUCCESS!" <<
endl;
                                                                                cout << "\tBOOKING
ADDED TO ACCOUNT." << endl;
                                                                                cout << "*(Review flights
from main menu to edit booking(s)*" << endl;
                                                                                cout <<
"================================================" << endl;
                                                                        }
                                                                        else if (choice == 2)
                                                                        {
                                                                                //do nothing, return to main
menu
                                                                                system("CLS");
                                                                        }
                                                                }
                                                                else
                                                                {
                                                                        system("CLS");
                                                                        //update seating array for specified
flight

flightList[selectedFlight].seats[seatStrToInt(seatChoice)] = 'X';
                                                                        //add booking to account

accountList[accountListIter].addBooking(flightList[selectedFlight].getFlightNumber(), accountList[accountListIter].getID(),
seatStrToInt(seatChoice));
                                                                        cout << "\t\tSUCCESS!" << endl;
                                                                        cout << "\tBOOKING ADDED TO
ACCOUNT." << endl;
                                                                        cout << "Card Used: " <<
accountList[accountListIter].getCreditCardNum() << endl;
                                                                        cout << "*(Review flights from main
menu to edit booking(s)*" << endl;
                                                                        cout <<
"================================================" << endl;
                                                                        }
                                                        }
                                                        else if (choice == 2)
                                                        {
                                                                //do nothing, return to main menu
                                                                system("CLS");
                                                        }
                                                }
                                                //return to main menu
                                                else if (choice == 2)
                                                {
```

```
                                                //do nothing, goes to main menu
                                                system("CLS");
                                                cout << flush;
                                        }
                                }
                                else if (choice == 5)
                                {
                                        //do nothing, goes to main menu
                                        //"system("CLS")" encountered below...
                                }
                        }
                        else if (selection.compare("2") == 0)
                        {
                                if (accountList[accountListIter].isBookingsEmpty() == true)
                                {
                                        cout << "No bookings were found on account. " << endl;
                                        cout << "Returning to main menu." << endl;
                                }
                                else
                                {
                                        string flightNum;
                                        string reviewMenuSelect;
                                        accountList[accountListIter].reviewBookings();
                                        cout << "Enter flight you want to modify, book, or print flight info for: " <<
endl;
                                        cout << "(Enter 0 to go back to main menu)" << endl;
                                        cout << "Flight #: ";
                                        cin >> flightNum;
                                        if (reviewMenuSelect != "0")
                                        {
                                                system("CLS");
                                                cout << "\tFlight Chosen: " << flightNum << endl;
                                                cout << "Choose option below: " << endl;
                                                cout << "\t1. Modify Flight " << endl;
                                                cout << "\t2. Cancel Booking " << endl;
                                                cout << "\t3. Print Flight Information " << endl;
                                                cout << "\t4. Return to main menu " << endl;
                                                cin >> reviewMenuSelect;

// ================================= MODIFY BOOKING ====================================

                                                /*
                                                a) Module Name: Modify Booking
                                                b) Date: July 31, 2018
                                                c) Programmer Name: Brian Trinh
                                                d) Description: Allows the customer to modify their booking.
                                                e) Important Data Structures: Vectors
                                                f) Choice of Algorithm: Brute Force (Searching through vectors)
                                                */

                                                if (reviewMenuSelect == "1")
                                                {
                                                        {
                                                                int bookingIndex = 0;
                                                                int bookingSeat = 0;

                                                                string seatChoice = "";

                                                                // Match user input to reservation from booking
vector
                                                                for (int i = 0; i <
```

```
accountList[accountListIter].bookingslist.size(); i++)
                                                                {
                                                                        if (flightNum ==
accountList[accountListIter].bookingslist[i].getFlightNumber())
                                                                        {
                                                                                bookingSeat =
accountList[accountListIter].bookingslist[i].getAssignedSeat();
                                                                                bookingIndex = i;
                                                                        }
                                                                }

