**HOSPITAL MANAGEMENT SYSTEM**

(TEAM HMS PRIME)

SRM University – AP, Andhra Pradesh

**Bachelor of Technology/Master of Technology**

In

**Computer Science and Engineering**

**School of Engineering and Sciences**



Under the Guidance of

**(Karnena Kavitha Rani)**

**Submitted by**

|  |  |  |
| --- | --- | --- |
| NAME | REG NO | BRANCH & CLASS |
| C. Mahendra Kumar | AP23110010328 | Btech CSE E |
| R. Sumanaswi | AP23110010300 | Btech CSE E |
| D. Sri Medha | AP23110010257 | Btech CSE E |
| T. Surya | AP23110010323 | Btech CSE E |

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

[November]

# Certificate

Date: 16-Nov-24

This is to certify that the work present in this Project entitled **“Hospital Management System”** has been carried out by **“Team HMS Prime”** under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in **School of Engineering and Sciences**.

**Supervisor**

Prof. Kavitha rani karnena

Assistance Professor.

Department of Computer Science.

**Acknowledgements**

This C++ Acknowledgments section sample was created especially for this Hospital Management System. In this section, contributors, tools, and resources that were crucial to the project's development are thanked.   
  
**Recognitions**  
  
The extraordinary assistance, direction, and resources of numerous people and tools were essential to the successful development of this Hospital Management System (HMS). We are appreciative of the contributions that enabled this C++ project, which has proven to be a fruitful undertaking.   
  
**1. Educators and Mentors**  
We would like to express our sincere gratitude to the instructors and mentors who helped us understand the fundamentals of object-oriented design and C++ programming. This project has greatly benefited from their knowledge of software development, project management, and debugging.   
  
**2. Normal Tools and Libraries**

The C++ Standard Library served as the project's cornerstone, offering crucial data structures and functions that made data management, error correction, and user interaction easier. Specifically:  
Commonly used libraries and functions were included in to facilitate effective program development.  
Components from the C++ Standard Template Library (STL), such as `vector` and `string`, allowed for dynamic data manipulation and storage, which is essential for efficiently managing patients, physicians, and therapies.  
Error management was enhanced via {\stdexcept>`and custom exception handling, which made the system more reliable and intuitive.  
  
**3. Code Contributors and Development Team**We appreciate all of the team members' efforts and dedication to this project. Each person worked on various aspects of the code, including scheduling, billing, error handling, and maintaining patient and physician information. Their cooperation and commitment have been pivotal in creating a cohesive and fully functional system.

**4. Open-Source Documentation and Inspiration**   
We also thank the open-source projects and the comprehensive C++ documentation that served as references and sources of inspiration for several of the features and design components. We were able to implement industry-standard procedures thanks to these resources, which increased the system's scalability and efficiency.

**Table of Contents**

Certificate

Acknowledgements

Table of Contents

Abstract

List of Tables

1. Patient
2. Doctor
3. Appointment
4. Bill

List of Figures

Fig 1: "Hospital Management System Functional Flow”

Fig 2: "Hospital Management System Module”

1. Introduction
   1. Hospital Management System’s Objective
   2. Range
   3. Goals
2. Methodology

2.1. Analysis of Requirements

2.2 Design of the System

2.2.1 Structure of Classes

2.2.2 Operations and Functions

2.3 Storage of Data

2.3.1 Dynamic Storage using Vectors

2.3.2 Verification of Data

2.4 Design that is extensible and modular

1. Discussion
2. HMS Prime Code
3. Input And Output
4. Concluding Remarks
5. Future Work

References

**Abstract**

This project uses C++ to construct a Hospital Management System (HMS) to manage patients, physicians, appointments, treatments, and billing in a more efficient manner. Key entities in the healthcare process are represented by the many classes that make up the system: `Person`, `Patient`, `Doctor`, {Appointment`, {Treatment`, and `Billing`.   
The `Person` class is a base class that shares attributes with the `name` and `id` classes. Other classes include Person, Patient, and Doctor. With altered display methods to present distinct information, the `Patient` and `Doctor` classes are descended from `Person`.   
  
**Appointment Class**: Uses `Patient`, `Doctor`, and `dateTime` properties to manage doctor-patient appointments.   
  
**Therapy and Billing Classes**: The `Billing` class aggregates treatments, determines the overall cost, and presents a summary, whereas the `Treatment` class specifies specifics for each therapy.   
  
