Hospital Management System

Using Object-Oriented Programming Concepts in C++

1. Introduction

The Hospital Management System is a comprehensive software solution designed to streamline and automate the operations of a hospital or healthcare facility. This project implements a fully functional hospital management system using C++ and Object-Oriented Programming principles to efficiently manage patients, doctors, appointments, and medical records.

1.1 Purpose

The primary purpose of this system is to provide an efficient, reliable, and user-friendly interface for hospital staff to manage daily operations without the need for manual record-keeping. It aims to reduce paperwork, minimize human errors, and improve the overall efficiency of hospital management.

A **Hospital Management System (HMS)** is a software solution designed to streamline and automate the day-to-day operations of a hospital or healthcare facility. It serves as a centralized platform for managing various activities such as patient registration, doctor information, appointment scheduling, treatment tracking, and billing. By digitalizing administrative and medical processes, a hospital management system ensures improved efficiency, accuracy, and service quality for both staff and patients.

In traditional hospital settings, data is often managed manually through paperwork, which increases the chances of human error, misplacement of files, and delays in accessing information. A computerized system eliminates these inefficiencies by storing data in structured formats, allowing for quick retrieval and updates. It enhances communication between departments and maintains accurate records of patients' medical histories, doctors' schedules, and treatment details.

The main objective of this system is to simplify hospital workflows and enhance data organization in an academic or small-scale healthcare environment. It is designed with simplicity and modularity in mind, allowing for easy future extensions like appointment management, file storage, or login-based access control. The interface is console-based, making it suitable for understanding the logic behind HMS implementations without the complexity of graphical user interfaces.

Overall, a hospital management system is a critical tool in modern healthcare infrastructure. While this project represents a simplified model, it illustrates the foundational logic and structure of larger-scale hospital software used in real-world environments. Such systems not only improve operational efficiency but also contribute significantly to better patient care and administrative control.

In today's fast-paced and technology-driven world, the healthcare sector increasingly depends on reliable software systems to manage large volumes of sensitive data and ensure

efficient delivery of medical services. One such solution is the **Hospital Management System (HMS)**, a software application designed to handle the comprehensive operations of a hospital. It centralizes and automates functions such as patient registration, doctor assignments, scheduling, treatment tracking, record management, and sometimes even billing and pharmacy operations. By doing so, it minimizes manual work, reduces errors, and enhances overall productivity within the hospital environment.

CODE:

```
#include <iostream>
#include <vector>
using namespace std;
//PERSON CLASS
class Person {
protected:
string name;
int age;
public:
Person() {}
Person(string n, int a) : name(n), age(a) {}
virtual void display() {
cout << "Name: " << name << ", Age: " << age << endl;
}
string getName() { return name; }
};
// Patient class
class Patient : public Person {
string patientID;
```

