**NBKR INSTITUTE OF SCIENCE AND TECHNOLOGY**

**HOSPITAL QUEUE MANAGEMENT SYSTEM**

**COURSE:** DATA STRUCTURES

**DEPARTMENT:** COMPUTER SCIENCE

**SECTION:** F

**YEAR:** 1

**SEMESTER:** 1

**DATE:** MAY 01 2025

**SUBMITTED TO:** ASHOK SELVA KUMAR E

**SUBMITTED BY:**

1. K. KUSHWANTH SAI 1 11(24KB1A05R3)11
2. N. VINEEL KRISHNA1 1(24KB1A05CE)11
3. M. PRAVEEN KUMAR1 1 1(24KB1A05X8)1
4. K. KUNDHAN PRAKASH 1(24KB1A05U0)1

**ACKNOWLEDGEMENT**

I would like to express my sincere gratitude to NBKR Institute of Science and Technology (NBKRIST) for providing the opportunity and resources to undertake this project.

I am deeply thankful to my project guide, Ashok Selva Kumar E, for their valuable guidance, continuous support, and insightful feedback throughout the development of the Hospital Queue Management System.

I also extend my appreciation to the faculty of the CSE Department for their encouragement and to my peers and family for their constant support and motivation.

**ABSTRACT**

The Hospital Queue Management System is a software-based solution designed to streamline patient flow in healthcare facilities. It automates patient registration, assigns queue positions based on priority (emergency, appointment, or walk-in), and updates status in real-time. The system improves efficiency by reducing waiting times, managing doctor availability, and securely storing patient visit data. It enhances the overall hospital experience for both patients and staff while ensuring better resource utilization and faster service delivery.

**INTRODUCTION**

The Hospital Queue Management System project is focused on developing an automated system that organizes and manages patient queues within hospitals or clinics. The system handles patient registration, assigns queue positions based on predefined priority rules (such as emergency, appointment, or walk-in), and facilitates smooth communication between patients and medical staff. It helps track patient status in real time, manages doctor availability, and ensures that healthcare delivery is orderly and efficient.

We chose this project because queue mismanagement is one of the most common and frustrating issues in healthcare facilities. Long wait times not only increase patient dissatisfaction but can also delay critical treatments. In many hospitals, manual queuing leads to confusion, bias, and stress for both patients and staff. With the increasing demand for digital solutions in healthcare, this project offers a practical and impactful way to address these operational challenges.

By developing this system, we aim to enhance patient experience, reduce administrative burden, and contribute to the ongoing digital transformation of healthcare services.

**OBJECTIVE**

The primary goal of the Hospital Queue Management System is to improve the efficiency and quality of patient handling in healthcare facilities through automation and real-time queue monitoring. This project aims to develop a user-friendly, reliable, and scalable system that benefits both patients and hospital staff.

Specific objectives include:

To automate patient registration and queue assignment based on priority levels such as emergency, scheduled appointments, and walk-ins.

To reduce patient waiting time and improve the overall experience by ensuring organized and timely consultations.

To optimize doctor availability and resource utilization by managing queues in real-time.

To maintain accurate records of patient visits and consultations for future reference and reporting.

To minimize human error and bias in queue handling through a fair and systematic process.

To provide transparency and live updates to patients regarding their queue position and expected waiting time.

To support hospital staff and administrators in monitoring, managing, and analyzing patient flow and service efficiency.

**SYSTEM REQUIRMENTS**

SOFTWARE: Turbo C ++, Github

**METHODOLOGY**

The methodology for this project was designed to address the research objectives and achieve the desired outcomes effectively. The project followed a systematic approach comprising the following key steps:

1. Problem Definition

Initially, the problem was clearly defined to outline the scope of the project. A detailed analysis of the subject matter was conducted to identify key challenges and formulate the research questions.

1. Literature Review

A comprehensive review of existing literature was conducted to understand the current state of knowledge in the area of study. Relevant articles, journals, books, and previous research reports were examined to establish a theoretical foundation.

1. Data Collection

Data was collected through [mention methods like surveys, experiments, interviews, etc.]. Primary and secondary data sources were used, and tools such as [mention any software or instruments used] were employed for gathering accurate and reliable data.

1. Data Analysis

The collected data was analyzed using [mention analysis methods, e.g., statistical analysis, qualitative coding, etc.]. The results were interpreted to identify trends, patterns, and insights relevant to the project objectives.

1. Implementation

Based on the findings from the data analysis, the next step was the implementation phase. [Describe any systems, prototypes, or solutions you developed or tested.]

