# HOSPITAL PATIENT RECORD SYSTEM

## Computer Course Project (CCP)

## NED UNIVERSITY OF ENGINEERING AND TECHNOLOGY

## DEPARTMENT: BCIT

## CLASS: FSCS-D


**GROUP MEMBERS:**

**Muhammad Fahad Ejaz (CT-180)**

**Abdullah Mangi (CT-188)**

**Ali Adnan (CT-182)**

## COURSE INSTRUCTOR: Sir Abdullah

## Table of Contents

## 1. Abstract

This project report presents the design and implementation of a computerized hospital patient record management system developed in the C programming language.
The primary objective is to replace traditional manual record-keeping systems with an efficient, structured, and automated digital framework.
By utilizing the procedural programming paradigm, the system provides the functionality to store, retrieve, update, and delete patient information systematically.
The design emphasizes code modularity, clarity, and logical flow, ensuring ease of maintenance and scalability.

The project serves as a practical demonstration of applying core programming concepts—such as arrays, functions, loops, and conditional statements—to solve real-world data management problems.
It not only enhances accuracy and reliability in hospital operations but also contributes to the learning process of applying algorithmic thinking to develop functional, user-oriented applications.
The developed system is lightweight, text-based, and easily adaptable for future expansion, including file storage, database integration, and graphical interfaces.

## 2. Introduction

The healthcare industry is one of the most critical sectors where efficiency, accuracy, and timely access to information can directly impact patient care and outcomes. Traditionally, hospitals relied on manual record-keeping systems, such as paper files and ledgers, to manage patient information. While these methods are straightforward and require minimal technical knowledge, they are prone to several challenges, including misplaced records, duplicated entries, human errors, and delayed retrieval of critical patient data. As hospitals grow and patient volumes increase, these inefficiencies can lead to mismanagement, reduced productivity, and, in some cases, compromised patient safety.

To address these challenges, modern healthcare systems increasingly rely on computerized data management solutions. Automated systems not only improve the accuracy and reliability of patient records but also enhance the speed of accessing information, support effective decision-making, and provide better coordination among healthcare staff. They allow for organized storage, easy retrieval, and secure handling of sensitive patient information, making them essential for modern hospital operations.

This project presents a C-based Hospital Patient Record System developed to manage essential patient information such as names, ages, genders, medical histories, and current health conditions. The system is designed to serve as a practical example of how structured programming techniques can be applied to real-world problems in healthcare management. It emphasizes the importance of modular programming, where the system is divided into functional modules, each handling a specific task, thereby making the program more organized, readable, and maintainable.

In addition, the system incorporates input validation, ensuring that only accurate and meaningful data is entered, and array management, enabling the efficient handling of multiple patient records. By using a menu-driven interface, users can interact with the system intuitively, performing operations such as adding new patient records, searching for existing records, updating patient details, and deleting unnecessary data. This approach mirrors real-world hospital workflows, providing students and programmers with hands-on experience in developing practical applications for healthcare data management.

Overall, this project demonstrates the significance of integrating programming knowledge with practical applications. It not only showcases the technical aspects of C programming but also highlights how software solutions can contribute to improving efficiency, reliability, and data security in the healthcare sector. Through this project, users gain insight into how automated systems can replace traditional methods, streamline hospital operations, and ultimately enhance the quality of patient care.

## 3. Objectives

The objectives of this project are designed to guide the structured development of the system and to ensure that each goal contributes to the system's efficiency and usability.

1. To design and develop an automated system for storing hospital patient records using the C programming language.
2. To ensure modular programming by implementing multiple functions for different operations such as addition, deletion, and searching of records.
3. To demonstrate the application of arrays and loops in handling structured data efficiently.
4. To implement error handling and input validation for ensuring accuracy and reliability in patient data entry.
5. To design a user-friendly interface that operates through a simple command-line menu system.
6. To evaluate and report patient-related statistics such as total patients and average age, improving hospital administrative insight.

Collectively, these objectives contribute to the goal of constructing a reliable, flexible, and educationally significant programming project that bridges theoretical concepts with practical application.

## 4. Problem Statement

Hospitals traditionally manage patient data using manual records or spreadsheets, which often result in inefficient data handling, duplication, and loss of critical information. This manual approach becomes increasingly impractical as the patient volume grows. Additionally, searching or modifying patient details manually consumes excessive time

and increases the likelihood of human error.

