Chandigarh University

University Institute of Computing

Report

On

HOSPITAL MANAGEMENT SYSTEM

SUBMITTED BY: RIDHI

UID: 24BCD10027

SUBMITTED TO: Ms. SHUCHI

SUBJECT: OBJECT ORIENTED PROGRAMMING

SUBJECT-CODE:24CAH-201

ACKNOWEDGMENT

We deem it a great pleasure to express our heartfelt gratitude to our project guide, **Ms. Shuchi**, under whose constant guidance, encouragement, and insightful suggestions we were able to carry out our project work successfully. Her incisive observations, objective feedback, and timely advice provided us with the motivation and clarity we needed at every stage of the project. Her unwavering support instilled in us a continuous flow of energy and enthusiasm to strive for excellence.

We also extend our sincere thanks to **Dr. Kavita Gupta (H.O.D., University Institute of Computing)**, whose valuable insights and constructive suggestions played a crucial role in shaping the direction and quality of our work. Her constant inspiration and supportive demeanour created a nurturing environment that fuelled our academic endeavours.

Our deepest gratitude goes to **Dr. Manisha Malhotra, Director, University Institute of Computing** for fostering a strong academic culture, and for providing us with a disciplined and intellectually stimulating atmosphere. Her leadership and commitment to academic rigor encouraged us to approach the project with utmost seriousness, dedication, and professionalism.

We are also immensely thankful to all the faculty members and staff of the University Institute of Computing for their kind cooperation and constant encouragement throughout the course of our project.

Finally, we would be remiss if we did not acknowledge the unconditional support of our **parents and friends**. No achievement is possible without sacrifices and moral strength provided by loved ones. Their patience, care, and unwavering belief in us were the pillars upon which we built our efforts. To them, we owe our deepest gratitude and love.

Introduction:

The **Hospital Management System** is a simple, yet effective **Object-Oriented Programming (OOP)** based project developed using **C++**.
The purpose of this system is to computerize the basic functionalities of a hospital such as registering patients, storing doctor details, booking appointments, and generating bills.

This project models real-life entities like **Doctors**, **Patients**, and **Appointments** as C++ classes and demonstrates the core concepts of **Encapsulation**, **Inheritance**, **Abstraction**, and **Polymorphism**.

The program helps hospital staff or administrators efficiently manage patient records and appointments while minimizing human error.
It uses arrays to store data instead of STL vectors, keeping it simple, lightweight, and easy to understand.

This project provides an excellent demonstration of how theoretical OOP concepts can be practically applied to solve real-world problems.

Objective:

- To develop a computerized system that automates hospital operations.
- To apply Object-Oriented Programming concepts in solving real-world problems.
- To understand abstraction, encapsulation, inheritance, and polymorphism through implementation.
- To store and manage doctor and patient records in an efficient way.
- To reduce manual paperwork and human error.
- To make hospital processes like appointment booking and billing faster and more accurate.

Techniques Used:

1. Encapsulation

Used to hide sensitive data like patient details and doctor's fees inside classes.
Data can be accessed or modified only through class functions.

2. Inheritance

The Doctor and Patient classes inherit from the base class Person.
Similarly, Outpatient and Inpatient inherit from Patient.
This helps reuse code and maintain a hierarchical structure.

3. Polymorphism

Functions like displayInfo() and calculateBill() are overridden in derived classes.
This allows different types of patients to have their own billing logic.

4. Abstraction

The Person class acts as an abstract base class containing pure virtual functions.
This hides unnecessary implementation details from the main program.

5. Composition

The Hospital class contains multiple Doctor, Patient, and Appointment objects — representing "has-a" relationships.

6. Manual Array Implementation

The project avoids STL containers like vector and instead manages data using arrays and counters.
This makes the code simple and understandable for beginners.

Features:

- Add and manage doctor information (name, age, specialization, fees).
- Register patients as *Inpatients* or *Outpatients*.
- Book appointments and link them with respective doctors and patients.
- Automatically calculate total bills for patients.
- Display doctor, patient, and appointment information neatly.
- Demonstrates all four major OOP concepts in a single program.

Process:

1. Register Doctors:
   The hospital admin adds doctor details like name, specialization, and consultation fees.
2. Register Patients:
   The system accepts details of new patients (name, age, disease).
   The patient can be an Inpatient or an Outpatient.
3. Book Appointments:
   Once doctors and patients are registered, appointments can be booked by selecting doctor and patient indexes along with a date.
4. Calculate Bills:
   The system automatically calculates the total bill by combining doctor consultation fees and patient charges.
5. Display Data:
   Doctors, patients, and appointment records are displayed in an organized way.

6. Memory Management:
   Dynamically allocated objects (like patients) are deleted at the end to prevent memory leaks.

Algorithm:

1. Start the program.
2. Create a hospital object.
3. Register doctors using the addDoctor() function.
4. Create and register patients (Inpatient or Outpatient).
5. Display doctors and patients.
6. Book appointments by linking doctor and patient objects.
7. Calculate and display bills.
8. Show all appointment details.
9. Delete dynamically allocated memory.
10. End program.

