Bahria University, Islamabad
Department of Software Engineering

OBJECT ORIENTED PROGRAMMING LAB (SPRING 2025)

PROJECT REPORT

Submitted To: Engr. FAISAL ZIA

Submitted By:

Umar Farooq (- 088)

Date: 05/27/2025

Hospital Management System –

Declaration

I certify that this research work titled “Hospital Management System” is my own work. The work has not been presented elsewhere for assessment. The material that has been used from other sources has been properly acknowledged / referred.

Abstract

This project demonstrates the development of a Hospital Management System (HMS) using ObjectOriented Programming (OOP) principles in C++. The system simulates real-world hospital operations by providing interfaces for administrators, doctors, and patients. It includes functionalities for doctor management (hiring and sacking), patient-doctor appointments, ward admissions, and lab test assignments. The program leverages file handling for persistent storage and encapsulates core functionality in logically structured class hierarchies. It employs inheritance and polymorphism to facilitate code reuse and modularity. Designed as a console application, the project aims to enhance understanding of OOP concepts and data management techniques in real-time systems.

Keywords: Hospital Management, OOP in C++, Encapsulation, Inheritance, File Handling

Table of Contents

CHAPTER 1: INTRODUCTION

Hospital Management Systems (HMS) are integral to modern healthcare operations, improving administrative and clinical workflows through digitized processes. This C++ project aims to replicate core hospital management functionalities in a console-based application.

The system supports three user panels:

1. **Administrator Panel**: Hire/sack doctors, manage hospital statistics.
2. **Doctor Panel**: Manage patients, assign lab tests, ward beds.
3. **Patient Panel**: Schedule appointments, undergo lab tests.

The architecture employs OOP principles—encapsulation (e.g., private data in doctor and patient classes), inheritance (specialized doctor and patient types), and polymorphism (e.g., appointment handling). Persistent data storage is maintained through binary/text files (Doctor.DAT, Patient.DAT, etc.), allowing recovery of state across sessions.

By implementing this system, students explore real-world OOP design and software engineering practices in healthcare automation.

CHAPTER 2: METHODOLOGY

2.1 System Design

The application follows a modular design structured around classes defined in doctor.h and patient.h.

