AIM: Design and implement an Airport Flight Management System to manage and track flight information, including schedules, bookings, and passenger details. The system should efficiently handle operations such as adding new flights, updating flight information, deleting flights, and querying flight details based on various criteria. Consider the relevant passenger and flight record attributes. Also, generate reports for flights by status (e.g., all scheduled flights, all cancelled flights). Use appropriate data structures to manage flight and passenger records.

```
#include <bits/stdc++.h>

using namespace std;


class Passenger {

public:

    string id;

    string name;

    string email;

    Passenger(string id, string name, string email)

        : id(id), name(name), email(email) {}

};

class Flight {

public:

    string flightNumber;

    string origin;

    string destination;

    string departureDate;

    string departureTime;

    string arrivalDate;

    string arrivalTime;
```

```
    string status; // scheduled, cancelled, completed

    vector<Passenger> passengers;

    // Default constructor

    Flight() : flightNumber(""), origin(""), destination(""), departureDate(""),
departureTime(""), arrivalDate(""), arrivalTime(""), status("scheduled") {}



    // Parameterized constructor

    Flight(string flightNumber, string origin, string destination,

          string departureDate, string departureTime, string arrivalDate, string arrivalTime,
string status)

        : flightNumber(flightNumber), origin(origin), destination(destination),

          departureDate(departureDate), departureTime(departureTime),
arrivalDate(arrivalDate), arrivalTime(arrivalTime), status(status) {}



    void addPassenger(const Passenger& passenger) {

        passengers.push_back(passenger);

    }



    void removePassenger(const string& passengerId) {

        passengers.erase(remove_if(passengers.begin(), passengers.end(),

            [&](Passenger& p) { return p.id == passengerId; }), passengers.end());

    }



    void generatePassengerReport() const {

        cout << "Passengers for flight " << flightNumber << ":\n";

        if (passengers.empty()) {

            cout << "No passengers.\n";
```

```
        } else {
            for (const Passenger& passenger : passengers) {
                cout << "ID: " << passenger.id << ", Name: " << passenger.name << ", Email: " << passenger.email << "\n";
            }
        }
    }
};

class AirportManagementSystem {
private:
    unordered_map<string, Flight> flights;

public:
    void addFlight(const Flight& flight) {
        if (flights.find(flight.flightNumber) != flights.end()) {
            cout << "Flight with number " << flight.flightNumber << " already exists.\n";
            return;
        }
        flights[flight.flightNumber] = flight;
        cout << "Added flight " << flight.flightNumber << "\n";
    }

    void updateFlight(const string& flightNumber, const Flight& updatedFlight) {
        auto it = flights.find(flightNumber);
        if (it != flights.end()) {
```

```
        it->second = updatedFlight;
        cout << "Updated flight " << flightNumber << "\n";
    } else {
        cout << "Flight with number " << flightNumber << " does not exist.\n";
    }
}

void deleteFlight(const string& flightNumber) {
    if (flights.erase(flightNumber) == 0) {
        cout << "Flight with number " << flightNumber << " does not exist.\n";
    } else {
        cout << "Deleted flight " << flightNumber << "\n";
    }
}

Flight* getFlight(const string& flightNumber) {
    auto it = flights.find(flightNumber);
    if (it != flights.end()) {
        return &it->second;
    } else {
        cout << "Flight with number " << flightNumber << " not found.\n";
        return nullptr;
    }
}
```

```
    vector<Flight> queryFlightsByStatus(const string& status) {
        vector<Flight> result;
        for (const auto& pair : flights) {
            if (pair.second.status == status) {
                result.push_back(pair.second);
            }
        }
        return result;
    }

    vector<Flight> queryFlightsByDate(const string& date, bool isDeparture) {
        vector<Flight> result;
        for (const auto& pair : flights) {
            if ((isDeparture && pair.second.departureDate == date) || (!isDeparture &&
pair.second.arrivalDate == date)) {
                result.push_back(pair.second);
            }
        }
        return result;
    }

    void generateReportByStatus(const string& status) {
        vector<Flight> flightsByStatus = queryFlightsByStatus(status);
        cout << "Flights with status " << status << ":\n";
        for (const Flight& flight : flightsByStatus) {
            cout << "Flight Number: " << flight.flightNumber
```

```
            << ", Origin: " << flight.origin
            << ", Destination: " << flight.destination
            << ", Departure Date: " << flight.departureDate
            << ", Departure Time: " << flight.departureTime
            << ", Arrival Date: " << flight.arrivalDate
            << ", Arrival Time: " << flight.arrivalTime
            << "\n";
    }
}

void generateReportByDate(const string& date, bool isDeparture) {
    vector<Flight> flightsByDate = queryFlightsByDate(date, isDeparture);
    string dateType = isDeparture ? "Departure" : "Arrival";
    cout << "Flights with " << dateType << " Date " << date << ":\n";
    for (const Flight& flight : flightsByDate) {