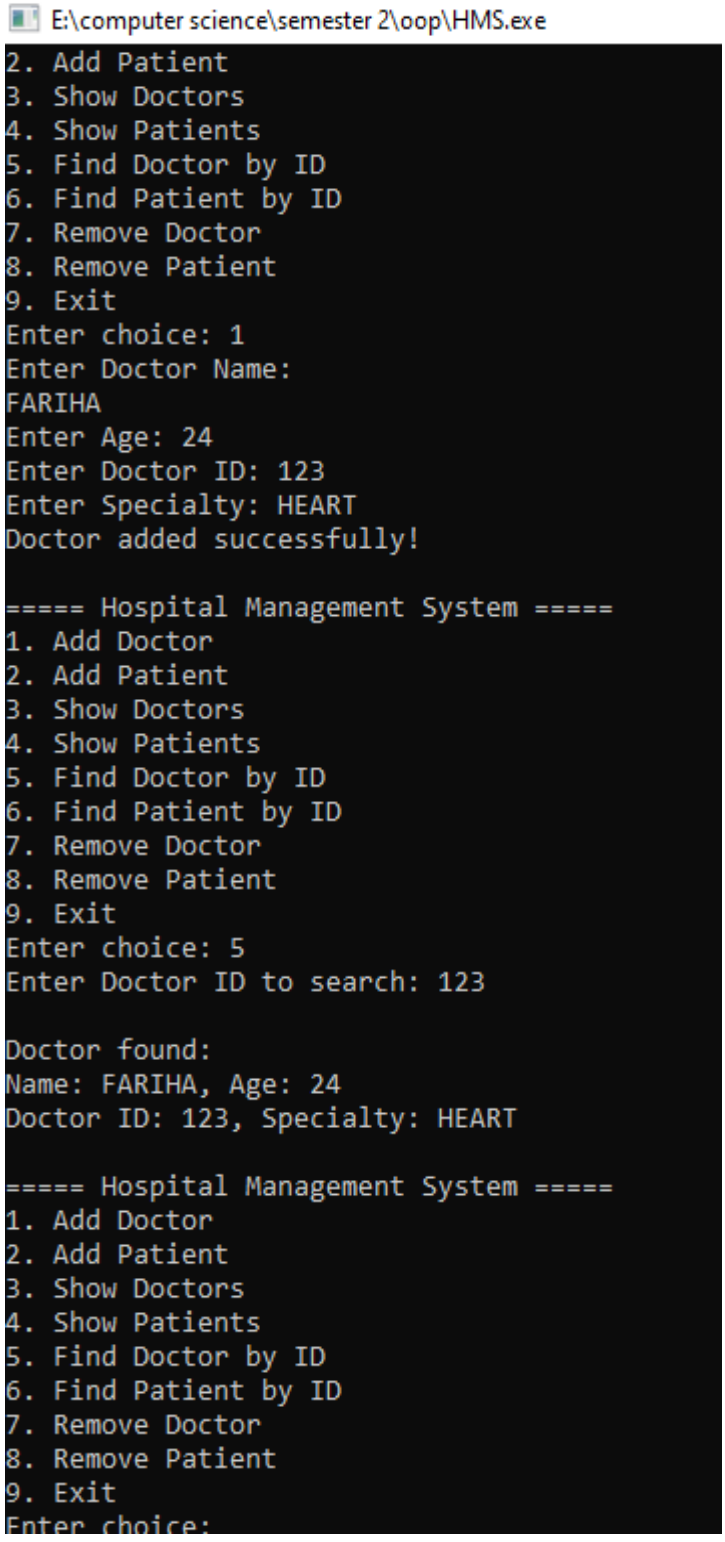
```
string disease;
public:
Patient() {}
Patient(string n, int a, string id, string dis)
: Person(n, a), patientID(id), disease(dis) {}
void display() {
Person::display();
cout << "Patient ID: " << patientID << ", Disease: " << disease << endl;
}
string getID() { return patientID; }
};
// Doctor class
class Doctor : public Person {
string doctorID;
string specialty;
public:
Doctor() {}
Doctor(string n, int a, string id, string spec)
: Person(n, a), doctorID(id), specialty(spec) {}
void display() {
Person::display();
cout << "Doctor ID: " << doctorID << ", Specialty: " << specialty << endl;
}
string getID() { return doctorID; }
};
// Hospital class
class Hospital {
string name;
vector<Doctor> doctors;
vector<Patient> patients;
public:
Hospital(string n) : name(n) {}
void addDoctor(Doctor d) {
doctors.push_back(d);
cout << "Doctor added successfully!\n";
}
void addPatient(Patient p) {
patients.push_back(p);
cout << "Patient added successfully!\n";
}
void showDoctors() {
cout << "\n--- List of Doctors ---\n";
for (Doctor d : doctors)
d.display();
}
void showPatients() {
cout << "\n--- List of Patients ---\n";
for (Patient p : patients)
p.display();
}
void findDoctorByID(string id) {
```

