**HOTEL MANAGEMENT SYSTEM**

#include <iostream>

using namespace std;

class node {

public:

int floor\_number;

int room\_id;

string room\_type;

string room\_status;

node\* next;

node(int floor\_number, int room\_id, string room\_type, string room\_status) {

this->floor\_number = floor\_number;

this->room\_id = room\_id;

this->room\_type = room\_type;

this->room\_status = room\_status;

next = NULL;

}

};

class Floors {

private:

node\* head;

public:

Floors() {

head = NULL;

}

void add\_Room(int floor\_number, int room\_id, string room\_type, string room\_status) {

node\* newnode = new node(floor\_number, room\_id, room\_type, room\_status);

if (head == NULL) {

head = newnode;

} else {

node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newnode;

}

}

void Display\_floors() {

cout << "Available floors:" << endl;

for (int i = 1; i <= 5; i++) {

cout << "Floor number: " << i << endl;

}

}

void Display\_rooms\_on\_floor(int floor\_number) {

if (head == NULL) {

cout << "No rooms are available." << endl;

} else {

node\* temp = head;

bool found = false;

cout << "Rooms on Floor " << floor\_number << ":\n";

while (temp != NULL) {

if (temp->floor\_number == floor\_number) {

cout << "Room ID: " << temp->room\_id

<< ", Type: " << temp->room\_type

<< ", Status: " << temp->room\_status << endl;

found = true;

}

temp = temp->next;

}

if (!found) {

cout << "No rooms found on this floor." << endl;

}

}

}

bool change\_room\_status(int floor\_num, int room\_id, const string& status) {

node\* temp = head;

while (temp != NULL) {

if (temp->floor\_number == floor\_num && temp->room\_id == room\_id) {

temp->room\_status = status;

return true; // Successfully changed room status

}

temp = temp->next;

}

return false; // Room not found

}

bool is\_room\_available(int floor\_num, int room\_id) {

node\* temp = head;

while (temp != NULL) {

if (temp->floor\_number == floor\_num && temp->room\_id == room\_id) {

return temp->room\_status == "Ready";

}

temp = temp->next;

}

return false; // Room not found or not available

}

};

class BookingNode {

public:

string room\_type, req\_type, customer\_name;

int stay\_nights, floor\_num, room\_id;

BookingNode\* next;

BookingNode(string customer\_name, string req\_type, string room\_type, int stay\_nights, int floor\_num, int room\_id) {

this->customer\_name = customer\_name;

this->req\_type = req\_type;

this->room\_type = room\_type;

this->stay\_nights = stay\_nights;

this->floor\_num = floor\_num;

this->room\_id = room\_id;

this->next = NULL;

}

};

class BookingQueue {

private:

BookingNode\* front;

public:

BookingQueue() {

front = NULL;

}

int booked\_rooms = 0;

void enqueue(string customer\_name, string req\_type, string room\_type, int stay\_nights, int floor\_num, int room\_id) {

BookingNode\* newNode = new BookingNode(customer\_name, req\_type, room\_type, stay\_nights, floor\_num, room\_id);

if (front == NULL) {

front = newNode;

} else {

if (req\_type == "priority") {

newNode->next = front;

front = newNode;

} else {

BookingNode\* temp = front;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

}

booked\_rooms++;

cout << "Room booked successfully! Total booked rooms: " << booked\_rooms << endl;

}

void total\_booked\_rooms() {

cout << "Total booked rooms: " << booked\_rooms << endl;

}

void cancel\_booking(int floor\_num, int room\_id, Floors& floors) {

if (front == NULL) {

cout << "No bookings to cancel!" << endl;

return;

}

BookingNode\* temp = front;

BookingNode\* prev = NULL;

while (temp != NULL) {

if (temp->floor\_num == floor\_num && temp->room\_id == room\_id) {

if (prev == NULL) {

front = temp->next;

} else {

prev->next = temp->next;

}

delete temp;

booked\_rooms--;

floors.change\_room\_status(floor\_num, room\_id, "Ready");

cout << "Booking cancelled! Total booked rooms: " << booked\_rooms << endl;

return;

}

prev = temp;

temp = temp->next;

}

cout << "Booking not found!" << endl;

}

};

class BookingStack {

private:

BookingNode\* top;

int count\_req;

public:

BookingStack() {

top = NULL;

count\_req = 0;

}

void Push\_Req(string customer\_name, string req\_type, string room\_type, int stay\_nights, int floor\_num, int room\_id) {

BookingNode\* newNode = new BookingNode(customer\_name, req\_type, room\_type, stay\_nights, floor\_num, room\_id);

if (top == NULL || req\_type == "priority") {

newNode->next = top;

top = newNode;

} else {

BookingNode\* temp = top;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

count\_req++;

}

void Cancel\_req(Floors& floors) {

if (top == NULL) {

cout << "There are no requests to cancel!" << endl;

} else {

BookingNode\* toDelete = top;

floors.change\_room\_status(toDelete->floor\_num, toDelete->room\_id, "Ready");

top = top->next;

delete toDelete;

}

count\_req--;

}

void Cancel\_request() {

if (top == NULL) {

cout << "There are no requests to cancel!" << endl;

} else {

BookingNode\* toDelete = top;

top = top->next;

delete toDelete;

}

count\_req--;

}

void display\_requests() {

if (top == NULL) {

cout << "No customer requests found." << endl;

return;

