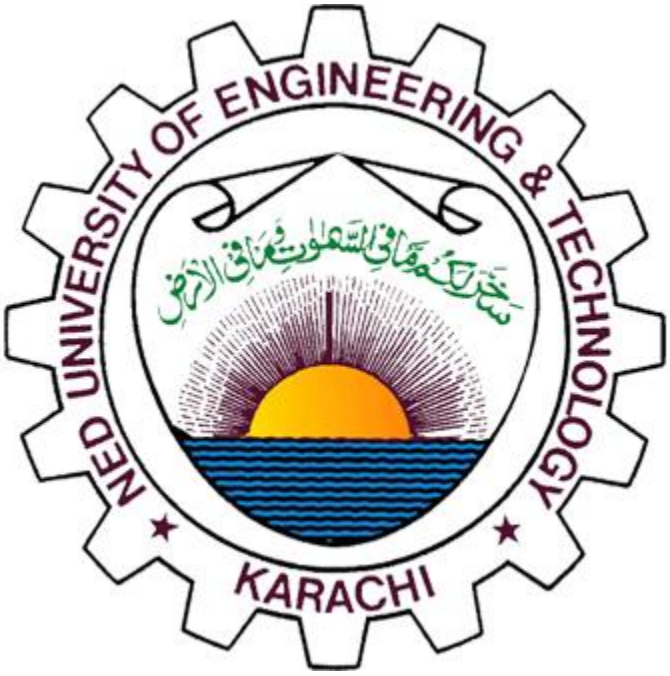
CCP-REPORT



GROUP MEMBERS:

EMAN SIDDIQUI CT-072

DUA FAROOQ CT-052

DISCIPLINE:

BCIT (1st Year)

TEACHERS:

MUHAMMAD ABDULLAH

FURQAN HUSSAIN

INDEX:

PROJECT TITLE:

HOSPITAL PATIENT RECORD SYSTEM

ABSTRACT:

In today's healthcare system, managing patient information properly plays a very important role in providing smooth and effective medical services. Every day, hospitals and clinics handle a large number of patients, and keeping their records accurate and up to date can be a difficult task. Many healthcare centers still depend on paper files to record patient details, which can easily be misplaced, damaged, or take too long to search through. These problems often lead to delays in treatment, repeated tests, and unnecessary confusion. To overcome such challenges, this project introduces a **Patient Record System** developed in the C programming language.

INTRODUCTION:

This system is designed to help hospitals and clinics store and manage patient information in a simple and organized way. It allows users to enter details such as the patient's name, disease, age, and a unique ID number. Once entered, the program makes it easy to view all patient records, search for a patient by ID or disease, and even update information whenever needed. The use of arrays and functions in C helps in storing and managing the data efficiently.

Although this version of the project stores data temporarily in memory, it can be further improved by adding new features such as saving records permanently using file handling, deleting old records, and adding data validation to prevent incorrect entries. In the future, it can also be connected to a database to handle a larger number of patients or linked with hospital management software for more advanced use.

Overall, this project shows how computer programming can solve real-life problems in the healthcare field. By converting manual record-keeping into a digital process, the **Patient Record System** makes it easier, faster, and safer to manage patient data. It is a small but meaningful step toward the digital transformation of healthcare, ensuring that technology benefits not only large hospitals but also small clinics and local health centers.

OBJECTIVES:

The main aim of this project is to develop a simple and effective **Patient Record System** that helps manage patient information digitally. The system is designed to:

1. **Provide Organized Record-Keeping:**
   Ensure that patient details are stored in a clear and structured way for easy access.
2. **Improve Efficiency:**
   Reduce the time and effort required to find, update, or maintain patient records.
3. **Minimize Errors:**
   Decrease mistakes that often occur in manual paper-based record-keeping.
4. **Be Easy to Use:**
   Create a user-friendly system that can be used by hospital or clinic staff without technical expertise.
5. **Support Learning and Understanding**
   Serve as a practical example for students and beginners to understand basic programming concepts like arrays, functions, and conditional statements, while solving a real-world problem.

TOOLS AND TECHNOLOGIES USED:

- **Programming Language:** C
- **Compiler:** Dev C++
- **Libraries Used:** `stdio.h` and `string.h`
- **Environment:** Command-Line Interface (CLI)
- **Programming Concepts:** Loops, Arrays, Conditional Statements and Functions

HOW IT WAS SOLVED?

To solve the problem of managing patient records manually, a simple and user-friendly program was developed using the **C programming language**. The goal was to design a system that could perform basic operations like adding, viewing, searching, and updating patient information easily

and efficiently. The project focuses on using simple programming logic, arrays, and functions to store and handle multiple patient details at once.

The first step in solving the problem was to identify what kind of data needed to be stored for each patient. It was decided that every patient would have four main pieces of information, **ID, name, disease, and age** After defining the data structure, different functions were created to perform specific tasks.

A menu-driven structure was used in the main program to make the system interactive and simple to use. The user can select from different options like adding a patient, searching, or updating data, just by entering a number from the menu. This approach makes the program easy to understand and operate even for someone with basic computer knowledge.

