C++ Programming and Software

Airways Management System

Group leader: Haoyuan Tong

Group members:

Student ID	Name	Responsibility	Contribution
2251507	Haoyuan Tong	System Design Coding	25%
2253126	Guo Yu	System Design Coding	25%
2251625	Yukun Zheng	Coding Report	25%
2255283	Boyuan Zheng	Coding Testing	25%

Abstract

This report describes in detail the development process of the aviation management system commissioned by the airline and its functions. The system is designed to streamline company operations, manage aircraft, flight and passenger information, improve operational efficiency and enhance the customer experience

CONTENTS

1. Specification

a) Overall Specifications

The aviation Management System is an integrated computer system that is responsible for most of the company's operations. The system requires the storage of aircraft, flights and passenger information that the company has operated over the years, including two aircraft types, the P62 and the P124, as well as the origin and destination of direct flights. The system is designed to process large amounts of data and ensure the accuracy and security of the data

b) Customer Specifications

The system must be able to store aircraft, flight and passenger information for the duration of the company's operations. System needs to provide:

- ✧ Administrator capabilities to add new flights, prices, dates, airports and new aircraft types
- ✧ A ticket agent with the flexibility to search for specific flights and process customer bookings
- ✧ Manager function that retrieves operational statistics of the company
- ✧ Customers can only book one ticket for themselves at a time
- ✧ Ticket agents are able to count the number of tickets sold during a specific period

c) System Specifications

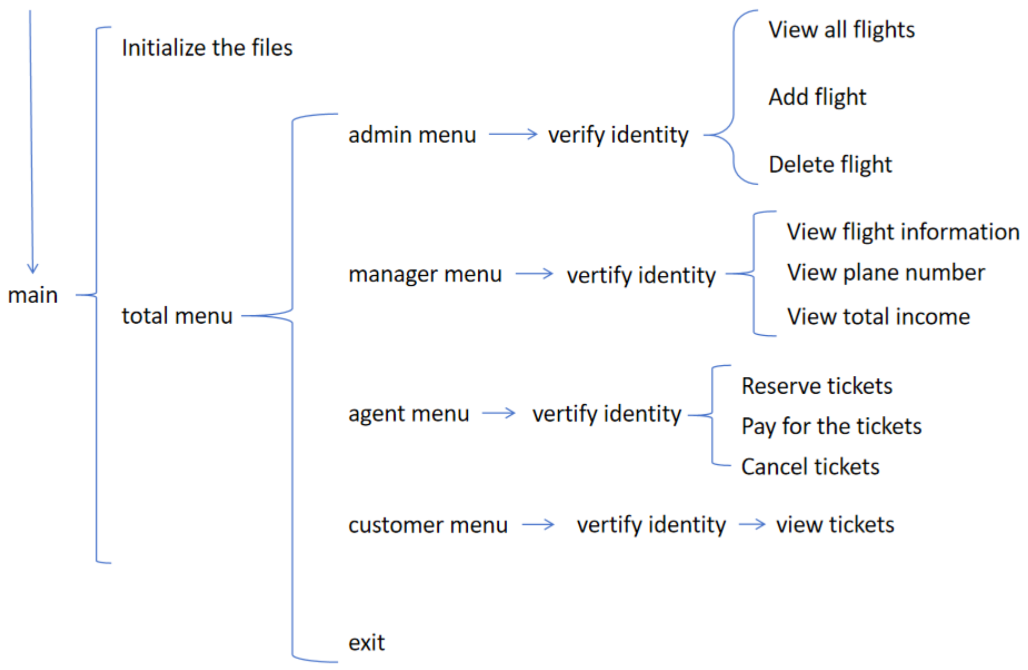
- Flight Management: Including functions such as adding, deleting, and viewing flights.
- User Management: Including functions such as user registration, login, and ticket viewing.
- Agent Management: Agents can book seats, process payments, and cancel bookings for passengers.
- Manager Management: Manager can view number of different flight type, total income, and total flight informations.

2. Analysis and Design

A. System Design

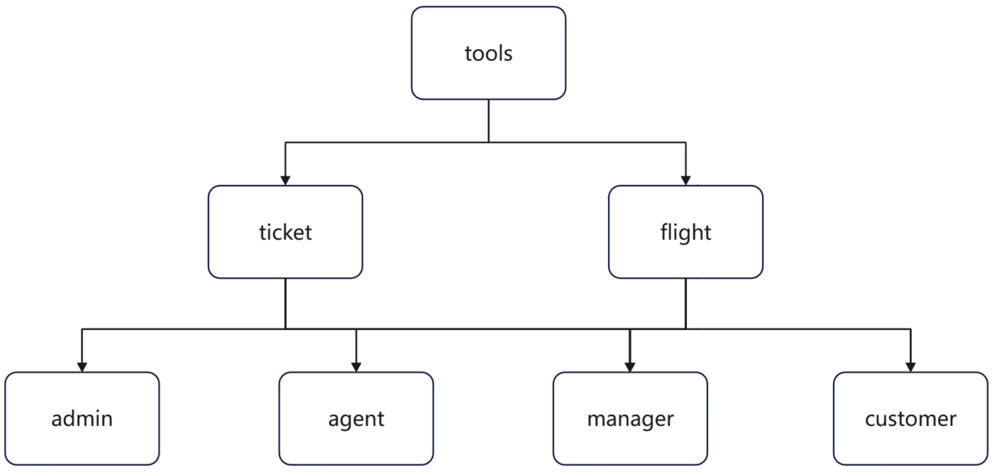
1) Overall Analysis

The system adopts a modular design, where modules interact through interfaces. The system architecture diagram is as follows:



2) Module Design—Requirement Analysis

To complete the system design, we need to implement five main modules: user management, agent management, administrator management, flight management, and file I/O operations. For the relationship between these five modules, we first need to establish two basic structures, ticket and flight, and then create different classes for each program user (user, administrator, agent, manager) and place them in different header files. Finally, integrate and call them in the main function to achieve the goals. The Following figure shows the basic relationship between all head files.