1. Testing and Validation

The implemented solution was subjected to rigorous testing to ensure its functionality and reliability. [Mention the testing methods, e.g., A/B testing, performance evaluation, etc.]

1. Conclusion and Recommendations

Finally, conclusions were drawn based on the findings, and recommendations were made for further research or improvements.

**PROJECT DESCRIPTION**

. The Hospital Queue Management System is a C-based solution designed to automate and optimize patient queue management in hospitals. It addresses the problem of long waiting times, inefficiency, and lack of real-time queue tracking, which can lead to patient dissatisfaction and operational challenges.

Problem Statement:

Manual queue management in hospitals leads to delays, confusion, and difficulty in prioritizing patients, negatively impacting patient care.

Proposed Solution:

This system automates the patient registration and queue process, assigning priority to patients based on urgency, and providing real-time updates on the queue status to hospital staff. It aims to reduce wait times, improve resource allocation, and enhance the overall patient experience.

Key Features:

Patient Registration: Allows patients to enter the queue.

Queue Prioritization: Assigns priority based on patient urgency.

Real-time Updates: Displays current queue status for staff.

Automatic Patient Calling: Calls patients based on queue order.

Admin Interface: Enables staff to manage patient details and priorities.

The system aims to enhance operational efficiency, improve patient satisfaction, and streamline hospital operations.

**ALGORITHM**

Start

Initialize an empty queue structure to store patient details.

Display Main Menu with options:

Register new patient

View queue

Call next patient

Exit

If user selects "Register new patient":

Input patient name, age, and priority (e.g., emergency, normal, check-up).

Create a patient record.

Insert the patient into the queue based on priority (higher priority patients are placed ahead).

If user selects "View queue":

Display all patients currently in the queue with their details (name, priority, etc.).

If user selects "Call next patient":

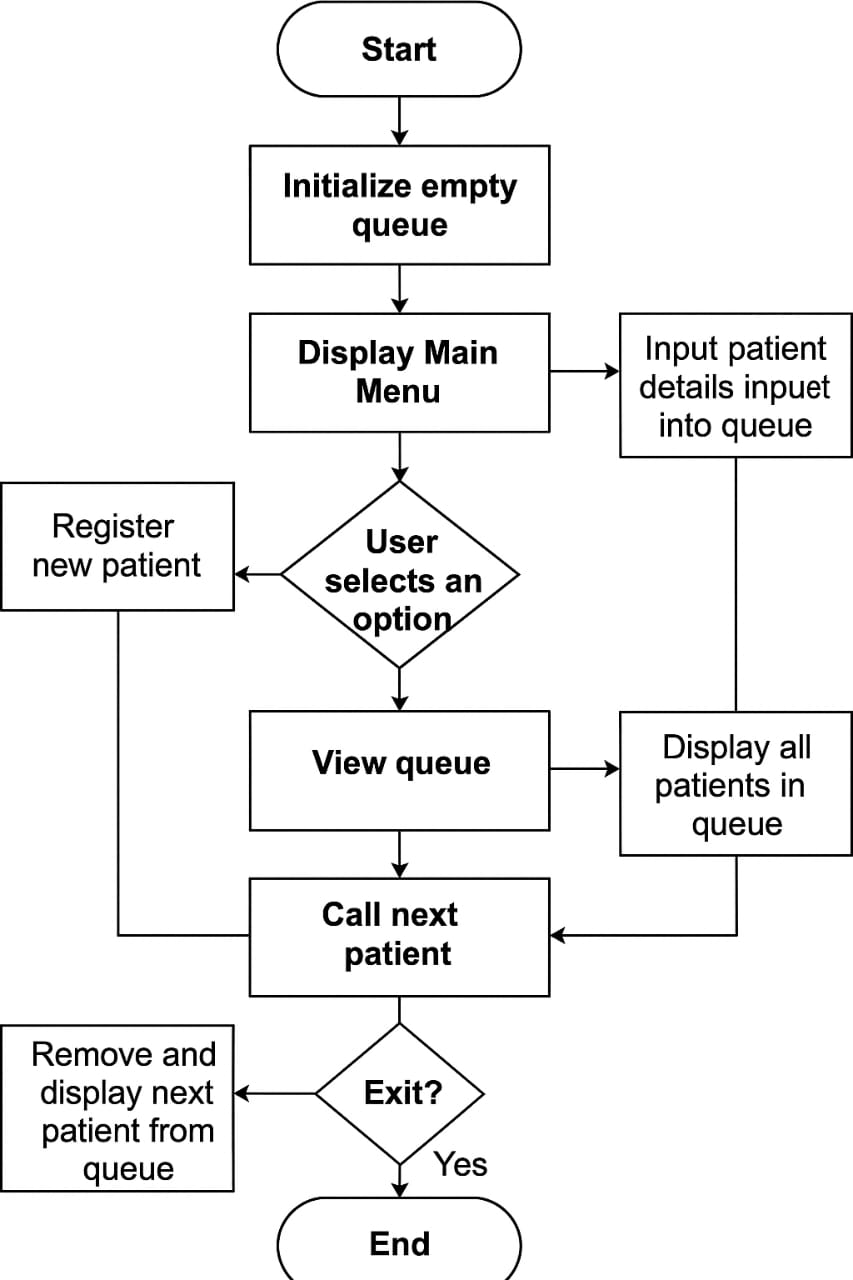
Remove the patient with the highest priority from the queue.