**HMSException Class:** Manages system-specific exceptions.

**Functions and Main Menu:** Dynamic data entry and storage are supported via functions like `addPatient`, `addDoctor`, `addAppointment`, and `addTreatment`. Users may successfully manage appointments and billing summaries by interacting with the system through the primary menu interface.

In a hospital setting, this HMS software facilitates effective data processing for patient records, appointments, and financial transactions by offering users an interactive, console-based interface.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S. No** | **Column\_name** | **Data Type** | **Constraints** | **Description** |
| 1. | P\_ID | Varchar (50) | Primary Key | Contains unique ID for each patient |
| 2. | Name | Varchar (50) | - | Contains the name of the patient |

**List of Tables**

**TABLE 1 – Patient**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S. No** | **Column\_name** | **Data Type** | **Constraints** | **Description** |
| 1. | D\_ID | Varchar (50) | Primary Key | Contains unique ID for each doctor |
| 2. | Name | Varchar (50) | - | Contains the name of the doctor |

**TABLE 2 – Doctor**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No** | **Column\_name** | **Data Type** | **Constraints** | **Description** |
| 1. | P\_ID | Varchar(50) | Foreign Key | Links to unique patient ID in the patient table |
| 2. | D\_ID | Varchar(50) | Foreign Key | Links to unique doctor ID in the doctor table |
| 3. | Date | Date | - | Contains date for appointment |
| 4. | Time | Time | - | Contains time for appointment |

**TABLE 3 – Appointment**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No** | **Column\_name** | **Data Type** | **Constraints** | **Description** |
| 1. | Treatment\_ID | Varchar(50) | Foreign Key | Contains unique Identifier for each treatment |
| 2. | Cost | Varchar(50) | Foreign Key | Contains the cost of each treatment |
| 3. | Total\_Bill | Date | - | Contains the total amount of the bill |

