Hospital Management System

Project submitted to the

SRM University – AP, Andhra Pradesh

Submitted in partial fulfillment of the requirement for the award of the degree

of

Bachelor of Technology

in

Computer Science and Engineering

School of Engineering and Sciences

Submitted By

G.VEERA CHARAN-(AP23110010895)

A.DINESH REDDY-(AP23110010662)

PV SAI KIRAN-(AP23110010650)

K.RAHUL-(AP23110010945)

Under the guidance of

Ms. Mary CH

Department of Computer Science and Engineering SRM University,AP

Neerukonda, Mangalagiri, Guntur
Andhra Pradesh – 522 240
[November,2024]

Department of Computer Science and Engineering

SRM University,AP

CERTIFICATE

This is to certify that the Project report entitled **"Hospital Management System"** is being submitted by **Gandla Veera Charan (AP23110010895), Andluru Dinesh Reddy (AP23110010662), Patnaikuni Venkata Sai Kiran (AP23110010650), K Rahul (AP23110010945)** a student of Department of Computer Science and Engineering, SRM University,AP, in partial fulfillment of the requirement for the degree of **"B.Tech(CSE)"** carried out by her/his during the academic year 2024-2025.

Signature of the Supervisor

Signature of Head of the Dept.

Acknowledgement

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success.

I am extremely grateful and express my profound gratitude and indebtedness to my project guide, **Mary CH**, Department of Computer Science & Engineering, SRM University,Andhra pradesh, for her kind help and for giving me the necessary guidance and valuable suggestions in completing this project work.

G.VEERA CHARAN-(AP23110010895)

A.DINESH REDDY-(AP23110010662)

PV SAI KIRAN-(AP23110010650)

K.RAHUL-(AP23110010945)

Table of Contents

Abstract

The **Hospital Management System (HMS)** is a C++ application designed to facilitate efficient and streamlined patient and doctor data management within a hospital setting. This system provides essential functionalities such as patient admission, assignment of doctors, room allocation, billing, and record-keeping. Through a structured, object-oriented approach, HMS manages patient information, including contact details, medical requirements, and billing, alongside tracking room assignments based on need.

The system's doctor management feature includes detailed records for each doctor, allowing easy assignment based on specialty, fees, and seniority. Patients can be registered, viewed, or discharged, with automatic bill calculation based on selected services. Additionally, a record of recently discharged patients is maintained temporarily, allowing reference for up to two months.

This project not only enhances administrative efficiency by reducing paperwork and manual errors but also serves as a foundational system that can be expanded with additional functionalities as per hospital needs. The HMS is aimed at providing a comprehensive and modular solution to streamline hospital management operations.

Introduction

In today's healthcare landscape, efficient management of hospital operations is essential for delivering high-quality patient care and ensuring smooth administrative workflows. Hospitals manage a vast amount of data daily, from patient records and doctor assignments to billing and room allocations. Without a streamlined system, tracking and organizing this data can be time-consuming and prone to errors, affecting the overall efficiency and quality of service. The **Hospital Management System (HMS)** in C++ addresses these challenges by providing a structured and user-friendly platform to manage critical hospital operations effectively.

This Hospital Management System is built using object-oriented principles to encapsulate essential entities such as patients, doctors, and room assignments into a cohesive program. The system organizes data through classes like Doctor and Patient, each of which represents key entities within the hospital, and a central HospitalManagementSystem class that facilitates interactions and manages operations. With functionalities like patient admission, doctor assignment, room allocation, billing, and record-keeping, the HMS serves as a comprehensive solution to hospital administrative needs.

Additionally, the system supports efficient room and billing management by offering automatic room assignments and calculating bills based on doctor fees and room charges. For enhanced record-keeping, the system maintains a temporary log of discharged patients, allowing hospital staff to reference recent discharges for a limited time. This modular and extendable structure enables hospitals to simplify and digitize their workflows, ultimately reducing manual errors, improving staff efficiency, and enhancing patient care quality.