Display the called patient's details.

Repeat steps 3–6 until the user selects "Exit".

End

**FLOW CHART**



**PROGRAM CODE**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 100

#define MAX\_DEPTS 20

typedef struct {

char name[50];

int age;

int priority;

} Patient;

typedef struct {

char deptName[30];

Patient patients[MAX];

int front, rear;

} Queue;

Queue departments[MAX\_DEPTS];

int dept\_count = 0;

const char \*filename = "hospital\_data.txt";

void initQueue(Queue \*q, const char \*name) {

strcpy(q->deptName, name);

q->front = q->rear = -1;

}

int isFull(Queue \*q) {

return (q->rear + 1) % MAX == q->front;

}

int isEmpty(Queue \*q) {

return q->front == -1;

}

void enqueue(Queue \*q, char name[], int age, int priority) {

if (isFull(q)) return;

if (isEmpty(q)) q->front = 0;

q->rear = (q->rear + 1) % MAX;

strcpy(q->patients[q->rear].name, name);

q->patients[q->rear].age = age;

q->patients[q->rear].priority = priority;

}

Patient dequeue(Queue \*q) {

Patient p = {"", -1, -1};

if (isEmpty(q)) return p;

int highIndex = q->front;

int i = q->front;

while (1) {

if (q->patients[i].priority > q->patients[highIndex].priority)

highIndex = i;

if (i == q->rear) break;

i = (i + 1) % MAX;

}

p = q->patients[highIndex];

int next = (highIndex + 1) % MAX;

while (highIndex != q->rear) {

q->patients[highIndex] = q->patients[next];

highIndex = next;

next = (next + 1) % MAX;

}

q->rear = (q->rear - 1 + MAX) % MAX;

if (q->rear < q->front) q->front = -1, q->rear = -1;

return p;

}

void sortPatientsByName(Patient temp[], int count) {

for (int i = 0; i < count - 1; i++) {

for (int j = i + 1; j < count; j++) {

if (strcmp(temp[i].name, temp[j].name) > 0) {

Patient t = temp[i];

temp[i] = temp[j];

temp[j] = t;

}

}

}

}

void sortPatientsByAge(Patient temp[], int count) {

for (int i = 0; i < count - 1; i++) {

for (int j = i + 1; j < count; j++) {

if (temp[i].age > temp[j].age) {

Patient t = temp[i];

temp[i] = temp[j];

temp[j] = t;

}

}

}

}

void displayQueue(Queue \*q) {

if (isEmpty(q)) {

printf("\033[33mNo patients in %s department.\033[0m\n", q->deptName);

return;

}

int count = (q->rear - q->front + MAX) % MAX + 1;

Patient temp[MAX];

int i = q->front;

int idx = 0;

while (1) {

temp[idx++] = q->patients[i];

if (i == q->rear) break;

i = (i + 1) % MAX;

}

int sortChoice;

printf("\nSort patients by:\n1. Name\n2. Age\n3. No Sorting\nEnter choice: ");

scanf("%d", &sortChoice);

if (sortChoice == 1)

sortPatientsByName(temp, count);

else if (sortChoice == 2)

sortPatientsByAge(temp, count);

printf("\n\033[1mPatients in %s Department:\033[0m\n", q->deptName);

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

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

printf("%sName: %s\033[0m | Age: %d | Priority: %s\n",

temp[i].priority == 1 ? "\033[1;31m" : "\033[1;32m",

temp[i].name,

temp[i].age,

temp[i].priority == 1 ? "CRITICAL" : "NORMAL");

}

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

}

void listDepartments() {

for (int i = 0; i < dept\_count; i++) {

printf("%d. %s\n", i + 1, departments[i].deptName);