**TABLE 4 – Bill**

**List of figures**

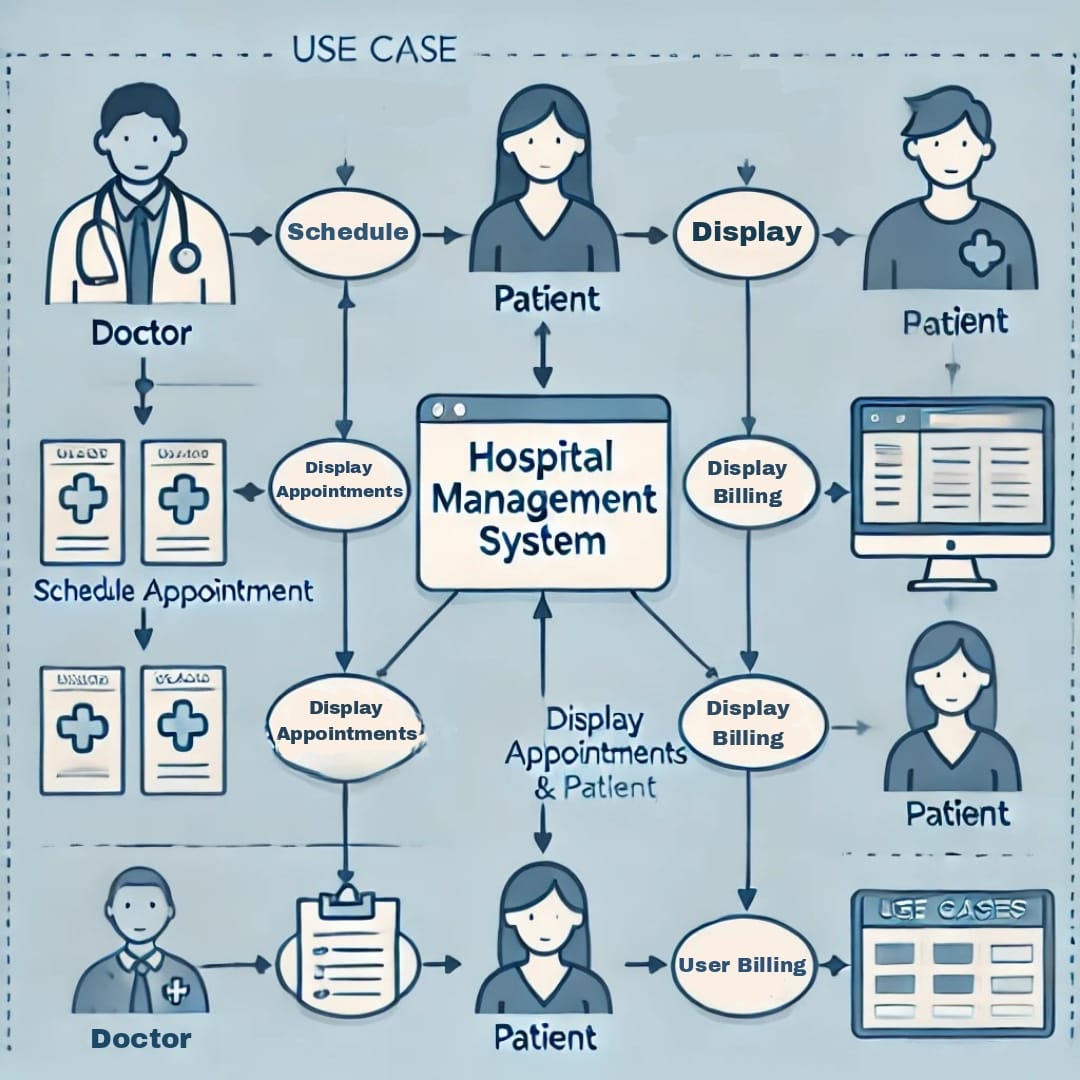


Fig 1: "Hospital Management System Functional Flow”

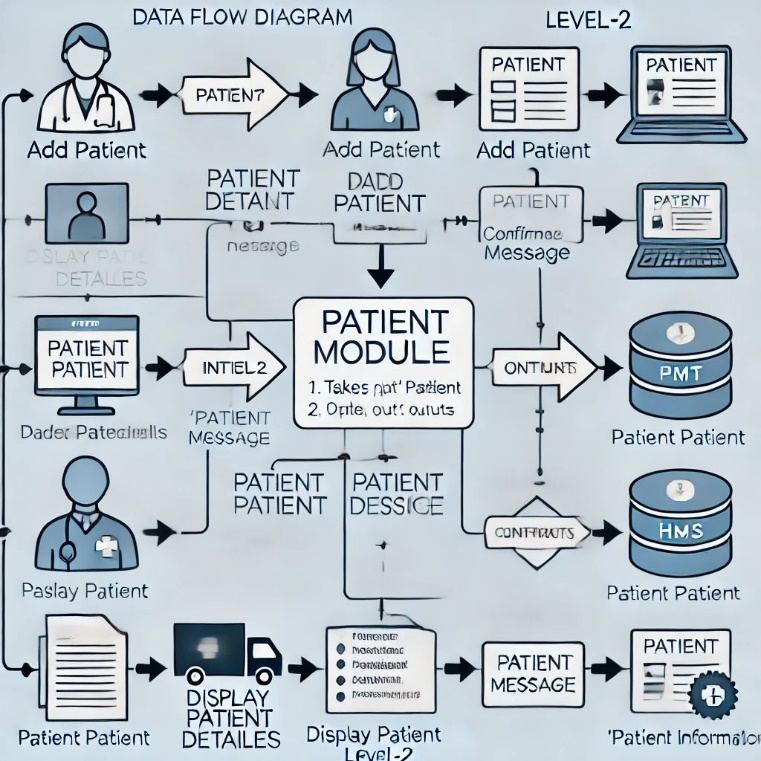


Fig 2: "Hospital Management System Module”

# Introduction

An important piece of software called the Hospital Management System (HMS) was created to automate and simplify a number of operational and administrative duties in hospitals. Hospital employees may more effectively handle patient data, doctor appointments, treatment plans, billing, and other crucial healthcare processes with the use of a well-implemented HMS. The total quality of healthcare services is improved, accuracy is increased, and manual labor is greatly reduced when a hospital management system is used.

The Hospital Management System (HMS), created in C++ and utilizing object-oriented programming (OOP) concepts to guarantee a modular, scalable, and maintainable system, is the subject of this paper. Core features of the system include billing, treatment logging, appointment scheduling, and patient and physician administration. In order to guarantee seamless operation and resilience, it also incorporates elements like error management and user interaction.