The logic behind this system was kept straightforward to ensure that it works smoothly without unnecessary complexity. Loops and conditional statements were used to control the flow of the program, while the strcmp() function was used to compare strings when searching by disease. This made the program efficient and responsive.

By combining these programming techniques, the problem of manual record keeping was effectively solved. The **Patient Record System** now provides a simple and organized way to store and retrieve patient data digitally. It reduces human errors, saves time, and makes patient management faster and more accurate.

ARCHITECTURE OF THE SYSTEM:

The **Patient Record System** is designed using a **modular and simple architecture** that allows easy management of patient information. The system is console-based and built using the **C programming language**. Its architecture can be understood in terms of **data storage, main modules, and program flow**.

1. Data Storage

All patient information is stored temporarily in **arrays**. Each patient has four main attributes:

- **ID** – a unique number automatically assigned to each patient.
- **Name** – the patient's full name.
- **Disease** – the illness or condition of the patient.
- **Age** – the patient's age.

Arrays are used to store multiple patient records, making it possible to manage up to 50 patients in the current system.

2. Modules of the System

The program is divided into different **functions (modules)**, each responsible for a specific task:

1. The **add_patient()** function lets users add a new patient by entering their name, disease, and age. The program automatically assigns a new ID to each patient.
2. The **view_patient()** function displays all the patient records stored in the system in a clean and organized way.
3. The **search_patient_by_id()** and **search_patient_by_disease()** functions make it easy to find a particular patient or a group of patients by entering their ID or disease name.
4. The **update_patient()** function allows the user to change any existing patient's details, such as name, disease, or age, if any corrections are needed.

3. Program Flow

The system uses a **menu-driven approach** to interact with the user. When the program runs, it repeatedly shows a menu with all available options. The user selects an option by entering a number, and the corresponding function is executed. This loop continues until the user chooses to exit the program.

4. Technical Terms and Data Handling

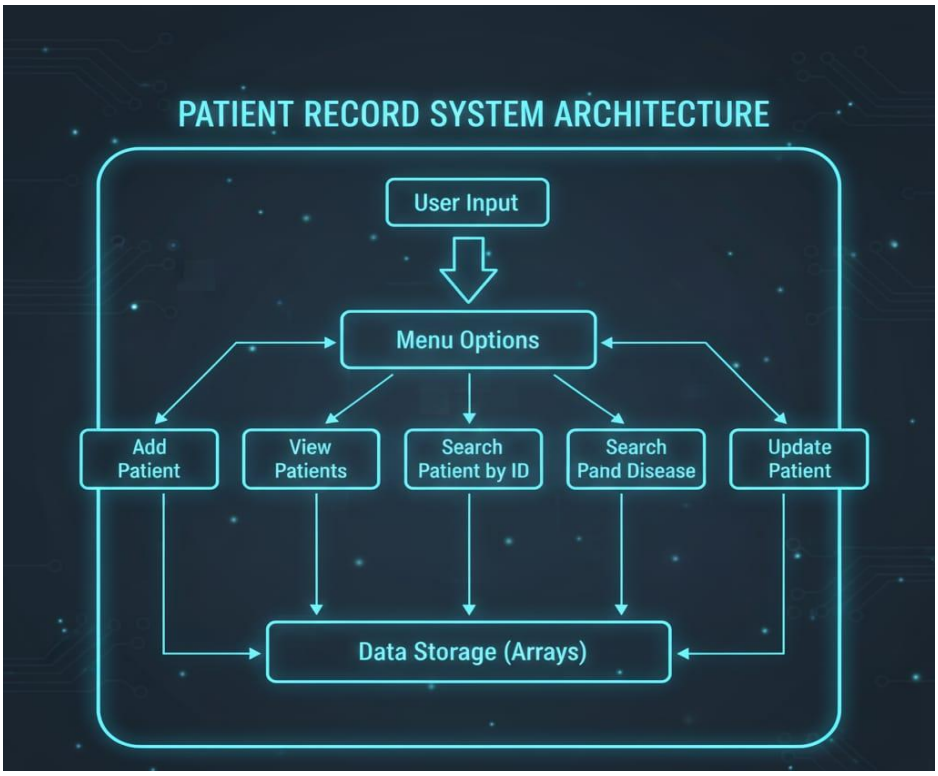
- **Arrays** are used to store multiple patient records. They allow multiple records to be managed simultaneously in memory.
- The program uses **loops** to traverse arrays and access records.
- Different **functions** are used for specific task. Each major operation, adding, viewing, searching, and updating patients is defined as a separate function to keep the program modular and organized.
- **Conditional statements** Employed to control program flow and handle user choices effectively within the menu-driven system.
- **String comparison functions** (like strcmp) are used when searching patients by disease.
- **Data Handling--** Data is stored temporarily in arrays during program execution. Input and output operations rely on standard console interaction for entering and displaying records.