}

}

void saveData() {

FILE \*fp = fopen(filename, "w");

if (!fp) return;

fprintf(fp, "%d\n", dept\_count);

for (int d = 0; d < dept\_count; d++) {

fprintf(fp, "%s\n", departments[d].deptName);

Queue \*q = &departments[d];

int i = q->front;

if (isEmpty(q)) {

fprintf(fp, "0\n");

continue;

}

fprintf(fp, "%d\n", (q->rear - q->front + MAX) % MAX + 1);

while (1) {

fprintf(fp, "%s;%d;%d\n", q->patients[i].name, q->patients[i].age, q->patients[i].priority);

if (i == q->rear) break;

i = (i + 1) % MAX;

}

}

fclose(fp);

}

void loadData() {

FILE \*fp = fopen(filename, "r");

if (!fp) return;

fscanf(fp, "%d\n", &dept\_count);

for (int i = 0; i < dept\_count; i++) {

fgets(departments[i].deptName, sizeof(departments[i].deptName), fp);

departments[i].deptName[strcspn(departments[i].deptName, "\n")] = '\0';

departments[i].front = departments[i].rear = -1;

int patient\_count;

fscanf(fp, "%d\n", &patient\_count);

for (int j = 0; j < patient\_count; j++) {

char line[100], pname[50];

int age, priority;

fgets(line, sizeof(line), fp);

sscanf(line, "%[^;];%d;%d", pname, &age, &priority);

enqueue(&departments[i], pname, age, priority);

}

}

fclose(fp);

}

int main() {

loadData();

if (dept\_count == 0) {

printf("\n\033[1m--- Setup Hospital Departments ---\033[0m\n");

printf("Enter number of departments: ");

scanf("%d", &dept\_count);

for (int i = 0; i < dept\_count; i++) {

printf("Enter Department %d Name: ", i + 1);

scanf(" %[^\n]", departments[i].deptName);

initQueue(&departments[i], departments[i].deptName);

}

}

int choice, dept, age, priority;

char name[50];

while (1) {

printf("\n\033[1m--- Hospital Queue Management ---\033[0m\n");

printf("1. Register Patient\n2. Process Patient\n3. Display Queue\n4. Exit\nEnter choice: ");

scanf("%d", &choice);

if (choice == 4) break;

printf("\nSelect Department:\n");

listDepartments();

printf("Enter choice: ");

scanf("%d", &dept);

if (dept < 1 || dept > dept\_count) {

printf("\033[31mInvalid Department!\033[0m\n");

continue;

}

Queue \*selected = &departments[dept - 1];

if (choice == 1) {

printf("Enter Patient Name: ");

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

printf("Enter Patient Age: ");

scanf("%d", &age);

printf("Priority (1-Critical, 0-Normal): ");

scanf("%d", &priority);

enqueue(selected, name, age, priority);

} else if (choice == 2) {

Patient p = dequeue(selected);

if (p.age == -1)

printf("\033[33mNo patient to process in %s.\033[0m\n", selected->deptName);

else

printf("\033[36mProcessing Patient: %s, Age: %d, Priority: %s\033[0m\n",

p.name, p.age, p.priority == 1 ? "CRITICAL" : "NORMAL");

} else if (choice == 3) {

displayQueue(selected);