Code:

🧱 Header Files and Constants

main.cpp

```cpp
#include <iostream>
#include <iomanip>
using namespace std;
#define MAX_DOCTORS 20
#define MAX_PATIENTS 20
#define MAX_APPOINTMENTS 50
```

🧩 Abstract Base Class – Person

main.cpp Share Run

```cpp
class Person {
protected:
    string name;
    int age;
    string gender;
public:
    Person(string n = "", int a = 0, string g = "") {
        name = n;
        age = a;
        gender = g;
    }
    virtual void displayInfo() = 0;
    string getName() {
        return name;
    }
};
```

🩺 Doctor Class (Derived from Person)

main.cpp Share Run

```
23
24  class Doctor : public Person {
25      string specialization;
26      double fee;
27  public:
28      Doctor(string n = "", int a = 0, string g = "", string s = "",
            double f = 0.0)
29          : Person(n, a, g) {
30          specialization = s;
31          fee = f;
32      }
33
34      void displayInfo() {
35          cout << "Dr. " << name << " (" << specialization << "), Fee:
                ₹" << fee << endl;
36      }
37
38      double getFee() {
39          return fee;
40      }
41  };
```

🧍 Patient Base Class (Abstract:

```
42  class Patient : public Person {
43  protected:
44      string disease;
45
46  public:
47      Patient(string n = "", int a = 0, string g = "", string d = "")
48          : Person(n, a, g) {
49          disease = d;
50      }
51
52      virtual double calculateBill() = 0;
53  };
```

👩‍⚕️ Outpatient Class (Derived from Patient)

```cpp
class Outpatient : public Patient {
public:
    Outpatient(string n = "", int a = 0, string g = "", string d = "")
        : Patient(n, a, g, d) {}

    void displayInfo() {
        cout << "Outpatient: " << name << ", Disease: " << disease << endl;
    }

    double calculateBill() {
        return 500.0;
    }
};
```

👩‍⚕️ Inpatient Class (Derived from Patient)

main.cpp Share Run

```cpp
class Inpatient : public Patient {
    int days;
    double chargePerDay;

public:
    Inpatient(string n = "", int a = 0, string g = "", string d = "", int dy = 0, double charge = 0.0)
        : Patient(n, a, g, d) {
        days = dy;
        chargePerDay = charge;
    }

    void displayInfo() {
        cout << "Inpatient: " << name << ", Disease: " << disease
             << ", Days Admitted: " << days << endl;
    }

    double calculateBill() {
        return (days * chargePerDay) + 1000; // room + base charge
    }
};
```

Appointment Class:

main.cpp

```
class Appointment {
    string doctorName;
    string patientName;
    string date;
    double bill;

public:
    Appointment(string d = "", string p = "", string dt = "", double
        b = 0.0) {
        doctorName = d;
        patientName = p;
        date = dt;
        bill = b;
    }

    void display() {
        cout << left << setw(15) << doctorName
             << setw(15) << patientName
             << setw(12) << date
             << "Bill: ₹" << bill << endl;
    }
};
```

Hospital Class (Main Management System):

main.cpp

```
class Hospital {
    Doctor doctors[MAX_DOCTORS];
    Patient* patients[MAX_PATIENTS];
    Appointment appointments[MAX_APPOINTMENTS];
    int doctorCount;
    int patientCount;
    int appointmentCount;

public:
    Hospital() {
        doctorCount = 0;
        patientCount = 0;
        appointmentCount = 0;
    }

    void addDoctor(string n, int a, string g, string s, double f) {
        if (doctorCount < MAX_DOCTORS) {
            doctors[doctorCount++] = Doctor(n, a, g, s, f);
        } else {
            cout << "Doctor list full!" << endl;
        }
    }

    void addPatient(Patient* p) {
        if (patientCount < MAX_PATIENTS) {
            patients[patientCount++] = p;
        } else {
            cout << "Patient list full!" << endl;
        }
    }
```

main.cpp

Share Run

```
142     void showDoctors() {
143         cout << "\nList of Doctors:\n";
144         for (int i = 0; i < doctorCount; i++) {
145             cout << i + 1 << ". ";
146             doctors[i].displayInfo();
147         }
148     }
149
150     void showPatients() {
151         cout << "\nList of Patients:\n";
152         for (int i = 0; i < patientCount; i++) {
153             patients[i]->displayInfo();
154         }
155     }
156
157     void bookAppointment(int docIndex, Patient* p, string date) {
158         if (docIndex < 0 || docIndex >= doctorCount) {
159             cout << "Invalid doctor number!" << endl;
160             return;
161         }
162
163         double totalBill = doctors[docIndex].getFee() + p->calculateBill();
164
165         if (appointmentCount < MAX_APPOINTMENTS) {
166             appointments[appointmentCount++] = Appointment(doctors[docIndex].getName(), p
                    ->getName(), date, totalBill);
167             cout << "Appointment booked successfully!\n";
168         } else {
169             cout << "Appointment list full!" << endl;
170         }
171     }
```

```
173     void showAppointments() {
174         if (appointmentCount == 0) {
175             cout << "\nNo appointments found.\n";
176             return;
177         }
178
179         cout << "\n===== Appointment Details =====\n";
180         cout << left << setw(15) << "Doctor"
181              << setw(15) << "Patient"
182              << setw(12) << "Date" << "Bill\n";
183         for (int i = 0; i < appointmentCount; i++) {
184             appointments[i].display();
185         }
186     }
187 };
```

