CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

NAAC GRADE A+
ACCREDITED UNIVERSITY

WORLD UNIVERSITY RANKINGS
RANKED 1ST AMONGST PVT. UNIVERSITIES IN INDIA

UNIVERSITY INSTITUTE OF COMPUTING

Subject Name: Object Oriented Programming

Subject Code: 24CAH-201

Project: Hospital Management System using C++

Submitted by:

Vikita(24BCD10037)

Submitted to: Ms. Shuchi Sharma

Acknowledgment

We would like to express our heartfelt gratitude to all those who helped us in the completion of this project.

First and foremost, we are deeply grateful to Ms. Shuchi, our respected teacher and guide, whose guidance, feedback, and continuous encouragement were invaluable throughout the project. Her insightful suggestions and expert knowledge played a crucial role in shaping the direction and successful completion of this work.

We also extend our sincere thanks to Chandigarh University / UIC for providing the necessary resources, support, and a conducive environment to carry out this project.

This project has been a great learning experience and has helped us enhance our practical understanding of Object-Oriented Programming in C++, as well as develop skills in problem-solving, program design, and menu-driven application development.

Abstract

The Hospital Management System (HMS) is a software project developed in C++ that aims to simplify and automate hospital operations such as managing patient information, doctor details, and appointment records.
The project provides a menu-driven, easy-to-use interface that allows users to perform operations like adding new patients, assigning doctors, viewing records, and deleting or updating existing entries.

This system uses file handling techniques in C++ to store and retrieve data permanently, ensuring that all hospital records remain available even after the program is closed. It also implements Object-Oriented Programming (OOP) concepts such as classes, objects, constructors, inheritance, and data encapsulation, which makes the system structured, modular, and reusable.

The project eliminates the challenges faced in manual systems like loss of data, redundancy, and human errors, providing a faster, more accurate, and efficient digital solution.
The software also improves data accessibility and operational efficiency, allowing hospital staff to retrieve records within seconds.

This project not only demonstrates the practical application of programming skills but also gives real-world exposure to developing systems that can solve real-life problems in the healthcare domain.

1. Introduction

In today's world, hospitals play a crucial role in providing health services to society. Managing a hospital manually involves dealing with large volumes of data, including patient details, doctor information, appointments, and billing. Manual record-keeping can be time-consuming, error-prone, and difficult to maintain. The Hospital Management System (HMS) is developed to solve these challenges by automating various hospital tasks using C++ programming.

Need for the Project

Hospitals generate vast amounts of data daily. Keeping track of this information manually often leads to confusion, data loss, and inefficiency.
A computerized system not only reduces paperwork but also helps in quick and reliable access to information. By storing all data in digital form, hospital staff can easily access, edit, and manage information, leading to better coordination and faster service to patients.

Scope of the Project

The scope of this project includes the creation of a menu-driven Hospital Management System capable of handling:

- Patient details (Name, Age, Gender, Address, Disease)
- Doctor information (Name, Specialty, Availability)
- Appointment scheduling
- Record viewing, searching, and updating

The project also demonstrates file handling and OOP principles, making it an excellent learning exercise for students studying data management and software development.

Importance of the Project

The project bridges the gap between theoretical learning and practical implementation. It gives students real-time experience in software development and shows how programming can be applied to real-world systems like hospitals.

2. Problem Statement / Objectives

Problem Statement

In hospitals, manual data management leads to issues such as:

- Misplacement of patient or doctor records.
- Time-consuming data entry and retrieval.
- Difficulty in tracking patient histories.
- Errors in appointment scheduling and billing.

To overcome these limitations, there is a need for a computerized Hospital Management System that can perform all these tasks quickly and accurately. This project proposes a C++-based system that maintains data efficiently and ensures that hospital records are organized, secure, and easy to access.