}

}

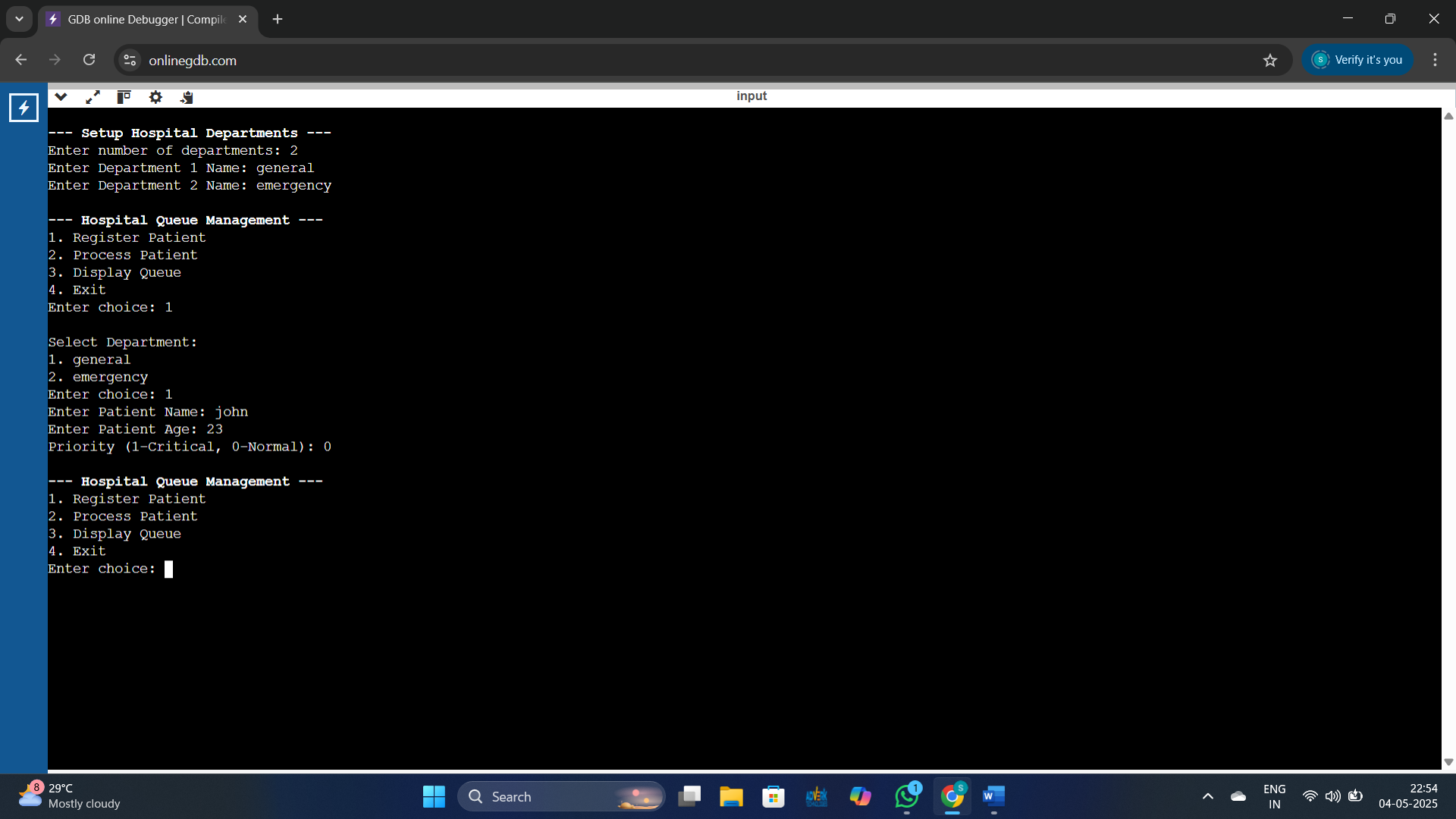
saveData();

printf("\033[35mData saved! Exiting... Thank you!\033[0m\n");

return 0;