}

BookingNode\* temp = top;

int count = 1;

cout << "Customer Requests: " << endl;

while (temp != NULL) {

cout << "Request number " << count++ << ": " << endl;

cout << "Customer Name: " << temp->customer\_name << endl;

cout << "Request Type: " << temp->req\_type << endl;

cout << "Room Type: " << temp->room\_type << endl;

cout << "Nights of Stay: " << temp->stay\_nights << endl;

cout << "Floor Number: " << temp->floor\_num << endl;

cout << "Room ID: " << temp->room\_id << endl;

cout << "----------------------------" << endl;

temp = temp->next;

}

}

void display\_req\_to\_admin(){

if (top == NULL) {

cout << "No customer requests found." << endl;

return;

}

BookingNode\* temp = top;

int count = 1;

cout << "Customer Requests: " << endl;

while (temp != NULL) {

cout << "Request " << count++ << ": " ;

cout << temp->room\_type << " room by " << temp->customer\_name <<" for " << temp->stay\_nights

<< " nights "<<endl;

cout << "----------------------------" << endl;

temp = temp->next;

}}

void approve\_req(Floors& floors) {

if (top == NULL) {

cout << "No requests to approve." << endl;

return;

}

BookingNode\* temp = top;

cout << "Press Y to approve the request: ";

char option;

cin >> option;

if (option == 'Y' || option == 'y') {

cout << "Request approved." << endl;

// Change the room status in Floors

if (floors.change\_room\_status(temp->floor\_num, temp->room\_id, "Occupied")) {

} else {

cout << "Failed to update the room status!" << endl;

}

Cancel\_request(); // Remove the approved request from the stack

} else {

cout << "Approval canceled." << endl;

}

}

void TotalReq() {

cout << "Total requests: " << count\_req << endl;

}

};

class booking {

public:

string room\_type, customer\_name, req\_type;

int stay\_nights, floor\_num, room\_id;

void customer\_req(BookingQueue& queue, Floors& floors, BookingStack& stack) {

cout << "Provide the following details:" << endl;

cout << "Request Type (Priority 50$ / Regular 20$): ";

cin >> req\_type;

cout << "Customer's name: ";

cin >> customer\_name;

cout << "Type of Room (Single/Double/Suite): ";

cin >> room\_type;

cout << "Nights of Stay: ";

cin >> stay\_nights;

if(stay\_nights > 30 || stay\_nights<1){

cout << "You can book a room for atleast 1 and not more than 30 nights! " << endl;

cout << "Nights of Stay: ";

cin >> stay\_nights;

}

cout << "Floor Number (1-5): ";

cin >> floor\_num;

cout << "Room ID: ";

cin >> room\_id;

if (floors.is\_room\_available(floor\_num, room\_id)) {

cout << "Room is available. Proceeding with booking." << endl;

queue.enqueue(customer\_name, req\_type, room\_type, stay\_nights, floor\_num, room\_id);

floors.change\_room\_status(floor\_num, room\_id, "Booked");

} else {

cout << "Room is not available. Please choose another room." << endl;

cout << "Room ID: ";

cin >> room\_id;

cout << "Room is available. Proceeding with booking." << endl;

}

stack.Push\_Req(customer\_name, req\_type, room\_type, stay\_nights, floor\_num, room\_id);

}

};

int main() {

int choice = 0;

Floors floors;

BookingQueue queue;

BookingStack stack;

// Initialize rooms

for (int i = 1; i <= 5; i++) {

for (int j = 1; j <= 10; j++) {

int room\_id = i \* 100 + j;

string room\_type;

if (j <= 3) {

room\_type = "Single";

} else if (j <= 7) {

room\_type = "Double";

} else {

room\_type = "Suite";

}

floors.add\_Room(i, room\_id, room\_type, "Ready");

}

}

while (choice != 6) {

cout << "\n---- WELCOME TO THE GALAXY HOTEL MANAGEMENT SYSTEM ----" << endl;

cout << "Select an option (1, 2, 3, 4, 5, 6): " << endl;

cout << "1. View Floors and Available Rooms" << endl;

cout << "2. Make a Room Booking" << endl;

cout << "3. Cancel Booking" << endl;

cout << "4. View all customer Requests" << endl;

cout << "5. Administrator Panel" << endl;

cout << "6. Exit the System" << endl;

cin >> choice;

booking book;

switch (choice) {

case 1:

int floor\_num;

floors.Display\_floors();

cout << "Enter the floor number to see rooms on that floor: ";

cin >> floor\_num;

floors.Display\_rooms\_on\_floor(floor\_num);

break;

case 2:

book.customer\_req(queue, floors, stack);

break;

case 3:

stack.Cancel\_req(floors);

break;

case 4:{

stack.TotalReq();

stack.display\_requests();

break;

}

case 5:

int admin\_choice;

do {

cout << "---ADMINISTRATOR PANEL---:\n";

cout << "Please select an option(1, 2, 3, 4): " << endl;

cout << "1. See Total requests\n";

cout << "2. Approve Requests\n";

cout << "3. See Total Booked Rooms\n";

cout << "4. Back to Main Menu\n";

cin >> admin\_choice;

switch (admin\_choice) {

case 1:

stack.TotalReq();

break;

case 2:

stack.display\_req\_to\_admin();

stack.approve\_req(floors);

break;

case 3:

queue.total\_booked\_rooms();

break;

case 4:

cout << "Returning to main menu..." << endl;

break;

default:

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

}

} while (admin\_choice != 4);

break;

case 6:

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

break;

default:

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

break;

}

}

return 0;

}