**HOTEL MANAGEMENT SYSTEM**

**(DSA PROJECT)**

**Introduction:**

The Hotel Management System presented here is a robust and modular C++ implementation designed to cater to the diverse needs of hotel operations. It encapsulates features such as efficient room management, guest information tracking, reservation handling, and event management. Leveraging object-oriented programming principles and incorporating key data structures, this system provides an organized and efficient solution for hotel management.

**Use & Advantages:**

The core purpose of this system is to streamline and automate various hotel-related tasks, offering several advantages to hotel staff and management:

**Efficient Room Management:**

The system allows for the seamless addition of rooms, providing a clear overview of room occupancy status and simplifying room allocation.

**Guest Information Management:**

Guests can be added to the system with their details, facilitating easy tracking of guest data. This feature aids in personalizing guest experiences and managing customer relationships effectively.

**Reservation Handling:**

The system streamlines the reservation process, making it convenient for staff to manage bookings and cancellations. The integration of a reservation queue ensures systematic handling of reservations.

**Event Management:**

The system supports the initiation and conclusion of events. Additionally, it allows staff to add guests to ongoing events, fostering a dynamic and engaging hotel environment.

**User-Friendly Interface:**

The main function provides a menu-driven interface, enhancing usability for hotel staff. This ensures that the system is accessible to users with varying levels of technical expertise.

**Outline:**

**Need for Project**

Manual hotel management often encounters challenges such as time-consuming processes, increased chances of errors, and difficulty in simultaneously tracking various aspects. The need for automation arises to address these challenges, leading to improved efficiency and a better guest experience. The Hotel Management System aims to streamline operations, reduce manual effort, and enhance overall management capabilities.

**Tools Used**

The choice of the C++ programming language for implementing this project is deliberate. C++ offers a fine balance between performance and expressiveness, making it suitable for developing complex systems. Its object-oriented features contribute to a modular and structured design, aligning with the system's requirements.

**DSA Concepts Used**

Data structures and algorithms (DSA) play a crucial role in the efficiency of any software system. In this project, two key DSA concepts are employed to enhance functionality and performance:

**Queue (BookingQueue class)**

The BookingQueue class utilizes a queue data structure to manage reservations. The FIFO (First-In-First-Out) nature of queues ensures that reservations are handled in a chronological order, aligning with the real-time sequence of booking.

**List (std::list used for managing rooms, guests, and event attendees)**

The std::list container is employed for managing dynamic sets of data related to rooms, guests, and event attendees. Lists provide dynamic memory allocation and efficient insertion and deletion operations, crucial for the dynamic nature of these entities.

**Code:**

The code is organized into well-defined classes, each responsible for a specific aspect of hotel management. The main function serves as an interactive interface, allowing users to perform a variety of actions seamlessly. Extensive comments are provided throughout the code to enhance readability and comprehension, ensuring that the code remains maintainable and adaptable.

#include <iostream>

#include <cstring>

#include <list>

#include <queue>

using namespace std;

const int MAX\_ROOMS = 100; // Defining max. number of rooms as constant

class Guest {

private:

char name[50];

int age;

public:

Guest(const char\* guestName, int guestAge) : age(guestAge) {

strncpy(name, guestName, sizeof(name) - 1);

name[sizeof(name) - 1] = '\0';

}

void displayInformation() const {

cout << "Guest Name: " << name << " | Age: " << age << endl;

}

};

class Room {

private:

int roomNumber;

bool isOccupied;

public:

Room() : roomNumber(0), isOccupied(false) {}

Room(int number) : roomNumber(number), isOccupied(false) {}

void occupyRoom() {

isOccupied = true;

}

void vacateRoom() {

isOccupied = false;

}

void displayInformation() {

cout << "Room Number: " << roomNumber << " | Status: " << (isOccupied ? "Occupied" : "Vacant") << endl;

}

};

class Reservation {

public:

Room\* room;

Guest\* guest;

Reservation(Room\* reservationRoom, Guest\* reservationGuest)

: room(reservationRoom), guest(reservationGuest) {}

void displayInformation() {

cout << "Reservation Details:" << endl;

room->displayInformation();

guest->displayInformation();

}

};

class BookingQueue {

private:

mutable std::queue<Reservation\*> reservations;

public:

void bookRoom(Reservation\* reservation) const {

reservations.push(reservation);

reservation->room->occupyRoom();

}

Reservation\* cancelBooking() const {

if (!reservations.empty()) {

Reservation\* canceledReservation = reservations.front();

reservations.pop();

canceledReservation->room->vacateRoom();

return canceledReservation;

}

return NULL;

}

const std::queue<Reservation\*>& getReservations() const {

return reservations;

}

};

class Event {

private:

char eventName[50];

std::list<Guest\*> attendees;

public:

Event(const char\* name) {

strncpy(eventName, name, sizeof(eventName) - 1);

eventName[sizeof(eventName) - 1] = '\0';