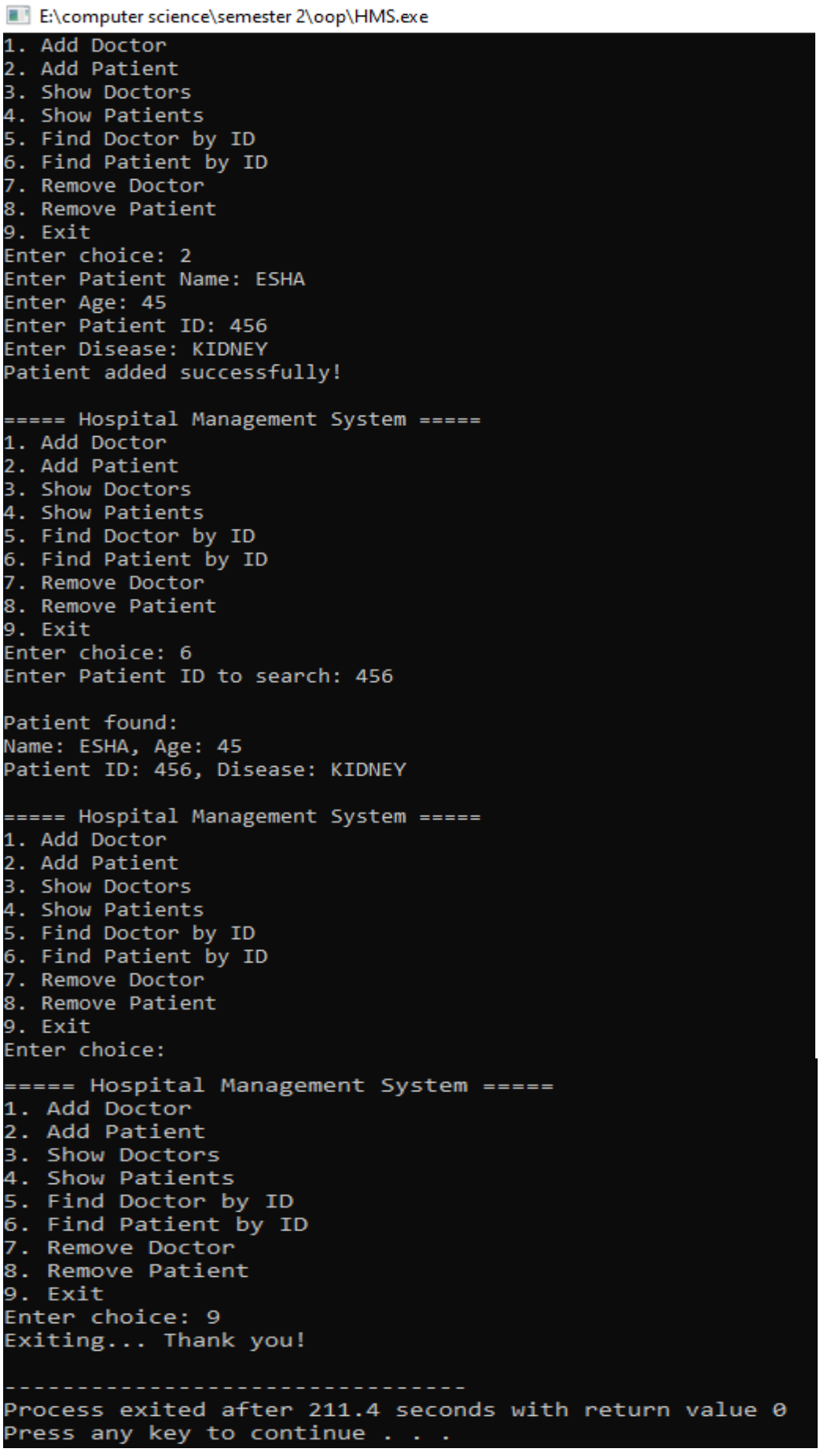
```
for (Doctor d : doctors) {
if (d.getID() == id) {
cout << "\nDoctor found:\n";
d.display();
return;
}
}
cout << "Doctor not found.\n";
}
void findPatientByID(string id) {
for (Patient p : patients) {
if (p.getID() == id) {
cout << "\nPatient found:\n";
p.display();
return;
}
}
cout << "Patient not found.\n";
}
void removeDoctor(string id) {
for (int i = 0; i < doctors.size(); i++) {
if (doctors[i].getID() == id) {
doctors.erase(doctors.begin() + i);
cout << "Doctor removed.\n";
return;
}
}
cout << "Doctor ID not found.\n";
}
void removePatient(string id) {
for (int i = 0; i < patients.size(); i++) {
if (patients[i].getID() == id) {
patients.erase(patients.begin() + i);
cout << "Patient removed.\n";
return;
}
}
cout << "Patient ID not found.\n";
}
};
int main() {
Hospital h("City Hospital");
int choice;
string name, id, disease, specialty;
int age;
while (true) {
cout << "\n===== Hospital Management System =====\n";
cout << "1. Add Doctor\n2. Add Patient\n3. Show Doctors\n4. Show Patients\n";
cout << "5. Find Doctor by ID\n6. Find Patient by ID\n";
cout << "7. Remove Doctor\n8. Remove Patient\n9. Exit\n";
cout << "Enter choice: ";
```

```
cin >> choice;
switch (choice) {
case 1:
cout << "Enter Doctor Name: ";
cin >> name;
cout << "Enter Age: ";
cin >> age;
cout << "Enter Doctor ID: ";
cin >> id;
cout << "Enter Specialty: ";
cin >> specialty;
h.addDoctor(Doctor(name, age, id, specialty));
break;
case 2:
cout << "Enter Patient Name: ";
cin >> name;
cout << "Enter Age: ";
cin >> age;
cout << "Enter Patient ID: ";
cin >> id;
cout << "Enter Disease: ";
cin >> disease;
h.addPatient(Patient(name, age, id, disease));
break;
case 3:
h.showDoctors();
break;
case 4:
h.showPatients();
break;
case 5:
cout << "Enter Doctor ID to search: ";
cin >> id;
h.findDoctorByID(id);
break;
case 6:
cout << "Enter Patient ID to search: ";
cin >> id;
h.findPatientByID(id);
break;
case 7:
cout << "Enter Doctor ID to remove: ";
cin >> id;
h.removeDoctor(id);
break;
case 8:
cout << "Enter Patient ID to remove: ";
cin >> id;
h.removePatient(id);
break;
case 9:
```

```
cout << "Exiting... Thank you!\n";
return 0;
default:
cout << "Invalid choice. Try again.\n";}}}
```

