**Patient Priority Management System**

**Name:** A. Sai lokesh

**Roll Number:** 24KB1A0501

**Course / Department:** CSE

**Date:** 01-05-2025

1. **Acknowledgement**

I would like to express my gratitude to my faculty and peers for their guidance and support throughout this project. Their insights and encouragement were instrumental in completing this work successfully.

1. **Abstract**

This project implements a patient queue management system that uses a priority queue based on a linked list. Patients are treated according to the urgency of their condition. Emergency patients are treated first, and others In the order of their arrival and severity.

1. **Introduction**

Hospitals need to treat patients based on the seriousness of their condition. This project simulates such a system using a priority queue. It helps understand how data structures can solve real-life problems.

1. **Objectives**

Implement a priority-based queue using a linked list

Ensure patients with higher emergency levels are treated first

Treat patients of the same priority based on arrival time

1. **System Requirements**

Software: GCC Compiler, Code::Blocks / VS Code

Hardware: Minimum 2 GB RAM, any basic PC or laptop

1. **Methodology**

The program was developed step-by-step:

Designed the structure for patient data

Used a linked list to manage the queue

Inserted patients based on their priority

Displayed and treated patients according to queue order

1. **Project Description**

Problem Statement: In real-world hospital systems, it’s important to treat patients based on urgency.

Proposed Solution: Use a priority queue to manage patient treatment order

Key Features:

Dynamic memory usage

Emergency patients get highest priority

FIFO order for same priority patients

1. **Algorithm**

**Step 1: Start**

**Step 2: Main Menu Logic**

**Declare an integer variable** choice.

**Repeat until** choice ==4 :

**Display name:**

Appoint Patient

Treat Patient

Display Records

Exit

**Read** choice.

**Perform based on** choice:

**If** choice == 1, **call** appoint\_patient().

**Else if** choice == 2, **call** treat\_patient().

**Else if** choice ==3, **call** diplay\_records().

**Else if** choice ==4, **display exit message.**

**Else,** display “**Invalid choice”.**

**Step 3: Appoint Patient**

**Create a new node.**

**Read patient’s** name, age, case type, **and** priority.

**Set** new\_node->next = NULL.

**If queue is empty OR** new\_node->priority **is higher (i.e., lower number)**   **than** head.

**Insert** new\_node **at the beginning.**

**Else:**

**Traverse the list to find correct position based on priority.**

**Insert** new\_node **after appropriate node.**

**Display “Patient Appointed Successfully”.**

**Step 4: Treat patient**

If queue is empty:

**Display** “No patients to treate”.

**Else:**

Remove the patient at the front.

Display patient’s details.

Free the memory of the treated patient.

Display “Patient treatement complete”.

**Step 5: Display Records**

If queue is empty:

**Display** “No patients appointed”.

**Else:**

Traverse from head node.

Display all patient’s details one by one.

**Step 6: End**

1. **Program Code**

#include<stdio.h>

#include<stdlib.h>

struct patient {

char name[20];

int age;

char case\_type[20];

int priority;

struct patient\* next;

};

struct patient\* head = NULL;

struct patient\* temp = NULL;

void clear\_buffer() {

while (getchar() != '\n');

}

void appoint\_patient() {

struct patient\* new\_patient = (struct patient\*)malloc(sizeof(struct patient));

if (new\_patient == NULL) {

printf("Memory allocation failed!\n");

return;

}

printf("\n--- Appoint New Patient ---\n");

printf("Enter patient name: ");

scanf("%s", new\_patient->name);

clear\_buffer();

printf("Enter age: ");

scanf("%d", &new\_patient->age);

clear\_buffer();

printf("Enter case type: ");

scanf("%s", new\_patient->case\_type);

clear\_buffer();

printf("Enter priority: ");

scanf("%d", &new\_patient->priority);

clear\_buffer();

new\_patient->next = NULL;

if (head == NULL || new\_patient->priority < head->priority) {

new\_patient->next = head;

head = new\_patient;

} else {

struct patient\* current = head;

while (current->next != NULL && current->next->priority <= new\_patient->priority) {

current = current->next;

}

new\_patient->next = current->next;

current->next = new\_patient;

}

printf("Patient appointed successfully.\n");

}

void treat\_patient() {

if (head == NULL) {

printf("\nNo patients to treat.\n");

return;

}

temp = head;

head = head->next;

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

printf("Name: %s\n", temp->name);

printf("Age: %d\n", temp->age);

printf("Case: %s\n", temp->case\_type);

printf("Priority: %d\n", temp->priority);

free(temp);

printf("Patient treatment complete.\n");

}

void display\_records() {

if (head == NULL) {

printf("\nNo patients appointed.\n");

return;

}

temp = head;

printf("\n\t---- PATIENT RECORDS ----\n");

while (temp != NULL) {

printf("Name: %s\n", temp->name);

printf("Age: %d\n", temp->age);

printf("Case: %s\n", temp->case\_type);

printf("Priority: %d\n", temp->priority);

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

temp = temp->next;

}

}

int main() {

int choice;

do {

printf("\n\n\t---- OPTIONS ----\n");

printf("1. Appoint Patient\n");

printf("2. Treat Patient\n");

printf("3. Display Records\n");

printf("4. Exit\n");

printf("Enter choice: ");

scanf("%d", &choice);

clear\_buffer();

switch (choice) {

case 1:

appoint\_patient();

break;

case 2:

treat\_patient();

break;

case 3:

display\_records();

break;

case 4:

printf("\nExiting... Have a nice day.\n");

break;

default:

printf("\nInvalid Choice. Try again.\n");

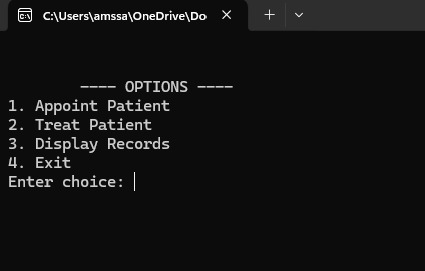
}

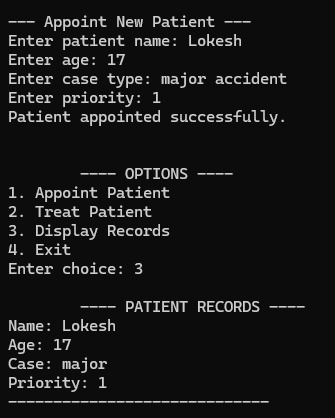
} while (choice != 4);

return 0;

}

1. **Output Screenshots**





1. **Testing / Validation**

The program was tested with:

Multiple patients of varying priority

Patients with same priority to check order

Emergency cases inserted after normal patients

1. **Limitations**

No GUI

No persistent storage of patient data after program ends

1. **Future Enhancements**

Add file storage to save patient records

Create a user-friendly GUI interface

Add categories for different departments (OPD, ICU, etc.)

1. **Conclusion**

This project helped me understand how linked lists can be used to implement real-world applications like priority queues. It also improved my logical thinking and coding skills in C.

1. **References**

Classroom notes

GeeksforGeeks (linked list tutorials)

TutorialsPoint – C Programming