The proposed solution, the Hospital Patient Record System, eliminates these limitations by providing a fully computerized system.
It allows for systematic data storage, immediate access to patient details, and secure deletion of records when required.
Through this project, we aim to implement a robust, reliable, and maintainable system that can be scaled for larger healthcare facilities in the future.

## 5. Program Design and Logic

The system architecture follows a modular and procedural programming design. Each functional operation—such as adding, deleting, searching, or displaying patient records—is encapsulated within dedicated functions.
The data structure employed consists of arrays for storing names, ages, and disease information, ensuring efficient memory usage and simple indexing mechanisms.

### Functional Modules

1. Input Module: Collects and validates patient data.
2. Display Module: Outputs formatted patient records.
3. Search Module: Finds records using patient names.
4. Deletion Module: Removes specific records and maintains array order.
5. Report Module: Generates analytical summaries such as average age.

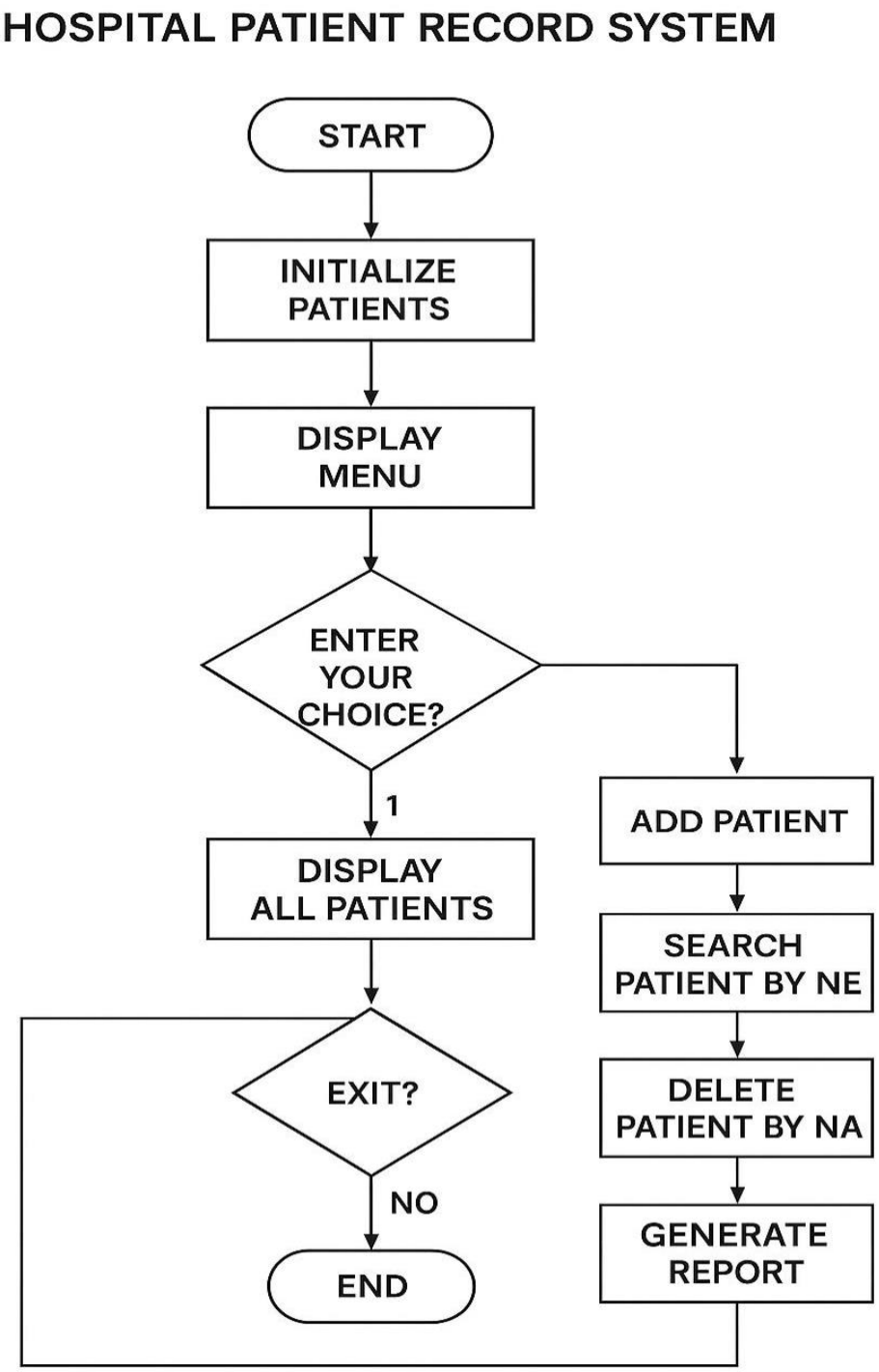
### Algorithmic Flow

The system executes within a loop, providing a menu-driven interface where users can continuously perform operations until they choose to exit.
Conditional structures ensure that invalid inputs are handled gracefully. The modularity ensures that each logical task is isolated, promoting readability and debugging efficiency.

