Hospital Management System –

Using Priority Queue

Abstract

The Hospital Management System is a software application designed to facilitate the efficient management of patients in a hospital setting. The system allows hospital staff to add new patients, prioritize their treatment based on the severity of their condition, serve patients in the order of their priority, and maintain a list of patients who have been added to the system.

This project aims to streamline the patient management process, ensuring that patients are attended to promptly and effectively, and reducing the overall administrative burden on hospital staff.

Introduction

In a hospital, managing patients effectively is crucial for providing quality healthcare services. Traditional manual methods of handling patient queues can lead to inefficiencies and delays, which may impact patient outcomes.

To address these challenges, the Hospital Management System offers an automated solution for managing patient queues based on their priority. The

system provides a user-friendly interface for hospital staff to enter patient details and assign priorities to patients based on their medical condition.

Features:

Add Patient:

Hospital staff can add new patients to the system by providing their name, age, weight, and selecting a priority based on the severity of their condition. The priority determines the order in which patients will be served.

Serve Patient:

The system allows hospital staff to serve patients in the order of their priority. Patients with higher priority levels, indicating more critical conditions, will be served first.

Display Non-Served Patients List:

Hospital staff can view the list of patients who are waiting to be served. This list is arranged according to the priority assigned to each patient.

Display All Patients Details:

The system maintains a separate list of all patients who have been added to the system, including those who have already been served. This list provides a comprehensive view of all patients' details.

How to use:

1) Add a patient:
   Users can add a patient to the priority queue by providing the patient's name, age, weight, and priority level.

2) Serve a patient:
   The program dequeues and serves the patient with the highest priority from the priority queue.

3) Display non-served patients list:
   It displays the patients still waiting in the priority queue.

4) Display non-served patients list:
   It displays the patients still waiting in the priority queue.

5) Display all patients details:
   It displays the patients still waiting in the priority queue.

6) Exit:
   Users can choose to exit the program.

Here's how to use the Hospital Management System:

1. Run the program:Compile and run the C program in your preferred C compiler.

2. Menu Options: After running the program, you will see a menu with several options. The program will prompt you to enter your choice by typing a number from 1 to 5.

3. Adding a patient: To add a patient, choose option 1. You will be asked to enter the patient's name, age, weight, and priority. Based on the priority, the patient will be added to the priority queue.

4. Serving a patient: To serve a patient, choose option 2. The patient with the highest priority will be dequeued and their details will be displayed.

5. Display non-served patients list: Choosing option 3 will display the details of all patients still in the priority queue.

6. Display all patients' details: Option 4 will show the details of all the patients that have been added to the system.

7. Exiting the program: To exit the program, choose option 5.

Explaining each part of code:

Before Main Function:

```
typedef struct
{
    char name[50];
    int age;
    int wt;
    int priority;
} Patient;

typedef struct
{
    Patient items[MAX_SIZE];
    int rear;
} PriorityQueue;
```

In this C code, the provided `typedef struct` statements define two custom data types: `Patient` and `PriorityQueue`, which are used in the Hospital Management System.

1. `Patient` struct:

The `Patient` struct represents a patient and contains the following member variables:

- `name`: An array of characters with a size of 50, used to store the name of the patient.

- `age`: An integer variable that stores the age of the patient.

- `wt`: An integer variable that stores the weight of the patient.

- `priority`: An integer variable that represents the priority level of the patient.

The `Patient` struct is designed to store information about individual patients, including their personal details (name, age, weight) and their priority level in the hospital system. This priority level will be used to manage the patients in the `PriorityQueue`.

2. `PriorityQueue` struct:

The `PriorityQueue` struct represents a queue data structure, specifically a priority queue, which is a collection of patients organized based on their priority levels.

The `PriorityQueue` struct contains the following member variables:

- `items`: An array of `Patient` structs with a size of `MAX_SIZE`. This array is used to store the patients in the priority queue.

- `rear`: An integer variable representing the index of the last element in the priority queue. It points to the end of the queue, where new patients will be added.

