# Project Title: Hospital Management System (Group Project)

## Overview

This project involves designing and implementing a **Hospital Management System** using the C programming language. The system aims to efficiently manage healthcare resources and patient data within a hospital environment. It addresses patient record management, doctor scheduling, and resource allocation challenges.

**Students will work in groups of 2**.

The project is divided into two phases:

1. **Phase 1:** Focuses on foundational programming concepts, including modular programming and arrays, to implement the basic functionality of the system.
2. **Phase 2:** Introduces advanced concepts such as dynamic memory allocation, file handling, and data structures, enabling the system to scale and handle more complex operations.

## Scenario

A medium-sized hospital is facing issues with manual management of operations. Tasks such as maintaining patient records, managing doctor schedules, and ensuring efficient resource utilization have become challenging, leading to inefficiencies and errors.

To address these challenges, the hospital requires a **Hospital Management System** that can:

1. Maintain accurate and accessible records of admitted patients, including diagnosis and room allocation.
2. Efficiently schedule doctors for daily shifts while avoiding conflicts.
3. Track patient discharges to free up hospital resources for incoming patients.

In this first phase, students will implement a simplified version of the system using modular programming and arrays to handle these tasks.

## Deliverable 1: Basic System Functionality (Week 6)

## Due Date: 15th Feb 2025

### Objective

Develop the foundational components of the Hospital Management System by implementing modular programming and arrays. The goal is to build a simple, functional system for managing patient data and doctor schedules.

### Features to Implement

#### 1. Patient Records Management

- Store and manage information for up to **50 patients** using an array of structures. Each record should include:
    - **Patient ID**: Integer (unique identifier for each patient)
    - **Name**: String (patient's full name)
    - **Age**: Integer (patient's age)
    - **Diagnosis**: String (reason for admission or medical condition)
    - **Room Number**: Integer (assigned hospital room)

#### 2. Basic Operations on Patient Records

- Implement functions to perform the following:
    1. **Add a New Patient Record:** Prompt the user to enter patient details and save them in the array.
    2. **View All Patient Records:** Display all stored records in a structured format.
    3. **Search for a Patient:** Search by either patient ID or name and display the corresponding record.
    4. **Discharge a Patient:** Remove a patient's record from the system once they are discharged. Update the array accordingly.

**3. Doctor Schedule Management**

- Use a **2D array** to manage doctor schedules for a week. Each row represents a day of the week, and each column represents a shift (morning, afternoon, evening).
- Features include:
    - Allow users to assign doctors to specific shifts for each day.
    - Display the full weekly schedule with assigned doctors.

**4. Input Validation**

- Ensure the integrity of the input data:
    - Patient IDs must be **unique**.
    - Age entries must be **valid positive integers**.
    - Ensure proper handling of invalid inputs **and prompt the user to re-enter data when necessary.**

**5. Menu-Driven Interface**

- Design a **text-based menu** to guide user interaction. Example menu:

```
1. Add Patient Record
2. View All Patients
3. Search Patient by ID
4. Discharge Patient
5. Manage Doctor Schedule
6. Exit
```

**Skills Covered in Phase 1**

Students will learn:

- **Functions:** Modular programming to break down the system into manageable components.
- **Arrays:** Implementation of 1D and 2D arrays to store and manipulate data.
- **Input/Output Operations:** Managing user input and displaying structured data output.
- **Error Handling:** Validating inputs and ensuring data integrity.

**Expected Output for Phase 1**

By the end of Phase 1, students will have a functional program that:

1. Efficiently manages patient records.
2. Displays weekly doctor schedules with shift assignments.
3. Provides a user-friendly, menu-driven interface for hospital administrators.

**What to Submit**

For the completion of Phase 1 of the Hospital Management System project, students are required to submit the following:

1. **Source Code:** The complete and well-documented source code of the implemented system.
2. **Project Report:** A comprehensive report, 2-5 pages long, covering the following headings:
    - Project Objectives and Scope: Outline the goals and scope of the project.
    - Design and Implementation Details: Describe the design approach and implementation steps.

- Challenges and Solutions: Discuss any challenges faced during the project and how they were overcome.
- Testing Procedures and Results: Explain the testing methods used and present the results.
- Conclusion: Summarize the project outcomes.

3. **Peer Evaluation:** Each group member will evaluate their peers, contributing 10% to the overall grade for this phase.

## Phase 2: Advanced Features and Concepts (Week 13)

## Due Date: 5th April 2025

### Objective

Expand the foundational Hospital Management System from Phase 1 to include advanced features and programming concepts. Phase 2 focuses on scalability, persistence, and efficient data handling. Students will learn how to use dynamic memory allocation, file handling, and advanced data structures to enhance the system's functionality, robustness, and efficiency.