                                                                for (int j = 0; j < flightList.size(); j++)
                                                                {
                                                                        if (flightNum ==
flightList[j].getFlightNumber())
                                                                        {
                                                                                system("CLS");
                                                                                cout << "Current Seating
Chart:\n";
                                                                                flightList[j].displaySeats();
                                                                                cout << "\nSelect an open
seat you would like to switch to (input as LETTER followed by ROW#): ";
                                                                                cin >> seatChoice;
                                                                                // Loop to ensure user
changes to an empty seat
                                                                                while
(flightList[j].seats[seatStrToInt(seatChoice)] == 'X')
                                                                                {
                                                                                        system("CLS");
                                                                                        cout << flush;
flightList[j].displaySeats();
                                                                                        cout <<
"\nINVALID, seat " << seatChoice << " is taken, try again." << endl;
                                                                                        cout << "Select
open seat (input as LETTER followed by ROW#): ";
                                                                                        cin >> seatChoice;
                                                                                }
                                                                                // Remove previous booking,
update seating chart, and create new booking with new seat.
accountList[accountListIter].bookingslist.erase(accountList[accountListIter].bookingslist.begin() + bookingIndex);

flightList[j].seats[bookingSeat] = '_';
flightList[j].seats[seatStrToInt(seatChoice)] = 'X';

accountList[accountListIter].addBooking(flightList[j].getFlightNumber(), accountList[accountListIter].getID(),
seatStrToInt(seatChoice));
                                                                                system("CLS");
                                                                                cout << "Booking has been
updated.\n\n";
                                                                        }
                                                                }
                                                        }
```

```
                                                }

// ================================= CANCEL BOOKING ====================================

                                                else if (reviewMenuSelect == "2")
                                                {
                                                        accountList[accountListIter].cancelBooking(flightNum,
flightList);
                                                }

// ================================= PRINT BOOKING ====================================

                                                else if (reviewMenuSelect == "3")
                                                {
                                                        accountList[accountListIter].printBooking(flightNum,
flightList);
                                                }
                                                else if (reviewMenuSelect == "4")
                                                {
                                                        //do nothing
                                                        //clear screen and go back to main menu
                                                        system("CLS");
                                                }
                                        }
                                        else
                                        {
                                                //do nothing
                                                //clear screen and go back to main menu
                                                system("CLS");
                                        }

                                }
                        }
                        else if (selection.compare("3") == 0) // Review account information
                        {
                                accountList[accountListIter].print();
                                system("pause");
                                system("CLS");
                        }
                        else if (selection.compare("4") == 0)
                        {
                                cout << "Thank you for using Northeast Airlines. Have a great day!" << endl;
                        }
                        else
                        {
                                cout << "Invalid selection." << endl;
                        }
                }
        }
        cout << endl;
        system("pause");
        return 0;
}
//===FUNCTION DEFINITIONS===

void printHeader()
{
        cout << "FLIGHT NUMBER\tTO\tFROM\tDEPARTURE DATE\tARRIVAL DATE\tAVAILABLE SEATS\n";
        cout << "-------------------------------------------------------------------------------\n";
}
```

```
string seatIntToStr(int x)
{
	string seatstring = "";
	int seatnum = x + 1;

	if(x == 0 || x == 4 || x == 8 || x == 12 || x == 16 ||  x == 20 ||  x == 24 ||
	 x == 28 || x == 32 ||  x == 36 ||  x == 40 ||  x == 44 ||  x == 48 ||  x == 52 ||
	  x == 56 ||  x == 60 ||  x == 64 ||  x == 68 )
	{
		seatstring += "A";
		if(x >= 4)
			seatnum = (seatnum/4) + 1;
		seatstring += to_string(seatnum);
	}
	else if(x == 1 || x == 5 || x == 9 || x == 13 || x == 17 ||  x == 21 ||  x == 25 ||
	 x == 29 || x == 33 ||  x == 37 ||  x == 41 ||  x == 45 ||  x == 49 ||  x == 53 ||
	  x == 57 ||  x == 61 ||  x == 65 ||  x == 69 )
	{
		seatstring += "B";
		if(x >= 5)
		{
			seatnum = (seatnum/4) + 1;
			seatstring += to_string(seatnum);
		}
		else
		seatstring += to_string(seatnum-1);
	}
	else if(x == 2 || x == 6 || x == 10 || x == 14 || x == 18 ||  x == 22 ||  x == 26 ||
	 x == 30 || x == 34 ||  x == 38 ||  x == 42 ||  x == 46 ||  x == 50 ||  x == 54 ||
	  x == 58 ||  x == 62 ||  x == 66 ||  x == 70 )
	{
		seatstring += "C";
		if(x >= 6)
		{
			seatnum = (seatnum/4) + 1;
			seatstring += to_string(seatnum);
		}
		else
		seatstring += to_string(seatnum-2);
	}
	else
	{
		seatstring += "D";
		if(x >= 7)
		{
			seatnum = (x/4) + 1;
			seatstring += to_string(seatnum);
		}
		else
		seatstring += to_string(seatnum-3);
	}
	return seatstring;
}