}

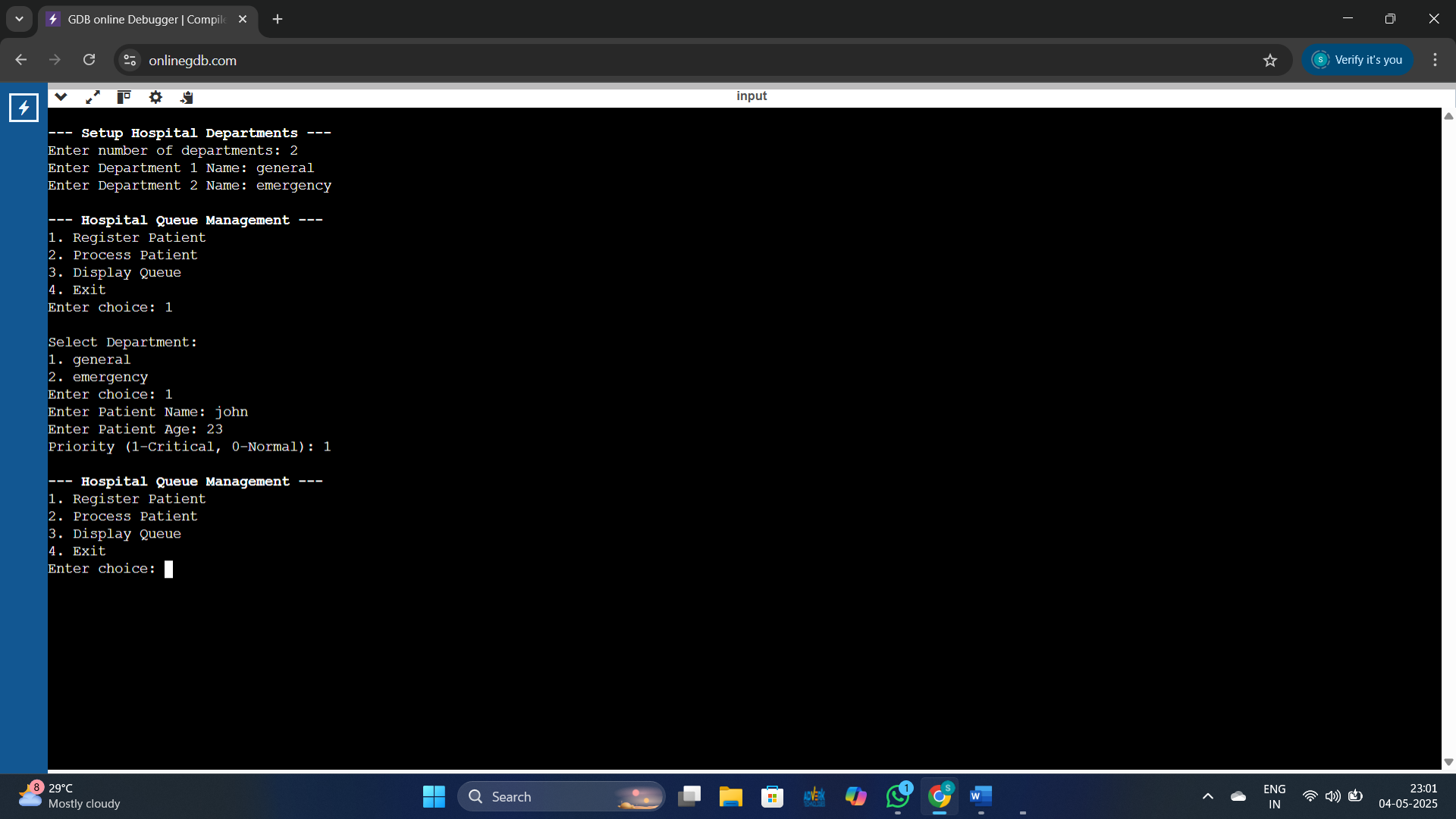
**OUTPUT SCREENSHOT**

****

**TESTING AND VALIDATION**

Input

1 (register patient)