Methodology

- The Hospital Management System (HMS) in C++ is designed using a modular, object-oriented approach, organizing functionality through the Doctor, Patient, and Hospital Management System classes.
- The Doctor class captures essential doctor details like ID, specialty, and fees, while the Patient class manages patient-specific data, including age, disease, assigned doctor, room needs, and billing.
- The Hospital Management System class acts as the main coordinator, handling patient admissions, doctor assignments, room allocation, and billing.
- Patients requiring rooms receive unique room numbers starting from 101, and total bills are calculated by combining doctor fees with room charges based on the room type.
- The system temporarily stores discharged patient records with timestamps, automatically removing records older than two months to keep data relevant.
- A menu-driven interface provides options for adding, viewing, deleting, and searching for patients, along with displaying doctor details.
- Built-in data validation ensures accurate entries, with checks for duplicate patient IDs and valid doctor selections, enhancing data integrity and user experience.
- This structure results in an efficient, expandable system that simplifies hospital management operations.

Functionalities

1. Admin Login

- The system includes a secure admin login feature that ensures only authorized personnel can access the system.
- The admin must enter a correct password to gain access to the system.
- If the login fails due to incorrect credentials, access is denied.

2. Doctor Management

- Adding Doctors: Doctors are predefined in the system with details such as ID, name, specialty, fee, and seniority status.
- Viewing Doctor List: The system displays a list of doctors, showing their ID, name, specialty, fee, and whether they are senior doctors.
- Doctor Fee Structure: Each doctor has a consultation fee associated with their specialty, which is used in patient billing.

3. Patient Management

- Adding Patients: Admin can add new patients to the system with details such as patient ID, name, age, disease, contact information, and doctor assigned.
- Room Allocation: The system allows assigning a patient to a room. If the patient requires a room, the system provides the option to choose between different room types (Normal/Deluxe). Each room is assigned a unique room number.
- Viewing Patients: The system allows the admin to view all patients with their details, including ID, name, disease, contact information, assigned doctor, room information, and total bill.
- Searching Patients: Admin can search for patients using their ID to display detailed information.
- Deleting Patients: Admin has the ability to discharge and delete patients from the system. Discharged patients’ data is stored for historical reference.

4. Billing and Prescription Generation

- Billing System: Each patient's total bill is calculated based on the doctor's consultation fee and room charges (if applicable).
- Saving Prescription: The system generates and saves a prescription for each patient, including the doctor’s details and the total bill, in a text file for record-keeping.

5. Discharge and Record Keeping

- Patient Discharge: When a patient is discharged, their details are moved from the active patient list to a queue of discharged patients. The discharge time is recorded for future reference.
- Historical Records: The system maintains a list of discharged patients, allowing for efficient tracking of past cases.

6. Room Management

- Room Assignment: The system assigns rooms to patients based on availability and their requirements (Normal/Deluxe). The room number is dynamically allocated, and the room type is recorded.

7. User Interface

- The user interface provides an intuitive menu-driven system for admins to interact with the hospital management functionalities. The options include adding/viewing patients, managing doctors, handling prescriptions, and viewing the entire hospital record.

8. File Handling

- Saving Patient Data: Patient details are saved in memory and can be printed to a file for record-keeping purposes, including prescriptions.
- Text Files for Prescriptions: Each patient’s prescription is stored in a text file, making it easy to retrieve and print when necessary.

9. Security Features

- The system requires admin authentication to ensure only authorized access to critical functions such as adding patients, doctors, and managing prescriptions.
- No user login is implemented, but the admin can control all operations through a secure password.

Objectivies

- **Efficient Patient Management**:

  To create a system that allows easy addition, modification, viewing, and deletion of patient records, ensuring accurate tracking of patient information such as disease, age, room requirements, and assigned doctor.
- **Doctor Management**:

  To manage and display the list of available doctors along with their specialties, consultation fees, and seniority, making it easy for the user to assign doctors to patients.
- **Room Allocation**:

  To efficiently allocate rooms to patients requiring them, ensuring each patient is assigned a unique room number and the appropriate room charge is added to their bill.
- **Billing and Fee Management**:

  To calculate and manage patient bills by adding doctor fees and room charges (if applicable), providing accurate billing details for patients.
- **Search and Discharge Patient Records**:

  To enable users to search for patients by their ID, view their medical details, and discharge them from the system, while maintaining the discharged patient records for reference.
- **Data Integrity and Validation**:

  To implement data validation checks to prevent duplicate patient records, invalid doctor assignments, and incorrect room allocations, ensuring the accuracy of all data within the system.
- **Discharged Patient Record Management**:

  To maintain and manage records of discharged patients for up to two months, automatically purging old records after the specified period to keep the system clean.