The `rear` variable acts as a pointer that keeps track of the current position in the `items` array where new patients will be inserted. When a new patient is added, the `rear` is updated to point to the next available position in the `items` array. Similarly, when a patient is dequeued from the priority queue, the `rear` is adjusted to reflect the updated size of the queue.

Main Function:

```c
int main() {
    PriorityQueue pq;
    initQueue(&pq);

    PatientList addedPatients;
    addedPatients.count = 0;

    int choice;
    do {
        printf(COLOR_BOLD "\n\t\t\tHospital Management System\n" COLOR_RESET);
        printf(COLOR_YELLOW "1. Add patient\n");
        printf("2. Serve patient\n");
        printf("3. Display non served patients list\n");
        printf("4. Display all patients details\n");
        printf("5. Exit\n" COLOR_RESET);
        printf(COLOR_BLUE "Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: {
                Patient p;
                printf("Enter patient name: ");
                scanf(" %[^\n]", p.name);
                printf("Enter patient age: ");
                scanf("%d", &p.age);
                printf("Enter weight:");
                scanf("%d", &p.wt);
                printf(COLOR_GREEN "Please refer to this table for your disease:\n");
                // Print the table
                printf("1. Emergency (accidents, cardiac arrest..)\n");
                printf("2. Chronic disease\n");
                printf("3. Illness\n");
                printf("4. General checkup\n" COLOR_RESET);
                printf(COLOR_BLUE "Enter patient priority: ");
                scanf("%d", &p.priority);
                insert(&pq, p);
                addToPatientList(&addedPatients, p); // Add patient to the separate list
                break;
            }
            case 2: {
                Patient p = dequeue(&pq);
                if (p.priority != 0) {
                    printf(COLOR_GREEN "Serving patient according to their priority:\n");
                    displayPatient(p);
                }
                break;
            }
            case 3: {
                displayQueue(&pq);
                break;
            }
            case 4: {
                printf(COLOR_BLUE "Added patients list:\n" COLOR_RESET);
                displayQueuer(&addedPatients); // Display the separate list of added patients
                break;
            }
            case 5: {
                printf(COLOR_RED "Exiting the program.\n" COLOR_RESET);
                break;
            }
            default: {
                printf(COLOR_RED "Invalid choice. Please try again.\n" COLOR_RESET);
                break;
            }
        }
    } while (choice != 5);

    return 0;
}
```

1. When the user selects option 1, they are prompted to enter the patient's details such as name, age, weight, and priority. The insert function is then called to add the patient to the priority queue. Additionally, the addToPatientList function is called to maintain a separate list of all patients who have been added to the system.

2. Option 2 allows the user to serve the patient with the highest priority. The dequeue function is used to remove the patient from the front of the priority queue and display their details.

3. Option 3 displays the list of patients who are waiting to be served (i.e., the contents of the priority queue). The displayQueue function is called for this purpose.

4. Option 4 displays the complete list of patients who have been added to the system. The displayQueuer function is used to display the contents of the separate patient list.
5. Option 5 allows the user to exit the program.

Insert Function:

```
void insert(PriorityQueue* pq, Patient p)
{
    if (isFull(pq))
    {
        printf("Priority queue is full. Cannot insert more patients.\n");
        return;
    }

    int i;
    // Find the correct position to insert the patient based on priority
    for (i = pq->rear; i >= 0; i--)
    {
        if (pq->items[i].priority > p.priority)
            pq->items[i + 1] = pq->items[i];
        else
            break;
    }
    pq->items[i + 1] = p;
    pq->rear++;
}
```

1. The insert function is responsible for adding a patient to the priority queue. It takes the address of the PriorityQueue (pq) and the patient (p) as arguments.

2. The function first checks if the priority queue is full using the isFull function. If the queue is full, it prints an error message and returns without adding the patient.

3. If the queue is not full, the function finds the correct position to insert the new patient based on their priority. It iterates through the queue from the end (pq->rear) towards the beginning to find the appropriate position.