Main Function (Program Execution):

```
188 int main() {
189     Hospital h;
190     cout << "===== HOSPITAL MANAGEMENT SYSTEM =====\n";
191     h.addDoctor("Ridhi", 20, "Female", "Cardiologist", 700);
192     h.addDoctor("Abhay", 38, "Male", "Neurologist", 800);
193     h.addDoctor("Harshita", 35, "Female", "Pediatrician", 600);
194     Patient* p1 = new Outpatient("Ashish", 28, "Male", "Fever");
195     Patient* p2 = new Inpatient("Aashna", 50, "Female", "Heart
            Surgery", 5, 2000);
196     h.addPatient(p1);
197     h.addPatient(p2);
198     h.showDoctors();
199     h.showPatients();
200     h.bookAppointment(0, p1, "18-Oct-2025");
201     h.bookAppointment(1, p2, "19-Oct-2025");
202     h.showAppointments();
203     delete p1;
204     delete p2;
205     return 0;
206 }
207
```

Output:

Output Clear

```
===== HOSPITAL MANAGEMENT SYSTEM =====

List of Doctors:
1. Dr. Ridhi (Cardiologist), Fee: ₹700
2. Dr. Abhay (Neurologist), Fee: ₹800
3. Dr. Harshita (Pediatrician), Fee: ₹600

List of Patients:
Outpatient: Ashish, Disease: Fever
Inpatient: Aashna, Disease: Heart Surgery, Days Admitted: 5
Appointment booked successfully!
Appointment booked successfully!

===== Appointment Details =====
Doctor          Patient         Date        Bill
Ridhi           Ashish          18-Oct-2025 Bill: ₹1200
Abhay           Aashna          19-Oct-2025 Bill: ₹11800


=== Code Execution Successful ===
```

Case Study

Hospitals today handle hundreds of patients daily. Managing such data manually leads to inefficiency and errors.
The proposed system automates the process by organizing information digitally.

Key Operations Compared to Real Hospital:

Real Process	System Function
Doctor registration	addDoctor()
Patient admission	addPatient()
Appointment scheduling	bookAppointment()
Billing and discharge	calculateBill()
Record viewing	showAppointments()

This makes the project not just academic but also relevant for small clinics and healthcare startups.

Conclusion

The **Hospital Management System** successfully automates hospital operations using **Object-Oriented Programming in C++**.
It demonstrates how programming principles can be applied to real-life applications.
The system reduces manual effort, increases efficiency, and ensures accurate record handling.

By implementing key OOP concepts — **Encapsulation**, **Inheritance**, **Abstraction**, and **Polymorphism** — this project serves as a strong learning foundation for object-oriented software design.

Summary:

- The system allows addition of doctors and patients.
- Uses inheritance to define Outpatient and Inpatient categories.
- Demonstrates polymorphism through overridden billing functions.
- Uses encapsulation to protect patient and doctor data.
- Displays appointment and billing details in a formatted manner.
- Simple and easy-to-understand code for academic use.

FUTURE ENHANCEMENTS:

- Integration with file handling to permanently store records.
- Creation of a graphical user interface (GUI) for easier navigation.
- Implementation of login authentication for staff and patients.
- Use of databases (MySQL or SQLite) for larger data management.
- Addition of medical report management and prescription modules.
- Ability to send email/SMS notifications for appointments.

REFERENCES

1. **E. Balagurusamy**, *Object-Oriented Programming with C++*, McGraw Hill Education, 8th Edition, 2020.
   → Provided core understanding of OOP concepts like Encapsulation, Inheritance, Abstraction, and Polymorphism used in this project.
2. **Herbert Schildt**, *C++: The Complete Reference*, McGraw Hill Education, 4th Edition, 2017.
   → Referred for syntax standards, class structures, and implementation of virtual functions in C++.
3. **Bjarne Stroustrup**, *The C++ Programming Language*, Addison-Wesley, 4th Edition, 2013.
   → Consulted to understand the real-world design philosophy and principles of Object-Oriented programming in C++.
4. **GeeksforGeeks**, "Hospital Management System in C++," *www.geeksforgeeks.org*, accessed October 2025.
   → Used as a reference for structuring real-world hospital management features like appointments and billing.