5. Scalability

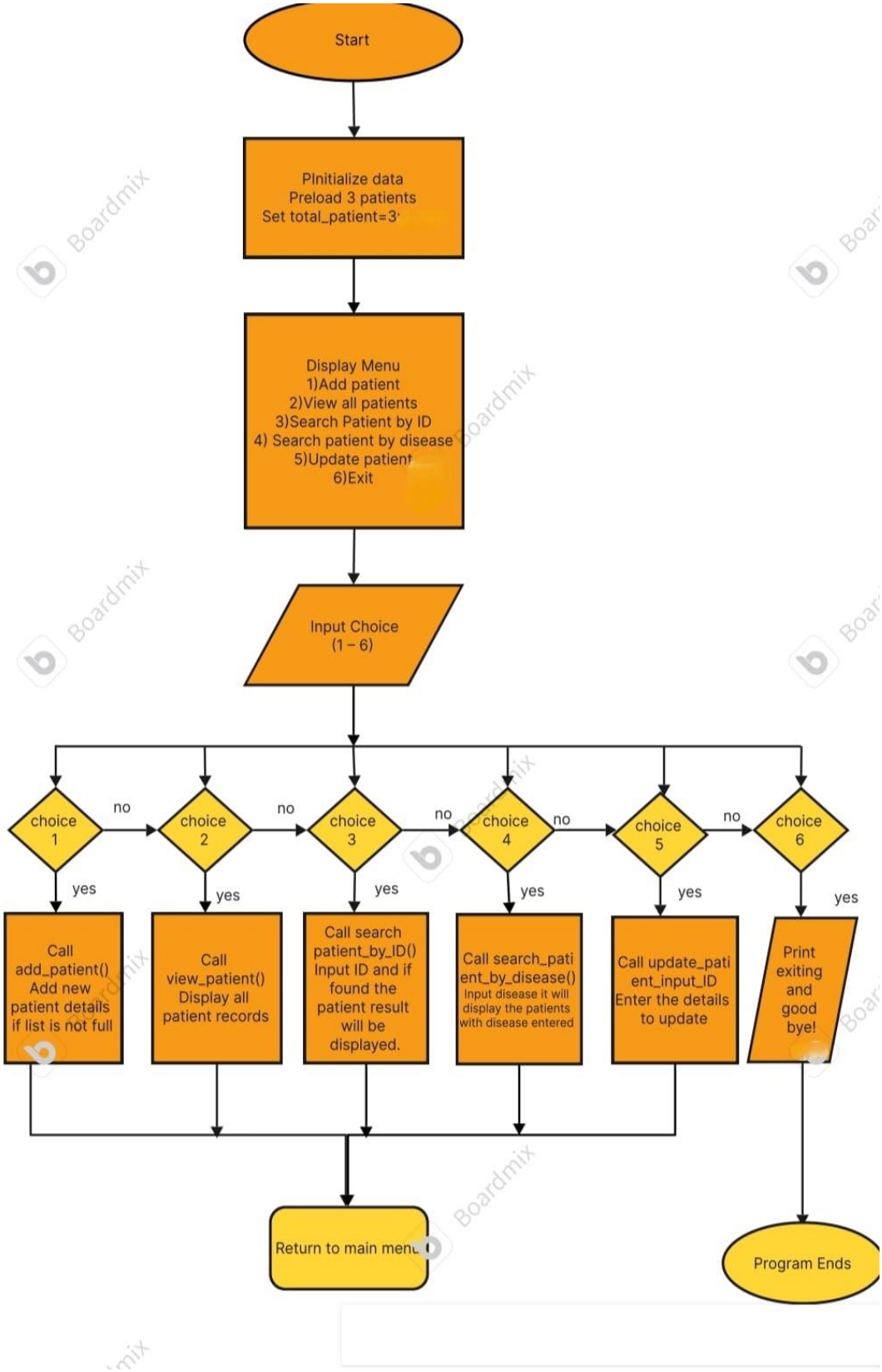
The system architecture is simple but **modular**, which allows future improvements, such as:

- Storing data permanently using **file handling**.
- Adding features like **deletion of records**.
- Integrating with a **database** for handling larger datasets.

In short, the architecture is **functional, easy to understand, and flexible**, making it suitable for small hospitals, clinics, and as a learning project for programming students.


PATIENT RECORD SYSTEM ARCHITECTURE
User Input
Menu Options
Add Patient
View Patients
Search Patient by ID
Search Pand Disease
Update Patient
Data Storage (Arrays)

FLOWCHART:


Start
PInitialize data
Preload 3 patients
Set total_patient=3
Display Menu
1)Add patient
2)View all patients
3)Search Patient by ID
4) Search patient by disease
5)Update patient
6)Exit
Input Choice
(1 – 6)
choice 1
no
choice 2
no
choice 3
no
choice 4
no
choice 5
no
choice 6
yes
yes
yes
yes
yes
yes
Call add_patient() Add new patient details if list is not full
Call view_patient() Display all patient records
Call search patient_by_ID() Input ID and if found the patient result will be displayed.
Call search_pati ent_by_disease() Input disease it will display the patients with disease entered
Call update_pati ent_input_ID Enter the details to update
Print exiting and good bye!
Return to main menu
Program Ends