- **User-Friendly Interface**:
  To design a simple, menu-driven interface that provides easy navigation through the functionalities, making the system accessible for hospital staff to use effectively.
- **Real-Time Updates**:
  To ensure that all changes, including patient admissions, discharges, and billing updates, are instantly reflected in the system, providing a real-time overview of hospital occupancy, doctor availability, and patient statuses.
- **Automated Room and Billing Adjustments**:
  To automate room allocation and billing adjustments based on room type and doctor consultation fees, reducing manual calculations and ensuring accurate invoicing for all patients.
- **Secure Record Management**:
  To implement secure record management practices, ensuring that patient data is handled with confidentiality and only accessible through the system’s authorized functionalities.
- **Reduction of Administrative Workload**:
  To reduce the administrative burden on hospital staff by automating repetitive tasks, such as record searching, billing, and room allocation, freeing staff time for other patient care tasks.

Code implementation

```
#include <iostream>
#include <vector>
#include <string>
#include <map>
#include <queue>
#include <ctime>
#include <fstream>
#include <iomanip>

using namespace std;

// Doctor class definition
class Doctor {
public:
    int id;
    string name;
    string specialty;
    float fee;
    bool isSenior;

    Doctor() : id(0), name(""), specialty(""), fee(0.0), isSenior(true) {}
    Doctor(int id, string name, string specialty, float fee, bool isSenior = true)
        : id(id), name(name), specialty(specialty), fee(fee), isSenior(isSenior) {}
};

// Patient class definition
class Patient {
public:
    int id;
    string name;
    int age;
    string disease;
    string contact;
    bool requiresRoom;
    string roomType;
    int roomNumber;
    int treatedDoctorId;
    float totalBill;

    Patient() : id(0), name(""), age(0), disease(""), contact(""),
              requiresRoom(false), roomType("Normal"), roomNumber(0),
              treatedDoctorId(0), totalBill(0) {}

    Patient(int id, string name, int age, string disease, string contact,
           bool requiresRoom, string roomType, int roomNumber,
           int doctorId, float bill)
        : id(id), name(name), age(age), disease(disease), contact(contact),
         requiresRoom(requiresRoom), roomType(roomType), roomNumber(roomNumber),
         treatedDoctorId(doctorId), totalBill(bill) {}

    void display(const map<int, Doctor>& doctorList) const {
        cout << left << setw(5) << id
            << setw(15) << name
            << setw(5) << age
            << setw(20) << disease
            << setw(15) << contact
            << setw(20) << (requiresRoom ? roomType + " (" + to_string(roomNumber) + ")" :
"None")
            << setw(20) << doctorList.at(treatedDoctorId).name
            << setw(10) << "?" << totalBill << "\n";
    }
};

// Structure to hold deleted patient information
struct DischargedPatient {
    int id;
    time_t deletionTime;
};

// Hospital Management System
class HospitalManagementSystem {
private:
    vector<Patient> patients;
    map<int, Doctor> doctors;
    queue<DischargedPatient> dischargedPatients;
    int nextRoomNumber; // To keep track of the next available room number
    const string adminPassword = "admin123"; // Admin password for login
    Patient* findPatientById(int id) {
        for (auto& patient : patients) {
            if (patient.id == id) return &patient;
        }
        return nullptr;
    }

public:
    HospitalManagementSystem() : nextRoomNumber(101) {
        doctors[1] = Doctor(1, "Dr. Dinesh", "Cardiologist", 250.0, true);
        doctors[2] = Doctor(2, "Dr. Charan", "Neurologist", 300.0, false);
        doctors[3] = Doctor(3, "Dr. Kiran", "Orthopedic", 150.0, true);
    }

    bool login() {
        string password;
        cout << "Enter admin password: ";
        cin >> password;
        if (password == adminPassword) {
            cout << "Login successful!\n";
            return true;
        } else {
            cout << "Incorrect password. Access denied.\n";
            return false;
        }
    }

    void showDoctorList() const {
        cout << left << setw(5) << "ID" << setw(15) << "Name" << setw(20) << "Specialty"
             << setw(10) << "Fee (?)" << setw(10) << "Senior" << "\n";
        cout << "------------------------------------------------------------\n";
        for (const auto& doctor : doctors) {
            cout << left << setw(5) << doctor.second.id << setw(15) << doctor.second.name
                 << setw(20) << doctor.second.specialty << setw(10) << doctor.second.fee
                 << setw(10) << (doctor.second.isSenior ? "Yes" : "No") << "\n";
        }
    }

    void addPatient(int id, string name, int age, string disease, string contact, bool
requiresRoom, string roomType, int doctorId) {
        if (findPatientById(id) != nullptr) {
            cout << "Patient with ID " << id << " already exists.\n";
            return;
        }