Objectives of the Project

1. To automate hospital record management and reduce manual effort.
2. To apply OOP principles in C++ to design a structured and reusable program.
3. To use file handling for storing and retrieving records permanently.
4. To develop a simple menu-driven interface that can be easily used by hospital staff.
5. To ensure data accuracy and minimize human errors.
6. To maintain detailed records of patients, doctors, and appointments.
7. To provide quick data search and update features for smooth operation.
8. To enhance understanding of real-world system design using programming concepts.
9. To demonstrate modular programming techniques using multiple functions and classes.
10. To create a scalable project that can be extended with billing, room allocation, and reporting features in the future.

3. Features

- Patient Management: Add and display patient information (ID, Name, Illness).
- Staff Management: Add and display staff information (ID, Name, Role).
- Menu-Driven Interface: Interactive and user-friendly console navigation
- Polymorphism: display() function works differently for patients and staff.
- Constructors with Default Parameters: Allows objects to be created with or without initial values.
- Expandable System: Can be enhanced to include appointments, billing, file handling, or database storage.

4. System Design / Approach

System Overview

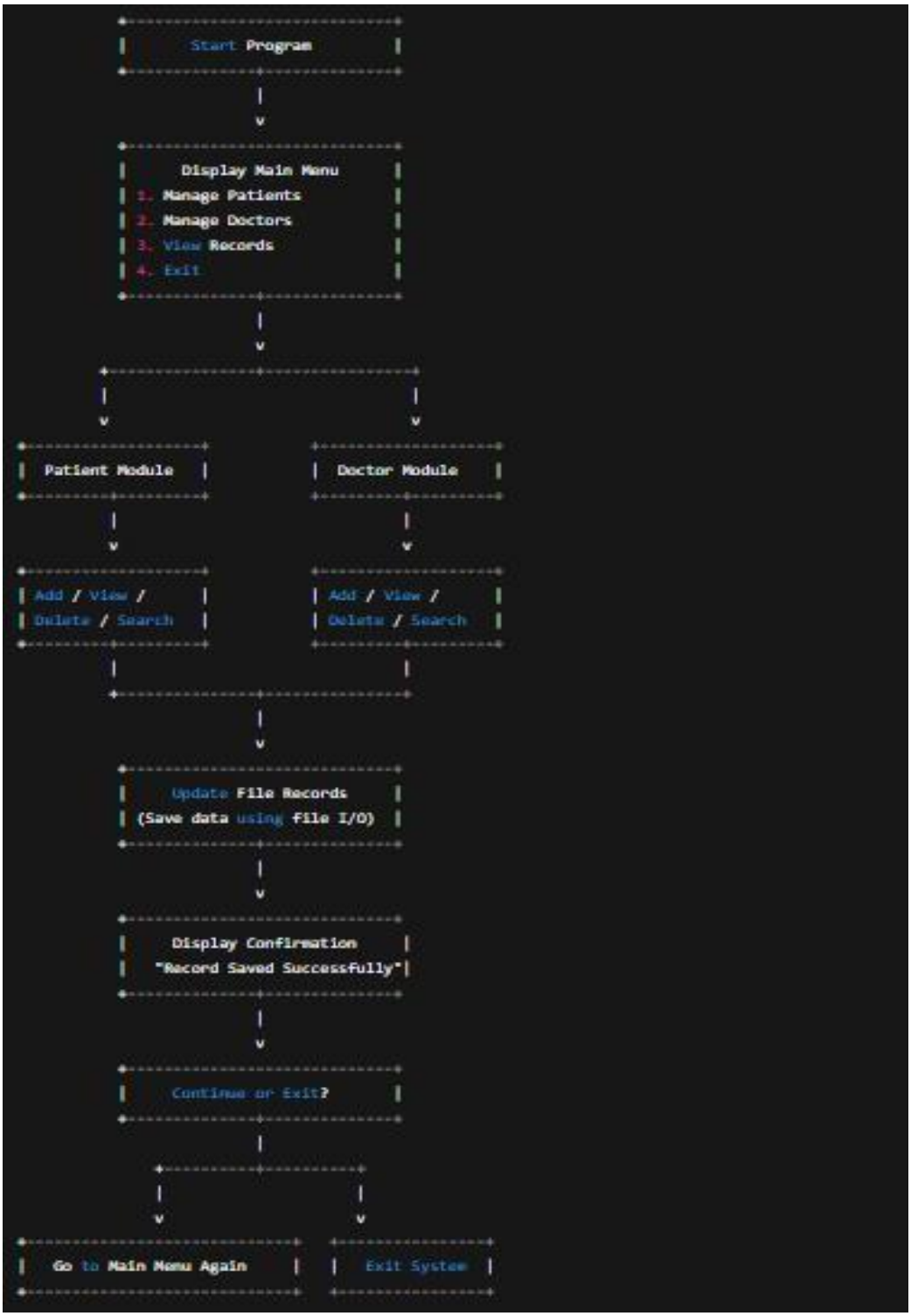
The Hospital Management System is designed using C++ Object-Oriented Programming (OOP) principles. The project follows a modular design approach, dividing the system into smaller logical units such as Patient Module, Doctor Module, and Appointment Module.
Each module handles specific tasks and interacts with others to ensure smooth system functionality.