        cout << "Flight Number: " << flight.flightNumber
            << ", Origin: " << flight.origin
            << ", Destination: " << flight.destination
            << ", Departure Date: " << flight.departureDate
            << ", Departure Time: " << flight.departureTime
            << ", Arrival Date: " << flight.arrivalDate
            << ", Arrival Time: " << flight.arrivalTime
            << ", Status: " << flight.status
            << "\n";
    }
```

```
    }

    void generatePassengerReport(const string& flightNumber) {
        Flight* flight = getFlight(flightNumber);
        if (flight != nullptr) {
            flight->generatePassengerReport();
        }
    }
};

int main() {
    AirportManagementSystem ams;
    int choice;

    while (true) {
        cout << "\nAirport Management System\n";
        cout << "1. Add Flight\n";
        cout << "2. Update Flight\n";
        cout << "3. Delete Flight\n";
        cout << "4. Add Passenger to Flight\n";
        cout << "5. Generate Report by Status\n";
        cout << "6. Generate Passenger Report\n";
        cout << "7. Generate Report by Departure Date\n";
        cout << "8. Generate Report by Arrival Date\n";
        cout << "9. Exit\n";
```

```
    cout << "Enter your choice: ";
    cin >> choice;
    cin.ignore(); // To ignore the newline character after the integer input

    if (choice == 1) {
        string flightNumber, origin, destination, departureDate, departureTime, arrivalDate, arrivalTime, status;
        cout << "Enter flight number: ";
        getline(cin, flightNumber);
        cout << "Enter origin: ";
        getline(cin, origin);
        cout << "Enter destination: ";
        getline(cin, destination);
        cout << "Enter departure date (YYYY-MM-DD): ";
        getline(cin, departureDate);
        cout << "Enter departure time (HH:MM): ";
        getline(cin, departureTime);
        cout << "Enter arrival date (YYYY-MM-DD): ";
        getline(cin, arrivalDate);
        cout << "Enter arrival time (HH:MM): ";
        getline(cin, arrivalTime);
        cout << "Enter status (scheduled, cancelled, completed): ";
        getline(cin, status);

        Flight flight(flightNumber, origin, destination, departureDate, departureTime, arrivalDate, arrivalTime, status);
```

```
        ams.addFlight(flight);

    } else if (choice == 2) {

        string flightNumber, origin, destination, departureDate, departureTime, arrivalDate,
arrivalTime, status;

        cout << "Enter flight number to update: ";

        getline(cin, flightNumber);

        Flight* flight = ams.getFlight(flightNumber);

        if (flight == nullptr) continue;


        cout << "Enter new origin (current: " << flight->origin << "): ";

        getline(cin, origin);

        cout << "Enter new destination (current: " << flight->destination << "): ";

        getline(cin, destination);

        cout << "Enter new departure date (current: " << flight->departureDate << "): ";

        getline(cin, departureDate);

        cout << "Enter new departure time (current: " << flight->departureTime << "): ";

        getline(cin, departureTime);

        cout << "Enter new arrival date (current: " << flight->arrivalDate << "): ";

        getline(cin, arrivalDate);

        cout << "Enter new arrival time (current: " << flight->arrivalTime << "): ";

        getline(cin, arrivalTime);

        cout << "Enter new status (current: " << flight->status << "): ";

        getline(cin, status);


        Flight updatedFlight(flightNumber, origin, destination, departureDate, departureTime,
arrivalDate, arrivalTime, status);
```

```
        ams.updateFlight(flightNumber, updatedFlight);
    } else if (choice == 3) {
        string flightNumber;
        cout << "Enter flight number to delete: ";
        getline(cin, flightNumber);
        ams.deleteFlight(flightNumber);
    } else if (choice == 4) {
        string flightNumber, passengerId, passengerName, passengerEmail;
        cout << "Enter flight number: ";
        getline(cin, flightNumber);
        Flight* flight = ams.getFlight(flightNumber);
        if (flight == nullptr) continue;

        cout << "Enter passenger ID: ";
        getline(cin, passengerId);
        cout << "Enter passenger name: ";
        getline(cin, passengerName);
        cout << "Enter passenger email: ";
        getline(cin, passengerEmail);

        Passenger passenger(passengerId, passengerName, passengerEmail);
        flight->addPassenger(passenger);
    } else if (choice == 5) {
        string status;
        cout << "Enter status (scheduled, cancelled, completed): ";
```

```
            getline(cin, status);
            ams.generateReportByStatus(status);
        } else if (choice == 6) {
            string flightNumber;
            cout << "Enter flight number: ";
            getline(cin, flightNumber);
            ams.generatePassengerReport(flightNumber);
        } else if (choice == 7) {
            string departureDate;
            cout << "Enter departure date (YYYY-MM-DD): ";
            getline(cin, departureDate);
            ams.generateReportByDate(departureDate, true);
        } else if (choice == 8) {
            string arrivalDate;
            cout << "Enter arrival date (YYYY-MM-DD): ";
            getline(cin, arrivalDate);
            ams.generateReportByDate(arrivalDate, false);
        } else if (choice == 9) {
            break;
        } else {
            cout << "Invalid choice. Please try again.\n";
        }
    }
    return 0;
}
```