4. During the iteration, if the current patient's priority is greater than the new patient's priority, it shifts the current patient one position up in the queue to make space for the new patient.

5. Once the correct position is found, the new patient is inserted into the queue at that position, and the rear pointer is incremented to reflect the new end of the queue.

Dequeue Function:

```
Patient dequeue(PriorityQueue* pq)
{
    if (isEmpty(pq))
    {
        printf("Priority queue is empty. No patients to dequeue.\n");
        Patient emptyPatient = { "", 0, 0 };//emptyPatient is avariable of Patient
        return emptyPatient;                //and intialising to NULL
    }
    Patient front = pq->items[0];
    for (int i = 0; i < pq->rear; i++)
    {
        pq->items[i] = pq->items[i + 1];
    }
    pq->rear--;
    return front;
}
```

1. The dequeue function is responsible for removing and returning the patient with the highest priority from the priority queue.

2. The function first checks if the priority queue is empty using the isEmpty function. If the queue is empty, it prints an error message and returns an empty Patient (i.e., a patient with blank fields).

3. If the queue is not empty, the function retrieves the front patient (highest priority) from the queue and stores it in the variable front.

4. The function then shifts all the patients one position forward to remove the front patient. This is done by iterating through the queue and copying each patient to the position of the previous patient.

5. Finally, the rear pointer is decremented to reflect the removal of the front patient from the queue, and the function returns the removed patient (front).

Display Function’s:

```c
//to display served Patient details
void displayPatient(Patient p)
{
    printf("Name: %s\n", p.name);
    printf("Age: %d\n", p.age);
    printf("weight: %d\n",p.wt);
    printf("Priority: %d\n", p.priority);
    printf("------------------------\n");
}
//to display all patients priority
void displayQueue(PriorityQueue* pq)
{
    if (isEmpty(pq))
    {
        printf("Priority queue is empty. No patients to display.\n");
        return;
    }

    printf("Patients in the priority queue:\n");
    for (int i = 0; i <= pq->rear; i++)
    {
        displayPatient(pq->items[i]);
    }
}
void displayQueuer(PriorityQueue* pq)
{
    if (isEmpty(pq))
    {
        printf("Priority queue is empty. No patients to display.\n");
        return;
    }

    printf("Patients in the priority queue:\n");
    for (int i = 0; i <= (pq->rear)-1; i++)
    {
        displayPatient(pq->items[i]);
    }
}
```

1. The displayPatient function is responsible for displaying the details of a single patient. It takes a Patient struct as input and prints the patient's name, age, weight, and priority.

2. The displayQueue function is used to display the list of patients in the priority queue. It takes the address of the PriorityQueue (pq) as input.

3. The function first checks if the queue is empty using the isEmpty function. If the queue is empty, it prints a message indicating that there are no patients to display.

4. If the queue is not empty, the function prints a header and then iterates through the queue to display each patient using the displayPatient function.

5. The displayQueuer function is similar to displayQueue, but it is used to display the separate list of added patients. The difference is that it iterates up to pq->rear - 1 to avoid displaying the patient who was last served and removed from the priority queue.

Note:

The program uses ANSI escape codes for colours to make the interface more user-friendly. These codes may not work in all terminals. If you encounter issues with colours, you can modify the output format or run the program in a terminal that supports ANSI escape codes.

Keep in mind that this is a simple implementation, and in a real-world scenario, you may need to add more features and error handling to make the system more robust. Nonetheless, it provides a basic framework for a Hospital Management System using a priority queue.