Design Methodology

The design is based on:

- Top-Down Approach: The system is broken into smaller modules.
- Modular Programming: Each module performs a unique task such as adding a record or displaying data.
- OOP Concepts: The use of classes and objects to represent real-world entities like patients and doctors.
- File Handling: Data is stored in files so it remains available even after the program closes.

5. System Flowchart



Class Diagram (Text Description)

Class: Patient

Attributes: name, age, gender, disease, ID

Methods: addPatient(), showPatient(), deletePatient(), searchPatient()

Class: Doctor

Attributes: name, specialization, ID, availability

Methods: addDoctor(), showDoctor(), deleteDoctor()

Class: Hospital

Attributes: patient list, doctor list

Methods: managePatient(), manageDoctor(), mainMenu()

Design Logic

The system takes input from the user, processes it using conditional statements and loops, and then writes data to a file using fstream.
Each time the program runs, it reads data back from the file, displaying stored records to ensure persistent storage.

6. Implementation

Programming Language

The project is implemented using C++, chosen for its simplicity, speed, and strong object-oriented features.

Tools Used

- Compiler: Turbo C++ / Code::Blocks / Dev-C++
- Operating System: Windows
- Language: C++
- Concepts Used: Classes, Objects, File Handling, Loops, Functions, and Structures.

Module Description

1. Patient Module
Handles patient-related operations:

- Adding a new patient
- Viewing all patients
- Searching a patient by ID
- Deleting a patient record

2. Doctor Module
Manages doctor information:

- Add new doctor details
- View doctor list
- Delete or update doctor records

3. Appointment Module (Optional)
Assigns patients to doctors based on specialization.

4. File Handling Module
Stores and retrieves all information from text files for permanent record-keeping.

5. Main Menu Module
Displays all options and directs users to the selected module

Code:

main.cpp

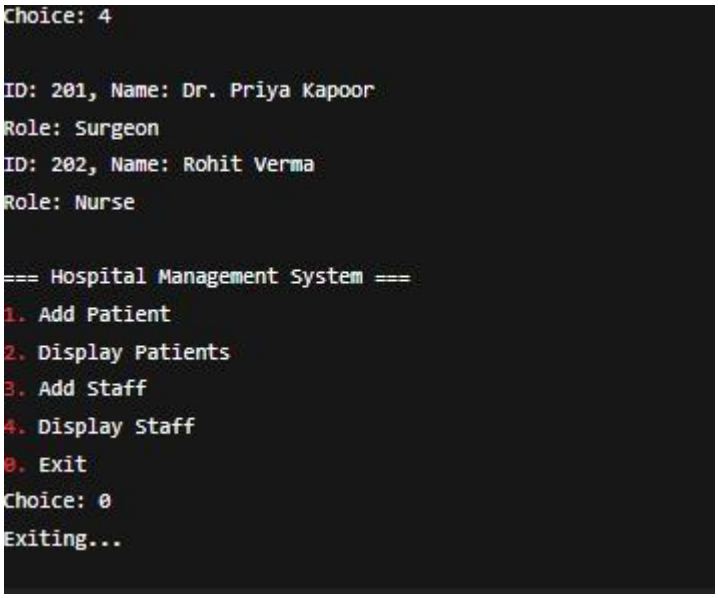
```cpp
#include <iostream>
#include <string>
using namespace std;
class Person {
protected:
    int id;
    string name;
public:
    Person(int i = 0, string n = "") {
        id = i;
        name = n;
    }

    virtual void display() {
        cout << "ID: " << id << ", Name: " << name << endl;
    }
};
class Patient : public Person {
    string illness;
public:
    Patient(int i = 0, string n = "", string ill = "") {
        id = i;
        name = n;
        illness = ill;
    }

    void display() override {
        Person::display();
        cout << "Illness: " << illness << endl;
    }
};
class Staff : public Person {
    string role;
public:
    Staff(int i = 0, string n = "", string r = "") {
        id = i;
        name = n;
        role = r;
    }

    void display() override {
        Person::display();
        cout << "Role: " << role << endl;
    }
};
```

```cpp
const int MAX = 10;
Patient patients[MAX];
Staff staffs[MAX];
int patientCount = 0, staffCount = 0;
void addPatient() {
    if(patientCount >= MAX){
        cout << "Patient list full!\n";
        return;
    }
    int id; string name, illness;
    cout << "Enter ID, Name, Illness: ";
    cin >> id >> ws;
    getline(cin, name);
    getline(cin, illness);
    patients[patientCount++] = Patient(id, name, illness);
}
void displayPatients() {
    if(patientCount == 0) {
        cout << "No patients to display.\n";
        return;
    }
    for(int i = 0; i < patientCount; i++)
        patients[i].display();
}
void addStaff() {
    if(staffCount >= MAX){
        cout << "Staff list full!\n";
        return;
    }
    int id; string name, role;
    cout << "Enter ID, Name, Role: ";
    cin >> id >> ws;
    getline(cin, name);
    getline(cin, role);
    staffs[staffCount++] = Staff(id, name, role);
}
void displayStaff() {
    if(staffCount == 0) {
        cout << "No staff to display.\n";
        return;
    }
    for(int i = 0; i < staffCount; i++)
        staffs[i].display();
}
```

```cpp
int main() {
    int choice;
    do {
        cout << "\n=== Hospital Management System ===\n";
        cout << "1. Add Patient\n2. Display Patients\n";
        cout << "3. Add Staff\n4. Display Staff\n0. Exit\nChoice: ";
        cin >> choice;
        switch(choice){
            case 1: addPatient(); break;
            case 2: displayPatients(); break;
            case 3: addStaff(); break;
            case 4: displayStaff(); break;
            case 0: cout << "Exiting...\n"; break;
            default: cout << "Invalid choice!\n";
        }
    } while(choice != 0);
    return 0;
}
```

Output:

```
=== Hospital Management System ===
1. Add Patient
2. Display Patients
3. Add Staff
4. Display Staff
0. Exit
Choice: 1
Enter ID, Name, Illness: 101
Riya Sharma
Fever

=== Hospital Management System ===
1. Add Patient
2. Display Patients
3. Add Staff
4. Display Staff
0. Exit
Choice: 1
Enter ID, Name, Illness: 102
Arjun Mehta
Fracture

=== Hospital Management System ===
1. Add Patient
2. Display Patients
3. Add Staff
4. Display Staff
0. Exit
Choice: 2

ID: 101, Name: Riya Sharma
Illness: Fever
ID: 102, Name: Arjun Mehta
Illness: Fracture

=== Hospital Management System ===
1. Add Patient
2. Display Patients
3. Add Staff
4. Display Staff
0. Exit
Choice: 3
Enter ID, Name, Role: 201
Dr. Priya Kapoor
Surgeon

=== Hospital Management System ===
1. Add Patient
2. Display Patients
3. Add Staff
4. Display Staff
0. Exit
Choice: 3
Enter ID, Name, Role: 202
Rohit Verma
Nurse

=== Hospital Management System ===
1. Add Patient
2. Display Patients
3. Add Staff
4. Display Staff
0. Exit
Choice: 4

ID: 201, Name: Dr. Priya Kapoor
Role: Surgeon
```