### Features to Implement in Phase 2

#### 1. Dynamic Memory Allocation

- **Problem Addressed:**
    - In Phase 1, the system was limited to handling only 50 patients due to the use of fixed-size arrays. This approach restricts scalability.
- **Enhancements in Phase 2:**
    - Replace static arrays with **dynamically allocated memory** using malloc and free.
    - Allow the system to grow dynamically as more patients are added. For example:
        - When the number of patients exceeds a certain threshold, reallocate memory to accommodate more patient records.
    - Use pointers to structures for efficient memory management.
- **Expected Outcome:**
    - A scalable system that can handle any number of patient records without predefined limits.

#### 2. File Handling for Data Persistence

- **Problem Addressed:**

- In Phase 1, all data was lost when the program terminated because no persistent storage was implemented.

- **Enhancements in Phase 2:**
  - Implement **file handling** to save and load data:
    - Store patient records in a text or binary file.
    - Store doctor schedules in a separate file.
  - Features to implement:
    - **Save to File:** At the end of a session, save all current patient records and schedules to files.
    - **Load from File:** At the beginning of a session, load existing patient records and schedules from files.
    - **Backup and Restore:** Provide functionality for periodic backups and restoring data.
- **Expected Outcome:**
  - The system will maintain data across multiple program sessions, ensuring no loss of information.

**3. Advanced Data Structures for Efficiency**

- **Problem Addressed:**
  - In Phase 1, searching and managing records relied on linear traversal of arrays, which is inefficient as the dataset grows.
- **Enhancements in Phase 2:**
  - Replace linear arrays with **linked lists** to improve data handling:
    - **Linked Lists:**
      - Use linked lists for patient records, allowing dynamic insertion and deletion without memory reallocation.

- **Expected Outcome:**
    - Improved efficiency in operations such as searching, adding, and removing records.

**4. Reporting and Analytics**

- **Problem Addressed:**
    - Phase 1 lacked the ability to generate summaries or reports, which are vital for administrative decision-making.
- **Enhancements in Phase 2:**
    - Add functions to generate reports such as:
        - Total number of patients admitted in a day, week, or month.
        - List of patients discharged on a specific day.
        - Doctor utilization reports (e.g., total shifts covered by each doctor in a week).
        - Room usage reports to identify underutilized or overutilized resources.
    - Format the reports neatly and allow saving them as text files for later reference.
- **Expected Outcome:**
    - Provide hospital administrators with actionable insights to improve efficiency.

**5. Error Handling**

- **Problem Addressed:**
    - Phase 1 included basic error handling for invalid inputs, but robustness was limited.
- **Enhancements in Phase 2:**
    - Implement advanced error handling techniques:
        - Handle file-related errors (e.g., missing files, read/write failures).

        - Prevent memory leaks by ensuring all dynamically allocated memory is freed before the program exits.
        - Implement input sanitization to prevent invalid data from corrupting records.

- **Expected Outcome:**
    - A highly robust system that gracefully handles unexpected inputs or system errors.

**Skills Covered in Phase 2**

By the end of Phase 2, students will have gained experience in:

- **Dynamic Memory Management:** Using malloc, calloc, and free effectively.
- **File Handling:** Saving and retrieving data using text or binary files.
- **Advanced Data Structures:** Implementing linked lists
- **Error Handling:** Writing robust code that handles invalid inputs and system errors.
- **Scalability:** Designing a system capable of handling larger and more complex datasets.

**Expected Output for Phase 2**

By the end of Phase 2, the enhanced system will:

1. Dynamically allocate memory to handle an arbitrary number of patient records.
2. Save and load patient records and schedules to/from files, ensuring data persistence.
3. Generate reports to assist hospital administrators in decision-making.
4. Provide a highly efficient, user-friendly, and robust interface for hospital management.

**What to Submit**

For the completion of Phase 2 of the Hospital Management System project, students are required to submit the following:

1. **Source Code**: The complete and well-documented source code of the enhanced system.

2. **PowerPoint Presentation**: A 10-15 minute presentation covering the main points of the project, including:
    - **Project Objectives and Scope**: Outline the goals and scope of the project.
    - **Design and Implementation Details**: Describe the design approach and implementation steps.
    - **Dynamic Memory Allocation**: Explain how dynamic memory allocation was implemented and its benefits.
    - **File Handling for Data Persistence**: Detail the file handling techniques used for data persistence.
    - **Advanced Data Structures**: Discuss the use of linked lists and their advantages.
    - **Reporting and Analytics**: Highlight the reporting features added and their importance.
    - **Error Handling**: Describe the advanced error handling techniques implemented.
    - **Testing Procedures and Results**: Present the testing methods used and the results obtained.
    - **Conclusion**: Summarize the project outcomes.
3. **Live Demo**: Prepare to show a live demonstration of the system during Week 13 or 14, as discussed in class.
4. **Peer Evaluation:** Each group member will evaluate their peers, contributing 10% to the overall grade for this phase.