oop project explanation

Hostel Management System

Header Files and Namespace

```
#include <iostream>     // For input/output operations (cin, cout)
#include <vector>       // For dynamic arrays (storing rooms, guests, bookings)
#include <string>       // For string data type
#include <iomanip>      // For formatting output (setw, setprecision, fixed)
#include <algorithm>    // For find_if function to search through vectors
#include <ctime>        // For time-related functions (though not used in this co
de)
using namespace std;    // Allows using std library without std:: prefix
```

Room Class Definition (Lines 8-108)

Private Data Members (Lines 10-16)

```
class Room {
private:
    int roomNumber;        // Unique identifier for each room
    string roomType;       // Type of room (e.g., "4-Bed Dorm", "Single Private")
    double pricePerNight;  // Cost per night for the room
    bool isOccupied;       // Flag to check if room is currently occupied
    string guestName;      // Name of current guest (empty if vacant)
    string checkInDate;    // Date when guest checked in
    int nights;            // Number of nights guest will stay
```

Constructor (Lines 18-20)

```
public:
    Room(int num, string type, double price)
        : roomNumber(num), roomType(type), pricePerNight(price),
          isOccupied(false), guestName(""), checkInDate(""), nights(0) {}
```

- **Constructor with initialization list**: Sets room number, type, and price from parameters
- **Default values**: New rooms start as unoccupied with empty guest details

Getter Methods (Lines 22-29)

```
// Getters - Allow access to private data members
int getRoomNumber() const { return roomNumber; }
string getRoomType() const { return roomType; }
double getPricePerNight() const { return pricePerNight; }
bool getIsOccupied() const { return isOccupied; }
string getGuestName() const { return guestName; }
string getCheckInDate() const { return checkInDate; }
int getNights() const { return nights; }
```

- **const methods**: Promise not to modify the object
- **Return copies**: Safely expose private data

Check-In Method (Lines 31-45)

```
void checkIn(string guest, string date, int numNights) {
    if (!isOccupied) {                  // Only if room is vacant
        isOccupied = true;              // Mark as occupied
        guestName = guest;              // Store guest name
        checkInDate = date;             // Store check-in date
        nights = numNights;             // Store duration
        cout << "Room " << roomNumber << " checked in successfully!" << end
l;
        cout << "Guest: " << guest << endl;
```

```
        cout << "Check-in Date: " << date << endl;
        cout << "Nights: " << numNights << endl;
        cout << "Total Cost: $" << fixed << setprecision(2) << (pricePerNight * n
umNights) << endl;
    } else {
        cout << "Room " << roomNumber << " is already occupied by " << guest
Name << "!" << endl;
    }
}
```

- **Conditional check**: Only allows check-in if room is vacant
- **Data updates**: Sets all relevant fields for the stay
- **Cost calculation**: Shows total cost (rate × nights)
- **Formatting**: `fixed` and `setprecision(2)` for currency display

Check-Out Method (Lines 47-71)

```
double checkOut() {
    if (isOccupied) {                    // Only if room is occupied
        double totalCost = pricePerNight * nights;  // Calculate total bill
        cout << "\n========== CHECK OUT SUMMARY ==========" << end
l;
        cout << "Room Number: " << roomNumber << endl;
        cout << "Guest Name: " << guestName << endl;
        cout << "Check-in Date: " << checkInDate << endl;
        cout << "Nights Stayed: " << nights << endl;
        cout << "Rate per Night: $" << fixed << setprecision(2) << pricePerNight
<< endl;
        cout << "Total Amount: $" << fixed << setprecision(2) << totalCost << e
ndl;
        cout << "========================================" << end
l;

        // Reset room to vacant state
```

```
        isOccupied = false;
        guestName = "";
        checkInDate = "";
        nights = 0;

        return totalCost;               // Return amount to collect
    } else {
        cout << "Room " << roomNumber << " is already vacant!" << endl;
        return 0;                       // No money to collect
    }
}
```

- **Bill calculation**: Computes total cost before clearing data
- **Detailed receipt**: Shows complete stay summary
- **Room reset**: Clears all guest data to make room available
- **Return value**: Amount due for payment processing