int seatStrToInt(string x)
{
	string rowNumAsString = "";
	rowNumAsString += x[1];
	if(x.length() > 2)
	{
		rowNumAsString += x[2];
```

```
	}
	//rowNumAsString now contains the one or two digit row (1-18),
	//here converted into int.
	int rowNumAsInt = stoi(rowNumAsString);
	//decrement by one to get as value from 0-17
	rowNumAsInt--;
	if (x[0] == 'A')
	{
		int arrayseatval = 0;
		for(int x = 0; x < rowNumAsInt; x++){
			arrayseatval += 4;
		}
		return arrayseatval;
	}
	else if (x[0] == 'B')
	{
		int arrayseatval = 1;
		for(int x = 0; x < rowNumAsInt; x++){
			arrayseatval += 4;
		}
		return arrayseatval;
	}
	else if (x[0] == 'C')
	{
		int arrayseatval = 2;
		for(int x = 0; x < rowNumAsInt; x++){
			arrayseatval += 4;
		}
		return arrayseatval;
	}
	else
	{
		int arrayseatval = 3;
		for(int x = 0; x < rowNumAsInt; x++){
			arrayseatval += 4;
		}
		return arrayseatval;
	}
}
```

== Account.h File ===

```
// Authors: Royce Nguyen, Jayro Alvarez, Brian Trinh, Danny Pham
// Course: CPSC 362
// Filename: Account.h
// Purpose: Holds user account information
// Last updated: 7/31/2018

/*
<EC>Account
Account entity class to hold account information
vector<Booking> bookingslist		<- Holds all bookings related to the account
static int id				<- Static variable used in account creation to generate a unique ID
*/

#ifndef ACCOUNT_H
#define ACCOUNT_H

#include <iostream>
#include <string>
#include <vector>
#include "Booking.h"
#include "Flight.h"
using namespace std;

class Account
{
	public:
		Account();
		Account(string, string, string, string, string, string, string, string, string, string, string);
		~Account();

		// Setters
		void setAccountHolderName(string);
		void setPassword(string);
		void setCreditCardNum(string);
		void setCreditCardExpiration(string);
		void setBillingAddress(string);
		void setEmail(string);
		void setDateOfBirth(string);
		void setHomePhone(string);
		void setCellPhone(string);

		// Getters
		int getID() const;
		string getName() const;
		string getUsername() const;
		string getCreditCardNum();

		// Functions
		bool validatePassword(string) const;
		void print() const;
		void addBooking(string, int, int);
		void reviewBookings();
		void printBooking(string flightNum, vector<Flight>);
		void modifyBooking(string flightNum, vector<Flight>&);
		void cancelBooking(string flightNum, vector<Flight>&);
		bool checkCC();
		string seatIntToStr(int);
		bool isBookingsEmpty();

		vector<Booking> bookingslist;
```

```
        private:
                int accountID;
                string userName;
                string password;
                string accountHolderName;
                string creditCardNum;
                string creditCardExpiration;
                string billingAddress;
                string email;
                string dateOfBirth;
                string homePhone;
                string cellPhone;
                string sex;

                static int id;
};

#endif
```

=== Account.cpp File ===

```
// Authors: Royce Nguyen, Jayro Alvarez, Brian Trinh, Danny Pham
// Course: CPSC 362
// Filename: Account.cpp
// Purpose: Holds user Account information
// Last updated: 7/31/2018