This system's code is straightforward but effective, providing a straightforward and effective method of hospital administration. It draws attention to the advantages of utilizing C++ for system development because of its flexibility, performance, and capacity to handle intricate data structures.

## 1.1 The Hospital Management System's Objective

The HMS's main goal is to provide a digital alternative to spreadsheets or paper-based approaches for hospital information management. By keeping vital patient data readily available and assisting medical personnel in providing prompt, efficient care, this system assists hospitals in keeping correct and current records.

## 1.2 Range

Designed as a command-line application, this system is appropriate for small to medium-sized hospitals to manage patient records. It is made to offer necessary features without necessitating database integration or a sophisticated graphical user interface. As the hospital expands, records may be easily added thanks to the present version's use of C++ vectors for dynamic data storage.

## 1.3 Goals

1. To make it possible for patient records to be added, updated, viewed, and deleted efficiently.
2. to prevent duplicate patient IDs and validate input in order to guarantee data integrity.
3. to offer a straightforward and user-friendly interface that enables hospital employees to handle records with little technical expertise.
4. to create an extensible and modular codebase that can be expanded with new features in later iterations.

# 2. Methodology

The Hospital Management System (HMS) is a modular, scalable, and user-friendly application that was developed using an organized process and object-oriented programming concepts in C++. In order to guarantee functional accuracy and data integrity, this technique describes the system's architecture, data storage, and validation strategy.

## 2.1. Analysis of Requirements

Understanding the needs of a normal hospital system was the first step in the technique, with an emphasis on routine tasks including keeping track of treatments, managing patient data, making doctor's appointments, and creating bills.

In order for hospital employees to quickly complete daily operations including adding patients, assigning doctors, scheduling appointments, documenting treatments, and creating billing summaries, the system needed to be easy to use.

## 2.2 Design of the System

A class-based architecture is used by the HMS, with distinct classes standing in for essential elements like as Person, Patient, Doctor, Appointment, Treatment, and Billing. Effective data management and interaction are made possible by the pertinent properties and methods that each class encapsulates.

### 2.2.1 Structure of Classes

**1. Class of Person**

* Acts as a base class abstraction for both patients and doctors.
* Includes standard properties such as id and name.
* To ensure implementation in derived classes, a pure virtual show() function is defined.

1. **Classes of Patients and Doctors:**

These classes, which are derived from the Person class, offer particular implementations of the show() function for presenting patient and physician information.

1. **Class of Appointment:**

* Symbolizes a connection between a patient, a doctor, and a time and date that has been set.
* It has characteristics for storing a string indicating the date and time of the appointment, a Patient object, and a Doctor object.
* Offers the display() function for displaying appointment information.

1. **Class of Treatment:**

* Depicts a medical procedure and includes details like the description and price.
* Contains instructions for retrieving the cost and displaying treatment details.

1. **Class of Billing:**

* Uses a vector to manage a group of Treatment objects.
* Offers ways to present a thorough billing summary, compute the overall cost, and add treatments

### 2.2.2 Operations and Functions

The HMS provides a number of features to efficiently manage hospital data, including:

**Add Doctor and Add Patient:** These features allow you to add new records for both patients and doctors by gathering user input and producing matching objects that are kept in different vectors.

**Schedule Appointment:** By confirming both parties' IDs and generating an Appointment object, this feature pairs a patient with a doctor for a given day and time.

**Add Treatment:** Gathers treatment information from the user and adds a new treatment to the billing system.

**Display Appointments**: To list all of the appointments that are scheduled, iterate over the vector of Appointment objects and invoke its display() method.

**Display Billing:** Shows a thorough billing summary that includes the overall cost and a breakdown of treatments.

## 2.3 Storage of Data

The std::vector data structure is used by the HMS to store and handle records in a dynamic manner. This method guarantees adaptability in managing varying volumes of records, allowing the system to grow in accordance with the hospital's needs.

### 2.3.1 Dynamic Storage using Vectors

The following benefits come from using vectors:

**Dynamic Sizing:** Removes the need for preset restrictions by automatically resizing as records are added or withdrawn.

**Effective Access and Modification:** Enables fast retrieval and updates by providing consistent time complexity for accessing items.

