Project Title : Hospital Patent Data Management

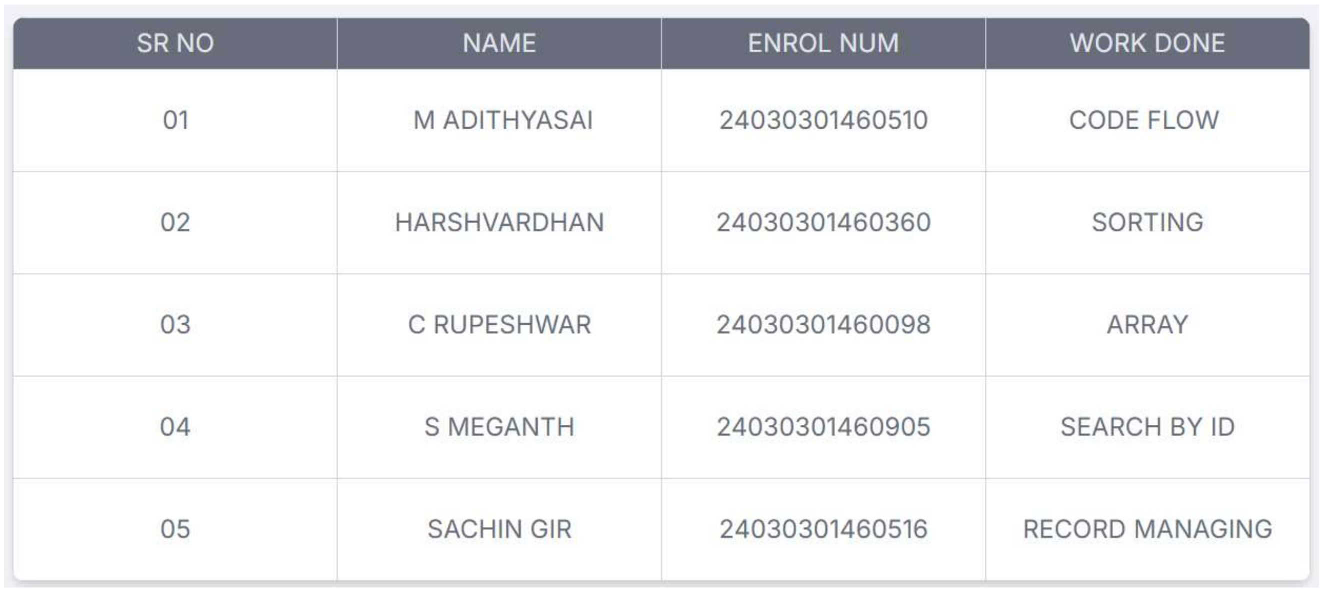
Table of Contents

1. Introduction

Every day, hospitals handle a vast amount of patient data, ranging from basic information like names and ages to medical conditions and past treatments. Accurate and prompt management of this data may mean the difference between critical delays and timely care.

The "Hospital Patient Data Management System," which is the name of our project, is a straightforward but useful program that helps store patient information, organize it (for instance, by age), and enable fast patient ID searches. Its purpose is to make information easily accessible and up-to-date, which is crucial in hectic hospital settings.

2. Team Members:



SR NO	NAME	ENROL NUM	WORK DONE
01	M ADITHYASAI	24030301460510	CODE FLOW
02	HARSHVARDHAN	24030301460360	SORTING
03	C RUPESHWAR	24030301460098	ARRAY
04	S MEGANTH	24030301460905	SEARCH BY ID
05	SACHIN GIR	24030301460516	RECORD MANAGING

3. Objective

The purpose of this project is to build a simple program that helps hospitals manage patient records.
It will:

- Save patient details like ID, name, age, and disease.
- Arrange patients by age so it's easier to see who is older or younger.
- Find a patient's details quickly by searching with their ID.
- Show all patient records in a clear list.

This makes it easier for hospital staff to handle information without wasting time.

4. Problem Statement

Hospitals need to keep patient information well-organized so that it can be found quickly, especially in emergencies.
Without a proper system, finding a patient's details can waste valuable time.
This project solves the problem by:

- Storing all patient details (ID, name, disease, age) in one place.
- Sorting patient records by age to make viewing easier.
- Searching for a patient using their ID in just a few seconds.