3) Basic Structures

1. Ticket Structure

This can be considered the most fundamental part of the entire program since almost all information originates from this structure. Each consumer can buy many tickets, and each flight can have many consumers. Therefore, we chose this one-to-many basic unit as the system's underlying unit for counting all consumption data. A ticket contains the following information:

- Order Number: Automatically generated, unique number. int ticket_id
- Corresponding Customer: One ticket corresponds to one customer string customerName
- Corresponding Flight: One ticket corresponds to one flight string flight_id
- Corresponding Seat: The format is “12A,” “23B,” where the first is the row and the second is the column, stored only as display information. To find the location, use the order number and corresponding flight number to search in the flight linked list. String seat_id
- Corresponding Price: Store the price of this ticket, which can be unpaid
- Depart time and Arrive time
- Payment Status: Reserved or paid
- Cancellation Status: Whether it has been canceled
- Linked List Next Node

2. Flight Structure

It includes all the information of each aircraft, linked together in the form of a linked list. When searching for a corresponding flight, just query the flight number (unique). Each flight contains the following information:

- Flight Number string flight_id
- Departure Location string depart
- Arrival Location string arrive
- Departure Time (Year, Month, Day) (Hour, Minute) int departDate [5]
- Arrival Time (Year, Month, Day) (Hour, Minute) int arriveDate[5]
- Ticket Price double price
- Aircraft Model: Use an array to store the length and width and print it in the format “P length width.” int type[2]
- Total Seats: The total number of seats on this flight
- Remaining Seats: Each time a user books a ticket, the remaining number decreases by 1. When the remaining number reaches zero, the flight is considered full
- Seat Layout: According to the “length, width” of the aircraft model, fill in the 100x12 matrix. If occupied, fill in the order number; 0 means available, -1 means non-existent. int seat[100][12]
- Cancellation Status bool ifCancel
- Linked List Next Node

These two structures form the system's underlying framework, and there is a close relationship between them. To make sure the system works well, we need to consider several things:

First, how to display time in a standard format. We hope the time format will be printed on the screen as “Year-Month-Day” + “Hour: Minute,” but if you want to keep each column aligned at the same time, it will be a challenge. Therefore, through online searches and AI assistance, we found some functions in the <iomanip> and <sstream> libraries that can meet the needs, which will also be analyzed in detail later.

Second, we need to ensure that the departure time is earlier than the arrival time. This can be judged by comparing each item. Additionally, printing the aircraft model also requires using the method of “merging strings,” such as printing the “12” and “6” in an array as “P126,” which will also be introduced later.

Finally, we do not intend to actually delete the canceled flights. To retain the historical record, we added the “ifCancel” variable, which means that when agents book tickets, these marked flights will not be displayed.

4) Basic Class

a) Admin Class

In the aviation information management system, administrators have extensive permissions, including reading, changing, adding, and deleting flights. The admin class defines functions for these tasks:

- **Set Administrator Account**: During system initialization, it checks for an existing admin file. If none exists, it prompts the admin to set an ID and password, ensuring an admin account is established.
- **Login Verification**: Reads the admin ID and password from a file and verifies the credentials during login to ensure only authorized access.
- **Add Flight**: Creates a new flight object with details like flight number, locations, times, price, model, and seats. Adds this flight to a linked list, ensuring the flight number is unique.
- **Delete Flight**: Marks a flight as canceled based on its flight number, retaining the historical record instead of deleting it physically.
- **Flight Viewing**: Displays all flight information, including canceled flights, allowing comprehensive management.
- **Message Notification**: Sends notifications to passengers about flight changes or cancellations, ensuring timely updates.

The admin class works closely with the flight structure, which stores flight information. Key functions include:

- **Add Flight**: Uses **newFlight()** to create a flight and **addFlight()** to add it to the flight linked list.
- **Delete Flight**: Uses **deleteFlight()** to mark flights as canceled.
- **View Flights**: Uses **showFlight()** to display all flight information.

These functions ensure the system efficiently and reliably manages flight information, forming the core data processing module of the aviation information management system.

b) Customer Class

In the aviation information management system, the **Customer** class manages user-related functions, including registration, login, and ticket management. The main functions of the **Customer** class are:

- **Set User Account**:
  - Checks and initializes user files. If no user file exists at system startup, it creates an empty file to record user information.
- **User Management**:
  - **Check Username**: Ensures new usernames are unique to prevent duplicate registrations.
  - **Username Existence**: Checks if a username exists to facilitate login and ticket booking.
  - **Login Verification**: Verifies username and password during login to ensure only authorized users can access the system.
- **User Linked List Management**:
  - **Add User**: Adds new users to the linked list and updates the system's user information.
  - **Write User Data**: Writes user information from the linked list to the file, ensuring data persistence.
  - **Read User Data**: Reads user information from the file to build the

linked list, allowing the system to load existing data at startup.

- **Ticket Management**:
    - **Read Tickets**: Loads user ticket information into the user's ticket linked list.
    - **Display Tickets**: Shows all ticket information, including order number, flight number, seat, price, payment status, and cancellation status, for user convenience.