Display Method (Lines 73-81)

```
void display() const {
    cout << setw(8) << roomNumber        // Column width of 8 characters
         << setw(15) << roomType         // Column width of 15 characters
         << setw(12) << fixed << setprecision(2) << pricePerNight
         << setw(12) << (isOccupied ? "Occupied" : "Available")
         << setw(20) << (isOccupied ? guestName : "N/A")
         << setw(12) << (isOccupied ? checkInDate : "N/A")
         << setw(8) << (isOccupied ? to_string(nights) : "N/A") << endl;
}
```

- **Formatted output**: Uses `setw()` for aligned columns
- **Conditional display**: Shows guest info only if occupied, "N/A" if vacant
- **Consistent spacing**: Creates tabular format for room listings

Setter Method (Lines 83-84)

```
void setPricePerNight(double newPrice) { pricePerNight = newPrice; }
```

- **Price update**: Allows modification of room rates

Guest Class Definition (Lines 86-137)

Private Data Members (Lines 88-94)

```
class Guest {
private:
    int guestId;          // Unique identifier for guest
    string name;          // Guest's full name
    string phone;         // Contact phone number
    string email;         // Email address
    string address;       // Physical address
    int roomNumber;       // Current room assignment (0 if none)
```

Constructor (Lines 96-98)

```
public:
    Guest(int id, string n, string p, string e, string addr, int room = 0)
        : guestId(id), name(n), phone(p), email(e), address(addr), roomNumber(r
oom) {}
```

- **Parameter list**: Takes all guest details
- **Default parameter**: `room = 0` means no room assigned initially

Getter Methods (Lines 100-107)

```
int getGuestId() const { return guestId; }
string getName() const { return name; }
string getPhone() const { return phone; }
string getEmail() const { return email; }
```

```
string getAddress() const { return address; }
int getRoomNumber() const { return roomNumber; }
```

Setter Method (Lines 109-110)

```
void setRoomNumber(int room) { roomNumber = room; }
```

- **Room assignment**: Updates guest's current room

Display Method (Lines 112-120)

```
void display() const {
    cout << setw(5) << guestId
         << setw(20) << name
         << setw(15) << phone
         << setw(25) << email
         << setw(30) << address
         << setw(8) << (roomNumber == 0 ? "N/A" : to_string(roomNumber)) <<
endl;
}
```

- **Formatted guest info**: Displays all guest details in columns
- **Room handling**: Shows "N/A" if no room assigned

Booking Class Definition (Lines 139-194)

Private Data Members (Lines 141-150)

```
class Booking {
private:
    int bookingId;        // Unique booking identifier
    int guestId;          // Links to guest who made booking
    string guestName;     // Guest name for quick reference
    int roomNumber;       // Room being booked
    string checkInDate;   // Planned arrival date
```

```
    string checkOutDate;   // Planned departure date
    int nights;            // Duration of stay
    double totalAmount;    // Total cost of booking
    string status;         // "Active", "Completed", "Cancelled"
```

Constructor (Lines 152-154)

```
public:
    Booking(int id, int gId, string gName, int room, string inDate, string outDate,
int n, double amount)
        : bookingId(id), guestId(gId), guestName(gName), roomNumber(room),
          checkInDate(inDate), checkOutDate(outDate), nights(n), totalAmount(a
mount), status("Active") {}
```

- **Default status**: All new bookings start as "Active"

Getter and Setter Methods (Lines 156-166)

```
int getBookingId() const { return bookingId; }
int getGuestId() const { return guestId; }
string getGuestName() const { return guestName; }
int getRoomNumber() const { return roomNumber; }
string getStatus() const { return status; }
double getTotalAmount() const { return totalAmount; }

void setStatus(string newStatus) { status = newStatus; }
```

Display Method (Lines 168-178)

```
void display() const {
    cout << setw(8) << bookingId
         << setw(8) << guestId
         << setw(20) << guestName
         << setw(8) << roomNumber
         << setw(12) << checkInDate
```

```
        << setw(12) << checkOutDate
        << setw(8) << nights
        << setw(12) << fixed << setprecision(2) << totalAmount
        << setw(12) << status << endl;
}
```

Main System Class (Lines 180-787)