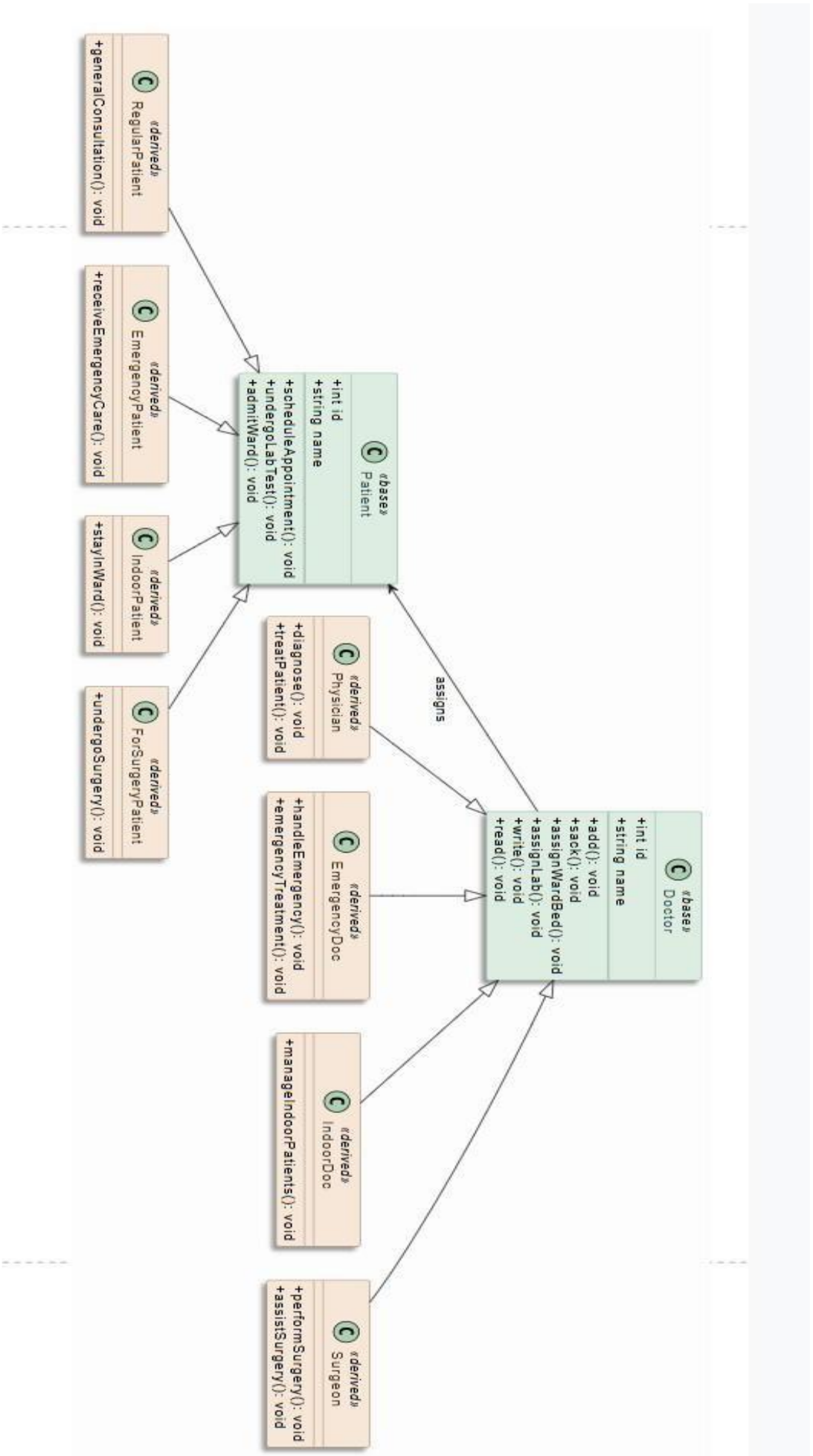
Core Classes and Relationships

- **Doctor (base)** ○ **Physician, Surgeon, EmergencyDoc, IndoorDoc (derived)**
- **Patient (base)** ○ **RegularPatient, EmergencyPatient, IndoorPatient, ForSurgeryPatient (derived)**

Relationships:

- Patients are assigned to doctors via **patToDocPatients().**
- Doctors manage beds (**assignWardBed()),** lab tests **(assignLab()),** and patient lists **(searchAllPatientsOfDoc()).**

🖼 *: Class Diagram of Hospital Management System*


«base»
Patient
+int id
+string name
+scheduleAppointment(): void
+undergoLabTest(): void
+admitWard(): void
«derived»
RegularPatient
+generalConsultation(): void
«derived»
EmergencyPatient
+receiveEmergencyCare(): void
«derived»
IndoorPatient
+stayInWard(): void
«derived»
ForSurgeryPatient
+undergoSurgery(): void
«derived»
Physician
+diagnose(): void
+treatPatient(): void
assigns
«base»
Doctor
+int id
+string name
+add(): void
+sack(): void
+assignWardBed(): void
+assignLab(): void
+write(): void
+read(): void
«derived»
EmergencyDoc
+handleEmergency(): void
+emergencyTreatment(): void
«derived»
IndoorDoc
+manageIndoorPatients(): void
«derived»
Surgeon
+performSurgery(): void
+assistSurgery(): void