The **Customer** class works closely with the ticket structure, which stores ticket information. Key functions include:

- **User Ticket Management**: Manages user tickets, utilizing the **readCustomerTicket()** function to load user-specific ticket data into the linked list.
- **Ticket Display**: Uses the **showCustomerTicket()** function to display all user tickets, providing comprehensive ticket status information.

These functions ensure users can effectively manage and view their tickets, forming the user data management part of the system. This integration with the ticket structure allows the system to run efficiently and reliably, meeting user needs while ensuring data integrity and consistency.

c) Manager Class

In the aviation information management system, the **Manager** class handles system statistics and reporting. It provides managers with key information such as aircraft counts, passenger numbers, and total revenue. The main functions of the **Manager** class are:

- **Aircraft and Flight Statistics**:
    - **Display Aircraft Count**: Counts the number of operational aircraft to understand the company's resources.
    - **Display Passenger Count**: Queries the number of passengers on specific flights to understand flight load and optimize resource allocation.

- **Data Management**:
    - **Set Aircraft Information**: Sets and updates the number of different aircraft types for effective resource management.
    - **Set Flight Information**: Records and updates passenger numbers on flights to understand operational status.
    - **Manage Revenue**: Obtains and updates total revenue data, aiding financial management and decision-making.

The **Manager** class works closely with the flight structure, which stores flight information. Key functions include:

- **Flight Statistics**: Obtains flight statistics by traversing the flight linked list. For example, counts the number of flights when displaying the number of aircraft.
- **Passenger Statistics**: Queries passenger numbers by finding specific flights in the linked list and counting booked seats.
- **Data Updating**: Records and updates passenger numbers based on seat information in the flight structure, ensuring data accuracy and timeliness.

These functions ensure managers can effectively obtain and manage key system data, forming the statistical and reporting part of the aviation information management system. This integration with the flight structure allows the system to run efficiently and reliably, meeting managers' needs while ensuring data integrity and consistency.

d) Agent Class

In the aviation information management system, the **Agent** class manages agent functions such as booking flight seats, processing passenger information, and ticket management. Key functions of the **Agent** class include:

- **Set Agent Account**:
    - **System Initialization**: Checks and initializes agent files. If no file exists at startup, prompts the agent to enter an ID and password to

create the agent file.

    - **File Reading**: Reads the agent ID and password from the file during system startup to load login information.
    - **Login Verification**: Verifies the ID and password entered by the agent to ensure legitimate access.

- **Seat Booking and Ticket Management**:
    - **Book Seats**: Agents can book specific flight seats according to passengers' needs. The system checks seat availability and updates the seat status and remaining seat count.
    - **Generate and Manage Tickets**: After booking a seat, the system generates new ticket information and adds it to the ticket linked list. Ticket information includes order number, customer name, flight ID, seat number, ticket price, payment status, and cancellation status.

Relationship with Flight Structure

The flight structure stores and manages flight information, including flight number, locations, times, ticket price, aircraft model, and seat layout.

- **Seat Booking**: When booking seats, the **Agent** class traverses the flight linked list to find the corresponding flight and checks seat availability. If the seat is available, it updates the seat status and remaining seat count. The **reserveSeat(flight*& specificFlight, ticket*& ticketHead, std::string seat, std::string name)** function achieves this.
- **Flight Information Synchronization**: After booking seats, the system updates the flight structure to ensure consistency and timeliness of seat booking information.

Relationship with Ticket Structure

The ticket structure stores and manages ticket information, including order number, customer information, flight ID, seat, price, payment status, and cancellation status.

- **Ticket Generation and Management**: After booking seats, the **Agent** class generates new ticket information and adds it to the ticket linked list.

Each ticket gets a unique order number, and records the passenger's name, flight ID, and seat number.

- **Data Persistence**: Ensures data persistence and security by writing ticket information to the file. The **reserveSeat** function handles ticket generation and addition to the linked list.

These functions ensure agents can effectively manage and book flight seats, forming a key part of the aviation information management system with the flight and ticket structures. This integration allows the system to run efficiently and reliably, meeting the needs of agents and passengers while ensuring data integrity and consistency.

5) Other design

The tools.h header file contains basic headers, input functions, and utility functions, making the code more concise and maintainable. It provides the following utilities:

- **Input Functions**:
    - **Number Input**: **double inputNumFun(int limSize, int maxSize)**: Encapsulates number input operations, ensuring the input is within a specified range and valid. Prompts for re-entry if out of range.
    - **Text Input**: **string inputStrFun()**: Ensures the input string does not contain spaces. Prompts for re-entry if spaces are present.
- Date Input Functions:
    - **Date Input: void inputDate(int& year, int& month, int& day, int& hour, int& minute):** Encapsulates date and time input operations, verifying the input is within a reasonable range.
- Date Conversion Functions:
    - **Aircraft Model Conversion**: **string convertType(int num[2])**: Converts the aircraft model array to string representation, such as "P62" or "P124".