OUTPUT:




E:\computer science\semester 2\oop\HMS.exe
1. Add Doctor
2. Add Patient
3. Show Doctors
4. Show Patients
5. Find Doctor by ID
6. Find Patient by ID
7. Remove Doctor
8. Remove Patient
9. Exit
Enter choice: 2
Enter Patient Name: ESHA
Enter Age: 45
Enter Patient ID: 456
Enter Disease: KIDNEY
Patient added successfully!
===== Hospital Management System =====
1. Add Doctor
2. Add Patient
3. Show Doctors
4. Show Patients
5. Find Doctor by ID
6. Find Patient by ID
7. Remove Doctor
8. Remove Patient
9. Exit
Enter choice: 6
Enter Patient ID to search: 456
Patient found:
Name: ESHA, Age: 45
Patient ID: 456, Disease: KIDNEY
===== Hospital Management System =====
1. Add Doctor
2. Add Patient
3. Show Doctors
4. Show Patients
5. Find Doctor by ID
6. Find Patient by ID
7. Remove Doctor
8. Remove Patient
9. Exit
Enter choice:
===== Hospital Management System =====
1. Add Doctor
2. Add Patient
3. Show Doctors
4. Show Patients
5. Find Doctor by ID
6. Find Patient by ID
7. Remove Doctor
8. Remove Patient
9. Exit
Enter choice: 9
Exiting... Thank you!

Process exited after 211.4 seconds with return value 0
Press any key to continue . . .