Private Data Members (Lines 182-186)

```
class HostelManagementSystem {
private:
    vector<Room> rooms;        // Collection of all rooms
    vector<Guest> guests;      // Collection of all guests
    vector<Booking> bookings;  // Collection of all bookings
    int nextGuestId;           // Counter for assigning guest IDs
    int nextBookingId;         // Counter for assigning booking IDs
```

Constructor (Lines 188-191)

```
public:
    HostelManagementSystem() : nextGuestId(1), nextBookingId(1) {
        initializeRooms();     // Set up initial room inventory
    }
```

Room Initialization (Lines 193-207)

```
void initializeRooms() {
    // Dormitory rooms (shared accommodation)
    rooms.push_back(Room(101, "4-Bed Dorm", 25.00));
    rooms.push_back(Room(102, "4-Bed Dorm", 25.00));
    rooms.push_back(Room(103, "6-Bed Dorm", 20.00));
    rooms.push_back(Room(104, "6-Bed Dorm", 20.00));
    rooms.push_back(Room(105, "8-Bed Dorm", 18.00));
```

```
    rooms.push_back(Room(106, "8-Bed Dorm", 18.00));

    // Private rooms (individual accommodation)
    rooms.push_back(Room(201, "Single Private", 45.00));
    rooms.push_back(Room(202, "Single Private", 45.00));
    rooms.push_back(Room(203, "Double Private", 35.00));
    rooms.push_back(Room(204, "Double Private", 35.00));
    rooms.push_back(Room(205, "Twin Private", 40.00));
    rooms.push_back(Room(206, "Twin Private", 40.00));
}
```

- **Room variety**: Creates different types of accommodation
- **Pricing strategy**: Dorms cheaper than private rooms, larger dorms cheapest
- **Room numbering**: 100s for dorms, 200s for private rooms

Main Menu Display (Lines 209-220)

```
void showMainMenu() {
    cout << "\n========== HOSTEL MANAGEMENT SYSTEM ==========="
<< endl;
    cout << "1. Room Management" << endl;
    cout << "2. Guest Management" << endl;
    cout << "3. Booking Management" << endl;
    cout << "4. Check-In/Check-Out" << endl;
    cout << "5. Reports" << endl;
    cout << "6. Settings" << endl;
    cout << "7. Exit" << endl;
    cout << "============================================" <
< endl;
    cout << "Enter your choice: ";
}
```

Room Management Menu System (Lines 222-246)

```
void roomManagement() {
    int choice;
    do {    // Menu loop - continues until user chooses to exit
        cout << "\n========== ROOM MANAGEMENT ==========" << endl;
        cout << "1. View All Rooms" << endl;
        cout << "2. View Available Rooms" << endl;
        cout << "3. View Occupied Rooms" << endl;
        cout << "4. Search Room" << endl;
        cout << "5. Update Room Rates" << endl;
        cout << "6. Back to Main Menu" << endl;
        cout << "====================================" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1: viewAllRooms(); break;
            case 2: viewAvailableRooms(); break;
            case 3: viewOccupiedRooms(); break;
            case 4: searchRoom(); break;
            case 5: updateRoomRates(); break;
            case 6: break;    // Exit the loop
            default: cout << "Invalid choice!" << endl;
        }
    } while (choice != 6);
}
```

View All Rooms (Lines 248-264)

```
void viewAllRooms() {
    cout << "\n========== ALL ROOMS ==========" << endl;
    cout << setw(8) << "Room#"          // Column headers
         << setw(15) << "Type"
         << setw(12) << "Rate/Night"
         << setw(12) << "Status"
```

```
        << setw(20) << "Guest Name"
        << setw(12) << "Check-In"
        << setw(8) << "Nights" << endl;
    cout << "----------------------------------------------------------------------
---------------------" << endl;
    for (const auto& room : rooms) {     // Range-based for loop
        room.display();                  // Call display method for each room
    }
    cout << endl;
}
```

- **Table format**: Creates professional-looking output
- **Range-based loop**: Modern C++ syntax for iterating through vector