Program:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_SIZE 100

typedef struct {
    char name[50];
    int age;
    int wt;    int priority;
} Patient;

typedef struct {
    Patient items[MAX_SIZE];
    int rear;
} PriorityQueue;

void insert(PriorityQueue* pq, Patient p);
Patient dequeue(PriorityQueue* pq);
void displayPatient(Patient p);
void displayQueue(PriorityQueue* pq);
void insert(PriorityQueue* pq, Patient p);
int isEmpty(PriorityQueue* pq);
void initQueue(PriorityQueue* pq);

// ANSI escape codes for colors
#define COLOR_RESET "\033[0m"
#define COLOR_BOLD "\033[1m"
#define COLOR_RED "\033[31m"
#define COLOR_GREEN "\033[32m"
#define COLOR_YELLOW "\033[33m"
#define COLOR_BLUE "\033[34m"

typedef struct {
    Patient items[MAX_SIZE];
    int count;
} PatientList;

void addToPatientList(PatientList* list, Patient p);

int main() {
    PriorityQueue pq;
    initQueue(&pq);

    PatientList addedPatients;
    addedPatients.count = 0;

    int choice;
    do {
        printf(COLOR_BOLD "\n\t\t\tHospital Management System\n" COLOR_RESET);
        printf(COLOR_YELLOW "1. Add patient\n");
        printf("2. Serve patient\n");
        printf("3. Display non served patients list\n");
        printf("4. Display all patients details\n");
        printf("5. Exit\n" COLOR_RESET);
        printf(COLOR_BLUE "Enter your choice: ");
        scanf("%d", &choice);
```

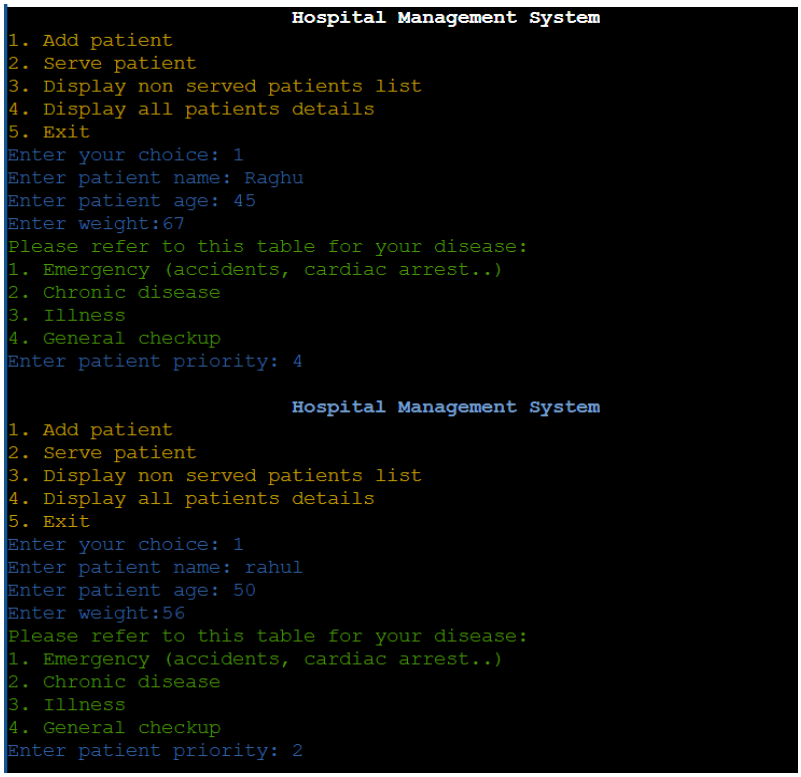
```c