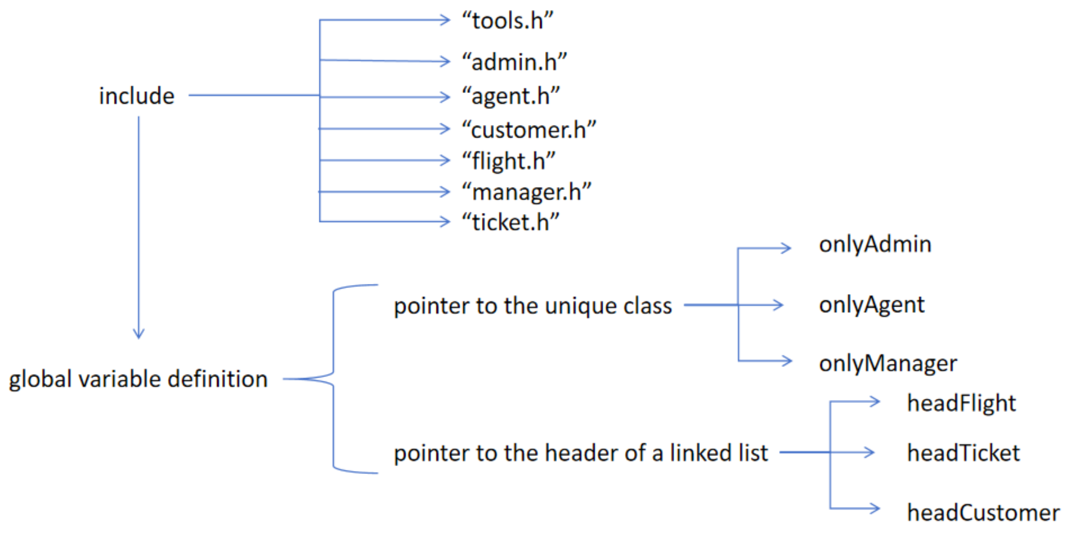
    - **Year-Month-Day Conversion**: **string convertYear(int num[5])**: Converts the year, month, and day array into string format, "YYYY-MM-DD".
    - **Hour-Minute Conversion**: **string convertHour(int num[5])**: Converts the time array into string format, "HH".
- Utility Functions:
    - **Return to Previous Menu: void backMenu():** Clears the screen and displays a prompt to return to the previous menu.

These utility functions simplify writing and maintaining code, making it more modular, readable, and easier to debug. They are widely used throughout the system, improving code reusability and consistency, ensuring the system's reliability and stability.

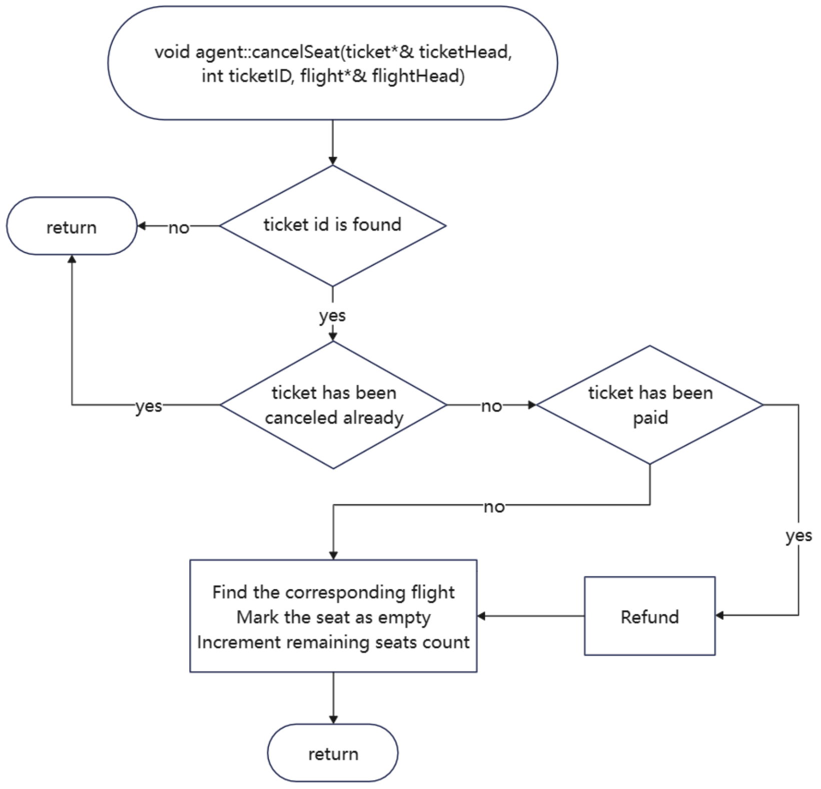
B. Program Design

After analyzing the demand and basic functionalities, here displays our core functions in our airway manage system.

1) Overall Program Design



2) Agent cancel tickets function


void agent::cancelSeat(ticket*& ticketHead, int ticketID, flight*& flightHead)
ticket id is found
no
return
yes
ticket has been canceled already
yes
no
ticket has been paid
no
yes
Find the corresponding flight
Mark the seat as empty
Increment remaining seats count
Refund
return

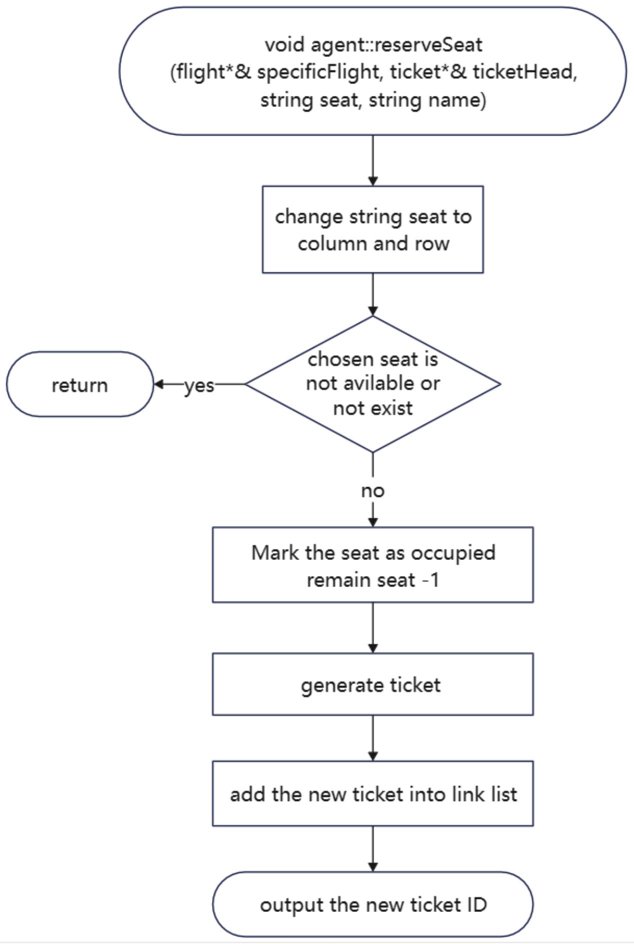
The function cancelSeat is called with parameters ticketHead, ticketID, and flightHead.