### 2.3.2 Verification of Data

Mechanisms for data validation are used to preserve the system's dependability and integrity:

**ID validation**: Prevents duplication by confirming that patient and physician IDs are distinct.

Verifies that user-provided data (such as price, date, and time) adheres to acceptable boundaries and formats.

**Error Handling:** Specific runtime faults are handled using custom exceptions (HMSException), which offer helpful feedback when an operation is invalid.

## 2.4 Design that is extensible and modular

Every class and method in the modular HMS codebase has a distinct purpose. Future improvements like adding a graphical user interface (GUI) or connecting a database are supported by this structure, which also makes maintenance simple.

**3.Discussion**

• The Hospital Management System (HMS) developed in C++ offers a comprehensive solution for managing key hospital operations, including patient data, doctor schedules, treatment tracking, and billing.

• It leverages object-oriented programming principles for modularity, reusability, and scalability.

• The system uses vectors for dynamic memory management and efficient sequential access of data.

• Appointment management allows scheduling and linking patients, doctors, and appointment times.

• Treatment and billing are handled with cost calculation and display functionality.

• The console-based user interface is simple and interactive but lacks the user-friendly experience of modern GUIs.

• Basic exception handling with the HMSException class provides meaningful error reporting and graceful failure.

• Potential improvements include enhancing data structures for faster lookups, expanding appointment features, incorporating advanced billing features, developing a user-friendly GUI, and implementing more comprehensive exception handling.

• The provided solution suggests expanding the exception handling to encompass a wider range of errors, including invalid input, out-of-range dates, and other exceptional scenarios.

• Moving forward, the system's scalability and future enhancements come into focus.

• While it effectively manages fundamental hospital operations, several areas offer opportunities for improvement.

• Integrating a database management system (DBMS) would enhance scalability, ensuring data persistence, simplified querying, and robust reporting capabilities.

• Advanced features such as patient medical records, doctor specialization, online appointment scheduling, payment processing, and reporting tools could be incorporated to make the system more comprehensive.

• Furthermore, implementing user roles and access controls would address security concerns, ensuring appropriate authorization and safeguarding sensitive information in a real-world application.

**4.Code**

#include <bits/stdc++.h>

using namespace std;

class Person {

protected:

string name;

int id;

public:

Person(string name, int id) : name(name), id(id) {}

virtual void display() const = 0;

int getId() const { return id; }

};

class Patient : public Person {

public:

Patient(string name, int id) : Person(name, id) {}

void display() const override {

cout << "Patient Name: " << name << ", ID: " << id << endl;

}

};

class Doctor : public Person {

public:

Doctor(string name, int id) : Person(name, id) {}

void display() const override {

cout << "Doctor Name: " << name << ", ID: " << id << endl;

}

};

class Appointment {

Patient patient;

Doctor doctor;

string dateTime;

public:

Appointment(Patient p, Doctor d, string dt) : patient(p), doctor(d), dateTime(dt) {}

void display() const {

cout << "Appointment for ";

patient.display();

cout << " with ";

doctor.display();

cout << " on " << dateTime << endl;

}

};

class Treatment {

string description;

double cost;

public:

Treatment(string desc, double c) : description(desc), cost(c) {}

double getCost() const {

return cost;

}

void display() const {

cout << "Treatment: " << description << ", Cost: $" << fixed << setprecision(2) << cost << endl;

}

};

class Billing {

vector<Treatment> treatments;

public:

void addTreatment(const Treatment& treatment) {

treatments.push\_back(treatment);

}

double calculateTotal() const {

double total = 0;

for (const auto& treatment : treatments) {

total += treatment.getCost();

}

return total;

}

void displayBill() const {

cout << "\nBilling Summary:\n";

for (const auto& treatment : treatments) {

treatment.display();

}

cout << "Total Bill: $" << fixed << setprecision(2) << calculateTotal() << endl;

}

};

class HMSException : public runtime\_error {

public:

HMSException(const string& message) : runtime\_error(message) {}

};

void addPatient(vector<Patient>& patients) {

string name;

int id;

cin.ignore(numeric\_limits<streamsize>::max(), '\n'); // Clear input buffer

cout << "Enter Patient Name: ";

getline(cin, name);

cout << "Enter Patient ID: ";

cin >> id;

patients.emplace\_back(name, id);

cout << "Patient added successfully.\n";