SOURCE CODE:

```
#include <stdio.h>

#include <string.h>


int choice;

int id[50];

char name[50][50];

char disease[50][50];

int age[50];

int total_patient = 3;


// Add Patient

void add_patient() {

   if(total_patient >= 50) {

      printf("\n         Patient list is full! Try again later!\n\n");

      return;

   }


   printf("\n             === Adding Patient ===\n\n");

   id[total_patient] = total_patient + 1;


   printf("Enter name: ");

   scanf(" %[^\n]s", name[total_patient]);


   printf("Enter disease: ");

   scanf(" %[^\n]s", disease[total_patient]);


   printf("Enter age: ");

   scanf("%d", &age[total_patient]);
```

```
    total_patient++;
    printf("\n         Patient added successfully!\n\n");
    printf("-------------------------------------------------\n\n");
}

// View All Patients
void view_patient() {
    if(total_patient == 0) {
        printf("\n         No patients available!\n\n");
        return;
    }

    printf("\n            === All Patients ===\n\n");
    for(int i = 0; i < total_patient; i++) {
        printf("Patient ID: %d\n", id[i]);
        printf("Name      : %s\n", name[i]);
        printf("Disease   : %s\n", disease[i]);
        printf("Age       : %d\n", age[i]);
        printf("-------------------------------------------------\n");
    }
    printf("\n");
}

// Search Patient by ID
void search_patient_by_id() {
    int searchID;
    printf("\nEnter patient ID: ");
    scanf("%d", &searchID);
```

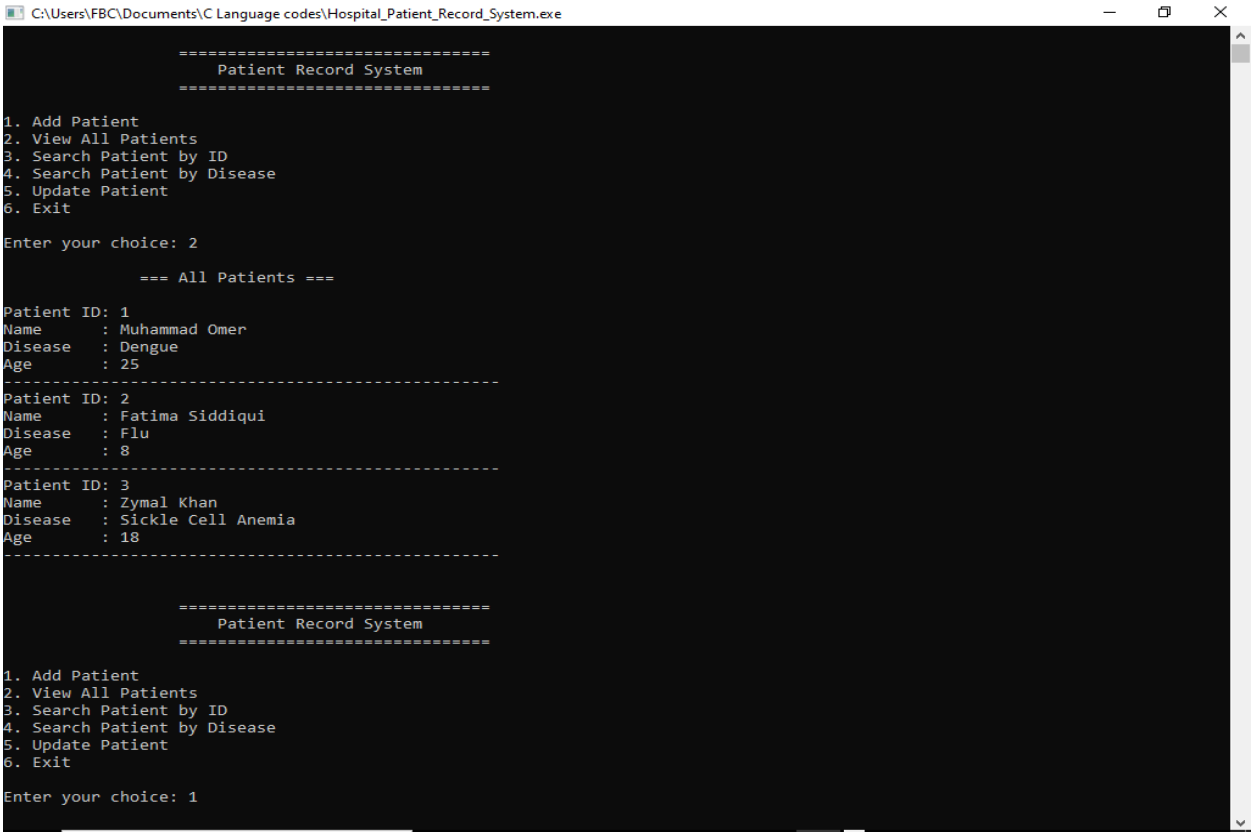
```
    printf("\n          === Searching Patient  ===\n\n");
    for(int i = 0; i < total_patient; i++) {
        if(id[i] == searchID) {
            printf("Record found!\n\n");
            printf("Patient ID: %d\n", id[i]);
            printf("Name      : %s\n", name[i]);
            printf("Disease   : %s\n", disease[i]);
            printf("Age       : %d\n", age[i]);
            printf("-------------------------------------------------\n\n");
            return;
        }
    }
    printf("No record found!\n\n");
}

// Search Patient by Disease
void search_patient_by_disease() {
    char searchDisease[50];
    int found = 0;

    printf("\nEnter disease name: ");
    scanf(" %[^\n]s", searchDisease);

    printf("\n          === Patients with %s ===\n\n", searchDisease);
    for(int i = 0; i < total_patient; i++) {
        if(strcmp(disease[i], searchDisease) == 0) {
            printf("Patient ID: %d\n", id[i]);
            printf("Name : %s\n", name[i]);
```