        if (doctors.find(doctorId) == doctors.end()) {
            cout << "Doctor with ID " << doctorId << " not found.\n";
            return;
        }
        float bill = doctors[doctorId].fee;
        int roomNumber = 0;

        if (requiresRoom) {
            bill += (roomType == "Normal" ? 500 : 1000);
            roomNumber = nextRoomNumber++;
        }

        patients.push_back(Patient(id, name, age, disease, contact, requiresRoom, roomType,
roomNumber, doctorId, bill));
        cout << "Patient added successfully.\n";
    }

    void viewPatients() const {
        if (patients.empty()) {
            cout << "No patients in the system.\n";
            return;
        }
        cout << left << setw(5) << "ID" << setw(15) << "Name" << setw(5) << "Age" << setw(20)
<< "Disease"
             << setw(15) << "Contact" << setw(20) << "Room" << setw(20) << "Doctor" << setw(10)
<< "Total Bill (?)\n";
        cout <<
"----------------------------------------------------------------------------------------------------\n";
        for (const auto& patient : patients) {
            patient.display(doctors);
        }
    }

    void savePrescription(int id) {
        Patient* patient = findPatientById(id);
        if (patient) {
            ofstream file("Patient_" + to_string(id) + "_Prescription.txt");
            if (file.is_open()) {
                file << "Prescription for Patient ID: " << id << "\n";
                file << "Name: " << patient->name << "\nDisease: " << patient->disease
                     << "\nDoctor: " << doctors[patient->treatedDoctorId].name << "\n";
                file << "Total Bill: ?" << patient->totalBill << "\n";
                file.close();
                cout << "Prescription saved successfully.\n";
            } else {
                cout << "Error opening file for writing.\n";
            }
        } else {
            cout << "Patient with ID " << id << " not found.\n";
        }
    }

    void deletePatient(int id) {
        for (auto it = patients.begin(); it != patients.end(); ++it) {
            if (it->id == id) {
                dischargedPatients.push({id, time(0)});
                patients.erase(it);
                cout << "Patient with ID " << id << " discharged.\n";
                return;
            }
        }
        cout << "Patient not found.\n";
    }

    void searchPatient(int id) {
        Patient* patient = findPatientById(id);
        if (patient) {
            cout << left << setw(5) << "ID" << setw(15) << "Name" << setw(5) << "Age" << setw(20)
<< "Disease"
             << setw(15) << "Contact" << setw(20) << "Room" << setw(20) << "Doctor" <<
setw(10) << "Total Bill (?)\n";
            patient->display(doctors);
        } else {
            cout << "Patient with ID " << id << " not found.\n";
        }
    }
};

int main() {
    HospitalManagementSystem hms;
```

```
if (!hms.login()) {
    return 0; // Exit if login fails
}

int choice, id, age, doctorId;
string name, disease, contact, roomType;
bool requiresRoom;

do {
    cout << "\nHospital Management System Menu:\n";
    cout << "1. Show Doctors List\n";
    cout << "2. Add Patient\n";
    cout << "3. View Patients\n";
    cout << "4. Save Prescription\n";
    cout << "5. Delete Patient\n";
    cout << "6. Search Patient\n";
    cout << "7. Exit\n";
    cout << "Enter your choice: ";
    cin >> choice;

    switch (choice) {
    case 1:
        hms.showDoctorList();
        break;
    case 2:
        cout << "Enter Patient ID: ";
        cin >> id;
        cout << "Enter Name: ";
        cin.ignore(); // Ignore newline character from previous input
        getline(cin, name);
        cout << "Enter Age: ";
        cin >> age;
        cout << "Enter Disease: ";
        cin.ignore(); // Ignore newline character
        getline(cin, disease);
        cout << "Enter Contact Number: ";
        getline(cin, contact);
        cout << "Does the patient require a room? (1 for yes, 0 for no): ";
        cin >> requiresRoom;
        if (requiresRoom) {
            cout << "Enter Room Type (Normal/Deluxe): ";
```

```
                cin >> roomType;
            } else {
                roomType = "Normal";
            }
            cout << "Enter Doctor ID: ";
            cin >> doctorId;
            hms.addPatient(id, name, age, disease, contact, requiresRoom, roomType, doctorId);
            break;
        case 3:
            hms.viewPatients();
            break;
        case 4:
            cout << "Enter Patient ID to save prescription: ";
            cin >> id;
            hms.savePrescription(id);
            break;
        case 5:
            cout << "Enter Patient ID to delete: ";
            cin >> id;
            hms.deletePatient(id);
            break;
        case 6:
            cout << "Enter Patient ID to search: ";
            cin >> id;
            hms.searchPatient(id);
            break;
        case 7:
            cout << "Exiting...\n";
            break;
        default:
            cout << "Invalid choice. Try again.\n";
        }
    } while (choice != 7);

    return 0;
}
```