Report -

2.2 Implementation

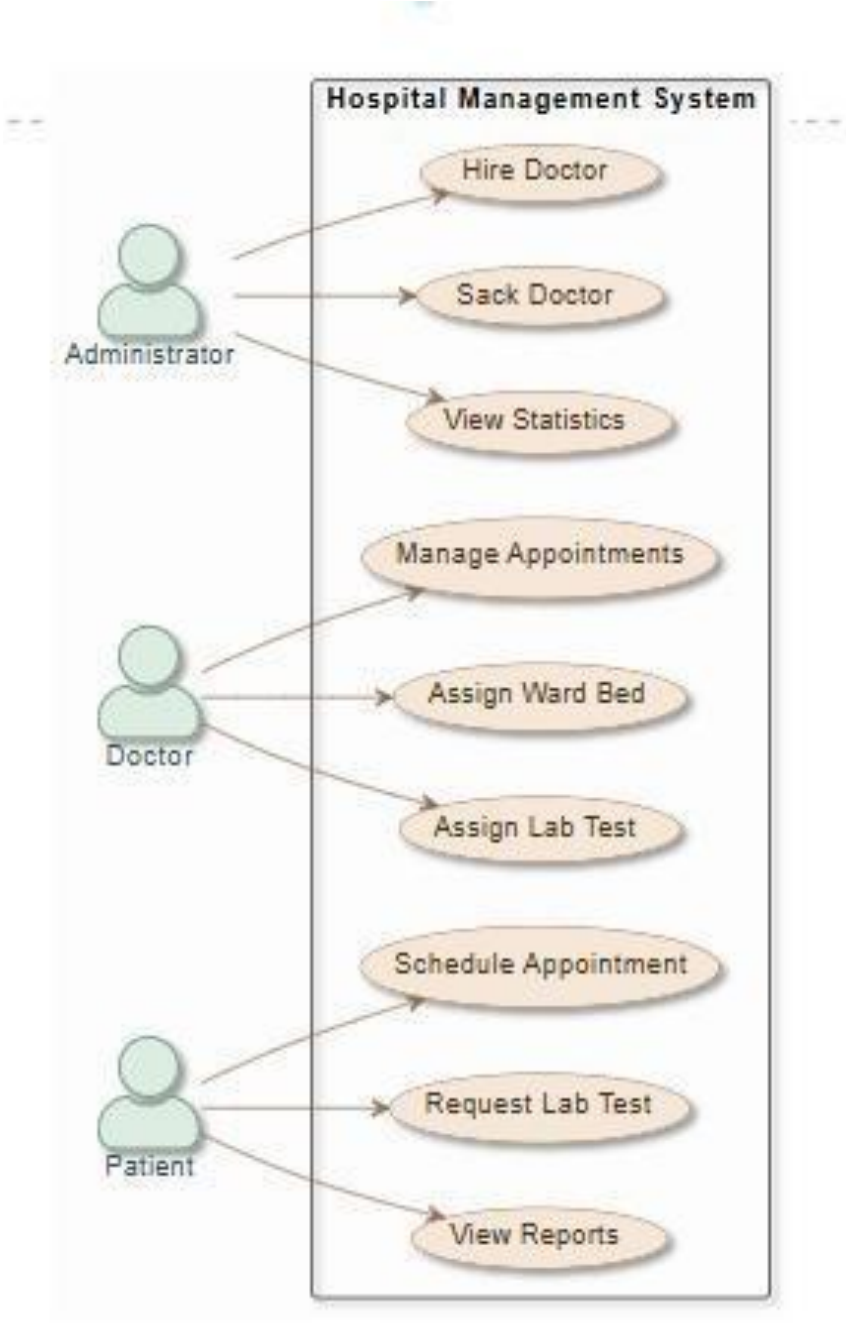
Implemented using C++ with core files:

- **main.cpp**: Handles UI and logic flow.
- **doctor.h:** Defines doctor-related classes and operations.
- **patient.h:** Defines patient types and operations.