}

void addDoctor(vector<Doctor>& doctors) {

string name;

int id;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Enter Doctor Name: ";

getline(cin, name);

cout << "Enter Doctor ID: ";

cin >> id;

doctors.emplace\_back(name, id);

cout << "Doctor added successfully.\n";

}

void addAppointment(const vector<Patient>& patients, const vector<Doctor>& doctors, vector<Appointment>& appointments) {

int patientId, doctorId;

string dateTime;

cout << "Enter Patient ID for appointment: ";

cin >> patientId;

cout << "Enter Doctor ID for appointment: ";

cin >> doctorId;

cout << "Enter Appointment Date and Time (YYYY-MM-DD HH:MM AM/PM): ";

cin.ignore();

getline(cin, dateTime);

const Patient\* selectedPatient = nullptr;

const Doctor\* selectedDoctor = nullptr;

for (const auto& p : patients) {

if (p.getId() == patientId) {

selectedPatient = &p;

break;

}

}

for (const auto& d : doctors) {

if (d.getId() == doctorId) {

selectedDoctor = &d;

break;

}

}

if (selectedPatient && selectedDoctor) {

appointments.emplace\_back(\*selectedPatient, \*selectedDoctor, dateTime);

cout << "Appointment scheduled successfully.\n";

} else {

cout << "Invalid Patient or Doctor ID.\n";

}

}

void addTreatment(Billing& bill) {

string description;

double cost;

cin.ignore(numeric\_limits<streamsize>::max(), '\n'); // Clear input buffer

cout << "Enter Treatment Description: ";

getline(cin, description);

cout << "Enter Treatment Cost: ";

cin >> cost;

bill.addTreatment(Treatment(description, cost));

cout << "Treatment added successfully.\n";

}

int main() {

vector<Patient> patients;

vector<Doctor> doctors;

vector<Appointment> appointments;

Billing bill;

int choice;

do {

cout << "\n--- Hospital Management System ---\n";

cout << "1. Add Patient\n";

cout << "2. Add Doctor\n";

cout << "3. Schedule Appointment\n";

cout << "4. Add Treatment\n";

cout << "5. Display Appointments\n";

cout << "6. Display Billing\n";

cout << "7. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: addPatient(patients); break;

case 2: addDoctor(doctors); break;

case 3: addAppointment(patients, doctors, appointments); break;

case 4: addTreatment(bill); break;

case 5:

for (const auto& appt : appointments) {

appt.display();

}

break;

case 6: bill.displayBill(); break;

case 7: cout << "Exiting system.\n"; break;

default: cout << "Invalid choice. Please try again.\n"; break;

}

} while (choice != 0);

return 0;

}

**5.Input And Output**

--- Hospital Management System ---

1. Add Patient

2. Add Doctor

3. Schedule Appointment

4. Add Treatment

5. Display Appointments

6. Display Billing

7. Exit

Enter your choice: 1

Enter Patient Name: SUMA

Enter Patient ID: 123

Patient added successfully.

--- Hospital Management System ---

1. Add Patient

2. Add Doctor

3. Schedule Appointment

4. Add Treatment

5. Display Appointments

6. Display Billing

7. Exit

Enter your choice: 2

Enter Doctor Name: MEDHA

Enter Doctor ID: 345

Doctor added successfully.

--- Hospital Management System ---

1. Add Patient

2. Add Doctor

3. Schedule Appointment

4. Add Treatment

5. Display Appointments

6. Display Billing

7. Exit

Enter your choice: 3

Enter Patient ID for appointment: 123

Enter Doctor ID for appointment: 345

Enter Appointment Date and Time (YYYY-MM-DD HH:MM AM/PM): 2006-06-12 10:10 AM

Appointment scheduled successfully.

--- Hospital Management System ---

1. Add Patient

2. Add Doctor

3. Schedule Appointment

4. Add Treatment

5. Display Appointments

6. Display Billing

7. Exit

Enter your choice: 4

Enter Treatment Description: FULL BODY CHECKUP

Enter Treatment Cost: 10000

Treatment added successfully.

