CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

UNIVERSITY INSITITUTE OF COMPUTING

(UIC)

TOPIC : A Project Report On HOSPITAL MANAGEMENT SYSTEM

Subject Name/Code: OOPs/CAH-201

Submitted By :

Name: Khushpreet Kaur

UID: 24bca10286

Section : BCA 2(B)

Program Name : BCA

Submitted To :

Ms. Rashmi Saluja

❖ Acknowledgement

I would like to express my sincere gratitude to our respected OOPs faculty [Rashmi Saluja mam], for their valuable guidance, motivation, and continuous support throughout the completion of this project titled “Hospital Management System.”

This project has provided me with deep insights into the concepts of Object-Oriented Programming, such as Encapsulation, Inheritance, Polymorphism, and File Handling, and how these can be practically implemented to design real-world applications.

I am also thankful to my college authorities for providing a conducive environment and resources that helped me in carrying out this project smoothly.

This project has not only strengthened my programming skills but also improved my problem-solving, teamwork, and analytical abilities.

1. Introduction

The rapid growth of technology has transformed every sector, and healthcare is no exception. Managing hospital data through manual systems has always been a tedious and error-prone process. The increasing number of patients, medical records, and billing requirements have made it necessary for hospitals to adopt computerized solutions. The **Hospital Management System (HMS)** is designed to overcome these limitations by providing a structured, efficient, and reliable method of handling hospital operations.

The main aim of this project is to create a **C++-based application** that demonstrates the core principles of **Object-Oriented Programming (OOP)** such as **Encapsulation, Inheritance, Polymorphism, and Abstraction**. The system manages essential hospital data like patient details, doctor information, appointment scheduling, and billing processes. Each function is implemented through different classes and objects to ensure data security, modularity, and easy code maintenance. In this system, the **Patient** class stores information such as name, gender, age, and disease. The **Doctor** class maintains records of doctors, their specializations, and assigned patients. The **Appointment** module helps connect patients with doctors efficiently, while the **Billing** module ensures accurate cost calculation and record storage using file handling in C++

2. Objective

The primary objective of the **Hospital Management System** project is to develop a user-friendly, reliable, and efficient computerized application that streamlines the operations of a hospital using the principles of **Object-Oriented Programming (OOP)** in **C++**.

The system aims to replace traditional manual record-keeping methods with a structured digital platform that can handle patient registration, doctor information, appointment scheduling, and billing processes seamlessly. Through the application of OOP concepts such as **Encapsulation, Inheritance, Polymorphism, and Abstraction**, the project seeks to demonstrate how software design can mirror real-world hospital operations in a logical and modular manner.

Specific objectives include:

1. **To design a modular system** that divides hospital activities into clearly defined classes and objects such as Patient, Doctor, Appointment, and Billing.
2. **To ensure data accuracy and security** through encapsulation and file handling mechanisms that allow only authorized access to information.
3. **To simplify hospital administration** by automating repetitive tasks like maintaining patient history, assigning doctors, and generating bills.
4. **To provide scalability and flexibility** so that new features such as pharmacy, laboratory, or ward management can be easily added in the future.
5. **To enhance learning outcomes** by practically applying OOP concepts to a real-life problem, improving both programming and analytical thinking skills.

3. Problem Statement

In most hospitals, patient and administrative records are still maintained manually using paper-based systems or basic spreadsheets. This traditional method is inefficient, time-consuming, and prone to human error. As the number of patients increases, it becomes difficult to manage data related to patient registration, doctor schedules, appointments, and billing accurately. Misplaced files, duplication of records, and slow data retrieval often lead to poor service quality and operational delays.

Furthermore, in emergency situations, manually searching for a patient's medical history or doctor's availability can waste valuable time, potentially risking lives. Hospitals also struggle to maintain accurate billing records, calculate charges, and generate reports without errors when relying solely on manual processes.

There is therefore a strong need for a **computerized system** that can automate these tasks efficiently and reliably. The **Hospital Management System (HMS)** aims to address these challenges by providing a structured, secure, and efficient platform to handle all hospital-related data. It will ensure that patient details, doctor information, and billing records are stored systematically and can be accessed instantly when needed.

By implementing this system using **Object-Oriented Programming (OOP)** in **C++**, the project not only solves the data management problem but also demonstrates how OOP principles can be applied to design modular and maintainable software. This solution minimizes manual effort, reduces paperwork, and enhances overall hospital administration efficiency.

4. Scope of the Project

The **Hospital Management System (HMS)** has a wide and practical scope in the healthcare sector. It is designed to simplify hospital operations by providing an efficient and reliable way to manage patients, doctors, appointments, and billing. The system is implemented using **Object-Oriented Programming (OOP)** in **C++**, which ensures modularity, reusability, and scalability.