#include "Account.h"

int Account::id = 1;

Account::Account()
{
	accountID = id;
	id++;

	userName = "N/A";
	password = "N/A";
	accountHolderName = "N/A";
	creditCardNum = "N/A";
	creditCardExpiration = "N/A";
	billingAddress = "N/A";
	email = "N/A";
	dateOfBirth = "N/A";
	homePhone = "N/A";
	cellPhone = "N/A";
	sex = '0';
}

Account::Account(string _username, string _pw, string _name, string _ccn, string _cce, string _addr, string _email, string _dob,
string _home, string _cell, string _sex)
{
	accountID = id;
	id++;

	userName = _username;
	password = _pw;
	accountHolderName = _name;
	creditCardNum = _ccn;
	creditCardExpiration = _cce;
	billingAddress = _addr;
	email = _email;
	dateOfBirth = _dob;
	homePhone = _home;
	cellPhone = _cell;
	sex = _sex;
}


Account::~Account()
{
}

void Account::setAccountHolderName(string _name)
{
	accountHolderName = _name;
}

void Account::setPassword(string _pw)
{
```

```
	password = _pw;
}

void Account::setCreditCardNum(string _num)
{
	creditCardNum = _num;
}

void Account::setCreditCardExpiration(string _exp)
{
	creditCardExpiration = _exp;
}

void Account::setBillingAddress(string _addr)
{
	billingAddress = _addr;
}

void Account::setEmail(string _email)
{
	email = _email;
}

void Account::setDateOfBirth(string _dob)
{
	dateOfBirth = _dob;
}

void Account::setHomePhone(string _home)
{
	homePhone = _home;
}

void Account::setCellPhone(string _cell)
{
	cellPhone = _cell;
}

int Account::getID() const
{
	return accountID;
}

string Account::getName() const
{
	return accountHolderName;
}

string Account::getUsername() const
{
	return userName;
}

bool Account::validatePassword(string _pw) const
{
	return password == _pw;
}


void Account::print() const
{
```

```
	cout << "Hello, " << accountHolderName << "!\tID: " << accountID << "\n";
	cout << "Addr: " << billingAddress << endl;
	cout << "Home Phone: " << homePhone << endl;
	cout << "Cell Phone: " << cellPhone << endl;
	cout << "Credit Card Num: " << creditCardNum << endl;
	cout << "Credit Card Exp: " << creditCardExpiration << endl;
	cout << "Email: " << email << endl;
	cout << "DOB: " << dateOfBirth << endl;
	cout << "Sex: " << sex << endl;
	cout << "Username: " << userName << endl;
	//cout << "PW: " << password << endl;
	cout << "PW: ****" << endl;
}

void Account::addBooking(string _flightNum, int _accNum, int _seat)
{
	bookingslist.push_back(Booking(_flightNum, _accNum, _seat));
}

void Account::reviewBookings()
{
	cout << "\t\tREVIEWING FLIGHTS\n";
	//Review Bookings Header to align info.
		cout << "Booking ID:\t" << "Flight#:\t" << "Seat:\n";

		//Print to screen: all bookings inside of bookingslist vector
		for (int x = 0; x < bookingslist.size(); x ++)
		{
			cout << bookingslist[x].bookingID << "\t\t" << bookingslist[x].flightNumber << "\t\t" <<
			seatIntToStr(bookingslist[x].assignedSeat) << endl;
		}
		cout << "================================================" << endl;
}

/*
a) Module Name: Print Booking
b) Date: July 31, 2018
c) Programmer Name: Brian Trinh
d) Description: Allows the customer to print their booking.
e) Important Data Structures: Vectors
f) Choice of Algorithm: Brute Force (Searching through vectors)
*/

void Account::printBooking(string flightNum, vector<Flight> flightList)
{
	int index = 0;

	// Match user input to reservation from booking vector
	for (int i = 0; i < bookingslist.size(); i++)
	{
		if (flightNum == bookingslist[i].getFlightNumber())
		{
			index = i;
		}
	}