        switch (choice) {
            case 1: {
                Patient p;
                printf("Enter patient name: ");
                scanf(" %[^\n]", p.name);
                printf("Enter patient age: ");
                scanf("%d", &p.age);
                printf("Enter weight:");
                scanf("%d", &p.wt);
                printf(COLOR_GREEN "Please refer to this table for your disease:\n");
                // Print the table
                printf("1. Emergency (accidents, cardiac arrest..)\n");
                printf("2. Chronic disease\n");
                printf("3. Illness\n");
                printf("4. General checkup\n" COLOR_RESET);
                printf(COLOR_BLUE "Enter patient priority: ");
                scanf("%d", &p.priority);
                insert(&pq, p);
                addToPatientList(&addedPatients, p); // Add patient to the separate list
                break;
            }
            case 2: {
                Patient p = dequeue(&pq);
                if (p.priority != 0) {
                    printf(COLOR_GREEN "Serving patient according to their priority:\n");
                    displayPatient(p);
                }
                break;
            }
            case 3: {
                displayQueue(&pq);
                break;
            }
            case 4: {
                printf(COLOR_BLUE "Added patients list:\n" COLOR_RESET);
                displayQueuer(&addedPatients); // Display the separate list of added patients
                break;
            }
            case 5: {
                printf(COLOR_RED "Exiting the program.\n" COLOR_RESET);
                break;
            }
            default: {
                printf(COLOR_RED "Invalid choice. Please try again.\n" COLOR_RESET);
                break;
            }
        }
    } while (choice != 5);

    return 0;
}

// ...

```

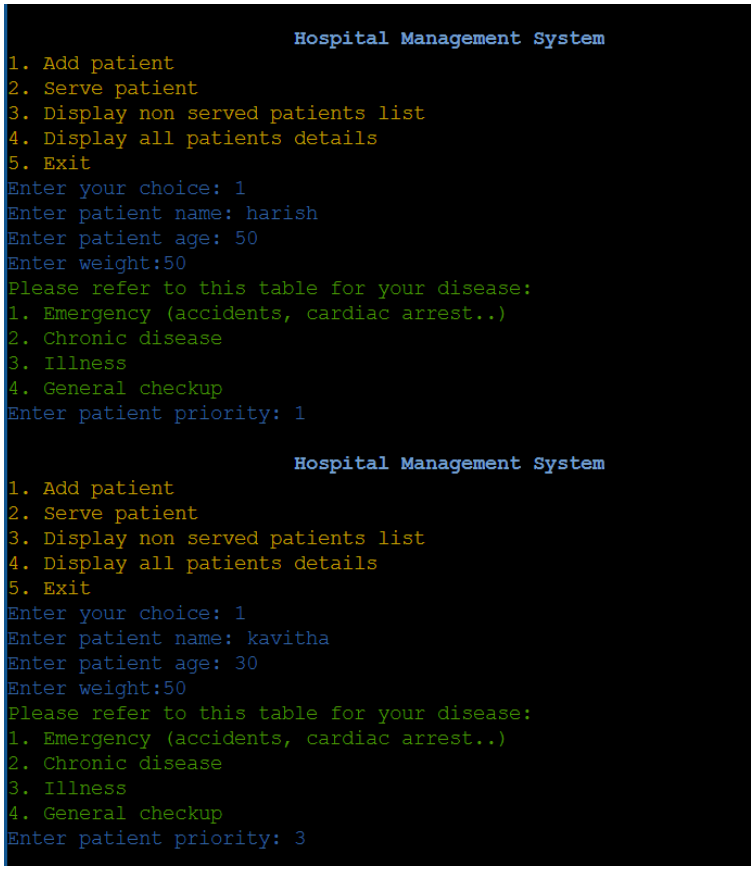
```c
void addToPatientList(PatientList* list, Patient p) {
    if (list->count >= MAX_SIZE) {
        printf("Added patients list is full. Cannot add more patients.\n");
        return;
    }
    list->items[list->count] = p;
    list->count++;
}
void initQueue(PriorityQueue* pq)
{
    pq->rear = -1;
}

//to check patients list(PriorityQueue)is empty or not
int isEmpty(PriorityQueue* pq)
{
    return pq->rear == -1;//it returns true if rear=-1
}