The project serves as a **foundation model** for real-world hospital management software. It can be used in small clinics, local health centers, or mid-level hospitals to reduce paperwork and human errors. By maintaining a digital record of patients and doctors, the system ensures quick access to data, accurate billing, and smooth coordination among hospital departments.

The HMS is flexible and can be **expanded or upgraded** with new modules such as:

- Pharmacy and Inventory Management
- Laboratory Test Records
- Room/Ward Allocation
- Emergency Case Tracking
- Staff and Payroll Management

5.System Design / Modules

1. Patient Module

- Stores patient details such as name, age, address, disease, and admission date.
- Allows updating and deleting records when necessary.

2. Doctor Module

- Maintains doctor details including specialization, experience, and timings.
- Uses inheritance to classify doctors as Permanent or Visiting.

3. Appointment Module

- Connects patients and doctors through composition.
- Manages scheduling, rescheduling, and cancellation of appointments.

4. Billing Module

- Automatically calculates charges for consultation, tests, and medicines.
- Demonstrates the concept of polymorphism through different billing types.

5. Report Module

- Generates reports like patient list, doctor availability, or daily summaries.
- Uses file handling to store and retrieve data for long-term use.

6. UML Diagram Description

The **Unified Modeling Language (UML)** is a standardized visual modeling language used to design and represent the structure and behavior of an object-oriented system. In the **Hospital Management System (HMS)**, UML diagrams help in understanding how different components of the software interact and how data flows between them.

The main UML diagrams used in this project are:

1. Use Case Diagram:

The Use Case Diagram illustrates the overall functionality of the system and identifies the main actors — *Admin, Doctor,* and *Patient.*

- **Admin** can add doctors, manage appointments, and view reports.
- **Doctor** can view assigned patients and update treatment details.
- **Patient** can register, book appointments, and view bills.

This diagram provides a clear picture of the system's interaction from a user's perspective.

2. Class Diagram:

The Class Diagram represents the core classes and their relationships. Major classes include:

- **Patient:** Stores patient details like name, age, gender, and disease.
- **Doctor:** Contains doctor name, specialization, and assigned patients.
- **Appointment:** Connects the doctor and patient based on schedule.
- **Billing:** Handles total payment and file storage.

Relationships such as **association**, **composition**, and **inheritance** are depicted, showing how classes interact while maintaining modularity.

3. Sequence Diagram (Optional):

This diagram can be used to represent the step-by-step interaction between objects during patient registration or billing generation.

Together, these UML diagrams provide a visual blueprint for the design and flow of the Hospital Management System, ensuring structured and efficient implementation.

.

7. OOP Concepts Implemented

The **Hospital Management System** is entirely based on the core principles of **Object-Oriented Programming (OOP)**, which make the system modular, maintainable, and scalable. The key OOP concepts implemented in this project are as follows:

1. Class and Object

The entire project revolves around real-world entities modeled as classes such as Patient, Doctor, Appointment, and Billing. Each class contains attributes (data members) and methods (functions) that define its behavior. For example, the Patient class holds details like name, age, gender, and disease, while the Doctor class stores specialization and assigned patients. Objects of these classes are used to represent actual data instances during program execution.

2. Encapsulation

Encapsulation is achieved by keeping class data members **private** and providing **public** functions to access or modify them. This ensures data security and prevents unauthorized manipulation. It also allows modular development, where each class performs specific responsibilities independently.

3. Inheritance

Inheritance is used to promote **code reusability**. For example, both Patient and Doctor classes can inherit common attributes such as name, gender, and age from a base class Person. This avoids code duplication and maintains cleaner structure.

4. Polymorphism

Polymorphism allows the same function name to behave differently depending on the object that calls it. This is implemented using **function overloading**, such as multiple display or input functions for different modules.

5. Abstraction

Abstraction is implemented by hiding the internal complexities of data management from the user. The user interacts only with simple menu-driven interfaces while the system internally manages files, records, and computations.

6. File Handling

An additional concept demonstrated is **File Handling**, which enables permanent storage of patient and billing details in external files like patients.txt. This adds real-world functionality to the system.

Overall, these OOP concepts ensure the software is well-structured, easy to understand, and extendable for future enhancements.

8.Advantages

1. **Automation of Tasks:**
   Manual record-keeping is replaced with an automated digital process, saving time and reducing human effort.
2. **Accuracy and Reliability:**
   Data such as patient details, doctor schedules, and billing are stored accurately, minimizing errors common in manual systems.
3. **Efficiency and Speed:**
   The system enables instant access to patient history and doctor information, improving hospital productivity and patient satisfaction.
4. **Data Security:**
   By using encapsulation and file handling, sensitive information is stored securely and can only be accessed through authorized operations.
5. **Scalability and Reusability:**
   The modular design allows easy integration of new features like pharmacy or laboratory modules without affecting the existing system.
6. **User-Friendly Interface:**
   The program uses a simple menu-driven interface that can be easily understood and operated by hospital staff with basic computer knowledge.
7. **Cost-Effectiveness:**
   Reduces paper usage and administrative workload, lowering the overall cost of hospital management.