It’s a simple but effective way to manage patient data for better hospital efficiency.

5. System Requirements

Software:

- Turbo C++ or Dev C++ (to run the program)
- Any text editor like VS Code (for writing and editing code)

Hardware:

- Minimum 4 GB RAM
- Processor with at least 1.0 GHz speed
- Laptop or desktop computer with keyboard and monitor

6. 📚 Data Structures Used

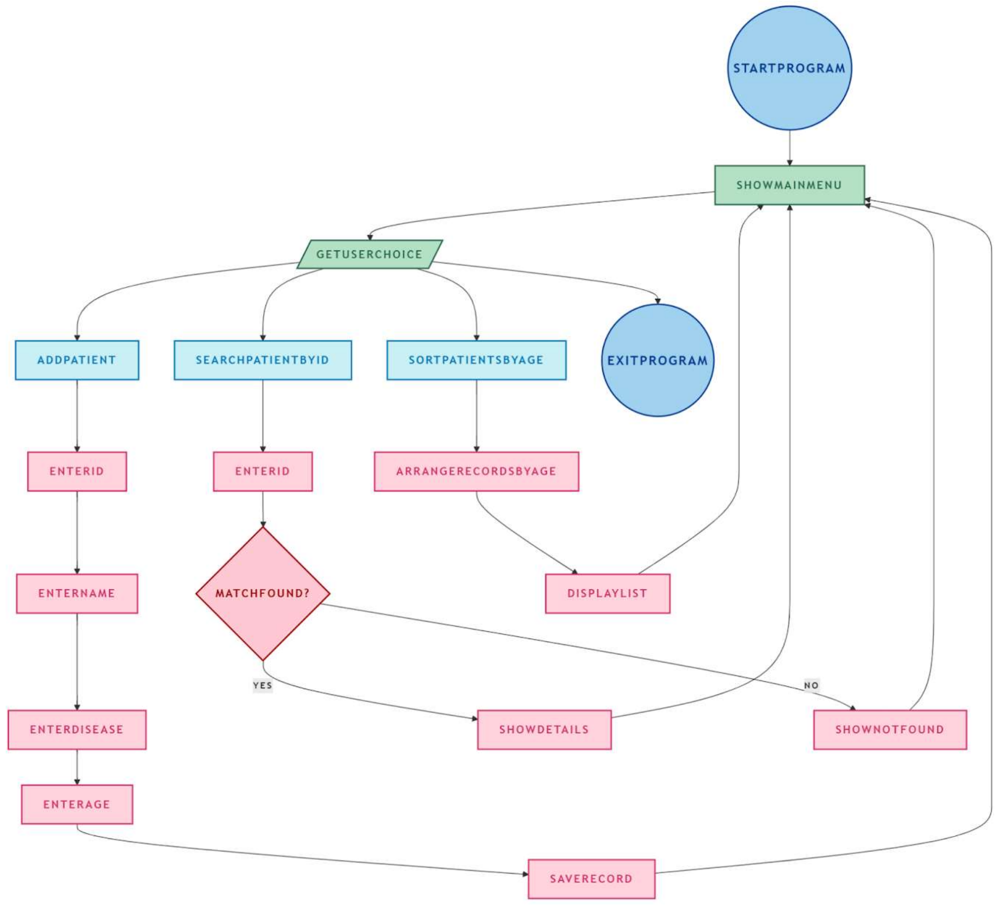
- **Array of Structures** – Used to store all patient details together (ID, name, disease, and age) in one place for easy access.
- **Bubble Sort** – Used to arrange patient records in order of age, from youngest to oldest.
- **Linear Search** – Used to quickly find a patient’s record by their ID.