1. The system checks if the ticket ID is found by a while loop:

- If not found, the function returns back.
- If found, it proceeds to the next step.

2. The system checks if the ticket has already been canceled:
   - If yes, the function returns back.
   - If no, it proceeds to the next step.
3. The system checks if the ticket has been paid:
   - If not paid, the function finds the corresponding flight, marks the seat as empty, increments the remaining seats count, and then returns.
   - If paid, the system processes a refund, finds the corresponding flight, marks the seat as empty, increments the remaining seats count, and then returns.

3) Agent reserve tickets function


void agent::reserveSeat (flight*& specificFlight, ticket*& ticketHead, string seat, string name)
change string seat to column and row
chosen seat is not avilable or not exist
yes
return
no
Mark the seat as occupied remain seat -1
generate ticket
add the new ticket into link list
output the new ticket ID

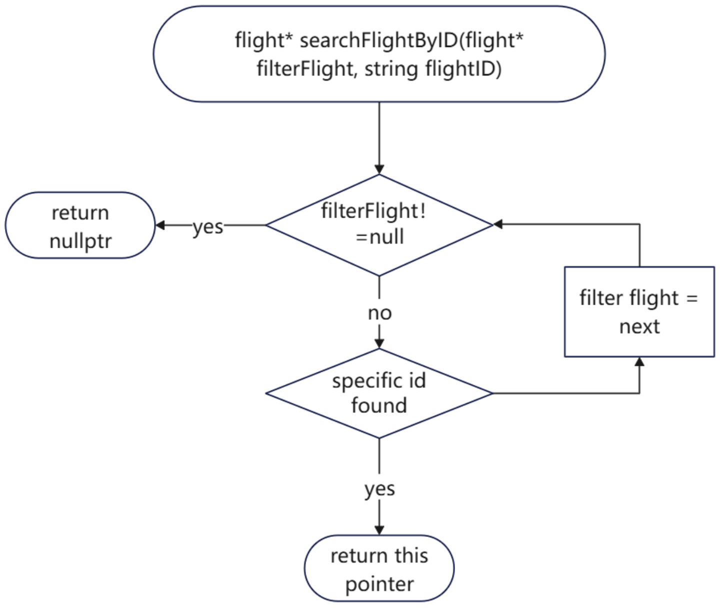
1. The function reserveSeat is called with parameters specificFlight, ticketHead, seat, and name.
2. The system converts the string seat into int column and int row format.

3. The system checks if the chosen seat is available or exists:

- If the seat is not available or does not exist, the function returns.
- If the seat is available, it proceeds to the next step.

4. The system marks the seat as occupied and decrease the remaining seat count by 1.
5. The system generates a ticket with unique id.
6. The system adds the new ticket into the ticket linked list.
7. The system shows the new ticket ID to the agent.

4) Find specific flight function



1. The function searchFlightByID is called with the parameters filterFlight (a pointer to the flight linked list) and flightID (a string representing the flight ID).
2. The system checks whether filterFlight is null:

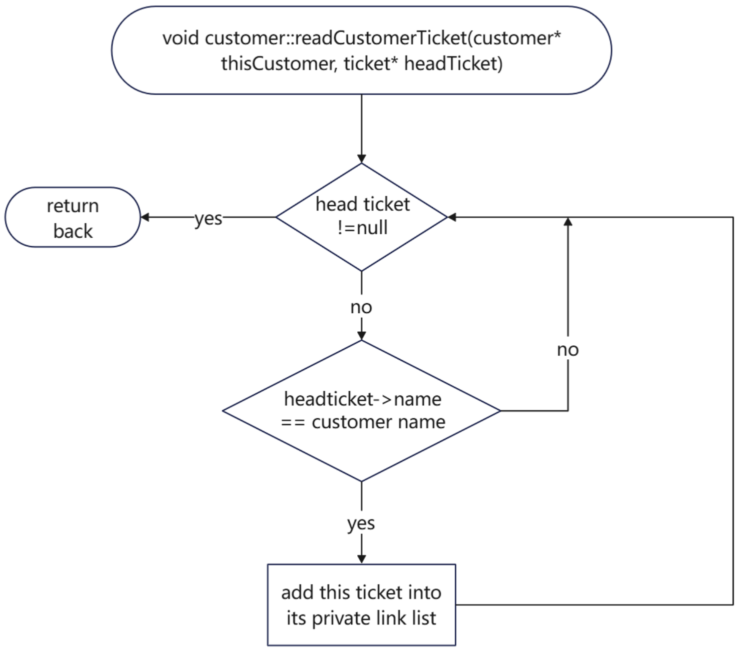
   • If filterFlight is null, the function returns nullptr.

   • If filterFlight is not null, proceed to the next step.
3. The system checks whether the flight ID of the current filterFlight matches flightID:

   • If a matching flight ID is found, the function returns the pointer to that flight.