9. Learning Outcomes

While developing this project, several technical and professional skills were acquired:

- **Practical Understanding of OOP Concepts:**
  Gained hands-on experience with classes, objects, inheritance, encapsulation, polymorphism, and abstraction.

- **Improved Coding Skills:**
  Enhanced ability to write structured, error-free C++ programs using file handling and modular design.

- **System Design Thinking:**
  Learned how to model real-world entities into software objects through UML diagrams and class relationships.

- **Problem-Solving and Logic Building:**
  Strengthened analytical skills by translating real-world problems into algorithmic solutions.

- **Teamwork and Project Documentation:**
  Understood the importance of collaboration, testing, and preparing professional documentation for academic projects.

- **Real-World Application Awareness:**
  Recognized how software solutions can enhance efficiency, accuracy, and service quality in healthcare institutions.

10.Conclusion

The **Hospital Management System** project successfully demonstrates how the concepts of **Object-Oriented Programming (OOP)** can be applied to solve real-world problems effectively. The system provides an organized and efficient way to manage hospital activities such as patient registration, doctor management, appointment scheduling, and billing operations. By automating these tasks, it reduces manual workload, minimizes human errors, and improves data accuracy and reliability.

Throughout the development of this project, various **OOP principles**—including **Encapsulation**, **Inheritance**, **Polymorphism**, and **Abstraction**—were effectively implemented to ensure modularity, flexibility, and reusability of code. The addition of **File Handling** furth

The project not only helped in understanding the technical aspects of programming but also emphasized the importance of structured design, logical thinking, and systematic problem-solving. It gave practical exposure to software development stages, from requirement analysis to testing and documentation.
In a broader sense, this system can be extended in the future by integrating additional features such as **pharmacy management**, **laboratory testing**, **online doctor consultation**, and **emergency services**. Such upgrades would make it more comprehensive and adaptable to real hospital environments.

In conclusion, the **Hospital Management System** serves as a solid example of how programming and technology can simplify critical healthcare operations, making services faster, more reliable, and better organized.

11.CODE

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

class Person {
protected:
    string name, gender;
    int age;
public:
    void getData() {
        cout << "Enter name: "; cin >> name;
        cout << "Enter gender: "; cin >> gender;
        cout << "Enter age: "; cin >> age;
    }
    void showData() {
        cout << "Name: " << name << "\nGender: " << gender << "\nAge: " << age << endl;
    }
    string getName() { return name; }
    string getGender() { return gender; }
    int getAge() { return age; }
};

class Patienthttps://www.onlinegdb.com/#editor_1 : public Person {
    string disease;
public:
    void getPatientData() {
        getData();
        cout << "Enter disease: "; cin >> disease;
    }
    void displayPatient() {
        showData();
        cout << "Disease: " << disease << endl;
    }
    string getDisease() { return disease; }
};

class Hospital {
public:
    void savePatient(Patient &p) {
        ofstream file("patients.txt", ios::app);
```

```
        file << p.getName() << "," << p.getGender() << ","
             << p.getAge() << "," << p.getDisease() << "\n";
        file.close();
        cout << "✅ Patient record saved successfully!\n";
    }
};
int main() {
    Patient p;
    Hospital h;
    int choice;

    cout << "\n=== Hospital Management System ===\n";
    cout << "1. Add Patient\n2. Display Patient\n3. Save Patient to File\n4. Exit\n";

    do {
        cout << "\nEnter your choice: ";
        cin >> choice;
        switch (choice) {
            case 1:
                p.getPatientData();
                break;
            case 2:
                cout << "\n--- Patient Details ---\n";
                p.displayPatient();
                break;
            case 3:
                h.savePatient(p);
                break;
            case 4:
                cout << "Exiting...\n";
                break;
            default:
                cout << "Invalid choice!\n";
        }
    } while (choice != 4);

    return 0;
}
```

12.RESULT/OUTPUT

```
=== Hospital Management System ===
1. Add Patient
2. Display Patient
3. Save Patient to File
4. Exit

Enter your choice: 1
Enter name: yo yo
Enter gender: Enter age: 78
Enter disease: fever

Enter your choice: 1
Enter name: nena
Enter gender: female
Enter age: 94
Enter disease: diabetes

Enter your choice: 2

--- Patient Details ---
Name: nena
Gender: female
Age: 94
Disease: diabetes

Enter your choice: 3
✅Patient record saved successfully!

Enter your choice: 4
Exiting...


...Program finished with exit code 0
Press ENTER to exit console.
```