CA 3: Experiential Learning

Group Members:

Sr. No.	PRN	Name of Student	Mail ID
1	22070122206	Shruti Bist	shruti.bist.btech2022@sitpune.edu.in
2	22070122216	Soumili Biswas	soumili.biswas.btech2022@sitpune.edu.in
3	22070122219	Srishti Parulekar	srishti.parulekar.btech2022@sitpune.edu.in

Problem Statement: Developing an efficient Hotel Management System for reservation, billing and invoicing purposes.

Introduction:

Our project is a user-friendly hotel management system designed to streamline room bookings, billing, and data management. Here's how users can interact with the system:

Features:

Room Selection: Users can choose from various room types (Classic, Deluxe, and Suite) on different floors, enabling personalized room selection.

Booking: Users can book rooms by specifying occupancy, room type, floor, and room number. The system automates room allocation and calculates bills based on room type and stay duration.

Billing: The system calculates total bills for guests per room, enhancing financial operations and customer service.

Search Room Details: Users can access room details, including status, occupants, check-in/check-out dates, and total charge presented in an organized tabular format.

Room Cancellation: Users can cancel their current bookings in case of emergency checkout.

Data Persistence: The code uses file handling to ensure data retention across program runs, initializing room data and saving details to a CSV file.

User Interaction: The main function features a user interaction loop, allowing users to book rooms, check billing, and continue the process efficiently.

Object Oriented Programming Concepts Used:

Encapsulation:

- '**Room**' class members : *'floor_no`, 'room_no`, 'status`, 'occupants`, 'checkin_date`, 'checkout_date` and 'total_bill`* are encapsulated within the class, ensuring data privacy and access control.

Inheritance:

- The classes '*ClassicRoom', 'DeluxeRoom*' and '*SuiteRoom'* inherit from the base '*Room*' class. They specialize in room types, demonstrating the concept of inheritance.

- In this code, the base class is Room, and there are three derived classes: ClassicRoom, DeluxeRoom, and SuiteRoom. Each of these derived classes inherits from the single base class Room. This is an example of *Single Inheritance.*

Polymorphism (Method Overriding):

- The *'calculateTotalBill'* function in the base '*Room*' class is declared as a virtual function, enabling dynamic method binding.

- The derived classes (*'ClassicRoom', 'DeluxeRoom', 'SuiteRoom'*) override this function to provide specialized implementations, showcasing polymorphism.

Composition:

- The '*Hotel*' class manages a map of rooms (*'rooms*'), demonstrating composition. It represents the relationship between a hotel and its constituent rooms.

Dynamic Memory Allocation:

- In the *'initializeCSVFile*' method, dynamic memory allocation using the '*new*' operator is used to create room objects at runtime, based on the specified room types and floors.

Friend Class:

- The '*Hotel*' class is declared as a friend class of the '*Room*' class, allowing it to access and manipulate private members of *'Room'*. This promotes encapsulation while enabling the hotel to manage room data effectively.

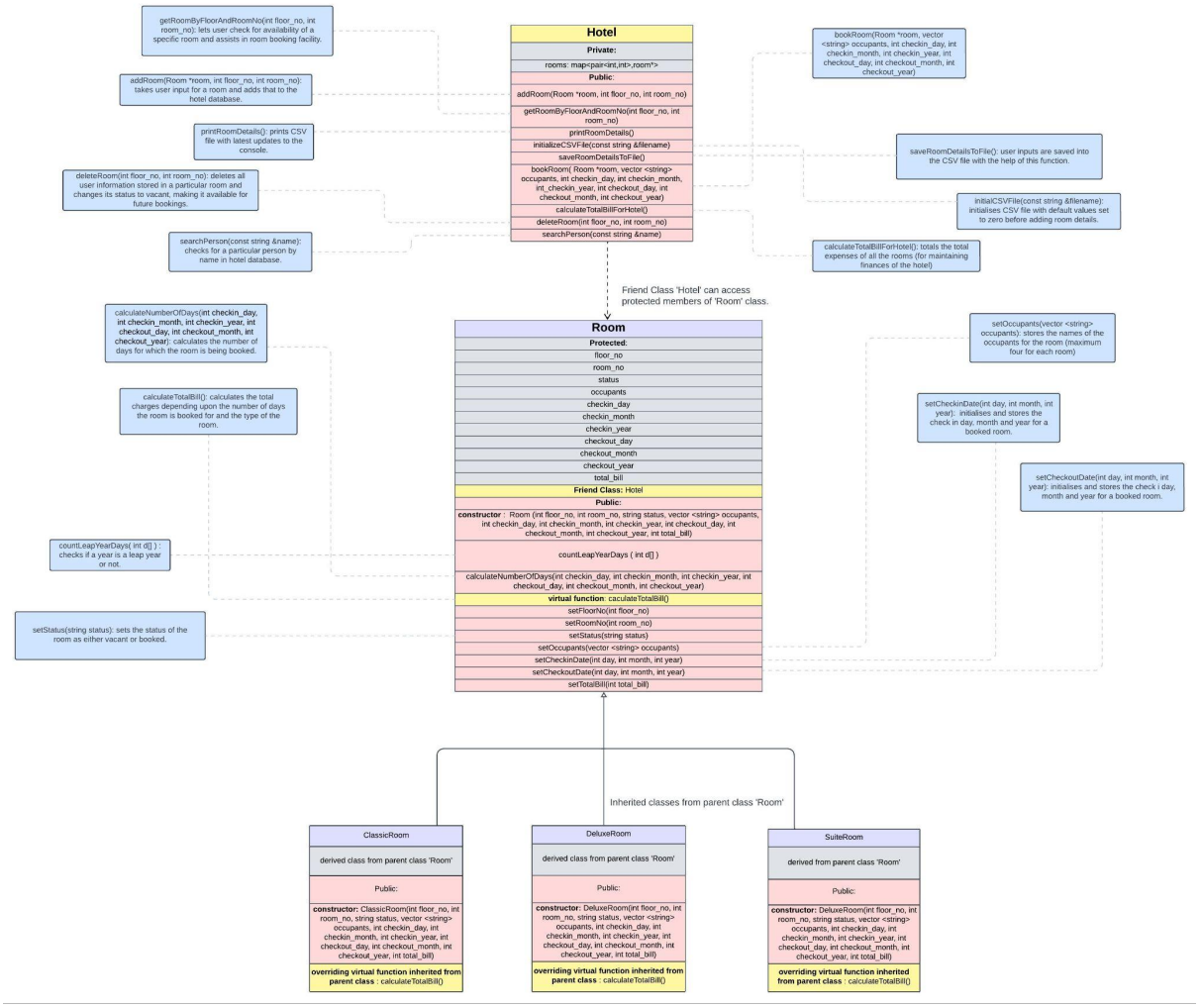
Abstraction:

- Abstraction in this code is primarily achieved through class definitions, providing simplified representations of real-world entities, hiding complex internal details.

Constructors:

- Constructors are used in the '*Room*' class to initialize room objects and in derived classes ('*ClassicRoom', 'DeluxeRoom', 'SuiteRoom*') to set their initial attributes. Additionally, constructors are indirectly employed during dynamic memory allocation when initializing room data in the '*initializeCSVFile*' method.

Class Diagram:



Code snippets:

```cpp
#include <fstream> //allows working with files, including reading from and writing to them
#include <iomanip> // provides tools for formatting input and output in a specified manner
#include <iostream>
#include <map> //allow you to store key-value pairs and quickly look up values by their associated keys
// here we made a map wherein it takes a pair of value as the key(floor and room
// no.) which is mapped to the room status(value)
#include <string> //provides functionality for working with strings
#include <vector> //working with dynamic arrays or lists

using namespace std;
const int monthDays[12] = {31,  59,  90,  120, 151, 181,
                           212, 243, 273, 304, 334, 365};

// Base class representing a room
class Room {
protected:
  // Member variables (attributes)
  int floor_no;
  int room_no;
  string status;
  vector<string> occupants;
  int checkin_day;
  int checkin_month;
  int checkin_year;
  int checkout_day;
  int checkout_month;
  int checkout_year;
  int total_bill;
  friend class Hotel; // Allowing the Hotel class to access private members

public:
  // Constructor for Room class
  Room(int floor_no, int room_no, string status, vector<string> occupants,
       int checkin_day, int checkin_month, int checkin_year, int checkout_day,
       int checkout_month, int checkout_year, int total_bill) {
    // Initialize the member variables
    this->floor_no = floor_no;
    this->room_no = room_no;
    this->status = status;
    this->occupants = occupants;
    this->checkin_day = checkin_day;
    this->checkin_month = checkin_month;
    this->checkin_year = checkin_year;
    this->checkout_day = checkout_day;
    this->checkout_month = checkout_month;
    this->checkout_year = checkout_year;
    this->total_bill = total_bill;
  }
  // Static method for calculating the number of days between two dates
  static int countLeapYearDays(int d[]) {
    int years = d[2];
    if (d[1] <= 2)
      years--;
    return ((years / 4) - (years / 100) + (years / 400));
  }

  static int calculateNumberOfDays(int checkin_day, int checkin_month,
                                   int checkin_year, int checkout_day,
                                   int checkout_month, int checkout_year) {
    int date1[3] = {checkin_day, checkin_month, checkin_year};
    int date2[3] = {checkout_day, checkout_month, checkout_year};

    long int dayCount1 = (date1[2] * 365);
    dayCount1 += monthDays[date1[1] - 1] + date1[0];
    dayCount1 += countLeapYearDays(date1);
    long int dayCount2 = (date2[2] * 365);
    dayCount2 += monthDays[date2[1] - 1] + date2[0];
    dayCount2 += countLeapYearDays(date2);

    return abs(dayCount2 - dayCount1);
  }
```

```
int getFloorNo() { return floor_no; }

int getRoomNo() { return room_no; }

string getStatus() { return status; }

vector<string> getOccupants() { return occupants; }

int getCheckinDay() { return checkin_day; }

int getCheckinMonth() { return checkin_month; }

int getCheckinYear() { return checkin_year; }

int getCheckoutDay() { return checkout_day; }

int getCheckoutMonth() { return checkout_month; }

int getCheckoutYear() { return checkout_year; }

int getTotalBill() { return total_bill; }

void setFloorNo(int floor_no) { this->floor_no = floor_no; }

void setRoomNo(int room_no) { this->room_no = room_no; }

void setStatus(string status) { this->status = status; }

void setOccupants(vector<string> occupants) { this->occupants = occupants; }

void setCheckinDate(int day, int month, int year) {
  checkin_day = day;
  checkin_month = month;
  checkin_year = year;
}
```

```
 void setCheckoutDate(int day, int month, int year) {
   checkout_day = day;
   checkout_month = month;
   checkout_year = year;
 }

 void setTotalBill(int total_bill) { this->total_bill = total_bill; }
 // Virtual function to calculate the total bill (to be overridden in derived
 // classes)
 virtual void calculateTotalBill() {}
};

// Derived class representing a Classic Room(inheritance)
class ClassicRoom : public Room {
public:
 // Constructor for ClassicRoom class (calls the base class constructor)
 ClassicRoom(int floor_no, int room_no, string status,
             vector<string> occupants, int checkin_day, int checkin_month,
             int checkin_year, int checkout_day, int checkout_month,
             int checkout_year, int total_bill)
     : Room(floor_no, room_no, status, occupants, checkin_day, checkin_month,
            checkin_year, checkout_day, checkout_month, checkout_year,
            total_bill) {}
 // Override the calculateTotalBill function for Classic Room (run time
 // polymorphism)
 void calculateTotalBill() override {
   // Calculate the total bill based on the room type and duration
   total_bill = 2000 * calculateNumberOfDays(checkin_day, checkin_month,
                                             checkin_year, checkout_day,
                                             checkout_month, checkout_year);
 }
};
```

```
// Derived class representing a Deluxe Room (inheritance)
class DeluxeRoom : public Room {
public:
  // Constructor for DeluxeRoom class (calls the base class constructor)
  DeluxeRoom(int floor_no, int room_no, string status, vector<string> occupants,
             int checkin_day, int checkin_month, int checkin_year,
             int checkout_day, int checkout_month, int checkout_year,
             int total_bill)
      : Room(floor_no, room_no, status, occupants, checkin_day, checkin_month,
             checkin_year, checkout_day, checkout_month, checkout_year,
             total_bill) {}
  // Override the calculateTotalBill function for Deluxe Room(run time
  // polymorphism)
  void calculateTotalBill() override {
    // Calculate the total bill based on the room type and duration
    total_bill = 2500 * calculateNumberOfDays(checkin_day, checkin_month,
                                              checkin_year, checkout_day,
                                              checkout_month, checkout_year);
  }
};

// Derived class representing a Suite Room(inheritance)
class SuiteRoom : public Room {
public:
  // Constructor for SuiteRoom class (calls the base class constructor)
  SuiteRoom(int floor_no, int room_no, string status, vector<string> occupants,
            int checkin_day, int checkin_month, int checkin_year,
            int checkout_day, int checkout_month, int checkout_year,
            int total_bill)
      : Room(floor_no, room_no, status, occupants, checkin_day, checkin_month,
             checkin_year, checkout_day, checkout_month, checkout_year,
             total_bill) {}
  // Override the calculateTotalBill function for Suite Room(run time
  // polymorphism)
  void calculateTotalBill() override {
    // Calculate the total bill based on the room type and duration
    total_bill = 3000 * calculateNumberOfDays(checkin_day, checkin_month,
                                              checkin_year, checkout_day,
                                              checkout_month, checkout_year);
  }
};

// Class representing a Hotel
class Hotel {
private:
  // Member variable to store rooms using a map
  map<pair<int, int>, Room *> rooms;

public:
  // const int monthDays[12] = {31, 59, 90, 120, 151, 181, 212, 243, 273, 304,
  // 334, 365};
  // Member function to add a room to the hotel
  void addRoom(Room *room, int floor_no, int room_no) {
    rooms[{floor_no, room_no}] = room;
  }

  // Member function to get a room by floor and room number
  Room *getRoomByFloorAndRoomNo(int floor_no, int room_no) {
    auto roomIt = rooms.find({floor_no, room_no});
    if (roomIt != rooms.end() && roomIt->second->getStatus() == "Vacant") {
      return roomIt->second;
    }
    return nullptr;
  }

  // Member function to book a room
  void bookRoom(Room *room, vector<string> occupants, int checkin_day,
                int checkin_month, int checkin_year, int checkout_day,
                int checkout_month, int checkout_year) {
    room->setStatus("Booked");
    room->setOccupants(occupants);
    room->setCheckinDate(checkin_day, checkin_month, checkin_year);
```

```
  room->setCheckoutDate(checkout_day, checkout_month, checkout_year);
  // Calculate the total bill
  int num_days = Room::calculateNumberOfDays(checkin_day, checkin_month,
                                             checkin_year, checkout_day,