}

void addAttendee(Guest\* guest) {

attendees.push\_back(guest);

}

void displayInformation() const {

cout << "Event Name: " << eventName << endl;

cout << "Number of Attendees: " << attendees.size() << endl;

for (auto attendee : attendees) {

attendee->displayInformation();

}

}

};

class Hotel {

private:

char name[50];

Room\* roomsArray; // Array for easy access to rooms

int numRooms;

std::list<Room\*> roomsList; // Linked list for managing rooms

std::list<Guest\*> guests;

BookingQueue bookingQueue;

Event\* currentEvent;

public:

Hotel(const char\* hotelName, int maxRooms) : numRooms(0), currentEvent(NULL) {

strncpy(name, hotelName, sizeof(name) - 1);

name[sizeof(name) - 1] = '\0';

roomsArray = new Room[maxRooms];

}

~Hotel() {

delete[] roomsArray;

delete currentEvent;

}

Room\* getRoomsArray() const {

return roomsArray;

}

const BookingQueue& getBookingQueue() const {

return bookingQueue;

}

void addRoom(int roomNumber) {

if (numRooms < MAX\_ROOMS) {

roomsArray[numRooms] = Room(roomNumber);

roomsList.push\_back(&roomsArray[numRooms]);

++numRooms;

} else {

cout << "Cannot add more rooms. Maximum limit reached." << endl;

}

}

void addGuest(Guest\* guest) {

guests.push\_back(guest);

}

Reservation\* makeReservation(Guest\* guest, Room\* room) const {

if (room && guest) {

Reservation\* reservation = new Reservation(room, guest);

bookingQueue.bookRoom(reservation);

return reservation;

}

return NULL;

}

void startEvent(const char\* eventName) {

if (currentEvent == NULL) {

currentEvent = new Event(eventName);

} else {

cout << "An event is already in progress." << endl;

}

}

void endEvent() {

delete currentEvent;

currentEvent = NULL;

}

void addGuestToEvent(Guest\* guest) const {

if (currentEvent != NULL) {

currentEvent->addAttendee(guest);

} else {

cout << "No event in progress to add guests to." << endl;

}

}

void displayInformation() const {

cout << "Hotel Name: " << name << endl;

cout << "Number of Rooms: " << numRooms << endl;

for (auto room : roomsList) {

room->displayInformation();

}

cout << "Number of Guests: " << guests.size() << endl;

for (auto guest : guests) {

guest->displayInformation();

}

if (currentEvent != NULL) {

currentEvent->displayInformation();

}

}

};

int main() {

// Main Code for the Hotel Management System

const int MAX\_ROOMS = 100;

Hotel hotel("Grand Hotel", MAX\_ROOMS);

int choice;

do {

cout << "Hotel Management System Menu:" << endl;

cout << "1. Add Room" << endl;

cout << "2. Add Guest" << endl;

cout << "3. Make Reservation" << endl;

cout << "4. Cancel Reservation" << endl;

cout << "5. Start Event" << endl;

cout << "6. End Event" << endl;

cout << "7. Add Guest to Event" << endl;

cout << "8. Display Information" << endl;

cout << "0. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: {

int roomNumber;

cout << "Enter Room Number: ";

cin >> roomNumber;

hotel.addRoom(roomNumber);

break;

}

case 2: {

char guestName[50];

int guestAge;

cout << "Enter Guest Name: ";

cin.ignore(); // Ignore any leftover newline characters

cin.getline(guestName, sizeof(guestName));

cout << "Enter Guest Age: ";

cin >> guestAge;

Guest\* guest = new Guest(guestName, guestAge);

hotel.addGuest(guest);

break;

}

case 3: {

// Implementation for making a reservation

break;

}

case 4: {

// Implementation for canceling a reservation

break;

}

case 5: {

char eventName[50];

cout << "Enter Event Name: ";

cin.ignore();

cin.getline(eventName, sizeof(eventName));

hotel.startEvent(eventName);

break;

}

case 6: {

hotel.endEvent();

break;

}

case 7: {

// Implementation for adding a guest to an event

break;

}

case 8: {

hotel.displayInformation();

break;

}

case 0: {

cout << "Exiting program." << endl;

break;

}

default:

cout << "Invalid choice. Please try again." << endl;

}

} while (choice != 0);

return 0;

}

**Output:**

Sample outputs are meticulously examined to showcase the functionality of the system. Demonstrations include the addition of rooms, guests, reservations, and events. The system's ability to display relevant information and manage data dynamically is underscored through these outputs.

**Conclusion:**

The Hotel Management System stands as a valuable tool for hotels aiming to automate and optimize their day-to-day operations. The modular design, incorporation of key DSA concepts, and the provision of a user-friendly interface contribute to its effectiveness in addressing the challenges associated with manual hotel management. While the current implementation meets basic requirements, there is ample room for enhancement and expansion based on specific hotel requirements and evolving industry standards.