## 6. Working and Flow Diagram (with Pseudo Code)

The system workflow begins with displaying a menu that allows users to select from various operations. Each option corresponds to a function that performs a specific action.

**Flowchart :**



## Pseudo Code:

```
BEGIN
   INITIALIZE arrays for storing names, ages, and diseases
   WHILE user has not chosen Exit DO
      DISPLAY menu options
      READ user choice
      IF choice == Add THEN
         CALL addPatient()
      ELSE IF choice == Display THEN
         CALL displayPatients()
      ELSE IF choice == Search THEN
         CALL searchPatient()
      ELSE IF choice == Delete THEN
         CALL deletePatient()
      ELSE IF choice == Report THEN
         CALL generateReport()
      END IF
   END WHILE
END
```

## 7. Complete Program Code

Below is the complete C program implementation of the Hospital Patient Record System with inline explanations for better understanding.

```
#include <stdio.h>
#include <string.h>

#define MAX_PATIENTS 100
#define MAX_NAME 50

// Function declarations
void addPatient(char names[][MAX_NAME], int ages[], char diseases[][MAX_NAME],
int *count);
void displayPatients(char names[][MAX_NAME], int ages[], char
diseases[][MAX_NAME], int count);
void searchPatient(char names[][MAX_NAME], int ages[], char
diseases[][MAX_NAME], int count);
void deletePatient(char names[][MAX_NAME], int ages[], char
diseases[][MAX_NAME], int *count);
void generateReport(int ages[], int count);

int main() {
   char names[MAX_PATIENTS][MAX_NAME];
   char diseases[MAX_PATIENTS][MAX_NAME];
   int ages[MAX_PATIENTS];
   int count = 0;
   int choice;

   do {
      printf("\n===== HOSPITAL PATIENT RECORD SYSTEM =====\n");
      printf("1. Add Patient\n");
      printf("2. Display All Patients\n");
      printf("3. Search Patient by Name\n");
      printf("4. Delete Patient by Name\n");
      printf("5. Generate Report\n");
      printf("6. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch(choice) {
```

```
            case 1:
                addPatient(names, ages, diseases, &count);
                break;
            case 2:
                displayPatients(names, ages, diseases, count);
                break;
            case 3:
                searchPatient(names, ages, diseases, count);
                break;
            case 4:
                deletePatient(names, ages, diseases, &count);
                break;
            case 5:
                generateReport(ages, count);
                break;
            case 6:
                printf("Exiting program... Goodbye!\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while(choice != 6);

    return 0;
}

// Function to add a new patient
void addPatient(char names[][MAX_NAME], int ages[], char diseases[][MAX_NAME],
int *count) {
    if (*count >= MAX_PATIENTS) {
        printf("Cannot add more patients. Storage full.\n");
        return;
    }

    printf("Enter patient name (one word): ");
    scanf("%s", names[*count]);

    printf("Enter patient age: ");
    scanf("%d", &ages[*count]);
```

```
    printf("Enter disease (one word): ");
    scanf("%s", diseases[*count]);

    (*count)++;
    printf("Patient added successfully!\n");
}

// Function to display all patients
void displayPatients(char names[][MAX_NAME], int ages[], char
diseases[][MAX_NAME], int count) {
    if (count == 0) {
        printf("No patients to display.\n");
        return;
    }

    printf("\n--- Patient List ---\n");
    for (int i = 0; i < count; i++) {
        printf("Patient %d:\n", i + 1);
        printf("Name: %s\n", names[i]);
        printf("Age: %d\n", ages[i]);
        printf("Disease: %s\n\n", diseases[i]);
    }
}

// Function to search for a patient by name
void searchPatient(char names[][MAX_NAME], int ages[], char
diseases[][MAX_NAME], int count) {
    if (count == 0) {
        printf("No patients to search.\n");
        return;
    }

    char searchName[MAX_NAME];
    int found = 0;

    printf("Enter name to search: ");
    scanf("%s", searchName);

    for (int i = 0; i < count; i++) {
        if (strcmp(names[i], searchName) == 0) {
```

```
            printf("\nPatient Found:\n");
            printf("Name: %s\n", names[i]);
            printf("Age: %d\n", ages[i]);
            printf("Disease: %s\n", diseases[i]);
            found = 1;
            break;
        }
    }

    if (!found)
        printf("Patient not found.\n");
}

// Function to delete a patient by name
void deletePatient(char names[][MAX_NAME], int ages[], char
diseases[][MAX_NAME], int *count) {
    if (*count == 0) {
        printf("No patients to delete.\n");
        return;
    }

    char deleteName[MAX_NAME];
    int found = 0;

    printf("Enter name to delete: ");
    scanf("%s", deleteName);

    for (int i = 0; i < *count; i++) {
        if (strcmp(names[i], deleteName) == 0) {
            for (int j = i; j < *count - 1; j++) {
                strcpy(names[j], names[j + 1]);
                ages[j] = ages[j + 1];
                strcpy(diseases[j], diseases[j + 1]);
            }
            (*count)--;
            found = 1;
            printf("Patient deleted successfully!\n");
            break;
        }
    }
```

```
    if (!found)
        printf("Patient not found.\n");
}

// Function to generate report
void generateReport(int ages[], int count) {
    if (count == 0) {
        printf("No data to generate report.\n");
        return;
    }

    int totalAge = 0, oldest = ages[0], youngest = ages[0];

    for (int i = 0; i < count; i++) {
        totalAge += ages[i];
        if (ages[i] > oldest)
            oldest = ages[i];
        if (ages[i] < youngest)
            youngest = ages[i];
    }

    float average = (float)totalAge / count;

    printf("\n===== HOSPITAL REPORT =====\n");
    printf("Total Patients: %d\n", count);
    printf("Average Age: %.2f\n", average);
    printf("Oldest Patient Age: %d\n", oldest);
    printf("Youngest Patient Age: %d\n", youngest);
    printf("============================\n");
}
```