   • If no matching flight ID is found, filterFlight is moved to the next flight node, and step 2 is repeated.

5) Customer read ticket function


void customer::readCustomerTicket(customer* thisCustomer, ticket* headTicket)
head ticket !=null
yes
return back
no
headticket->name == customer name
no
yes
add this ticket into its private link list

1. The function agent::paySeat is called with the parameters ticketHead (a reference to the ticket linked list pointer) and ticketID (the ticket ID).
2. The system checks whether ticketHead is null:

   • If ticketHead is null, the function returns, indicating there are no tickets to process.

   • If ticketHead is not null, proceed to the next step.
3. The system searches for the corresponding ticketID in the ticketHead linked list:

   • If the corresponding ticket is found, proceed to the next step.

   • If the corresponding ticket is not found, the function returns.
4. The system checks whether the found ticket is unpaid:

   • If the ticket is already paid, the function returns.

   • If the ticket is unpaid, proceed to the next step.
5. The system checks whether the ticket is not canceled:

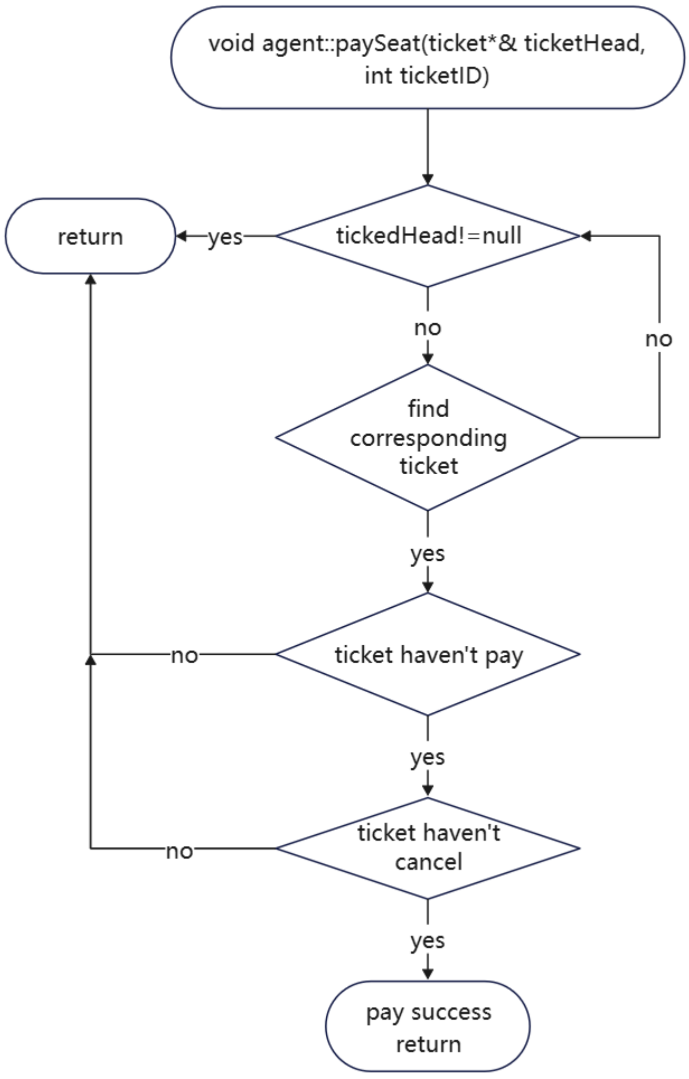
   • If the ticket is canceled, the function returns.

   • If the ticket is not canceled, proceed to the next step.
6. The system performs the payment operation and updates the payment status to

successful.

7. The function returns, indicating the payment was successful.

6) Agent purchase ticket function



1. The function agent::paySeat is called with the parameters ticketHead (a reference to the ticket linked list pointer) and ticketID (the ticket ID).
2. The system checks whether ticketHead is null: • If ticketHead is null, the function returns, indicating there are no tickets to process. • If ticketHead is not null, proceed to the next step.
3. The system searches for the corresponding ticketID in the ticketHead linked list: • If the corresponding ticket is not found, the function returns.

• If the corresponding ticket is found, proceed to the next step.

4. The system checks whether the found ticket is unpaid:

   • If the ticket is already paid, the function returns.

   • If the ticket is unpaid, proceed to the next step.

5. The system checks whether the ticket is not canceled:

   • If the ticket is canceled, the function returns.

   • If the ticket is not canceled, proceed to the next step.

6. The system performs the payment operation and updates the payment status to successful.

7. The function returns, indicating the payment was successful.

3. Testing

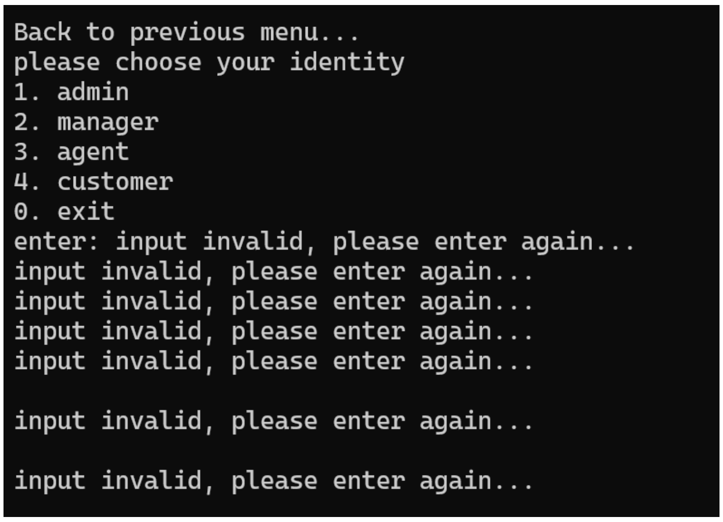
Roles	Aim	Input	Expected	Got
	Initialize admin,	1. No space	1. success	1. Correct