	/*
	If booking vector contained flight, match to flight vector
	Print ticket using information contained in flight vector index
	*/
	for (int j = 0; j < flightList.size(); j++)
```

```
	{
		if (flightNum == flightList[j].getFlightNumber())
		{
			system("CLS");
			cout << "Here is your ticket.\n";
			cout <<
"================================================================================\n"
				<< "\tNortheast Airlines Ticket" << "\tFlight Number: " <<
flightList[j].getFlightNumber() << "\n\n"
				<< "\tPassenger: " << accountHolderName << "\t\tAccount ID: " << accountID << "\n\n"
				<< "\tFrom: " << flightList[j].getDepartureLocation() << "\tSeat: " <<
seatIntToStr(bookingslist[index].assignedSeat) << "\tDeparture Time: " << flightList[j].getDepartureTime() << "\n"
				<< "\tTo: " << flightList[j].getArrivalLocation() << "\t\t\t\tArrival Time: " <<
flightList[j].getArrivalTime() << "\n"
				<<
"================================================================================\n\n";
		}
	}
}

/*
a) Module Name: Cancel Booking
b) Date: July 31, 2018
c) Programmer Name: Brian Trinh
d) Description: Allows the customer to cancel their booking.
e) Important Data Structures: Vectors
f) Choice of Algorithm: Brute Force (Searching through vectors)
*/

void Account::cancelBooking(string flightNum, vector<Flight> & flightList)
{
	int index = 0;
	int bookingSeat = 0;

	// Match user input to reservation from booking vector
	for (int i = 0; i < bookingslist.size(); i++)
	{
		if (flightNum == bookingslist[i].getFlightNumber())
		{
			bookingSeat = bookingslist[i].getAssignedSeat();
			index = i;
		}
	}

	for (int j = 0; j < flightList.size(); j++)
	{
		if (flightNum == flightList[j].getFlightNumber())
		{
			bookingslist.erase(bookingslist.begin() + index);

			flightList[j].seats[bookingSeat] = '_';

			system("CLS");
			cout << "Your booking has been cancelled.\n"
				<< "Refund amount: " << flightList[j].getprice() << "\n\n";
		}
	}
}

bool Account::checkCC()
{
```

```
	if (creditCardNum != "N/A" && creditCardExpiration != "N/A")
	{
		return true;
	}
	else
		return false;
}
string Account::getCreditCardNum()
{
	return creditCardNum;
}

//takes in seat# as int in seating array, returns seat# as string version
//(i.e: input 0 => returns "A1")
//(i.e: input 71 => returns "D18")
//(i.e: input 4 => "A2"
string Account::seatIntToStr(int x)
{
	string seatstring = "";
	int seatnum = x + 1;

	if (x == 0 || x == 4 || x == 8 || x == 12 || x == 16 || x == 20 || x == 24 ||
		x == 28 || x == 32 || x == 36 || x == 40 || x == 44 || x == 48 || x == 52 ||
		x == 56 || x == 60 || x == 64 || x == 68)
	{
		seatstring += "A";
		if (x >= 4)
			seatnum = (seatnum / 4) + 1;
		seatstring += to_string(seatnum);
	}
	else if (x == 1 || x == 5 || x == 9 || x == 13 || x == 17 || x == 21 || x == 25 ||
		x == 29 || x == 33 || x == 37 || x == 41 || x == 45 || x == 49 || x == 53 ||
		x == 57 || x == 61 || x == 65 || x == 69)
	{
		seatstring += "B";
		if (x >= 5)
		{
			seatnum = (seatnum / 4) + 1;
			seatstring += to_string(seatnum);
		}
		else
			seatstring += to_string(seatnum - 1);
	}
	else if (x == 2 || x == 6 || x == 10 || x == 14 || x == 18 || x == 22 || x == 26 ||
		x == 30 || x == 34 || x == 38 || x == 42 || x == 46 || x == 50 || x == 54 ||
		x == 58 || x == 62 || x == 66 || x == 70)
	{
		seatstring += "C";
		if (x >= 6)
		{
			seatnum = (seatnum / 4) + 1;
			seatstring += to_string(seatnum);
		}
		else
			seatstring += to_string(seatnum - 2);
	}
	else
	{
		seatstring += "D";
		if (x >= 7)
		{
```