                                             checkout_month, checkout_year);
  room->calculateTotalBill();
}

// Member function to calculate the total bill for the entire hotel
void calculateTotalBillForHotel() {
  int total_bill = 0;
  for (const auto &roomPair : rooms) {
    total_bill += roomPair.second->getTotalBill();
  }
  cout << "Total bill for the hotel: " << total_bill << endl;
}

// Member function to print room details
void printRoomDetails() {
  int prevFloor = -1; // To keep track of the previous floor
  cout << left << setw(11) << "Floor No." << setw(11) << "Room No."
       << setw(10) << "Status" << setw(15) << "Occupant 1" << setw(15)
       << "Occupant 2" << setw(15) << "Occupant 3" << setw(15) << "Occupant 4"
       << setw(12) << "Checkin" << setw(12) << "Checkout" << setw(12)
       << "Total Bill" << endl;

  for (const auto &roomPair : rooms)

  {
    Room *room = roomPair.second;
    // Accessor methods to get member variables
    if (room->getFloorNo() != prevFloor) {
      cout << "----------------------------------------------------------------"
              "----------------------------------------------------------------"
              "-\n"; // Add a separator line
      prevFloor = room->getFloorNo();
```

```
    }

    cout << left << setw(11) << room->getFloorNo() << setw(11)
         << room->getRoomNo() << setw(10) << room->getStatus();

    const vector<string> &occupants =
        room->getOccupants(); // stores at max 4 occupants in each floor in a
                              // vector
    for (int i = 0; i < 4; i++) {
      if (i < occupants.size()) {
        cout << setw(15) << occupants[i];
      } else {
        cout << setw(15) << ""; // Empty column
      }
    }
    cout << room->getCheckinDay() << "/" << room->getCheckinMonth() << "/"
         << setw(10) << room->getCheckinYear();
    cout << room->getCheckoutDay() << "/" << room->getCheckoutMonth() << "/"
         << setw(11) << room->getCheckoutYear();
    cout << room->getTotalBill() << endl;
  }
}

// Member function to initialize a CSV file
void initializeCSVFile(const string &filename) {
  ofstream outfile(filename);

  // Write the header to the CSV file
  outfile << left << setw(11) << "Floor No." << setw(11) << "Room No."
          << setw(10) << "Status" << setw(15) << "Occupant 1" << setw(15)
          << "Occupant 2" << setw(15) << "Occupant 3" << setw(15)
          << "Occupant 4" << setw(10) << "Checkin" << setw(10) << "Checkout"
          << setw(10) << "Total Bill" << endl;

  for (int floor = 1; floor <= 5; floor++) {
    if (floor == 1 || floor == 2) {
```

```
      for (int roomNum = 1; roomNum <= 10; roomNum++) {
        ClassicRoom *room =
            new ClassicRoom(floor, roomNum, "Vacant", {}, 0, 0, 0, 0, 0, 0,
                            0);        // Use dynamic allocation (new)
        addRoom(room, floor, roomNum); // Add the room to the hotel
      }
    } else if (floor == 3 || floor == 4) {
      for (int roomNum = 1; roomNum <= 10; roomNum++) {
        DeluxeRoom *room =
            new DeluxeRoom(floor, roomNum, "Vacant", {}, 0, 0, 0, 0, 0, 0,
                           0);         // Use dynamic allocation (new)
        addRoom(room, floor, roomNum); // Add the room to the hotel
      }
    } else if (floor == 5) {
      for (int roomNum = 1; roomNum <= 10; roomNum++) {
        SuiteRoom *room =
            new SuiteRoom(floor, roomNum, "Vacant", {}, 0, 0, 0, 0, 0, 0,
                          0);          // Use dynamic allocation (new)
        addRoom(room, floor, roomNum); // Add the room to the hotel
      }
    }
  }

  // Close the file
  outfile.close();
}
// Member function to save room details to a file
void saveRoomDetailsToFile() {
  ofstream outfile("room_details.csv");
  // Ensure the file is open
  if (!outfile.is_open()) {
    cerr << "Failed to open the CSV file for appending." << endl;
    return;
  }
  outfile << left << setw(11) << "Floor No." << setw(11) << "Room No."
          << setw(10) << "Status" << setw(15) << "Occupant 1" << setw(15)
```

```
        << "Occupant 2" << setw(15) << "Occupant 3" << setw(15)
        << "Occupant 4" << setw(10) << "Checkin" << setw(10) << "Checkout"
        << setw(10) << "Total Bill" << endl;
int prevFloor = -1; // To keep track of the previous floor

for (const auto &roomPair : rooms) {
  Room *room = roomPair.second;

  if (room->getFloorNo() != prevFloor) {
    outfile << "-------------------------------------------------------------"
            "-------------------------------------------------------------"
            "-------\n"; // Add a separator line
    prevFloor = room->getFloorNo();
  }

  outfile << left << setw(11) << room->getFloorNo() << setw(11)
          << room->getRoomNo() << setw(10) << room->getStatus();

  const vector<string> &occupants = room->getOccupants();
  for (int i = 0; i < 4; i++) {
    if (i < occupants.size()) {
      outfile << setw(15) << occupants[i];
    } else {
      outfile << setw(15) << ""; // Empty column
    }
  }

  outfile << room->getCheckinDay() << "/" << room->getCheckinMonth() << "/"
          << setw(10) << room->getCheckinYear() << " ";
  outfile << room->getCheckoutDay() << "/" << room->getCheckoutMonth()
          << "/" << setw(11) << room->getCheckoutYear() << " ";
  outfile << room->getTotalBill() << endl;
}

// Close the file
outfile.close();
```

```
  }

  // member function to delete booking
  void deleteRoom(int floor_no, int room_no) {
    auto roomIt = rooms.find({floor_no, room_no});
    if (roomIt != rooms.end()) {
      Room *room = roomIt->second;

      // Reset the room status to "Vacant" and clear occupants
      room->setStatus("Vacant");
      room->setOccupants({});
      room->setCheckinDate(0, 0, 0);
      room->setCheckoutDate(0, 0, 0);
      room->setTotalBill(0);

      cout << "Booking for Room on Floor " << floor_no << ", Room No. "
           << room_no << " has been deleted." << endl;
    } else {
      cout << "Room not found. Make sure to enter the correct floor number and "
              "room number."
           << endl;
    }
  }

  // Search for a person by name and display their details
  void searchPerson(const string &name) {
    cout << "Searching for " << name << ":\n";
    for (const auto &roomPair : rooms) {
      Room *room = roomPair.second;
      const vector<string> &occupants = room->getOccupants();

      for (const string &occupant : occupants) {
        if (occupant == name) {
          cout << "Name: " << occupant << endl;
          cout << "Floor No.: " << room->getFloorNo() << endl;
          cout << "Room No.: " << room->getRoomNo() << endl;
          cout << "Check-in Date: " << room->getCheckinDay() << "/"
               << room->getCheckinMonth() << "/" << room->getCheckinYear()
               << endl;
          cout << "Check-out Date: " << room->getCheckoutDay() << "/"
               << room->getCheckoutMonth() << "/" << room->getCheckoutYear()
               << endl;
          cout << "Total Bill: " << room->getTotalBill() << endl;
          return; // Exit the function after finding the person
        }
      }
    }

    // If the person is not found
    cout << "Person not found." << endl;
  }
};

int main() {
  string csvFileName = "room_details.csv";
  char ans = 'y';
  Hotel hotel;
  hotel.initializeCSVFile(csvFileName);
  int flag = 0;
  while (ans == 'y' || ans == 'Y') {
    int choice;
    cout << "Menu:\n"; // user menu
    cout << "1. Book a Room\n";
    cout << "2. Delete Booking\n";
    cout << "3. Search Booking\n";
    cout << "4. Display\n";
    cout << "5. Exit\n";
    cout << "Enter your choice (1/2/3/4/5): ";
    cin >> choice;
    switch (choice) {
      int minRooms;
    case 1:
```

```
// Book a room
int numMembers;
cout << "Enter the number of members: ";
flag = 1;
cin >> numMembers;

// Calculate the minimum number of rooms needed
minRooms = (numMembers + 3) / 4; // Ceiling division
cout << "You'll need at least " << minRooms
     << " room(s) to accommodate all members." << endl;

while (numMembers >
       0) // loops that runs till all the members are appointed a room
{
  cout << "\nRoom Types:\n1. Classic Room\n2. Deluxe Room\n3. Suite "
          "Room\n";
  int roomTypeChoice;
  cout << "Choose a room type (1/2/3): ";
  cin >> roomTypeChoice;

  int floorNo, roomNum;
  if (roomTypeChoice == 1) {
    cout << "Classic Room - Choose the floor (1/2): ";
    cin >> floorNo;
    cout << "Enter a room number (1-10): ";
    cin >> roomNum;
  } else if (roomTypeChoice == 2) {
    cout << "Deluxe Room - Choose the floor (3/4): ";
    cin >> floorNo;
    cout << "Enter a room number (1-10): ";
    cin >> roomNum;
  } else if (roomTypeChoice == 3) {
    floorNo = 5; // Suite rooms are on the 5th floor
    cout << "Enter a room number (1-10): ";
    cin >> roomNum;
  } else {
```