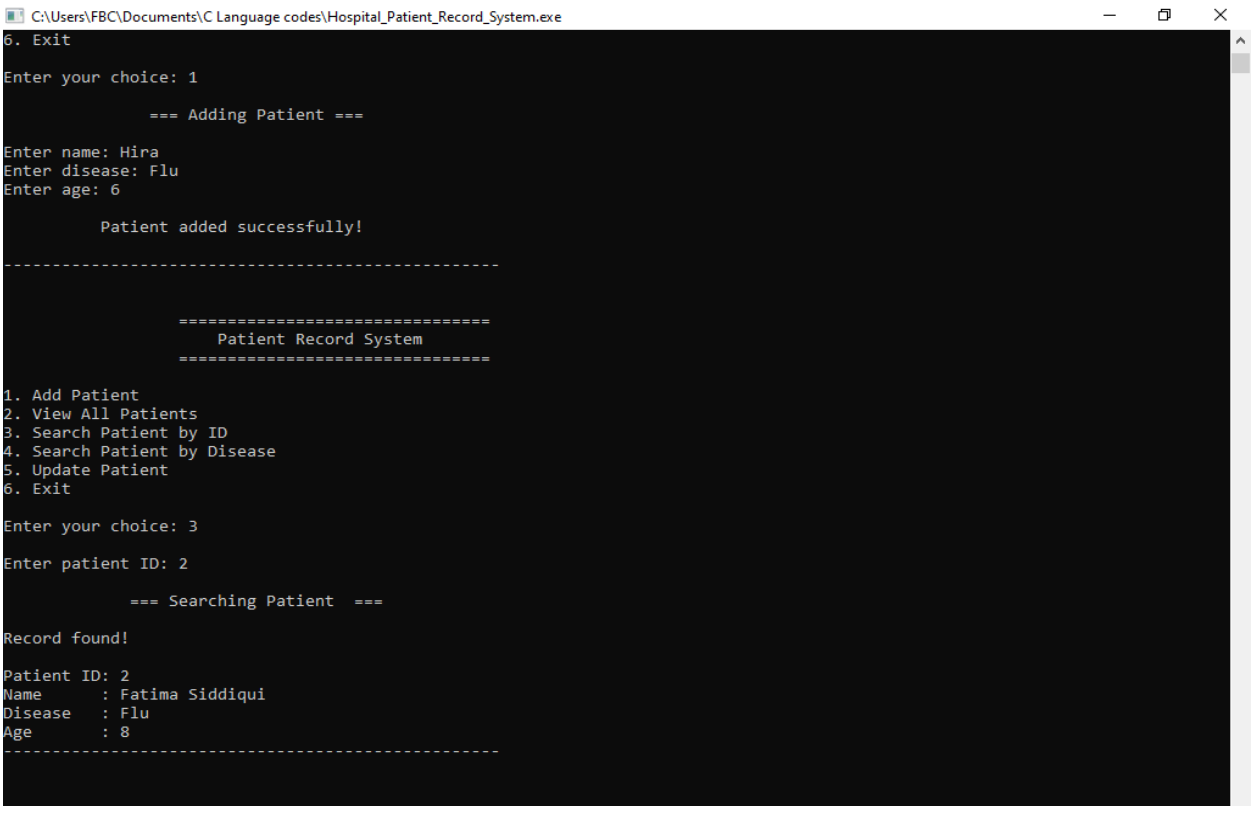
```
            printf("Age : %d\n", age[i]);
            printf("-------------------------------------------------\n");
            found = 1;
        }
    }

    if(!found) {
        printf("No patients found with this disease.\n");
    }
    printf("\n");
}

// Update Patient
void update_patient() {
    int updateID;
    printf("\nEnter patient ID to update: ");
    scanf("%d", &updateID);

    printf("\n             === Updating Patient ===\n\n");
    for(int i = 0; i < total_patient; i++) {
        if(id[i] == updateID) {
            printf("Enter new name: ");
            scanf(" %[^\n]s", name[i]);

            printf("Enter new disease: ");
            scanf(" %[^\n]s", disease[i]);

            printf("Enter new age: ");
            scanf("%d", &age[i]);
```

```
            printf("\n Record updated successfully!\n\n");
            printf("-----------------------------------------------------\n\n");
            return;
        }
    }
    printf("No patient found with ID %d\n\n", updateID);
}

// Main Function
int main(void) {
    // Initial patients
    id[0] = 1; strcpy(name[0], "Muhammad Omer");
            strcpy(disease[0], "Dengue"); age[0] = 25;
    id[1] = 2; strcpy(name[1], "Fatima Siddiqui");
            strcpy(disease[1], "Flu"); age[1] = 8;
    id[2] = 3; strcpy(name[2], "Zymal Khan");
            strcpy(disease[2], "Sickle Cell Anemia"); age[2] = 18;

    while(1) {
        printf("\n              ================================\n");
        printf("                 Patient Record System\n");
        printf("              ================================\n\n");

        printf("1. Add Patient\n");
        printf("2. View All Patients\n");
        printf("3. Search Patient by ID\n");
        printf("4. Search Patient by Disease\n");
        printf("5. Update Patient\n");
```