```
                        seatnum = (x / 4) + 1;
                        seatstring += to_string(seatnum);
                }
                else
                        seatstring += to_string(seatnum - 3);
        }
        return seatstring;
}

bool Account::isBookingsEmpty()
{
        if (bookingslist.empty() == true)
                return true;
        else
                return false;
}
```

=== Booking.h File ===

```
// Authors: Royce Nguyen, Jayro Alvarez, Brian Trinh, Danny Pham
// Course: CPSC 362
// Filename: Booking.h
// Purpose: Holds flight number, account number, assigned seat, price paid
// Last updated: 7/31/2018

#ifndef BOOKING_H
#define BOOKING_H

#include <iostream>
#include <string>
using namespace std;

class Booking
{
	public:
		Booking();
		~Booking();
		Booking(string, int, int);

		// Getters
		int getBookingID();
		string getFlightNumber();
		int getAccountNumber();
		int getAssignedSeat();

		// Variables
		int bookingID;
		string flightNumber;
		int accountNumber;
		int assignedSeat;

	private:
		static int id;
};

#endif
```

=== Booking.cpp File ===

```
// Authors: Royce Nguyen, Jayro Alvarez, Brian Trinh, Danny Pham
// Course: CPSC 362
// Filename: Booking.cpp
// Last updated: 7/31/2018

#include "Booking.h"

int Booking::id = 1;

Booking::Booking(){}

Booking::~Booking(){}

Booking::Booking(string flightNum, int accountNum, int seat)
{
	bookingID = id;
	id++;
	flightNumber = flightNum;
	accountNumber = accountNum;
	assignedSeat = seat;
}

int Booking::getBookingID()
{
	return bookingID;
}

string Booking::getFlightNumber()
{
	return flightNumber;
}

int Booking::getAccountNumber()
{
	return accountNumber;
}

int Booking::getAssignedSeat()
{
	return assignedSeat;
}
```

=== Flight.h File ===

```
// Authors: Royce Nguyen, Jayro Alvarez, Brian Trinh, Danny Pham
// Course: CPSC 362
// Filename: Flight.h
// Purpose: Declares Flight class
// Last updated: 7/31/18

#ifndef FLIGHT_H
#define FLIGHT_H

#include <iostream>
#include <string>
using namespace std;

class Flight
{
	public:
		Flight();
		~Flight();
		Flight(string, string, string, int, int, int, int, int, int, int, int, char[], double);

		// Functions
		string getFlightNumber();
		int getDepartureTime();
		int getArrivalTime();
		string getDepartureLocation();
		string getArrivalLocation();
		void print();
		void displaySeats();
		double getprice();
		void addFlight(Flight);
		string getDepartureDate();
		string getArrivalDate();
		int getAvailableSeats();

		// Variables
		string flightNumber;
		string toLocation;
		string fromLocation;
		int departureMonth;
		int departureDay;
		int departureYear;
		int departureTime;
		int arrivalMonth;
		int arrivalDay;
		int arrivalYear;
		int arrivalTime;
		char seats[72];
		double price;
	private:
};

#endif
```

=== Flight.cpp File ===

```
// Authors: Royce Nguyen, Jayro Alvarez, Brian Trinh, Danny Pham
// Course: CPSC 362
// Filename: Flight.cpp
// Last updated: 7/31/18

#include "Flight.h"

Flight::Flight(){}
Flight::~Flight(){}

// Constructor
Flight::Flight(string flightNum, string toLoc, string fromLoc, int depMonth, int depDay, int depYear, int depTime, int arrMonth,
int arrDay, int arrYear, int arrTime, char seating[], double _price)
{
	flightNumber = flightNum;
	toLocation = toLoc;
	fromLocation = fromLoc;
	departureMonth = depMonth;
	departureDay = depDay;
	departureYear = depYear;
	departureTime = depTime;
	arrivalMonth = arrMonth;
	arrivalDay = arrDay;
	arrivalYear = arrYear;
	arrivalTime = arrTime;
	price = _price;
	for (int x = 0; x < 72; x++)
	{
		seats[x] = seating[x];
	}
}

