AHSANULLAH UNIVERSITY OF SCIENCE AND TECHNOLOGY

DEPERTMENT OF ELECTRICAL & ELECTRONIC ENGINEERING

Project Report

Course No	: EEE 1110
Course Name	: Programming Language Laboratory

Project name : Hotel Management System

Submitted by

Name	: Adittya Sinha Arko
Student ID	: 00724105131029
Year	: 1st Year
Semester	: 1st Semester
Section	: A2

Hotel Management System

1. Introduction

The **Hotel Management System** is a C++ **console-based application** designed to streamline room reservations, guest management, and checkouts. It allows users to:
- View available rooms
- Book a room
- Check out guests
- Search for guest details

The system uses **file handling** to maintain persistent data, ensuring that room information is saved across sessions.

2. Problem Description

The **Hotel Management System** addresses the following problems associated with manual hotel management:

- **Inefficient Room Booking:** Manual hotel booking may lead to **double bookings** or **incorrect booking**.
- **Lack of Guest Tracking:** It is difficult to fetch guest details without a system.
- **Billing Inconsistencies:** Manual bill calculations might cause inconsistencies.
- **Data Loss:** Traditional logbooks lack **data persistence**.

Challenges Faced While Developing

- Using a **dynamic data structure** to effectively store rooms and guest information.
- Making sure proper file handling is implemented for storing and fetching room information.
- Implementing a **simple console-based user interface**.

3. System Design

The program follows **Object-Oriented Programming (OOP)** principles, utilizing **three primary classes**:

3.1 Room Class

Purpose: Represents a hotel room with details such as type, price, and availability.

Attribute	**Description**
roomNum (int)	Unique room number
roomType (string)	Type of room (Standard/Deluxe)
roomPrice (double)	Room price per night
occupied (bool)	Indicates whether the room is booked

Methods:

```
getNumber(), getType(), getPrice(), getStatus() – Get Functions
bookRoom() – Marks the room as booked
vacateRoom() – Sets the room as available
display() – Prints room details
toFileString() – Converts room details to a string for file
storage
```

Code Snippets:

```
class Room
{
private:
    int roomNum;
    string roomType;
    double roomPrice;
    bool occupied;
public:
    Room(int num, string type, double price)
    {
        roomNum = num;
        roomType = type;
        roomPrice = price;
        occupied = false;
    }
};
```

3.2 Customer Class

Purpose: Represents a guest who books a room.

Attribute	**Description**
name (string)	Customer's name
contact (string)	Customer’s Contact Number
roomNumber (int)	Room number allocated to the customer
days (int)	Number of days the customer remains

Methods:

`Customer(string n, string c, int rn, int d)` - Constructor is applied to set up customer details.

`“getName()”, “getContact()”, “getRoomNumber()”, “getDays()”` - These are the getter methods.

`calculateBill(const vector<Room>& rooms)` - Calculates total bill based on room rate and stay.

`display()` - Displays customer details.

`toFileString()` - Outputs customer details as string for saving purposes.

Code Snippets:

```
class Customer {
  private:
    string name;
    string contact;
    int roomNumber;
    int days;
  public:
    Customer(string n, string c, int rn, int d) {
        name = n;
     contact = c;
     roomNumber = rn;
     days = d;
  }
};
```

3.3 Hotel Class

Purpose: Manages the hotel operations, including room booking, check-out, and guest information retrieval.

Attribute	**Description**
vector<Room>roomsList	Stores all rooms in the hotel
vector<Customer>guestlist	Stores all active customer bookings

Methods:

`loadRooms()` - Loads room data from a file.

`saveRooms()` - Saves room data to a file.

`bookRoom(int roomNum, const string& name, const string& contact, int days)` - Books a room if available.

`checkOut(int roomNum)` - Processes guest checkout, calculates bill, and frees the room.

viewAllRooms() - Displays all rooms along with their status.

`searchGuest(int roomNum)` - Retrieves guest data by room number.

4. How the Code Works

4.1 Flow of Execution

1. The program starts up and starts running the hotel system.
2. It reads room data from rooms.txt (or creates default rooms if no such file is present).
3. The user is presented with a menu-driven interface to:
    - I. Display available rooms.
    - II. Book a room.
    - III. Check out a guest.
    - IV. Search for a guest.
    - V. Exit the system.
4. Based on what the user selects, the corresponding function is called.
5. Room and guest information are updated dynamically in memory and written to **rooms.txt** for persistence.

4.2 User Interaction

1. The application asks the user for input (e.g., room number, guest name, stay duration).
2. Upon successful booking, the room is indicated as occupied, and customer information is stored.
3. Upon checkout, the application computes the bill and releases the room for subsequent bookings.
4. If there is an invalid input (e.g., selecting an already reserved room), the system displays an appropriate message.

4.3 Error Handling

1. If the user enters an invalid room number, the system prevents errors and prompts valid input.
2. If there are no free rooms, the system displays a message instead of crashing.
3. If rooms.txt is not available or corrupted, default rooms are created automatically.

4.4 File Handling Details

1. **room.txt** has room details and occupancy status.
2. When the program is run, it reads from this file and re-creates the list of rooms.
3. When closing down, the program writes the room status to the file.

5. Design Decisions

5.1 Use of Vectors
- Choosing **Vector** instead of arrays for dynamic storage, better memory management and handling flexibility.

5.2 File Handling for Data Persistence
- Room information is stored in **rooms.txt** to ensure data is not lost on program shutdown.

5.3 Encapsulation & Abstraction for Data Protection
- Private attributes with get and set functions for controlled access.

5.4 Efficiency Considerations:
- "const" and "&" (pass by reference) to prevent unnecessary object copies.

5.5 Separation of Concerns:
- Each class does a well-defined job, making the code modular and easy to maintain.

6. Conclusion

This Hotel Management System is a good implementation of OOP concepts to organize hotel functionalities in an efficient manner. The use of file handling, encapsulation, and dynamic storage gives a good and scalable design.

This project demonstrates a good understanding of the OOP concepts and is a good foundation for further development.