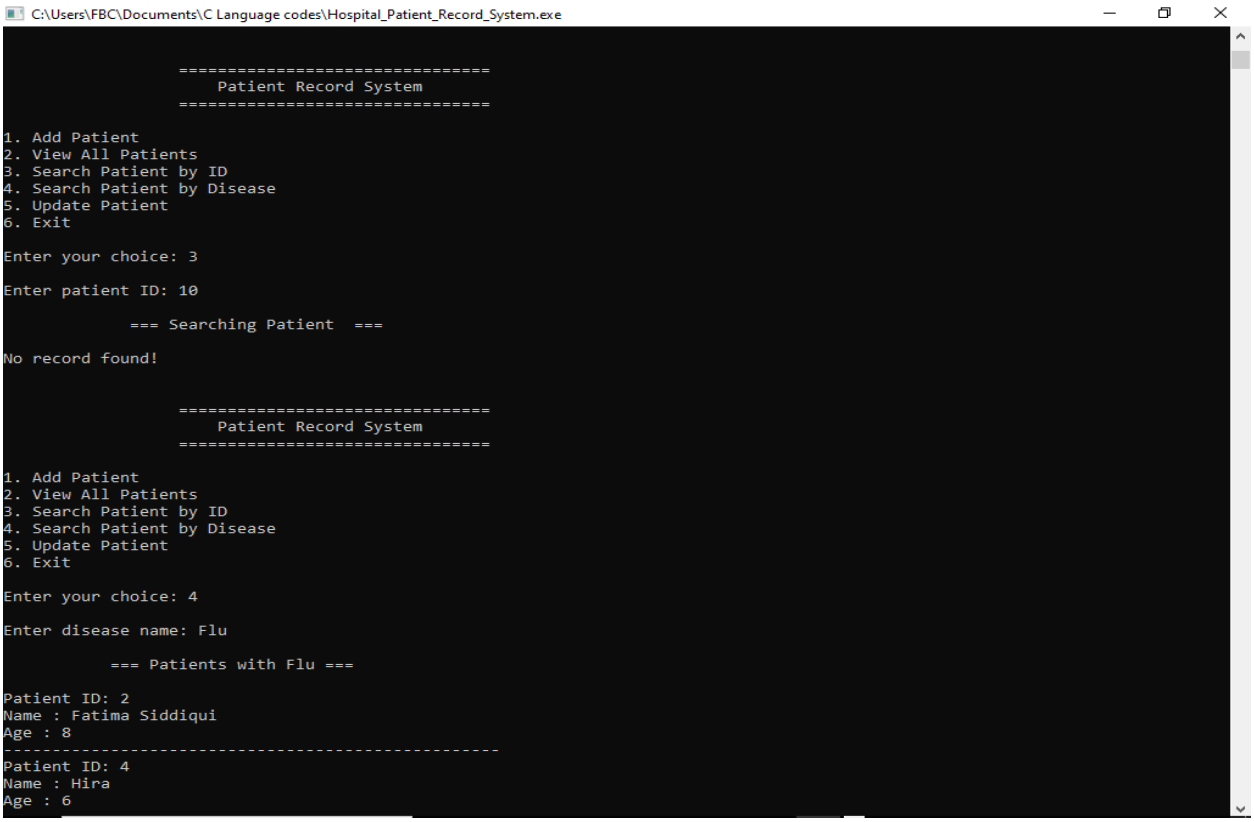
```
        printf("6. Exit\n\n");

        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1: add_patient();
                                                                    break;
            case 2: view_patient();
                                                                    break;
            case 3: search_patient_by_id();
                                                                    break;
            case 4: search_patient_by_disease();
                                                                    break;
            case 5: update_patient();
                                                                    break;
            case 6:
                printf("\n Exiting... Goodbye!\n\n");
                return 0;
            default:
                printf("\nInvalid choice! Enter again.\n\n");
        }
    }

    return 0;
}
```