Code -

```
#include <stdio.h>
#include <string.h>
#define MAX 1000
struct Patient {
 int id;
 char name[50];
 char disease[100];
 int age;
};
struct Patient patients[MAX];
int count = 0;
void addPatient() {
 if (count >= MAX) {
 printf("Patient list is full!\n");
 return;
 }
 printf("\\n--- Add Patient ---\\n");
 printf("Enter ID: ");
```

```c
 scanf("%d", &patients[count].id);
 printf("Enter Name (no spaces): ");
 scanf("%s", patients[count].name);
 printf("Enter Disease (no spaces): ");
 scanf("%s", patients[count].disease);
 printf("Enter Age: ");
 scanf("%d", &patients[count].age);
 count++;
 printf("Patient added successfully.\\n");
}
void sortByAge() {
 for (int i = 0; i < count - 1; i++) {
 for (int j = 0; j < count - i - 1; j++) {
 if (patients[j].age > patients[j + 1].age) {
 struct Patient temp = patients[j];
 patients[j] = patients[j + 1];
 patients[j + 1] = temp;
 }
 }
 }
 printf("\\n--- Patients Sorted by Age ---\\n");
 for (int i = 0; i < count; i++) {
 printf("ID: %d, Name: %s, Age: %d, Disease: %s\\n", patients[i].id, patients[i].name,
patients[i].age, patients[i].disease);
 }
}
void searchByID() {
 int id, found = 0;
 printf("\\nEnter Patient ID to search: ");
 scanf("%d", &id);
 for (int i = 0; i < count; i++) {
 if (patients[i].id == id) {
 printf("\\n--- Patient Found ---\\n");
 printf("ID: %d\\n", patients[i].id);
 printf("Name: %s\\n", patients[i].name);
 printf("Disease: %s\\n", patients[i].disease);
 printf("Age: %d\\n", patients[i].age);
 found = 1;
 break;
 }
 }
 if (!found) {
 printf("No patient found with ID %d.\\n", id);
 }
}
int main() {
 int choice;
 while (1) {
 printf("\\n--- Hospital System ---\\n");
 printf("1. Add Patient\\n");
 printf("2. Search Patient by ID\\n");
 printf("3. Sort by Age\\n");
 printf("4. Exit\\n");
 printf("Enter your choice: ");
 scanf("%d", &choice);
 switch (choice) {
 case 1: addPatient(); break;
 case 2: searchByID(); break;
 case 3: sortByAge(); break;
 case 4: return 0;
 default: printf("Invalid choice. Try again.\\n");
 }
 }
}
```

Flowchart of Program



7. Modules and Description

- **Input Module** – Lets the user add new patient records by entering their ID, name, disease, and age.
- **Display Module** – Shows all saved patient details in a neat and readable format.
- **Sorting Module** – Arranges patient records in order of age, from youngest to oldest, so hospital staff can prioritize as needed.
- **Searching Module** – Finds and shows the details of a specific patient using their ID, making it quick to locate their record.

8. Algorithms Used

A. Sorting by Age – Bubble Sort

Step 1: Repeat for each patient (except the last one)

Step 2: Compare current patient's age with the next patient's age

Step 3: If current age > next age → Swap them

Step 4: Continue until the list is sorted from youngest to oldest

```c
void sortByAge() {
    for (int i = 0; i < count - 1; i++) {
        for (int j = 0; j < count - i - 1; j++) {
            if (patients[j].age > patients[j + 1].age) {
                struct Patient temp = patients[j];
                patients[j] = patients[j + 1];
                patients[j + 1] = temp;
            }
        }
    }
    printf("\n--- Patients Sorted by Age ---\n");
    for (int i = 0; i < count; i++) {
        printf("ID: %d, Name: %s, Age: %d, Disease: %s\n",
               patients[i].id, patients[i].name, patients[i].age, patients[i].disease);
    } }
```

B. Searching by ID – Linear Search

Step 1: Enter the patient ID to search

Step 2: Go through each patient in the list one by one

Step 3: If ID matches → Display patient details

Step 4: If no match is found → Show "Patient Not Found"

```c
void searchByID() {
    int id, found = 0;
    printf("\nEnter Patient ID to search: ");
    scanf("%d", &id);
    for (int i = 0; i < count; i++) {
        if (patients[i].id == id) {
            printf("\n--- Patient Found ---\n");
            printf("ID: %d\n", patients[i].id);
            printf("Name: %s\n", patients[i].name);
            printf("Disease: %s\n", patients[i].disease);
            printf("Age: %d\n", patients[i].age);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("No patient found with ID %d.\n", id);
    } }
```