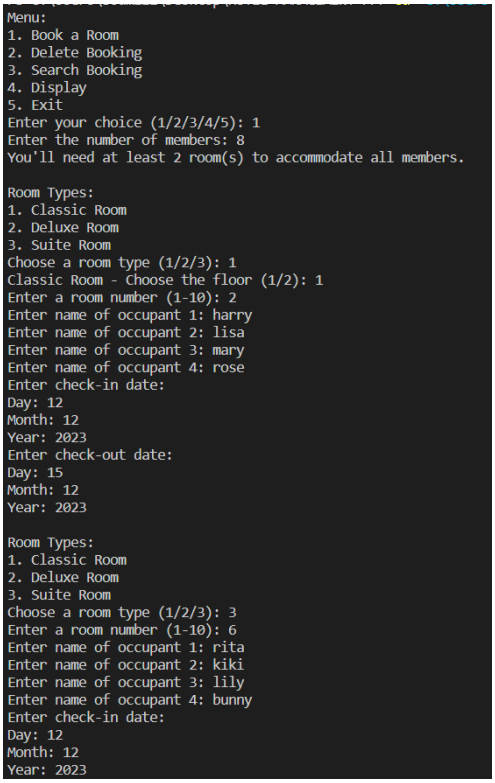
```
    cout << "Invalid room type choice. Please select again." << endl;
    continue;
  }

  // Get a vacant room
  Room *room = hotel.getRoomByFloorAndRoomNo(floorNo, roomNum);

  if (room == nullptr) {
    cout << "The room is not vacant. " << floorNo
         << ". Choose another floor or room type." << endl;
    continue;
  }

  vector<string> occupants;
  int checkin_day, checkin_month, checkin_year;
  int checkout_day, checkout_month, checkout_year;

  for (int i = 0; i < 4 && numMembers > 0; i++) {
    cout << "Enter name of occupant " << i + 1 << ": ";
    string occupantName;
    cin >> occupantName;
    occupants.push_back(occupantName);
    numMembers--;
  }

  cout << "Enter check-in date:" << endl;
  cout << "Day: ";
  cin >> checkin_day;
  cout << "Month: ";
  cin >> checkin_month;
  cout << "Year: ";
  cin >> checkin_year;

  cout << "Enter check-out date:" << endl;
  cout << "Day: ";
  cin >> checkout_day;
```

```
      cout << "Month: ";
      cin >> checkout_month;
      cout << "Year: ";
      cin >> checkout_year;

      // Book the room
      hotel.bookRoom(room, occupants, checkin_day, checkin_month,
                     checkin_year, checkout_day, checkout_month,
                     checkout_year);
    }
    break;
  case 2:
    // Delete booking
    int deleteChoice;
    cout << "Do you want to delete a room? (1 for Yes, 0 for No): ";
    cin >> deleteChoice;
    if (deleteChoice == 1) {
      int floorNo, roomNum;
      cout << "Enter the floor number of the room to delete: ";
      cin >> floorNo;
      cout << "Enter the room number of the room to delete: ";
      cin >> roomNum;
      hotel.deleteRoom(floorNo, roomNum);
    }
    break;
  case 3:
    // Search booking
    int searchBook;
    cout << "Do you want to search a booking? (1 for Yes, 0 for No): ";
    cin >> searchBook;
    if (searchBook == 1) {
      string searchName;
      cout << "Enter the name to search for: ";
      cin >> searchName;
      hotel.searchPerson(searchName);
    }
```