Admin	manager and agent id and password	2. have space	2. please enter a valid number	2. Correct
	Log-in	1. Correct password 2. Wrong password 3. with space	1. Pass 2. password wrong 3. please enter a valid input	1. Correct 2. Correct 3. Correct
	Check information	---	---	Information shown
	Manipulate flights For Add/ Delete flights	1. flight number with character 2. Wrong seat length/ width 3. departure city with space 4. invalid date	1. Pass 2. invalid, input again 3. contain space, enter again 4. invalid, input again	1. Correct 2. Correct 3. Correct 4. Correct
Manager	View total income	---	---	Information shown
Agent	Reserve seat And view booking information	1. valid cities	1. Valid information including flight ID, seat booking state, seat price and so on 2. Printed cabin seating picture 3. Unique Ticket ID if booked successfully	1. Correct 2. Correct 3. Correct
	Pay for customer	1. Invalid name of customer 2. Invalid flight number	1. customer does not exist 2. please enter a valid flight number	1. Correct 2. Correct
	Cancel booking for customer	1. Valid customer name 2. Invalid Ticket ID	1. Pass 2. please enter a valid ID	1. Correct 2. Correct
Customer	Register	1. Combinations of numbers and characters (no space) 2. with space	1. valid 2. no space allowed	1. Correct 2. Correct
	Log-in	1. Name of changing the capital Letter but valid	1. Wrong ID of password	1. Correct

		password		
	View information	1. “1” 2. ”0” 3. “asdfgh”	1. Ticket information display 2. Successfully back to the customer menu 3. system crash	1. Correct 2. Correct 3.wrong

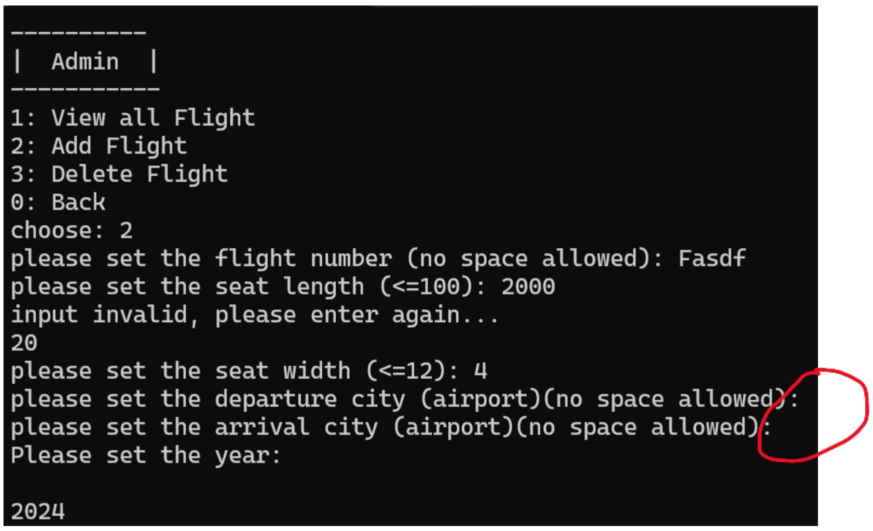
4. Bugs report

- Please do not enter a non-digit when entering your choose in any menu, that will cause the program crash.
  Solution: restart the program, data will not lose.

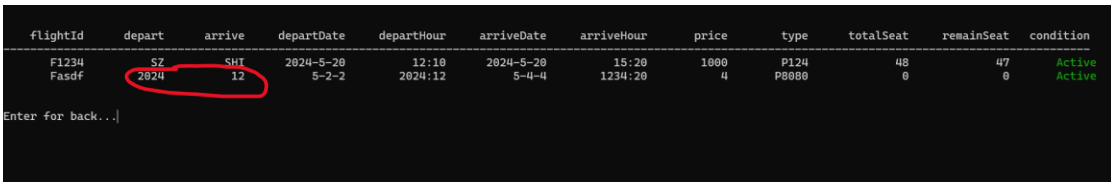


```
Back to previous menu...
please choose your identity
1. admin
2. manager
3. agent
4. customer
0. exit
enter: input invalid, please enter again...
input invalid, please enter again...
input invalid, please enter again...
input invalid, please enter again...
input invalid, please enter again...

input invalid, please enter again...

input invalid, please enter again...
```

- Please do not enter nothing when asked to input a string, this will cause bug in future using.
  Solution: delete the wrong data in txt file and try to input again