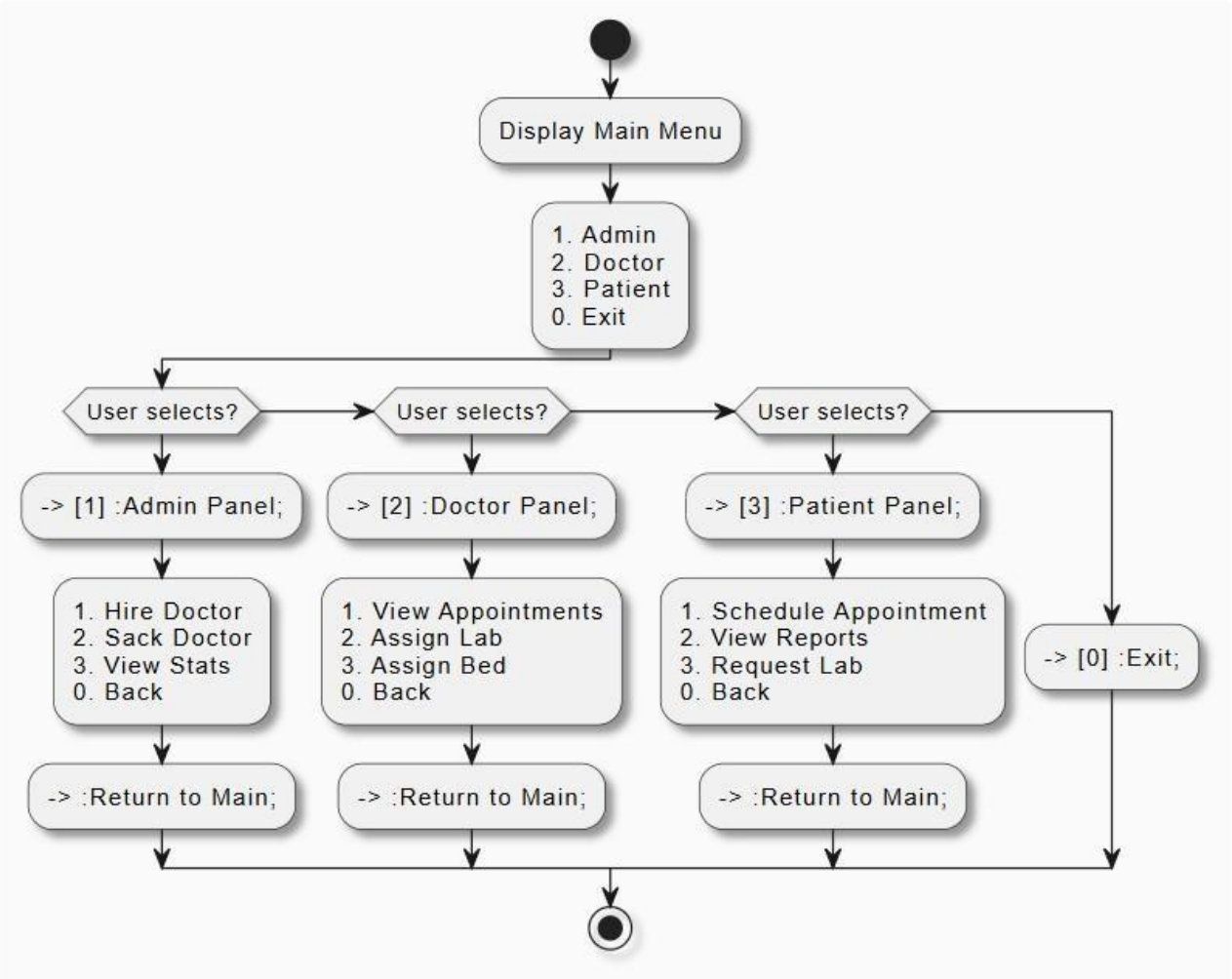
Key Functionalities:

- **add() / sack():** Modify doctor list.
- **assignWardBed():** Bed allocation logic.
- **write() / read():** File I/O for persistence. • **print(), printDept():** UI display for users.