//to check if the patients list(PriorityQueue) is full or not
int isFull(PriorityQueue* pq)
{
    return pq->rear == MAX_SIZE - 1; // it returns true if rear= max-1
}

void insert(PriorityQueue* pq, Patient p)
{
    if (isFull(pq))
    {
        printf("Priority queue is full. Cannot insert more patients.\n");
        return;
    }

    int i;
    // Find the correct position to insert the patient based on priority
    for (i = pq->rear; i >= 0; i--)
    {
        if (pq->items[i].priority > p.priority)
            pq->items[i + 1] = pq->items[i];
        else
            break;
    }
    pq->items[i + 1] = p;
    pq->rear++;
}
Patient dequeue(PriorityQueue* pq)
{
    if (isEmpty(pq))
    {
        printf("Priority queue is empty. No patients to dequeue.\n");
        Patient emptyPatient = { "", 0, 0 };//emptyPatient is avariable of Patient
        return emptyPatient;                //and intialising to NULL
    }
    Patient front = pq->items[0];
    for (int i = 0; i < pq->rear; i++)
    {
        pq->items[i] = pq->items[i + 1];
```

```
    }
    pq->rear--;
    return front;
}
//to display served Patient details
void displayPatient(Patient p)
{
    printf("Name: %s\n", p.name);
    printf("Age: %d\n", p.age);
    printf("weight: %d\n",p.wt);
    printf("Priority: %d\n", p.priority);
    printf("------------------------\n");
}
//to display all patients priority
void displayQueue(PriorityQueue* pq)
{
    if (isEmpty(pq))
    {
        printf("Priority queue is empty. No patients to display.\n");
        return;
    }

    printf("Patients in the priority queue:\n");
    for (int i = 0; i <= pq->rear; i++)
    {
        displayPatient(pq->items[i]);
    }
}
void displayQueuer(PriorityQueue* pq)
{
    if (isEmpty(pq))
    {
        printf("Priority queue is empty. No patients to display.\n");
        return;
    }