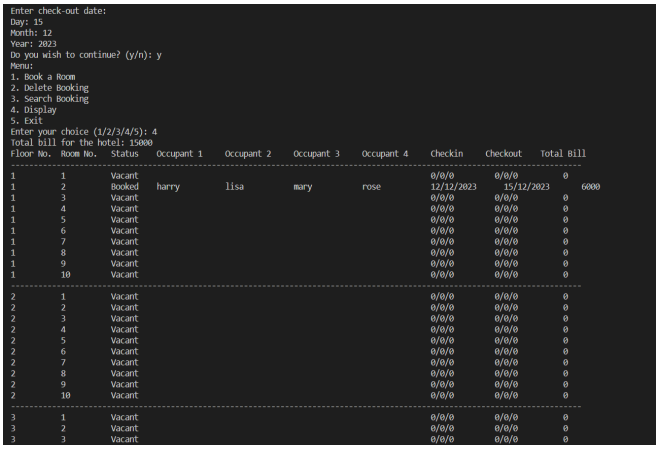
```
      break;
    case 4:
      // Display
      hotel.calculateTotalBillForHotel();
      hotel.printRoomDetails();
      hotel.saveRoomDetailsToFile();
      break;
    case 5:
      // Exit
      return 0;
    default:
      cout << "Invalid choice. Please select again." << endl;
      break;
    }
    cout << "Do you wish to continue? (y/n): ";
    char ans1;
    cin >> ans1;
    ans = ans1;
  }
  return 0;
}
```