Output

```
Enter admin password: admin123
Login successful!

Hospital Management System Menu:
1. Show Doctors List
2. Add Patient
3. View Patients
4. Save Prescription
5. Delete Patient
6. Search Patient
7. Exit
Enter your choice: 1
ID   Name           Specialty           Fee (?)   Senior
------------------------------------------------------------
1    Dr. Dinesh     Cardiologist        250       Yes
2    Dr. Charan     Neurologist         300       No
3    Dr. Kiran      Orthopedic          150       Yes

Hospital Management System Menu:
1. Show Doctors List
2. Add Patient
3. View Patients
4. Save Prescription
5. Delete Patient
6. Search Patient
7. Exit
Enter your choice: 2
Enter your choice: 2
Enter Patient ID: 01
Enter Name: Dinesh
Enter Age: 19
Enter Disease: Headache
Enter Contact Number: 9573320409
Does the patient require a room? (1 for yes, 0 for no): 1
Enter Room Type (Normal/Deluxe): Deluxe
Enter Doctor ID: 2
Patient added successfully.
```

```
Hospital Management System Menu:
1. Show Doctors List
2. Add Patient
3. View Patients
4. Save Prescription
5. Delete Patient
6. Search Patient
7. Exit
Enter your choice: 3
ID   Name           Age  Disease              Contact        Room
              Doctor              Total Bill (?)
----------------------------------------------------------------------
    ----------------------------------
1    dinesh         19   Headache             9573320409     Deluxe
    (101)       Dr. Charan          ?         1300

Hospital Management System Menu:
1. Show Doctors List
2. Add Patient
3. View Patients
4. Save Prescription
5. Delete Patient
6. Search Patient
7. Exit
Enter your choice: 5
Enter your choice: 5
Enter Patient ID to delete: 01
Patient with ID 1 discharged.

Hospital Management System Menu:
1. Show Doctors List
2. Add Patient
3. View Patients
4. Save Prescription
5. Delete Patient
6. Search Patient
7. Exit
Enter your choice: 7
Exiting...


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

Conclusion

The Hospital Management System developed in C++ provides a robust and streamlined approach to managing essential hospital operations, including patient admissions, doctor assignments, billing, room allocation, and record-keeping. By centralizing and automating these tasks, the system significantly reduces administrative burdens, minimizes errors, and improves operational efficiency. This allows hospital staff to dedicate more time and resources to patient care, enhancing the overall healthcare experience for both patients and staff.

This project exemplifies the power of Object-Oriented Programming (OOP) concepts, which have been instrumental in organizing, structuring, and expanding the system's functionality. Key OOP concepts like **Encapsulation** are used to group related data and functions within the Doctor, Patient, and Hospital ManagementSystem classes, creating distinct and secure modules for handling each type of hospital operation. This encapsulation ensures that each class is responsible for managing its own data, maintaining data integrity and security across the system.

Additionally, **Abstraction** is employed to simplify complex processes such as billing, room management, and discharge tracking. These functions are abstracted into specific methods, allowing users to access these features without needing to understand the underlying code. By focusing on simplicity in user interaction, abstraction enhances usability and accessibility. The system's design also allows for **Inheritance** and **Polymorphism**, making it adaptable for future extensions. For instance, it would be easy to add specialized subclasses for different types of patients or doctors, or to use polymorphism to handle diverse billing structures. This flexibility not only makes the system scalable but also supports code reusability and efficient maintenance.

In conclusion, this Hospital Management System successfully integrates core hospital functions with OOP principles, providing a comprehensive, efficient, and adaptable solution for managing hospital resources. With its structured design, real-time updates, and potential for expansion, the system is well-equipped to meet the needs of a modern healthcare facility and lays a strong foundation for further development and integration with other healthcare systems. The project demonstrates how OOP can significantly enhance the functionality, maintainability, and scalability of software systems in critical domains like healthcare.