🖼 *Use case diagram*



🖼 *Console Interface Flow*



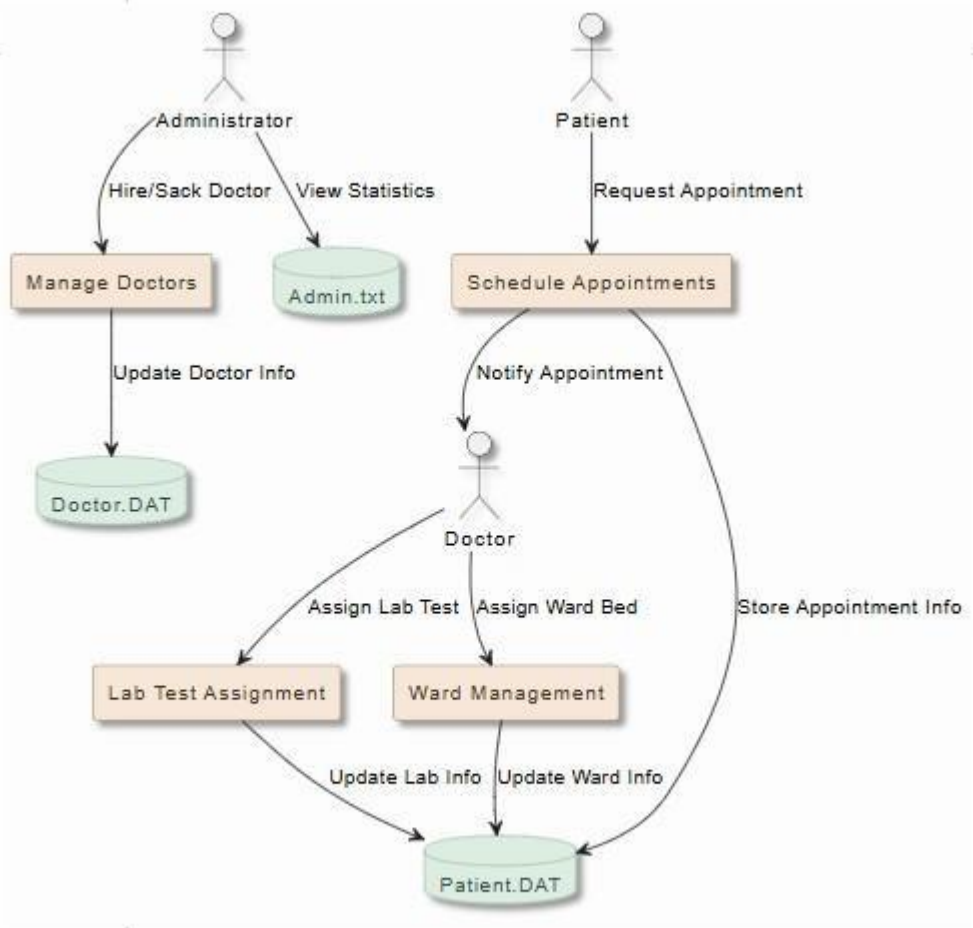
2.3 Data-File Handling

The system uses file I/O for maintaining persistent records:

- **Doctor.DAT, Patient.DAT:** Binary files for structured storage.
- **Admin.txt:** Textual admin info.
- **Uses fstream** to read/write objects via serialization.

All CRUD operations reflect in real-time file updates.

🖼 *Data Flow diagram*



CHAPTER 3: TESTING

Test Cases Covered:

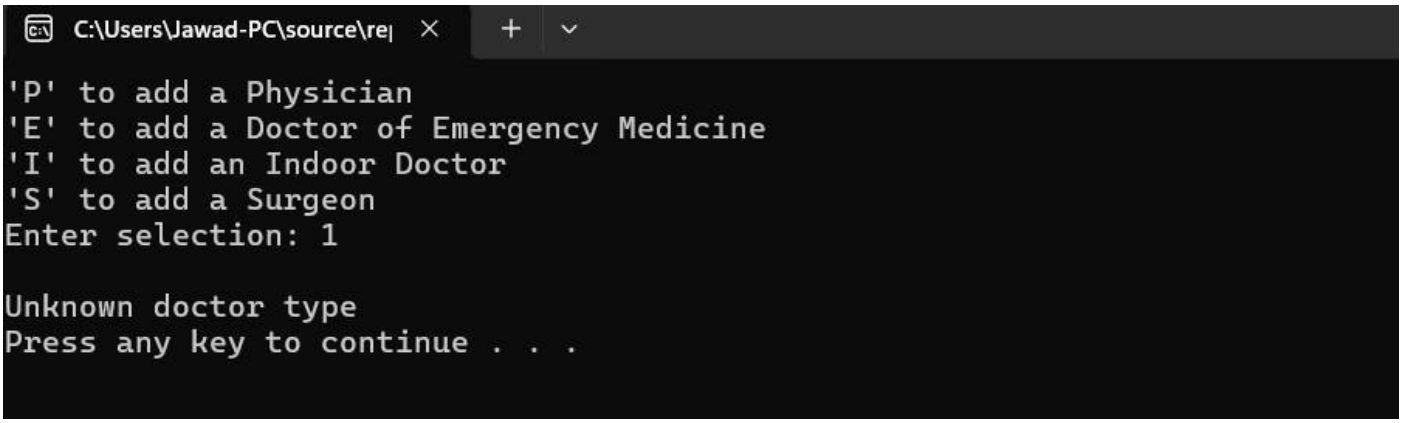
- Admin adds/removes doctor → Verify update in Doctor.DAT.
- Doctor assigns lab test → Lab info correctly stored.
- Ward assignment logic → Proper error handling for full wards.
- Patient appointment → Record consistency checked.