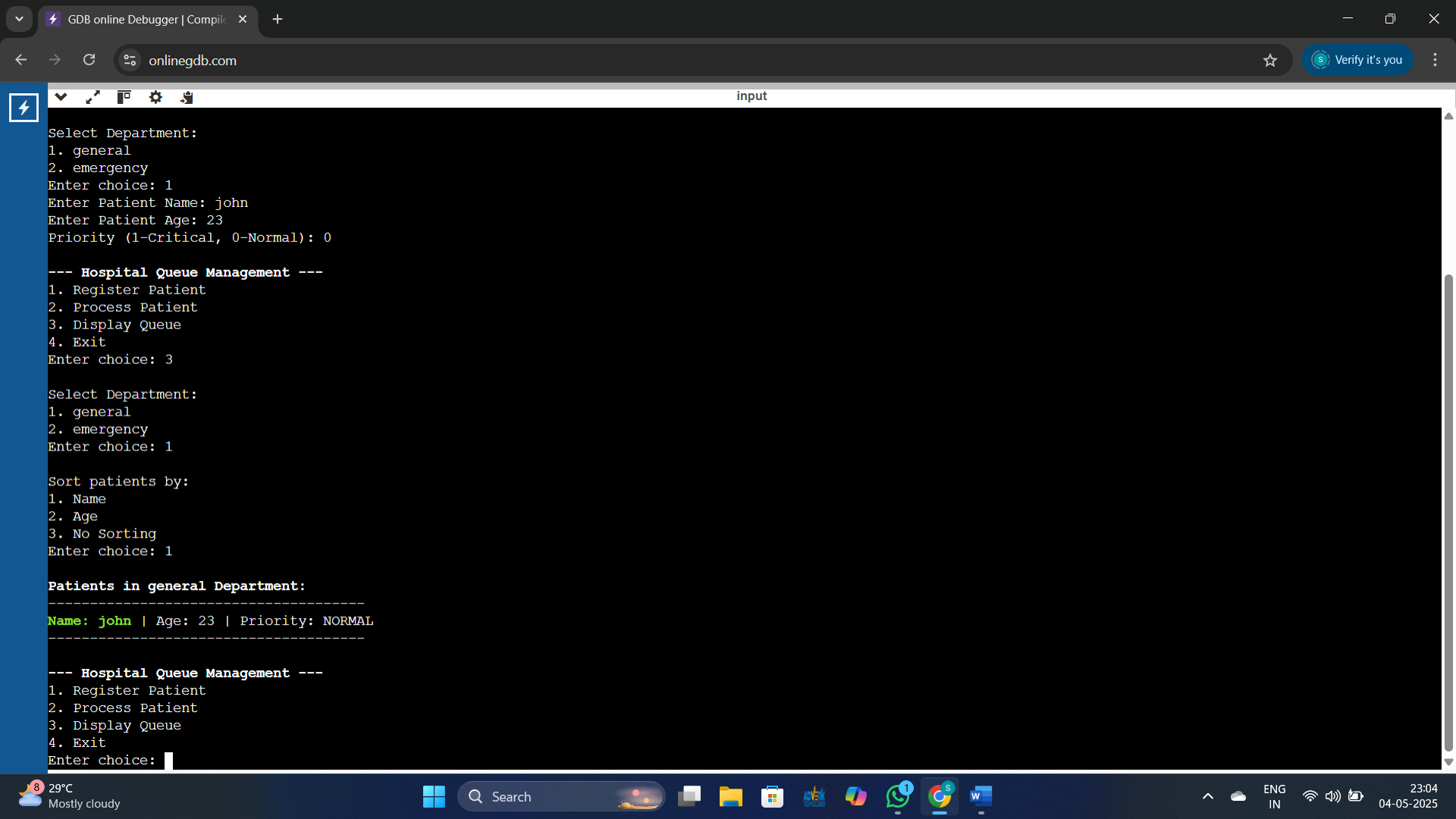


Input

1(Display Queue)