Input/Output:

The menu is displayed initially and the user books a room for eight people. Therefore, a minimum of 2 rooms is needed.

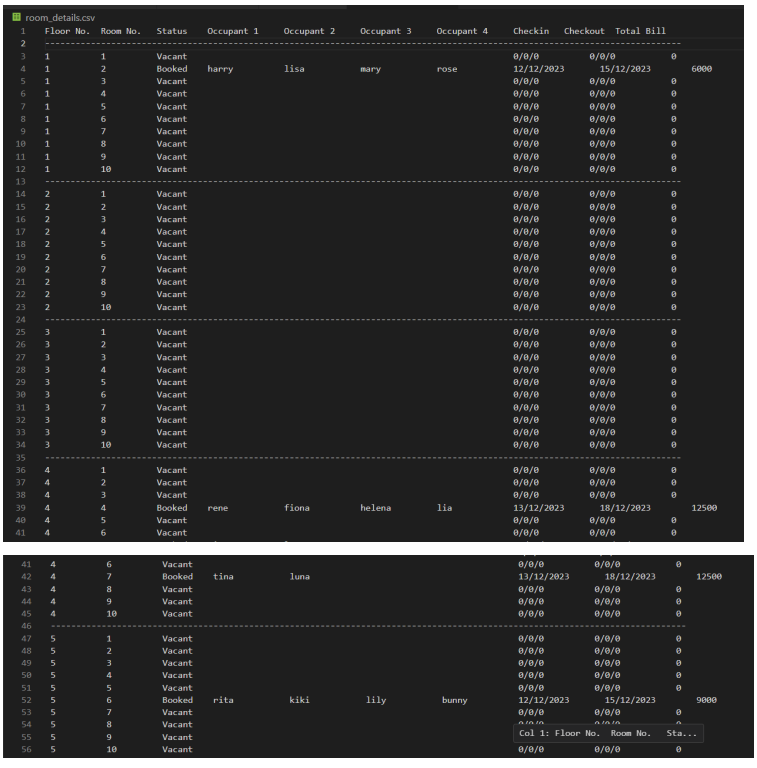


```
Menu:
1. Book a Room
2. Delete Booking
3. Search Booking
4. Display
5. Exit
Enter your choice (1/2/3/4/5): 1
Enter the number of members: 8
You'll need at least 2 room(s) to accommodate all members.

Room Types:
1. Classic Room
2. Deluxe Room
3. Suite Room
Choose a room type (1/2/3): 1
Classic Room - Choose the floor (1/2): 1
Enter a room number (1-10): 2
Enter name of occupant 1: harry
Enter name of occupant 2: lisa
Enter name of occupant 3: mary
Enter name of occupant 4: rose
Enter check-in date:
Day: 12
Month: 12
Year: 2023
Enter check-out date:
Day: 15
Month: 12
Year: 2023

Room Types:
1. Classic Room
2. Deluxe Room
3. Suite Room
Choose a room type (1/2/3): 3
Enter a room number (1-10): 6
Enter name of occupant 1: rita
Enter name of occupant 2: kiki
Enter name of occupant 3: lily
Enter name of occupant 4: bunny
Enter check-in date:
Day: 12
Month: 12
Year: 2023
```



```
Enter check-out date:
Day: 15
Month: 12
Year: 2023
Do you wish to continue? (y/n): y
Menu:
1. Book a Room
2. Delete Booking
3. Search Booking
4. Display
5. Exit
Enter your choice (1/2/3/4/5): 4
Total bill for the hotel: 15000
Floor No.  Room No.  Status   Occupant 1  Occupant 2  Occupant 3  Occupant 4  Checkin   Checkout   Total Bill
-------------------------------------------------------------------------------------------------------------
1          1         Vacant                                                   0/0/0     0/0/0      0
1          2         Booked   harry       lisa        mary        rose        12/12/2023  15/12/2023  6000
1          3         Vacant                                                   0/0/0     0/0/0      0
1          4         Vacant                                                   0/0/0     0/0/0      0
1          5         Vacant                                                   0/0/0     0/0/0      0
1          6         Vacant                                                   0/0/0     0/0/0      0
1          7         Vacant                                                   0/0/0     0/0/0      0
1          8         Vacant                                                   0/0/0     0/0/0      0
1          9         Vacant                                                   0/0/0     0/0/0      0
1          10        Vacant                                                   0/0/0     0/0/0      0
-------------------------------------------------------------------------------------------------------------
2          1         Vacant                                                   0/0/0     0/0/0      0
2          2         Vacant                                                   0/0/0     0/0/0      0
2          3         Vacant                                                   0/0/0     0/0/0      0
2          4         Vacant                                                   0/0/0     0/0/0      0
2          5         Vacant                                                   0/0/0     0/0/0      0
2          6         Vacant                                                   0/0/0     0/0/0      0
2          7         Vacant                                                   0/0/0     0/0/0      0
2          8         Vacant                                                   0/0/0     0/0/0      0
2          9         Vacant                                                   0/0/0     0/0/0      0
2          10        Vacant                                                   0/0/0     0/0/0      0
-------------------------------------------------------------------------------------------------------------
3          1         Vacant                                                   0/0/0     0/0/0      0
3          2         Vacant                                                   0/0/0     0/0/0      0
3          3         Vacant                                                   0/0/0     0/0/0      0
```