C. Displaying Records

Step 1: Loop through all stored patient records

Step 2: Print ID, name, disease, and age for each patient

Step 3: End when all records are shown

```c
void displayPatients() {

    printf("\n--- Patient Records ---\n");

    for (int i = 0; i < count; i++) {

        printf("ID: %d, Name: %s, Age: %d, Disease: %s\n",

                 patients[i].id, patients[i].name, patients[i].age, patients[i].disease);

    }

}
```

9. Sample Input/Output:

1. Input:

```
--- Hospital System ---
1. Add Patient
2. Search Patient by ID
3. Sort by Age
4. Exit
Enter your choice: 1

--- Add Patient ---
Enter ID: 201
Enter Name (no spaces): Alice
Enter Disease (no spaces): Flu
Enter Age: 29

--- Hospital System ---
1. Add Patient
2. Search Patient by ID
3. Sort by Age
4. Exit
Enter your choice: 1

--- Add Patient ---
Enter ID: 105
Enter Name (no spaces): Bob
Enter Disease (no spaces): Malaria
Enter Age: 35

--- Hospital System ---
1. Add Patient
2. Search Patient by ID
```

```
3. Sort by Age
4. Exit
Enter your choice: 1

--- Add Patient ---
Enter ID: 312
Enter Name (no spaces): Carol
Enter Disease (no spaces): Dengue
Enter Age: 27

--- Hospital System ---
1. Add Patient
2. Search Patient by ID
3. Sort by Age
4. Exit
Enter your choice: 3
```

2. Stored Output:

```
3
201 , Alice , Flu , 29
105 , Bob , Malaria , 35
312 , Carol , Dengue , 27
Sort by Age
```

3. Sorting by Age (Ascending Age):

```
ID: 312, Name: Carol, Age: 27, Disease: Dengue
ID: 201, Name: Alice, Age: 29, Disease: Flu
ID: 105, Name: Bob, Age: 35, Disease: Malaria
```

4. Sorting by Age (Reverse Order — Descending Age): *(If implemented)*

```
ID: 105, Name: Bob, Age: 35, Disease: Malaria
ID: 201, Name: Alice, Age: 29, Disease: Flu
ID: 312, Name: Carol, Age: 27, Disease: Dengue
```

5. Searching by Patient ID:

```
Enter Patient ID to search: 201

--- Patient Found ---
ID: 201
Name: Alice
Disease: Flu
Age: 29
```

10. ✅ Test Cases:

Test Case	Input	Sorting or Searching
Case 1	201, Alice, Flu, 29 105, Bob, Malaria, 35 312, Carol, Dengue, 27	Patients sorted by Age (asc): Rank 1: ID 312, Carol, Age: 27, Disease: Dengue Rank 2: ID 201, Alice, Age: 29, Disease: Flu Rank 3: ID 105, Bob, Age: 35, Disease: Malaria
Case 2	201, Alice, Flu, 29 105, Bob, Malaria, 35 312, Carol, Dengue, 27	Patients sorted by ID (asc): Rank 1: ID 105, Bob, Age: 35, Disease: Malaria Rank 2: ID 201, Alice, Age: 29, Disease: Flu Rank 3: ID 312, Carol, Age: 27, Disease: Dengue
Case 3	201	Patient found:
Case 4	500	No patient found with ID 500

11.Future Enhancements:

1. **Allow multi-word input** for patient names and diseases (using `fgets` instead of `scanf`).
2. **Case-insensitive search** for patient names and diseases.
3. **Multiple sorting options** – by ID, Age, Name, or Disease.
4. **Edit patient details** after they are added.
5. **Delete patient records** from the system.
6. **Data persistence** – save and load patient data from a file (so it remains after the program closes).
7. **Input validation** – prevent invalid IDs, negative ages, or duplicate IDs.
8. **Graphical or web-based interface** for easier interaction.

9. **Search by partial name or disease** (substring search).
10. **Statistics reports** – average patient age, count by disease, oldest/youngest patient.

12. Conclusion:

Basic features like adding patient records, searching by ID, and sorting by age are all successfully implemented by the hospital management system. It offers a straightforward and efficient interface for handling minor patient data in a medical facility. The program is easily extensible for future development because, despite its simplicity, it establishes the groundwork for more sophisticated features like persistent storage, improved search options, and improved user interaction.

13. References:

1. Chatgpt / NotebookLM
2. *Data Structures classes by Jashwanth Sir*

—-----END—----