// Getter functions
string Flight::getFlightNumber()
{
	return flightNumber;
}

int Flight::getDepartureTime()
{
	return departureTime;
}

int Flight::getArrivalTime()
{
	return arrivalTime;
}

string Flight::getDepartureLocation()
{
	return fromLocation;
}

string Flight::getArrivalLocation()
{
	return toLocation;
}

void Flight::print()
{
```

```
	cout << flightNumber << "\t\t";
	cout << toLocation << "\t";
	cout << fromLocation << "\t";
	cout << departureMonth << "/" << departureDay << "/" << departureYear << " ";

	if (departureTime < 1000)
	{
		cout << "0" << departureTime << "\t";
	}
	else
	{
		cout << departureTime << "\t";
	}

	cout << arrivalMonth << "/" << arrivalDay << "/" << arrivalYear << " ";
	if (arrivalTime < 1000)
	{
		cout << "0" << arrivalTime << "\t";
	}
	else
	{
		cout << arrivalTime << "\t";
	}

	cout << getAvailableSeats() << endl;
}

void Flight::displaySeats()
{
	cout << "\t\t  RESERVE SEAT(S)" << endl << endl;
	cout << "\t\t    A B\t  C D" << endl;
	cout << "\t\t1   ";
	for (int i = 0; i < 4; i++)
	{
		if (seats[i] == 'X')
		{
			cout << "X ";
			if (i == 1)
			{
				cout << "   ";
			}
		}
		else
		{
			cout << "_ ";
			if (i == 1)
			{
				cout << "   ";
			}
		}
	}

	cout << "\n\t\t2   ";
	for (int i = 4; i < 8; i++)
	{
		if (seats[i] == 'X')
		{
			cout << "X ";
			if (i == 5)
			{
				cout << "   ";
```

```
                }
        }
        else
        {
                cout << "_ ";
                if (i == 5)
                {
                        cout << "  ";
                }
        }
}

cout << "\n\t\t3   ";
for (int i = 8; i < 12; i++)
{
        if (seats[i] == 'X')
        {
                cout << "X ";
                if (i == 9)
                {
                        cout << "  ";
                }
        }
        else
        {
                cout << "_ ";
                if (i == 9)
                {
                        cout << "  ";
                }
        }
}

cout << "\n\t\t4   ";
for (int i = 12; i < 16; i++)
{
        if (seats[i] == 'X')
        {
                cout << "X ";
                if (i == 13)
                {
                        cout << "  ";
                }
        }
        else
        {
                cout << "_ ";
                if (i == 13)
                {
                        cout << "  ";
                }
        }
}

cout << "\n\t\t5   ";
for (int i = 16; i < 20; i++)
{
        if (seats[i] == 'X')
        {
                cout << "X ";
                if (i == 17)
```

```
                {
                        cout << "  ";
                }
        }
        else
        {
                cout << "_ ";
                if (i == 17)
                {
                        cout << "  ";
                }
        }
}

cout << "\n\t\t6   ";
for (int i = 20; i < 24; i++)
{
        if (seats[i] == 'X')
        {
                cout << "X ";
                if (i == 21)
                {
                        cout << "  ";
                }
        }
        else
        {
                cout << "_ ";
                if (i == 21)
                {
                        cout << "  ";
                }
        }
}

cout << "\n\t\t7   ";
for (int i = 24; i < 28; i++)
{
        if (seats[i] == 'X')
        {
                cout << "X ";
                if (i == 25)
                {
                        cout << "  ";
                }
        }
        else
        {
                cout << "_ ";
                if (i == 25)
                {
                        cout << "  ";
                }
        }
}

cout << "\n\t\t8   ";
for (int i = 28; i < 32; i++)
{
        if (seats[i] == 'X')
        {
```