C++ Concepts Used in Hospital Management System

Concept	Where Used	Why Used / Purpose
Class & Object	Classes: Person, Doctor, Patient, Hospital	To model real-world entities and structure code using objects.
Inheritance	Doctor and Patient inherit from Person	To reuse common attributes like name and age.
Encapsulation	Data members protected/private, accessed via methods	To protect internal data and expose only necessary functionality.
Polymorphism (Overriding)	display() method overridden in subclasses	To allow different behaviors using the same function name.
Constructors	Used in all classes for initialization	To initialize objects with values during creation.
Vector (std::vector)	Vectors to store patients and doctors	To dynamically manage lists of records with easy add/remove.
Control Structures	switch, if-else, used in menu system	To perform actions based on user input.
Input/Output	cin and cout	To interact with users via console input/output.
Loops	for loops for iteration, while(true) for menu	To repeat actions like showing lists or looping menu.
Abstraction	Public methods like addPatient(), showDoctors()	To hide internal implementation and simplify usage.

The Hospital Management System in C++ is a menu-driven console application that utilizes object-oriented programming (OOP) concepts to manage doctors and patients in a hospital. It defines a base class Person containing shared attributes like name and age, which is inherited by Doctor and Patient classes, each adding specific attributes such as doctorID and specialty for doctors, and patientID and disease for patients. A Hospital class manages lists of doctors and patients using std::vector, allowing dynamic addition, removal, and search of records. The program features constructors for object initialization, function overriding for polymorphic behavior, and encapsulation to protect data access. Users interact with the system through a looped menu that provides options to add, view, search, and delete doctors or patients. It uses control structures (switch, if-else), input/output (cin, cout), and loops (for, while) to manage the flow and ensure an interactive and user-friendly experience. The design promotes modularity, abstraction, and clean separation of responsibilities among classes.

Advantages of Hospital Management System

- **Improved Data Management:** Efficiently stores and organizes patient and doctor information using classes and dynamic data structures.
- **Faster Access to Information:** Enables quick search and retrieval of records, reducing delays and improving service speed.
- **Reduced Manual Errors:** Automation minimizes human errors related to manual record keeping.
- **Modular and Scalable Design:** Object-oriented structure allows easy maintenance and future enhancements.
- **User-friendly Menu System:** Simple console-based menu makes it easy for hospital staff to navigate.
- **Reusability of Code:** Inheritance and polymorphism reduce code duplication and improve maintainability.
- **Dynamic Data Storage:** Using vectors allows flexible management of varying numbers of patients and doctors.
- **Cost-effective Prototype:** Low development cost and complexity, suitable for educational purposes and testing.

Disadvantages of Hospital Management System

- **No Persistent Storage:** Data is lost when the program ends since no files or databases are used.
- **Limited User Interface:** Console-based input/output lacks the usability and appeal of graphical interfaces.
- **Lack of Advanced Features:** Missing critical hospital functions like appointment scheduling and billing.
- **No Multi-user Access or Security:** Lacks user authentication and simultaneous multi-user support.
- **Scalability Issues for Large Data:** Inefficient for handling very large datasets without a database backend.
- **Manual Input Errors Possible:** User input can cause errors if validation and error handling are not implemented.
- **Basic Functionality Only:** Designed primarily for learning and small-scale use, not full hospital deployment.

CONCLUSION

The Hospital Management System developed using C++ and Object-Oriented Programming principles is a foundational yet effective model for managing essential operations in a healthcare environment. It successfully demonstrates how core OOP concepts such as classes, inheritance, encapsulation, and polymorphism can be applied to real-world scenarios. Through this system, users can efficiently handle patient and doctor information, perform basic operations like adding, viewing, searching, and deleting records, and experience structured data flow in a modular program design.

Despite being a console-based prototype, the system brings clarity to how digital transformation in hospitals can reduce manual errors, speed up operations, and improve record-keeping. The use of vectors enables dynamic data handling, while the menu-driven interface ensures ease of use and interaction. This project not only strengthens programming and problem-solving skills but also provides practical insight into how software solutions are built for the healthcare sector.

However, the current version has limitations, such as the absence of persistent storage, multi-user support, and advanced features like appointment scheduling or billing. These can be considered areas for future development. Adding file handling, GUI elements, or database integration would further enhance the system's real-world usability.