    printf("Patients in the priority queue:\n");
    for (int i = 0; i <= (pq->rear)-1; i++)
    {
        displayPatient(pq->items[i]);
    }
}
```

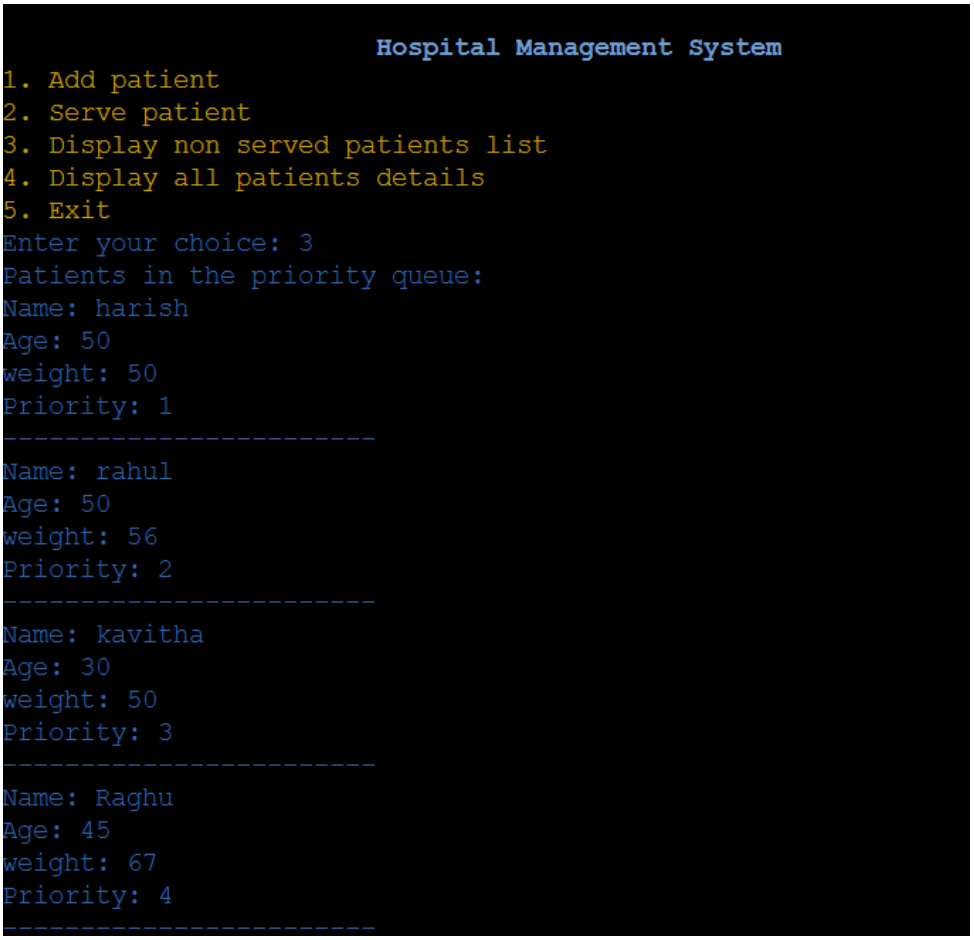
Input and Output:

Adding patients

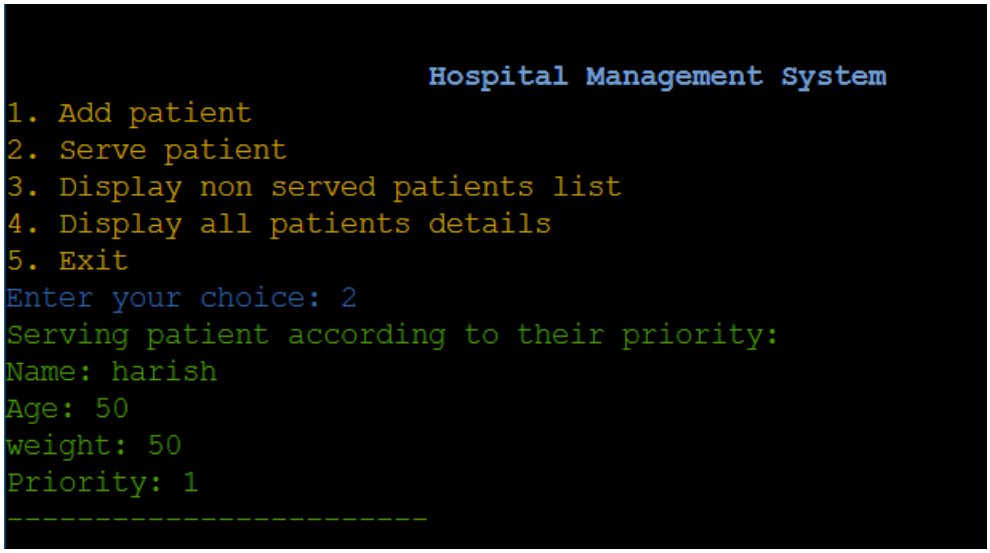




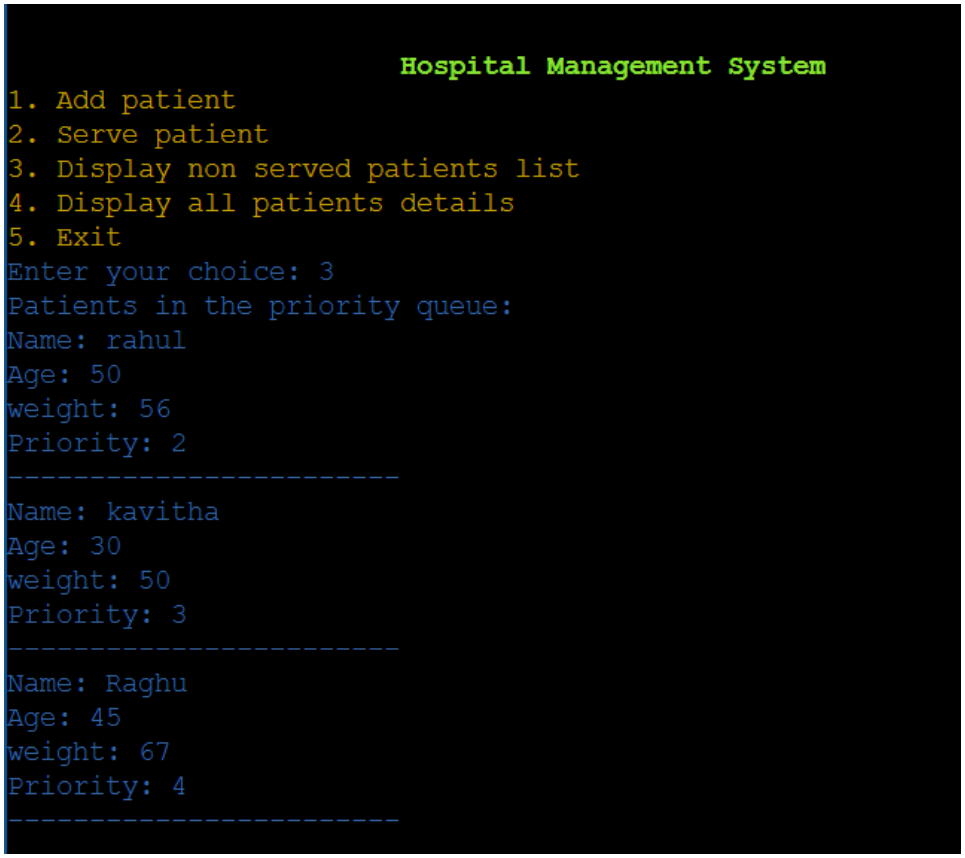
Displaying non served patients



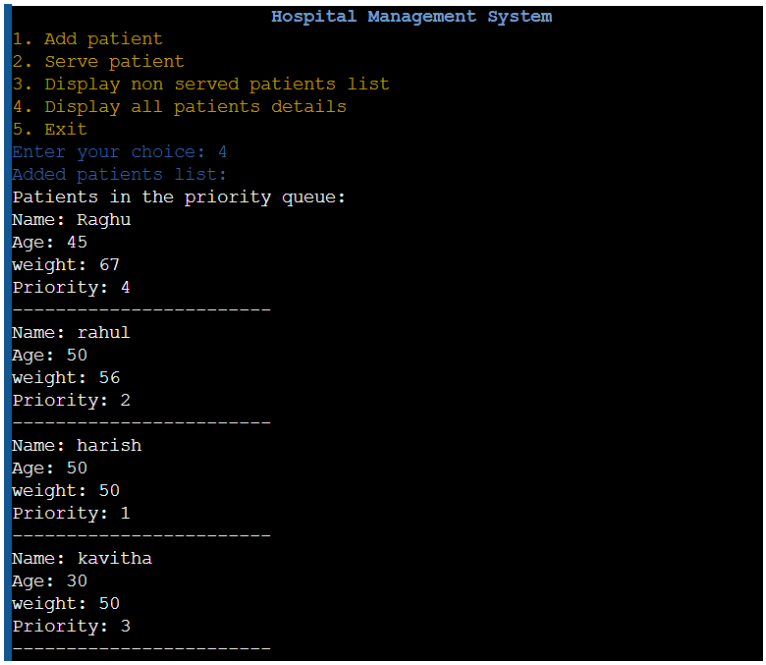
Serving Patient



Displaying non served patients after serving according to priority



Displaying all Patients



Conclusion:

The Hospital Management System provides an efficient solution for managing patient queues in a hospital setting. By utilizing a priority queue, the system ensures that patients with more critical conditions receive prompt attention. The project team successfully developed and implemented this system, contributing to improved patient care and hospital management.

Throughout the development process, the team collaborated effectively, with each member fulfilling their assigned roles and responsibilities. The project manager led the team with clear planning and organization, while the developers and testers worked diligently to create a reliable and functional system. The documentation and support team ensured that the project's documentation was comprehensive and provided assistance when needed.

In conclusion, the Hospital Management System project is a valuable tool for hospitals to enhance their patient management processes, optimize resources, and ultimately deliver better healthcare services. The team takes pride in this accomplishment and hopes that the system will have a positive impact on the healthcare industry.