A database is created and an output is displayed in the CSV file.



room_details.csv

	Floor No.	Room No.	Status	Occupant 1	Occupant 2	Occupant 3	Occupant 4	Checkin	Checkout	Total Bill
2										
3	1	1	Vacant					0/0/0	0/0/0	0
4	1	2	Booked	harry	lisa	mary	rose	12/12/2023	15/12/2023	6000
5	1	3	Vacant					0/0/0	0/0/0	0
6	1	4	Vacant					0/0/0	0/0/0	0
7	1	5	Vacant					0/0/0	0/0/0	0
8	1	6	Vacant					0/0/0	0/0/0	0
9	1	7	Vacant					0/0/0	0/0/0	0
10	1	8	Vacant					0/0/0	0/0/0	0
11	1	9	Vacant					0/0/0	0/0/0	0
12	1	10	Vacant					0/0/0	0/0/0	0
13										
14	2	1	Vacant					0/0/0	0/0/0	0
15	2	2	Vacant					0/0/0	0/0/0	0
16	2	3	Vacant					0/0/0	0/0/0	0
17	2	4	Vacant					0/0/0	0/0/0	0
18	2	5	Vacant					0/0/0	0/0/0	0
19	2	6	Vacant					0/0/0	0/0/0	0
20	2	7	Vacant					0/0/0	0/0/0	0
21	2	8	Vacant					0/0/0	0/0/0	0
22	2	9	Vacant					0/0/0	0/0/0	0
23	2	10	Vacant					0/0/0	0/0/0	0
24										
25	3	1	Vacant					0/0/0	0/0/0	0
26	3	2	Vacant					0/0/0	0/0/0	0
27	3	3	Vacant					0/0/0	0/0/0	0
28	3	4	Vacant					0/0/0	0/0/0	0
29	3	5	Vacant					0/0/0	0/0/0	0
30	3	6	Vacant					0/0/0	0/0/0	0
31	3	7	Vacant					0/0/0	0/0/0	0
32	3	8	Vacant					0/0/0	0/0/0	0
33	3	9	Vacant					0/0/0	0/0/0	0
34	3	10	Vacant					0/0/0	0/0/0	0
35										
36	4	1	Vacant					0/0/0	0/0/0	0
37	4	2	Vacant					0/0/0	0/0/0	0
38	4	3	Vacant					0/0/0	0/0/0	0
39	4	4	Booked	rene	fiona	helena	lia	13/12/2023	18/12/2023	12500
40	4	5	Vacant					0/0/0	0/0/0	0
41	4	6	Vacant					0/0/0	0/0/0	0
41	4	6	Vacant					0/0/0	0/0/0	0
42	4	7	Booked	tina	luna			13/12/2023	18/12/2023	12500
43	4	8	Vacant					0/0/0	0/0/0	0
44	4	9	Vacant					0/0/0	0/0/0	0
45	4	10	Vacant					0/0/0	0/0/0	0
46										
47	5	1	Vacant					0/0/0	0/0/0	0
48	5	2	Vacant					0/0/0	0/0/0	0
49	5	3	Vacant					0/0/0	0/0/0	0
50	5	4	Vacant					0/0/0	0/0/0	0
51	5	5	Vacant					0/0/0	0/0/0	0
52	5	6	Booked	rita	kiki	lily	bunny	12/12/2023	15/12/2023	9000
53	5	7	Vacant					0/0/0	0/0/0	0
54	5	8	Vacant					0/0/0	0/0/0	0
55	5	9	Vacant					0/0/0	0/0/0	0
56	5	10	Vacant					0/0/0	0/0/0	0

Col 1: Floor No. Room No. Sta...