Output

Name : John | Age : 23 | Priority : Normal

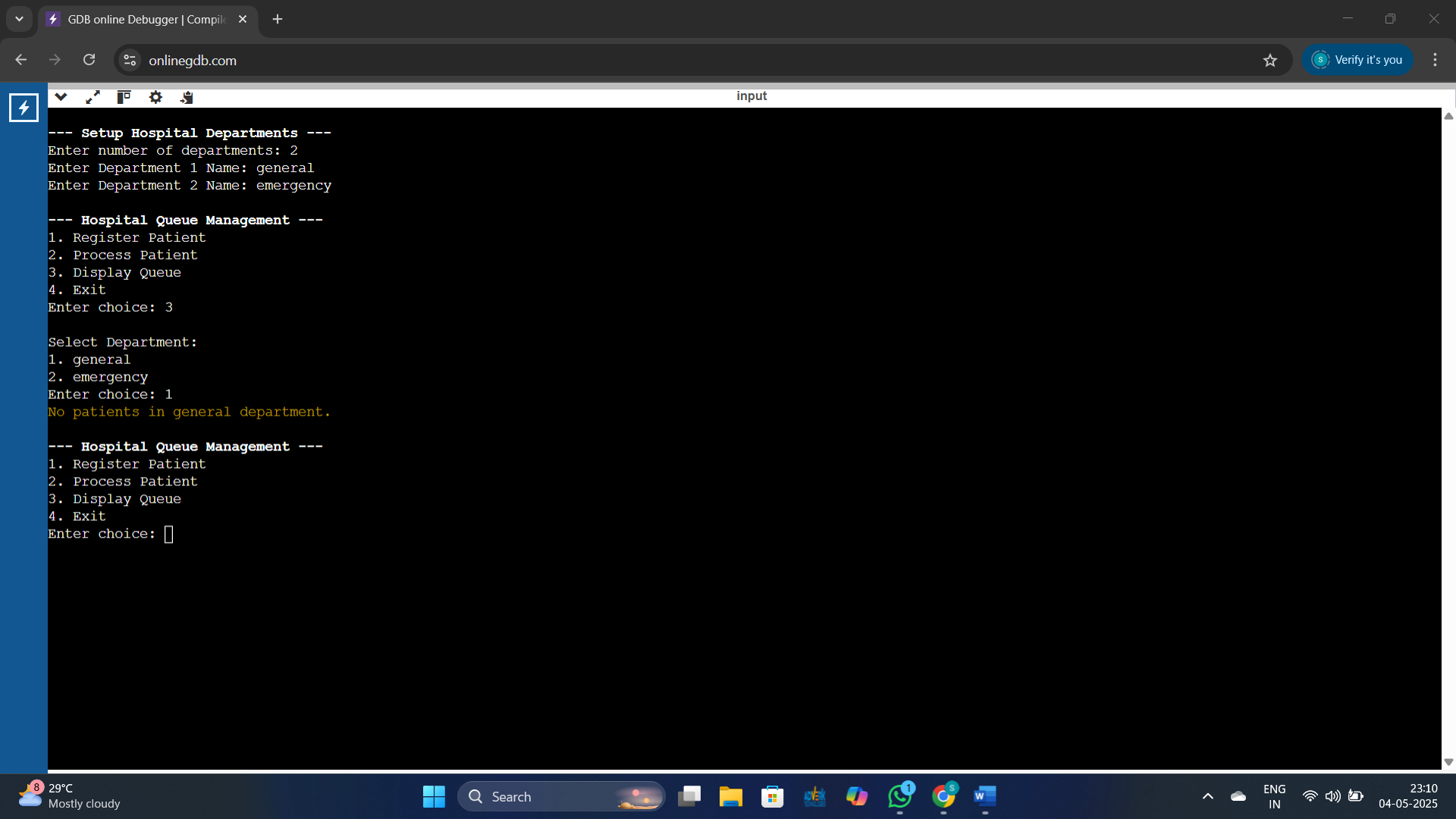


Input

2 (Display Queue in general)

Output

No patients in general department



**LIMITATIONS**

Console-Based Interface: The system uses a text-based interface, which may not be user-friendly for non-technical hospital staff compared to graphical user interfaces (GUIs).

No Data Persistence: Patient data is stored temporarily in memory. Once the program is closed, all records are lost unless file handling is added.

Limited Priority Logic: The system uses basic priority levels and may not fully reflect complex real-world medical triage requirements.

Single-User Access: The application does not support multi-user access or concurrent handling by multiple staff members.

No Authentication or Security: There is no login or security mechanism, which makes the system vulnerable to unauthorized access or accidental misuse.

Scalability Constraints: Being a simple C program, it may not perform efficiently with large numbers of records or in a large hospital setting.

Lack of Integration: The system does not integrate with other hospital systems like patient databases, billing, or electronic health records.

**FUTURE ENHANCEMENTS**

Future Enhancements

Graphical User Interface (GUI)

Upgrade the system to a GUI-based application for easier use and better user experience, especially for non-technical hospital staff.

Data Storage and Retrieval

Implement file handling or database integration (e.g., SQLite or MySQL) to store patient data permanently and allow for historical record tracking.

Advanced Priority System

Introduce a more dynamic and medically accurate triage system based on patient symptoms and severity levels.

Multi-User Access

Enable multiple users (e.g., receptionists, doctors) to access and update the queue simultaneously, possibly over a network.

Authentication and Security

Add user login functionality with roles (admin, staff) and secure access controls to prevent unauthorized usage.

Integration with Hospital Systems

Integrate with other modules like patient management, billing, and medical records for a complete hospital management solution.

SMS/Email Notifications

Implement a notification system to alert patients about their queue status or appointment times.

Mobile App or Web Version

Develop a mobile or web-based version of the system for broader accessibility and real-time updates.

**REFERENCE**

Text Books:

1. Data Structures and algorithm analysis in C, Mark Allen Weiss, Pearson, 2 nd Edition.

2. Data Structures, Algorithms, and Software Principles in C, by Thomas A Standish, ADDISON-WESLEY PUBLISHING COMPANY, 1995.

3. Fundamentals of data structures in C, Ellis Horowitz, Sartaj Sahni, Susan AndersonFreed, Silicon Press, 2008

Reference Books:

1. Algorithms and Data Structures: The Basic Toolbox by Kurt Mehlhorn and Peter Sanders

2. C Data Structures and Algorithms by Alfred V. Aho, Jeffrey D. Ullman, and John E. Hopcroft

3. Problem Solving with Algorithms and Data Structures" by Brad Miller and David Ranum

4. Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein

5. Algorithms in C, Parts 1-5 (Bundle): Fundamentals, Data Structures, Sorting, Searching, and Graph Algorithms" by Robert Sedgewick.

**CONCLUSION**

The Hospital Queue Management System developed in C effectively addresses the challenges of manual patient queue handling in hospitals. By automating the registration, prioritization, and calling process, the system enhances efficiency, reduces patient wait times, and supports better workflow management for hospital staff.

Though simple in design, the system lays the foundation for more advanced healthcare queue management solutions. With future enhancements like GUI, data storage, and integration with hospital systems, it can be scaled into a comprehensive and user-friendly tool for real-world hospital environments.

This project has not only improved our understanding of data structures and system design but also demonstrated how software can solve practical problems in healthcare settings.