```
                        cout << "X ";
                        if (i == 29)
                        {
                                cout << "  ";
                        }
                }
                else
                {
                        cout << "_ ";
                        if (i == 29)
                        {
                                cout << "  ";
                        }
                }
}

cout << "\n\t\t9   ";
for (int i = 32; i < 36; i++)
{
                if (seats[i] == 'X')
                {
                        cout << "X ";
                        if (i == 33)
                        {
                                cout << "  ";
                        }
                }
                else
                {
                        cout << "_ ";
                        if (i == 33)
                        {
                                cout << "  ";
                        }
                }
}

cout << "\n\t\t10  ";
for (int i = 36; i < 40; i++)
{
                if (seats[i] == 'X')
                {
                        cout << "X ";
                        if (i == 37)
                        {
                                cout << "  ";
                        }
                }
                else
                {
                        cout << "_ ";
                        if (i == 37)
                        {
                                cout << "  ";
                        }
                }
}

cout << "\n\t\t11  ";
for (int i = 40; i < 44; i++)
{
```

```
		if (seats[i] == 'X')
		{
			cout << "X ";
			if (i == 41)
			{
				cout << "  ";
			}
		}
		else
		{
			cout << "_ ";
			if (i == 41)
			{
				cout << "  ";
			}
		}
}

cout << "\n\t\t12  ";
for (int i = 44; i < 48; i++)
{
		if (seats[i] == 'X')
		{
			cout << "X ";
			if (i == 45)
			{
				cout << "  ";
			}
		}
		else
		{
			cout << "_ ";
			if (i == 45)
			{
				cout << "  ";
			}
		}
}

cout << "\n\t\t13  ";
for (int i = 48; i < 52; i++)
{
		if (seats[i] == 'X')
		{
			cout << "X ";
			if (i == 49)
			{
				cout << "  ";
			}
		}
		else
		{
			cout << "_ ";
			if (i == 49)
			{
				cout << "  ";
			}
		}
}

cout << "\n\t\t14  ";
```

```
for (int i = 52; i < 56; i++)
{
	if (seats[i] == 'X')
	{
		cout << "X ";
		if (i == 53)
		{
			cout << "  ";
		}
	}
	else
	{
		cout << "_ ";
		if (i == 53)
		{
			cout << "  ";
		}
	}
}

cout << "\n\t\t15  ";
for (int i = 56; i < 60; i++)
{
	if (seats[i] == 'X')
	{
		cout << "X ";
		if (i == 57)
		{
			cout << "  ";
		}
	}
	else
	{
		cout << "_ ";
		if (i == 57)
		{
			cout << "  ";
		}
	}
}

cout << "\n\t\t16  ";
for (int i = 60; i < 64; i++)
{
	if (seats[i] == 'X')
	{
		cout << "X ";
		if (i == 61)
		{
			cout << "  ";
		}
	}
	else
	{
		cout << "_ ";
		if (i == 61)
		{
			cout << "  ";
		}
	}
}
```

```
	cout << "\n\t\t17  ";
	for (int i = 64; i < 68; i++)
	{
		if (seats[i] == 'X')
		{
			cout << "X ";
			if (i == 65)
			{
				cout << "   ";
			}
		}
		else
		{
			cout << "_ ";
			if (i == 65)
			{
				cout << "   ";
			}
		}
	}

	cout << "\n\t\t18  ";
	for (int i = 68; i < 72; i++)
	{
		if (seats[i] == 'X')
		{
			cout << "X ";
			if (i == 69)
			{
				cout << "   ";
			}
		}
		else
		{
			cout << "_ ";
			if (i == 69)
			{
				cout << "   ";
			}
		}
	}
	cout << endl;
}

double Flight::getprice()
{
	return price;
}

string Flight::getDepartureDate()
{
	string date = "";
	date += to_string(departureMonth);
	date += "/";
	date += to_string(departureDay);
	date += "/";
	date += to_string(departureYear);
	return date;
}
```

```
string Flight::getArrivalDate()
{
	string date = "";
	date += to_string(arrivalMonth);
	date += "/";
	date += to_string(arrivalDay);
	date += "/";
	date += to_string(arrivalYear);
	return date;
}

int Flight::getAvailableSeats()
{
	int availableSeats = 0;
	for (int i = 0; i < 72; i++)
	{
		if (seats[i] == 'O')
		{
			availableSeats++;
		}
	}

	return availableSeats;
}
```