--- Hospital Management System ---

1. Add Patient

2. Add Doctor

3. Schedule Appointment

4. Add Treatment

5. Display Appointments

6. Display Billing

7. Exit

Enter your choice: 5

Appointment for Patient Name: SUMA, ID: 123

with Doctor Name: MEDHA, ID: 345

on 2006-06-12 10:10 AM

--- Hospital Management System ---

1. Add Patient

2. Add Doctor

3. Schedule Appointment

4. Add Treatment

5. Display Appointments

6. Display Billing

7. Exit

Enter your choice: 6

Billing Summary:

Treatment: FULL BODY CHECKUP, Cost: $10000.00

Total Bill: $10000.00

--- Hospital Management System ---

1. Add Patient

2. Add Doctor

3. Schedule Appointment

4. Add Treatment

5. Display Appointments

6. Display Billing

7. Exit

Enter your choice: 7

Exiting system.

--- Hospital Management System ---

1. Add Patient

2. Add Doctor

3. Schedule Appointment

4. Add Treatment

5. Display Appointments

6. Display Billing

7. Exit

Enter your choice: 8

Invalid choice. Please try again.

# 6. Concluding Remarks

The C++-developed Hospital Management System (HMS) is a straightforward but efficient way to oversee important hospital functions. The system arranges hospital data and procedures in a structured, modular, and scalable way by applying object-oriented programming concepts like inheritance, polymorphism, and encapsulation. The system provides a basic method for mimicking actual healthcare administration duties by effectively managing patients, physicians, appointments, treatments, and invoicing.

In order to maintain flexibility as the number of entities (patients, physicians, and therapies) increases, the code makes use of vectors for dynamic data storage. Appointment scheduling and treatment cost calculation are made simple by the Appointment and Billing classes, which offer controllable and transparent interactions between entities. Furthermore, input validation and simple exception handling give the system an extra degree of resilience, guaranteeing that mistakes made by users are identified and dealt with properly.

Nevertheless, the system is still in its infancy and might be improved in a number of areas:

**User Interface:** Although effective, the text-based terminal interface might be improved with a Graphical User Interface (GUI), particularly for hospital employees who might not be accustomed to command-line operations.

**Data Persistence:** By incorporating a database or file storage method, the system's current lack of permanent data storage could be fixed. This would make the system acceptable for long-term usage by allowing it to maintain data even after the program is closed.

**Advanced Features:** To represent a more complete hospital administration system, more features could be included, such as online appointment scheduling, insurance management, doctor specializations, and patient medical records.

**Scalability:** Although the current method works well for small to medium-sized businesses, managing vast amounts of hospital data may call for more effective data structures (such as databases or hashmaps) to maintain peak performance

.

To sum up, this hospital management system is a good place to start when developing a tool for healthcare administration. It demonstrates how fundamental C++ ideas may be used to construct important functionality, with room to grow to meet the demands of an actual hospital. The existing implementation offers a solid platform upon which to expand, but additional improvements and feature additions will be required to keep up with changing requirements as healthcare systems grow more sophisticated.

# 7. Future Work

• The current Hospital Management System (HMS) in C++ provides basic functionalities for patient registration, doctor appointments, treatment tracking, and billing.

• Future improvements include database integration for persistent data storage and efficient querying, development of a user-friendly Graphical User Interface (GUI), implementation of user authentication and role-based access control for enhanced security, advanced billing features such as discounts, payment plans, insurance management, and invoice generation.

• Additionally, expanding the system to include detailed patient medical records management, enhancing appointment management with features like status tracking, reminders, calendar integration, and online scheduling, and generating reports for management, financial planning, and medical analysis are important areas for future development.

• A mobile application integration, multilingual support, and cloud integration for remote access, scalability, and data backup are also potential avenues for future work.

• These enhancements will transform the HMS into a comprehensive and robust system that meets the demands of a real-world hospital setting.

• The HMS can be transformed into a more comprehensive, user-friendly, and scalable system by incorporating enhancements that cater to the requirements of both medical professionals and patients, thereby optimizing hospital operations and elevating the overall standard of healthcare service delivery.

# References

* Thinking in C++ by Bruce Eckel, Pearson, Second Edition, Volume 1, 2002.
* Lafore, Robert. Course Sams Publishing, Fourth Edition, 2001. Object-Oriented Programming in C++.
* Stroustrup, Bjarne. Addison-Wesley Professional, Fourth Edition, 2013. The C++ Programming Language.
* The C++ Standard Library: A Tutorial and Reference by Nicolai M. Josuttis, Addison-Wesley, Second Edition, 2012.