Exception Handling:

- Invalid doctor/patient ID inputs.
- File open/read/write errors.
- Overflow checks for ward beds.

Report -

🖼: *Error Messages and Valid Input Cases*



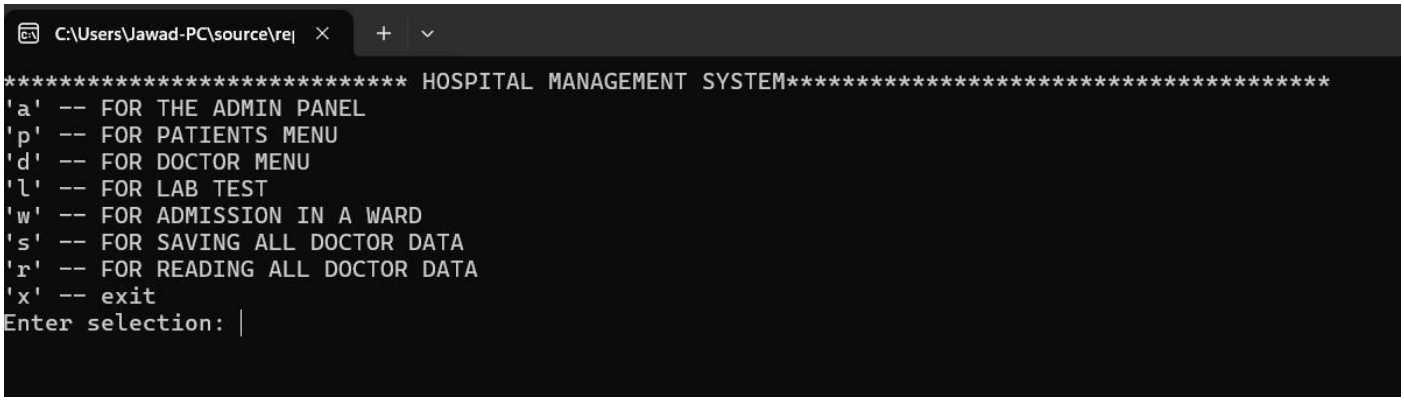
CHAPTER 4: PROJECT OUTPUT

The project features an interactive console menu and structured outputs for each user type.

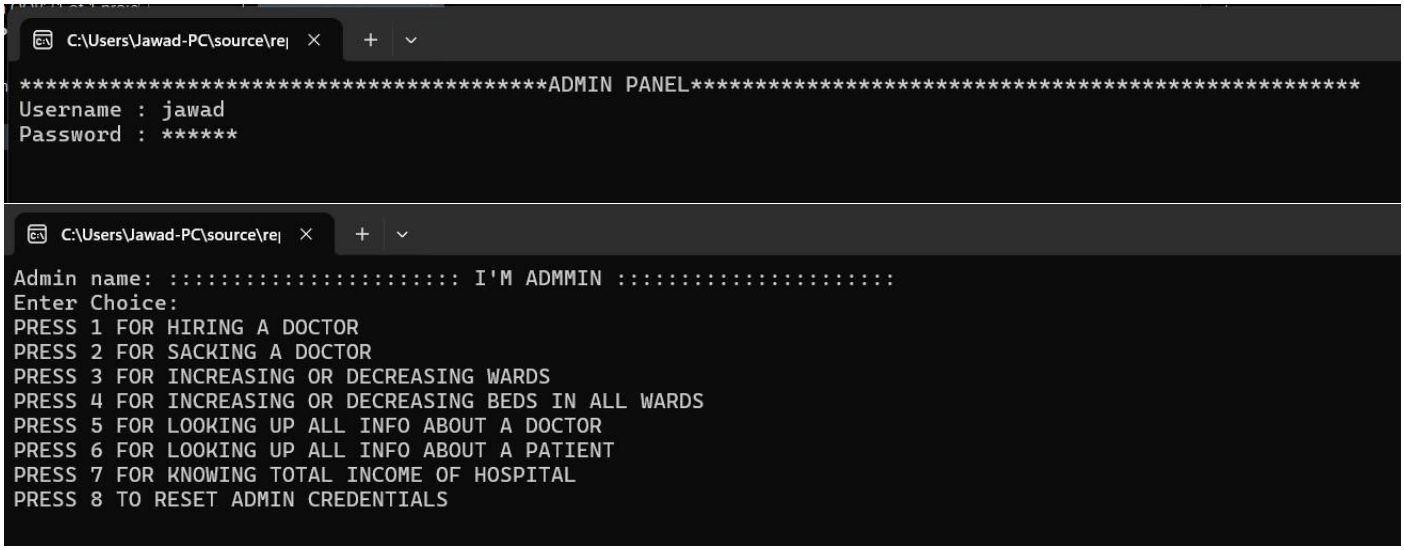
Sample Outputs:

- Doctor Added → Confirms and shows updated list.
- Patient Appointed → Lists time and doctor ID.
- Lab Assigned → Displays test and lab number.
- Ward Status → Shows total/available beds.

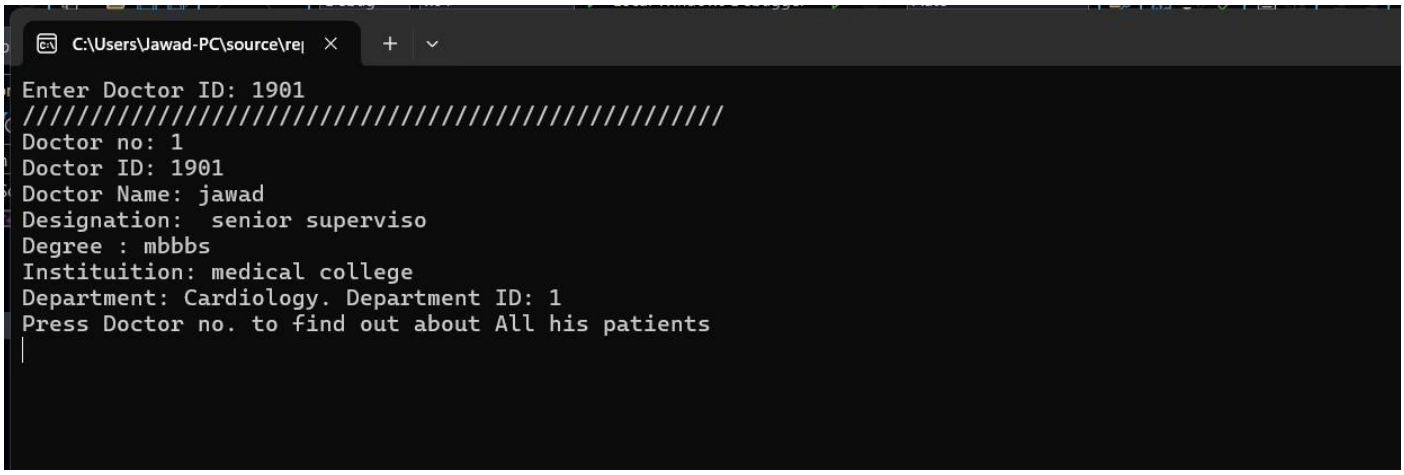
🖼 *Console Output for Main Page*



🖼 *Console Output for Admin Panel & Login*



🖼 *Console Output for Doctor Panel*



🖼 *Console Output for Patient Panel*

Conclusion

This Hospital Management System efficiently showcases the application of Object-Oriented Programming concepts to real-world problems. The modular design aids maintainability and reusability. Students gain hands-on experience in:

- Class hierarchy and object relationships
- File handling for data persistence
- Exception and input validation

Report -

Future Enhancements:

- GUI version with Qt or C++/CLI
- Database integration (MySQL)
- Web-based dashboard using REST APIs

APPENDIX A

Code Listing (Excerpts)

```
// doctor.h class doctor {
protected:     int id;
string name; public:     virtual
void add();     virtual void
sack();     virtual void
assignWardBed();      ... };
```

```
// main.cpp int main()
{     int choice;
do {          print();
cin >> choice;
...
    } while (choice != 0); }
```

(Refer to full code in included project files)