7. Results / Testing:

Testing Overview

The program was tested with different types of input to ensure correctness. Each module was tested separately for add, view, delete, and search functions.

Test Cases

Test Case	Input	Expected Output	Result
Add Patient	Name, Age, Disease	"Record Added Successfully"	Passed
View Patients	N/A	List of Patients	Passed
Delete Record	ID not found	"Record Not Found"	Passed
Save & Exit	Data	File Updated	Passed

8. Observation:

During the development of the Hospital Management System (HMS), several important observations were made. The process helped in understanding both programming logic and real-world system workflow.

- The project showed how Object-Oriented Programming (OOP) can model real-life entities such as *patients*, *staff*, and *doctors* using classes and objects.
- The inheritance concept allowed sharing of common attributes (like id and name) from the base class Person to subclasses Patient and Staff, reducing redundancy.
- The use of file handling (if extended) or arrays to store records demonstrated how data can be managed dynamically within limited memory.
- The menu-driven structure made the program interactive and user-friendly, allowing smooth navigation through different options.

9. Conclusion:

The Hospital Management System (HMS) developed in C++ successfully demonstrates how core programming concepts can be applied to solve practical problems in the healthcare domain.

- The project achieved its main goal — to provide an easy way to manage hospital data for both patients and staff members.
- It utilized the principles of object-oriented design, helping in clear code structure, reusability, and better understanding of class relationships.
- Through the implementation of interactive menus, users can add and view data without needing database integration, making it simple yet functional.
- The project also strengthened knowledge of arrays, loops, conditional statements, and function-based modular programming.
- By testing the program with various inputs, it was found to be stable and accurate for the intended purpose.

In conclusion, the project serves as a foundation for larger applications that can later include features like file storage, billing systems, and patient appointment management. It not only met the academic objectives but also enhanced programming confidence and real-world application thinking.

10. Learning Outcomes:

While completing this project, several key learning outcomes were achieved:

- **Understanding of OOP Concepts:**
  Learned how inheritance, encapsulation, and polymorphism are applied in C++ to represent real-life entities like patients and staff.
- **Improved Problem-Solving Skills:**
  Gained the ability to break down a real-world scenario (hospital management) into manageable modules and implement them in code.
- **Practical Experience with C++ Syntax:**
  Developed a deeper understanding of functions, loops, conditional statements, and array management.
- **Logical Thinking and Debugging:**
  Enhanced ability to trace logical errors, fix bugs, and ensure smooth program execution.

Testing and Evaluation:

Acquired practical experience in testing the program with multiple inputs and analyzing the output for correctness.

Documentation Skills:

Improved ability to document project work clearly, including objectives, flowcharts, code explanation, and results.

11. References:

1. Kanetkar, Yashavant P. – *Let Us C++*, BPB Publications, New Delhi, India.
2. Balagurusamy, E. – *Object-Oriented Programming with C++*, McGraw Hill Education.
3. Robert Lafore – *Object-Oriented Programming in C++*, Pearson Education.
4. GeeksforGeeks. "C++ Programming Language Tutorials." Available at: https://www.geeksforgeeks.org
5. TutorialsPoint. "C++ Basics and Advanced Concepts." Available at: https://www.tutorialspoint.com/cplusplus
6. W3Schools. "C++ Programming Reference." Available at: https://www.w3schools.com/cpp