Airport Flight Management System Report

Objective

The objective of this project is to design and implement an Airport Flight Management System using C++. This system will manage and track flight information, including schedules, bookings, and passenger details. It will handle operations such as adding new flights, updating flight information, deleting flights, and querying flight details based on various criteria. Additionally, the system will generate reports for flights based on their status, scheduled departure date, and arrival date, as well as detailed passenger reports for specific flights.

Introduction

Managing flight information is a critical task for any airport. This includes keeping track of flight schedules, managing passenger bookings, and generating reports to monitor the status of flights. This project aims to create a comprehensive system that can efficiently handle these tasks. The system is designed to be interactive, allowing users to input and manage flight and passenger details at runtime.

System Overview

The Airport Flight Management System consists of three main classes:

1. **Passenger Class**: Represents a passenger with attributes such as ID, name, and email.
2. **Flight Class**: Represents a flight with attributes such as flight number, origin, destination, departure date, departure time, arrival date, arrival time, status, and a list of passengers.
3. **AirportManagementSystem Class**: Manages the collection of flights. Provides methods to add, update, delete flights, and generate reports based on different criteria.

Modules

Passenger Class

The `Passenger` class encapsulates the details of a passenger. It includes:

- `id`: A unique identifier for the passenger.
- `name`: The name of the passenger.
- `email`: The email address of the passenger.

Flight Class

The `Flight` class encapsulates the details of a flight. It includes:

- `flightNumber`: A unique identifier for the flight.
- `origin`: The origin airport of the flight.
- `destination`: The destination airport of the flight.
- `departureDate`: The date of departure.
- `departureTime`: The time of departure.
- `arrivalDate`: The date of arrival.
- `arrivalTime`: The time of arrival.
- `status`: The status of the flight (scheduled, cancelled, completed).
- `passengers`: A list of `Passenger` objects associated with the flight.

The `Flight` class also includes methods to add and remove passengers, and to generate a passenger report.

AirportManagementSystem Class

The `AirportManagementSystem` class manages a collection of `Flight` objects. It includes:

- `flights`: An unordered map to store flights, with the flight number as the key.
- Methods to add, update, and delete flights.
- Methods to query flights by status, departure date, and arrival date.
- Methods to generate reports based on the status, departure date, and arrival date of flights.
- Methods to generate detailed passenger reports for specific flights.