View Available Rooms (Lines 266-285)

```
void viewAvailableRooms() {
    cout << "\n========== AVAILABLE ROOMS ==========" << endl;
    cout << setw(8) << "Room#"
         << setw(15) << "Type"
         << setw(12) << "Rate/Night" << endl;
    cout << "-----------------------------------" << endl;

    bool hasAvailable = false;           // Flag to track if any rooms available
    for (const auto& room : rooms) {
        if (!room.getIsOccupied()) {     // Check if room is vacant
            hasAvailable = true;
            cout << setw(8) << room.getRoomNumber()
                 << setw(15) << room.getRoomType()
                 << setw(12) << fixed << setprecision(2) << room.getPricePerNight
() << endl;
        }
    }

    if (!hasAvailable) {                 // Handle case when all rooms occupied
```

```
        cout << "No rooms available!" << endl;
    }
    cout << endl;
}
```

- **Filtering**: Only shows unoccupied rooms
- **Conditional display**: Shows message if no rooms available

Search Room Function (Lines 307-325)

```
void searchRoom() {
    int roomNum;
    cout << "Enter room number to search: ";
    cin >> roomNum;

    auto it = find_if(rooms.begin(), rooms.end(),
                      [roomNum](const Room& r) { return r.getRoomNumber() == room
Num; });

    if (it != rooms.end()) {            // If room found
        cout << "\n========== ROOM DETAILS ==========" << endl;
        // Display headers...
        it->display();                  // Show room details
    } else {
        cout << "Room not found!" << endl;
    }
}
```

- **Lambda function**: `[roomNum](const Room& r) { return r.getRoomNumber() == roomNum; }`
- **Iterator**: `find_if` returns iterator to found element or `end()` if not found

Guest Management (Lines 366-408)

```
void addGuest() {
    string name, phone, email, address;
```

```
    cout << "Enter guest name: ";
    cin.ignore();                    // Clear input buffer
    getline(cin, name);              // Read full line (allows spaces in names)
    cout << "Enter phone: ";
    getline(cin, phone);
    cout << "Enter email: ";
    getline(cin, email);
    cout << "Enter address: ";
    getline(cin, address);

    guests.push_back(Guest(nextGuestId++, name, phone, email, address));
    cout << "Guest added successfully with ID: " << (nextGuestId - 1) << endl;
}
```

- **Input handling**: `cin.ignore()` prevents issues with mixed cin/getline usage
- **ID management**: Post-increment ensures unique IDs

Main System Loop (Lines 774-787)

```
void run() {
    int choice;
    cout << "========================================" << endl;
    cout << "   Welcome to Hostel Management System" << endl;
    cout << "========================================" << endl;

    do {
        showMainMenu();
        cin >> choice;

        switch (choice) {
            case 1: roomManagement(); break;
            case 2: guestManagement(); break;
            case 3: bookingManagement(); break;
            case 4: checkInOut(); break;
            case 5: generateReports(); break;
```

```
            case 6: settings(); break;
            case 7:
                cout << "\n========================================"
<< endl;
                cout << "Thank you for using Hostel Management System!" << endl;
                cout << "========================================" <
< endl;
                break;
            default:
                cout << "Invalid choice! Please try again." << endl;
        }
    } while (choice != 7);
}
```

Main Function (Lines 789-793)

```
int main() {
    HostelManagementSystem hms;          // Create system instance
    hms.run();                           // Start the system
    return 0;                            // Successful program termination
}
```

Key Programming Concepts Used

Object-Oriented Design

- **Encapsulation**: Private data members with public getter/setter methods
- **Classes**: Room, Guest, Booking, and HostelManagementSystem classes
- **Constructor initialization lists**: Efficient member initialization

STL (Standard Template Library)

- **Vectors**: Dynamic arrays for storing collections
- **Algorithms**: `find_if` for searching

- **Lambda functions**: Modern C++ anonymous functions for search criteria

Input/Output Formatting

- **iomanip**: `setw()`, `fixed`, `setprecision()` for professional output
- **String handling**: `getline()` for multi-word input

Control Structures

- **Menu systems**: do-while loops for user interaction
- **Switch statements**: Clean handling of menu choices
- **Range-based loops**: Modern iteration through containers

This system demonstrates a complete business application with proper separation of concerns, user-friendly interface, and robust data management.