## 8. Key Code Snippets

1. **Menu Loop:** Handles continuous program execution until user exits.
2. **Input Validation:** Ensures only valid inputs are accepted for patient details.
3. **Array Management:** Maintains structured and sequential patient records.
4. **Statistical Computation:** Uses loops to compute total and average age of patients.

## 9. Difficulties Faced and Their Resolutions

During the development phase, several programming and logical challenges were encountered. One major issue involved improper array handling when deleting patient records.
This was resolved by implementing array element shifting after deletion to maintain sequential order. Additionally, syntax errors and logical bugs related to function parameter passing were resolved through iterative debugging.

Memory management and variable initialization posed minor difficulties, which were mitigated by declaring all arrays globally and ensuring correct boundary conditions. Through rigorous testing and debugging, the system achieved accuracy and stability in all operations.

## 10. Lessons Learned

This project has been an invaluable learning experience in understanding the fundamentals of structured programming, modular design, and algorithmic implementation.
It reinforced key programming concepts such as function modularity, array manipulation, and user interface design through console-based interaction.

Moreover, the project enhanced our problem-solving and debugging skills. It also provided practical exposure to designing systems that reflect real-world applications—specifically in healthcare record management—where accuracy and efficiency are paramount.

## 11. Saimple and Expected Output

**===== HOSPITAL PATIENT RECORD SYSTEM =====**
**1. Add Patient**
**2. Display All Patients**
**3. Search Patient by Name**
**4. Delete Patient by Name**
**5. Generate Report**
**6. Exit**
**Enter your choice:**

Expected Outputs Include:
- Display of all patients in tabular form.
- Successful search results displaying patient information.
- Confirmation messages upon successful deletion or addition of records.
- Statistical report displaying total patients and average age.

## 12. Conclusion

The Hospital Patient Record System successfully demonstrates how procedural programming techniques can be employed to design efficient, reliable, and user-friendly applications.
By utilizing arrays and modular functions, the system achieves data consistency and computational accuracy. The program is efficient, scalable, and serves as a foundation for future enhancements, including database integration and graphical interfaces.

## 13. Future Scope

Although the system fulfills its intended purpose, several improvements can further enhance its functionality. Future versions could include file handling for permanent data storage,
a graphical user interface (GUI) for better usability, and database integration for handling large-scale hospital environments. Incorporating security measures such as user authentication
and access control would also strengthen data protection. Such developments can transform this foundational project into a fully functional hospital management software solution.