If the user wishes to delete a room booking (here room number 10 on the fifth floor) :

41	4	6	Vacant					0/0/0	0/0/0	0
42	4	7	Booked	tina	luna			13/12/2023	18/12/2023	12500
43	4	8	Vacant					0/0/0	0/0/0	0
44	4	9	Vacant					0/0/0	0/0/0	0
45	4	10	Vacant					0/0/0	0/0/0	0
46	---									
47	5	1	Vacant					0/0/0	0/0/0	0
48	5	2	Vacant					0/0/0	0/0/0	0
49	5	3	Vacant					0/0/0	0/0/0	0
50	5	4	Vacant					0/0/0	0/0/0	0
51	5	5	Vacant					0/0/0	0/0/0	0
52	5	6	Booked	rita	kiki	lily	bunny	12/12/2023	15/12/2023	9000
53	5	7	Vacant					0/0/0	0/0/0	0
54	5	8	Vacant					0/0/0	0/0/0	0
55	5	9	Vacant					0/0/0	0/0/0	0
56	5	10	Booked	freddie	george	greg	tammy	3/6/2023	4/5/2023	87000
57										

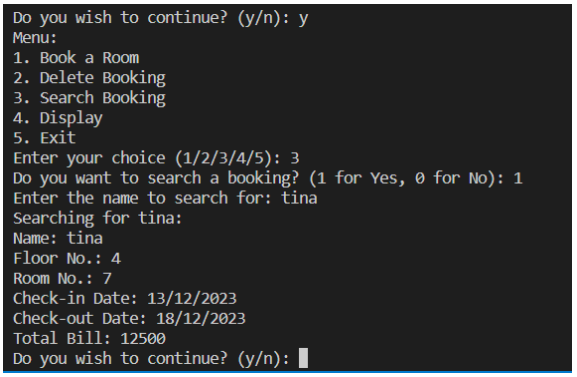
```
Do you wish to continue? (y/n): y
Menu:
1. Book a Room
2. Delete Booking
3. Search Booking
4. Display
5. Exit
Enter your choice (1/2/3/4/5): 2
Do you want to delete a room? (1 for Yes, 0 for No): 1
Enter the floor number of the room to delete: 5 10
Enter the room number of the room to delete: Booking for Room on Floor 5, Room No. 10 has been deleted.
Do you wish to continue? (y/n): y
Menu:
1. Book a Room
2. Delete Booking
3. Search Booking
4. Display
5. Exit
Enter your choice (1/2/3/4/5): 4
Total bill for the hotel: 51000
```

Thus, the 10th room on the 5th floor gets deleted.

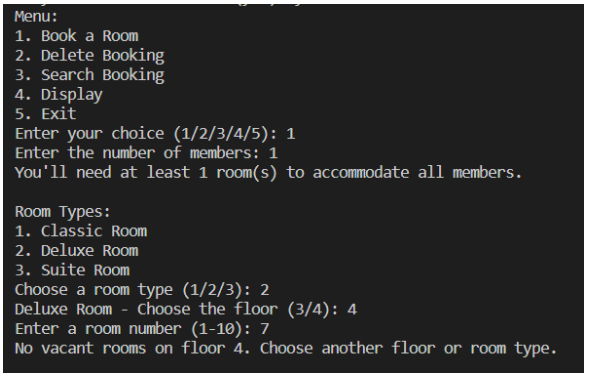
room_details.csv

35	---									
36	4	1	Vacant					0/0/0	0/0/0	0
37	4	2	Vacant					0/0/0	0/0/0	0
38	4	3	Vacant					0/0/0	0/0/0	0
39	4	4	Booked	rene	fiona	helena	lia	13/12/2023	18/12/2023	12500
40	4	5	Vacant					0/0/0	0/0/0	0
41	4	6	Vacant					0/0/0	0/0/0	0
42	4	7	Booked	tina	luna			13/12/2023	18/12/2023	12500
43	4	8	Vacant					0/0/0	0/0/0	0
44	4	9	Vacant					0/0/0	0/0/0	0
45	4	10	Vacant					0/0/0	0/0/0	0
46	---									
47	5	1	Vacant					0/0/0	0/0/0	0
48	5	2	Vacant					0/0/0	0/0/0	0
49	5	3	Vacant					0/0/0	0/0/0	0
50	5	4	Vacant					0/0/0	0/0/0	0
51	5	5	Vacant					0/0/0	0/0/0	0
52	5	6	Booked	rita	kiki	lily	bunny	12/12/2023	15/12/2023	9000
53	5	7	Vacant					0/0/0	0/0/0	0
54	5	8	Vacant					0/0/0	0/0/0	0
55	5	9	Vacant					0/0/0	0/0/0	0
56	5	10	Vacant					0/0/0	0/0/0	0
57										

Searching for a booked room:



Showing no vacancy for an already booked room:



Github repository link:

https://github.com/soumili-03/HOTEL-MANAGEMENT-SYSTEM