Main Function

The `main` function provides an interactive menu for the user to perform various operations on the flight management system. The operations include:

- Adding a new flight.
- Updating an existing flight.
- Deleting a flight.
- Adding a passenger to a flight.
- Generating reports by flight status.
- Generating passenger reports for specific flights.
- Generating reports by departure date.
- Generating reports by arrival date.

Implementation Details

Adding a New Flight

The system allows users to add new flights by providing details such as flight number, origin, destination, departure date, departure time, arrival date, arrival time, and status. The new flight is added to the collection of flights managed by the `AirportManagementSystem`.

Updating an Existing Flight

Users can update the details of an existing flight by specifying the flight number and providing the updated details. The system checks if the flight exists and updates its details accordingly.

Deleting a Flight

Users can delete a flight by specifying its flight number. The system checks if the flight exists and removes it from the collection of flights.

Adding a Passenger to a Flight

Users can add passengers to a flight by specifying the flight number and providing the passenger's details (ID, name, and email). The passenger is added to the list of passengers associated with the specified flight.

Generating Reports

The system provides several options for generating reports:

- **Report by Status**: Generates a list of flights with a specified status (scheduled, cancelled, completed).
- **Passenger Report**: Generates a detailed report of passengers for a specified flight.
- **Report by Departure Date**: Generates a list of flights with a specified departure date.
- **Report by Arrival Date**: Generates a list of flights with a specified arrival date.

Conclusion

The Airport Flight Management System is a comprehensive solution for managing flight and passenger information. It provides robust functionality for adding, updating, and deleting flights, managing passenger details, and generating various reports. The system is designed to be user-friendly and interactive, allowing users to input and manage details at runtime efficiently. This project demonstrates the use of object-oriented programming concepts and data structures in C++ to build a practical application for airport management.

Outputs

```
Airport Management System
1. Add Flight
2. Update Flight
3. Delete Flight
4. Add Passenger to Flight
5. Generate Report by Status
6. Generate Passenger Report
7. Generate Report by Departure Date
8. Generate Report by Arrival Date
9. Exit
Enter your choice: 1
Enter flight number: 12
Enter origin: Boston
Enter destination: NewYork
Enter departure date (YYYY-MM-DD): 2024-07-01
Enter departure time (HH:MM): 08:15
Enter arrival date (YYYY-MM-DD): 2024-07-01
Enter arrival time (HH:MM): 10:00
Enter status (scheduled, cancelled, completed): scheduled
Added flight 12
```

```
Airport Management System
1. Add Flight
2. Update Flight
3. Delete Flight
4. Add Passenger to Flight
5. Generate Report by Status
6. Generate Passenger Report
7. Generate Report by Departure Date
8. Generate Report by Arrival Date
9. Exit
Enter your choice: 4
Enter flight number: 12
Enter passenger ID: 24
Enter passenger name: John
Enter passenger email: john@gmail.com
```

```
Airport Management System
1. Add Flight
2. Update Flight
3. Delete Flight
4. Add Passenger to Flight
5. Generate Report by Status
6. Generate Passenger Report
7. Generate Report by Departure Date
8. Generate Report by Arrival Date
9. Exit
Enter your choice: 6
Enter flight number: 12
Passengers for flight 12:
ID: 24, Name: John, Email: john@gmail.com
```

```
Airport Management System
1. Add Flight
2. Update Flight
3. Delete Flight
4. Add Passenger to Flight
5. Generate Report by Status
6. Generate Passenger Report
7. Generate Report by Departure Date
8. Generate Report by Arrival Date
9. Exit
Enter your choice: 2
Enter flight number to update: 12
Enter new origin (current: Boston): Boston
Enter new destination (current: NewYork): NewYork
Enter new departure date (current: 2024-07-01): 2024-07-01
Enter new departure time (current: 08:15): 08:15
Enter new arrival date (current: 2024-07-01): 2024-07-01
Enter new arrival time (current: 10:00): 10:30
Enter new status (current: scheduled): scheduled
Updated flight 12
```

```
Airport Management System
1. Add Flight
2. Update Flight
3. Delete Flight
4. Add Passenger to Flight
5. Generate Report by Status
6. Generate Passenger Report
7. Generate Report by Departure Date
8. Generate Report by Arrival Date
9. Exit
Enter your choice: 3
Enter flight number to delete: 12
Deleted flight 12
```