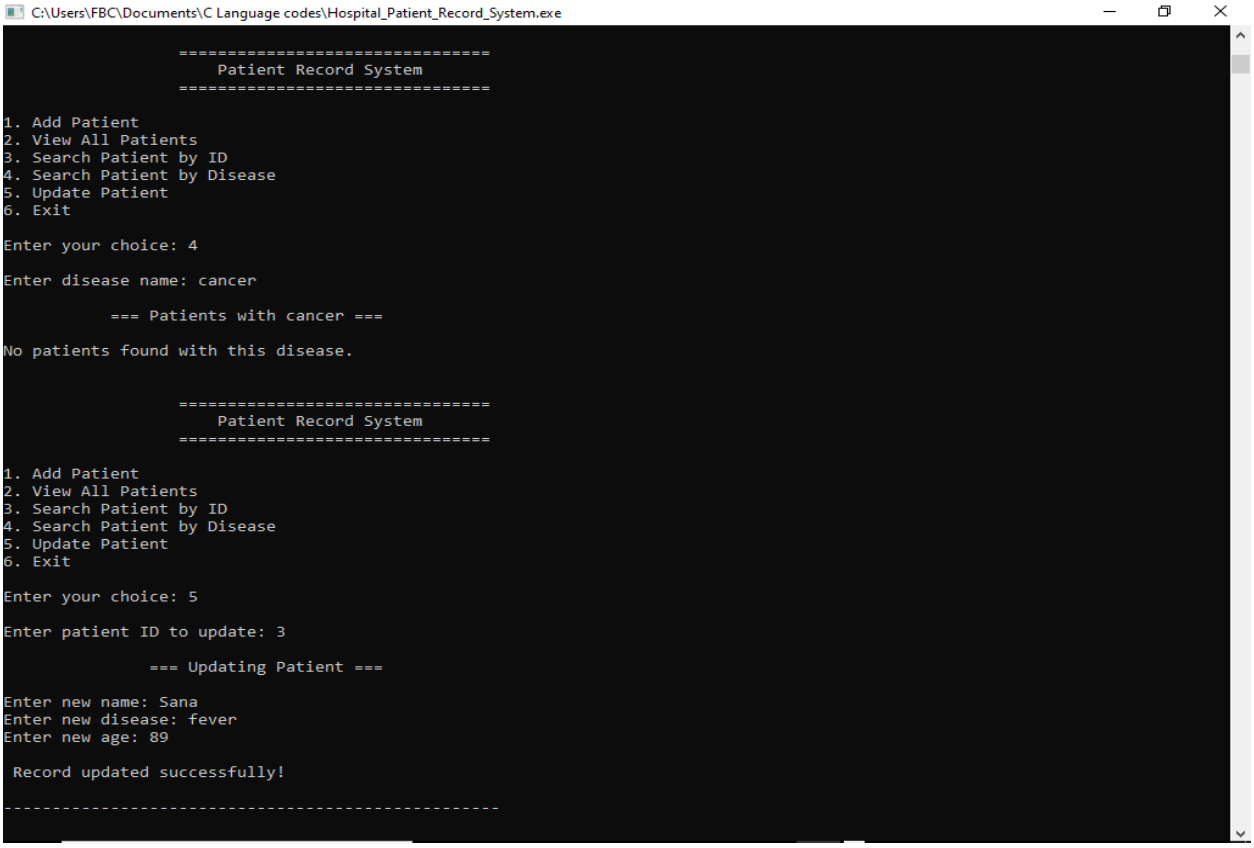
OUTPUT:

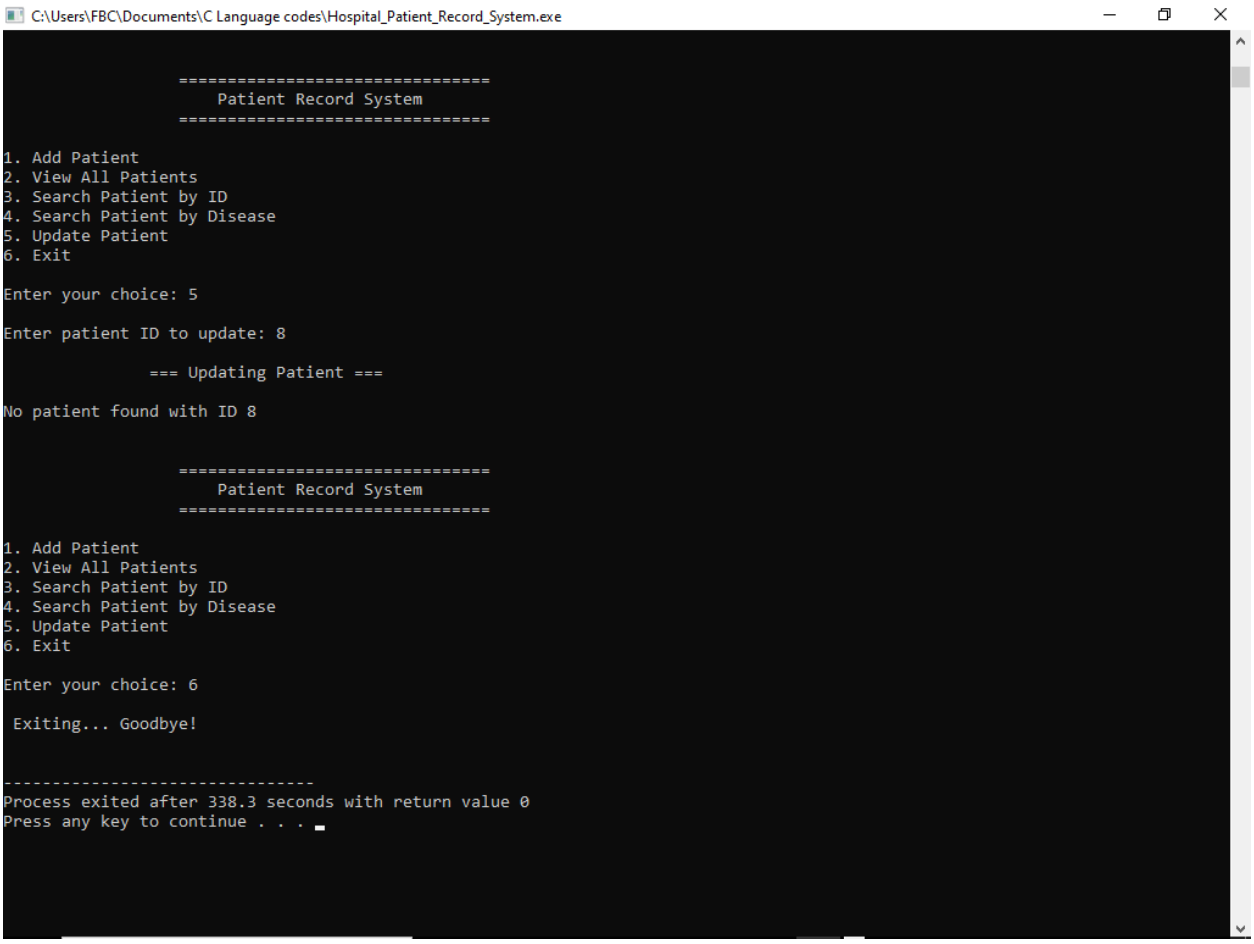

C:\Users\FBC\Documents\C Language codes\Hospital_Patient_Record_System.exe
==============================
Patient Record System
==============================
1. Add Patient
2. View All Patients
3. Search Patient by ID
4. Search Patient by Disease
5. Update Patient
6. Exit
Enter your choice: 2
=== All Patients ===
Patient ID: 1
Name : Muhammad Omer
Disease : Dengue
Age : 25
Patient ID: 2
Name : Fatima Siddiqui
Disease : Flu
Age : 8
Patient ID: 3
Name : Zymal Khan
Disease : Sickle Cell Anemia
Age : 18
==============================
Patient Record System
==============================
1. Add Patient
2. View All Patients
3. Search Patient by ID
4. Search Patient by Disease
5. Update Patient
6. Exit
Enter your choice: 1


C:\Users\FBC\Documents\C Language codes\Hospital_Patient_Record_System.exe
6. Exit
Enter your choice: 1
=== Adding Patient ===
Enter name: Hira
Enter disease: Flu
Enter age: 6
Patient added successfully!
==============================
Patient Record System
==============================
1. Add Patient
2. View All Patients
3. Search Patient by ID
4. Search Patient by Disease
5. Update Patient
6. Exit
Enter your choice: 3
Enter patient ID: 2
=== Searching Patient ===
Record found!
Patient ID: 2
Name : Fatima Siddiqui
Disease : Flu
Age : 8

HOSPITAL PATIENT RECORD SYSTEM







RESULTS:

- Successfully added and displayed multiple patient records.
- Search operations by ID and disease performed accurately.
- Updates to existing records reflected immediately.
- Menu interface functioned smoothly without logical errors.
- System validated through multiple input test cases.

ASSUMPTIONS:

- A maximum of 50 patient records can be managed.
- The system runs in a standard console-based C environment.
- Data is entered correctly by users (no validation errors).
- Data is stored temporarily (no file handling implemented).

CONCLUSION:

The Patient Record System successfully establishes a **reliable digital framework** for keeping track of patient details. By storing all essential information in an organized manner, the system ensures that data is consistent and ready for immediate access whenever needed. The menu-driven design and modular functions provide a clear structure, allowing smooth execution of tasks without confusion.

The system highlights how even a simple software solution can **streamline administrative tasks**, reduce the possibility of mistakes, and maintain proper records over time. Its straightforward architecture also allows easy expansion, making it suitable for **smaller healthcare centers** as well as for learning purposes.

Overall, this project proves that **well-structured digital tools** can bring efficiency, order, and reliability to patient management, supporting better operational practices in healthcare environments.