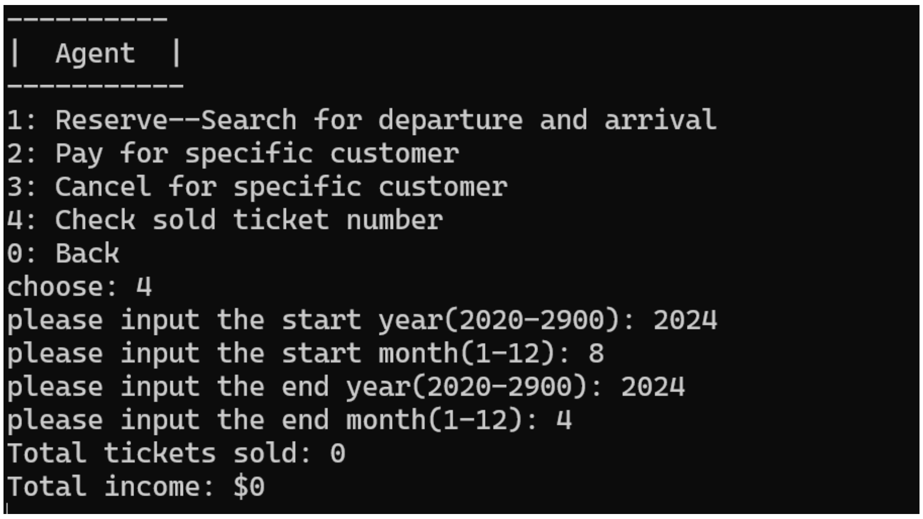


```
----------
|  Admin  |
-----------
1: View all Flight
2: Add Flight
3: Delete Flight
0: Back
choose: 2
please set the flight number (no space allowed): Fasdf
please set the seat length (<=100): 2000
input invalid, please enter again...
20
please set the seat width (<=12): 4
please set the departure city (airport)(no space allowed):
please set the arrival city (airport)(no space allowed):
Please set the year:

2024
```



```
   flightId      depart      arrive    departDate     departHour     arriveDate     arriveHour       price        type      totalSeat      remainSeat   condition
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
      F1234          SZ         SHI    2024-5-20          12:10     2024-5-20          15:20        1000        P124             48              47      Active
      Fasdf        2024          12        5-2-2        2024:12         5-4-4        1234:20           4       P8080              0               0      Active

Enter for back...
```

- **For agent identity, please do not input a larger time first when view the income of a time interval.**
  Solution: try again, this will not cause the system crash



```
----------
|  Agent  |
-----------
1: Reserve--Search for departure and arrival
2: Pay for specific customer
3: Cancel for specific customer
4: Check sold ticket number
0: Back
choose: 4
please input the start year(2020-2900): 2024
please input the start month(1-12): 8
please input the end year(2020-2900): 2024
please input the end month(1-12): 4
Total tickets sold: 0
Total income: $0
```

- More bugs are being tested…

5. User Manual

1. System Introduction

The Airline Information Management System aims to streamline and optimize the operational management of airlines, including flight information management, ticketing management, and user management. The system has four user roles: administrator, agent, manager and passenger, each with different permissions and functions.

2. System Requirements

- Operating System: Windows 11 and above
- Compiler: C++ supporting compiler (e.g., Visual Studio)

3. Installation and Setup

- Download and extract the system source package.
- Use a C++ supporting compiler to open the project file in the source folder.
- Compile and run the project file.
- The system will automatically initialize necessary files such as user files and flight files.

4. System Functions for different identities

a) Administrator Functions

Login

- Select administrator identity after starting the system.
- Enter administrator ID and password to log in (id and password can't be change).

View All Flights

- Select "View all Flight" after logging in.
- The system will display detailed information of all flights including canceled flights.

Add Flight

- Select "Add Flight" after logging in.
- Follow the prompts to enter flight number, departure city, arrival city, departure time, arrival time, fare, seat type, etc.
- The system will automatically generate and save the new flight.

Delete Flight

- Select "Delete Flight" after logging in.
- Enter the flight id to be deleted, and the system will mark the flight as canceled, which means the history will be save.

b) Agent Functions

Login

- Select agent identity after starting the system.

- Enter agent ID and password to log in.

Reserve Ticket

- Select "Reserve--Search for departure and arrival" after logging in.
- Enter departure city and arrival city, and the system will display matching flights.
- Select a flight and enter the seat number and passenger name. The system will reserve a seat for the passenger and generate a ticket.

Cancel Ticket

- Select "Cancel for specific customer" after logging in.
- Enter the ticket ID, and the system will find and mark the ticket as canceled.
- If the ticket has been paid, the system will display the refund amount.

Pay for Ticket

- Select "Pay for specific customer" after logging in.
- Enter the ticket ID, and the system will find and mark the ticket as paid.
- The system will check if the passenger has overlapping flight tickets. If so, it will prompt an error.

Income Statistics

- Select "Check Income" after logging in.
- Enter the start and end time, and the system will count and display the number of paid and non-canceled tickets within that time range and the total income.

c) Manager Functions

Login

- Select manager identity after starting the system.
- Enter manager ID and password to log in.

View flight information

- Print out all the available flights
- Find the number of specific flight type using length and width

View Total Income

- Check the total income of this system

d) Customer Functions

Registration

- Select passenger identity after starting the system.
- Select "Registration" and follow the prompts to enter the name and password to register, and the system will automatically make sure the name is unique.

Login

- Select passenger identity after starting the system.
- Enter the Customer name and password to log in.

View Tickets

- Select "View Tickets" after logging in.
- The system will display this Customer's all tickets, including ticket ID,

flight number, seat number, price, time, payment status, and ticket status.

5. FAQs and Solutions

Q1: Unable to log in to the system?
Please check if the entered ID and password are correct, and ensure the system has been initialized.
Q2: Unable to add a flight?
Please ensure the entered flight number is not duplicated, and all information is complete.
Q3: Unable to reserve a seat?
Please check if the selected seat is already occupied and ensure the entered passenger name exists in the system.

6. Contact Us

If you have any questions or suggestions, please contact the technical support team:
Email: support@airsystem.com
Phone: +86 123 4567 890

✧ Note: code will not be print out since it is more than 2000 lines, please check our code in the .cpp and .h files, thanks for understanding.

admin.h agent.h customer.h flight.h manager.h

test.cpp ticket.h tools.h