**NBKR INSTITUTE OF SCIENCE AND TECHNOLOGY**

**VIDYANAGAR,VAKADA.PIN:524413**

**HOSPITAL MANAGEMENT SYSTEM**

**Course**: Data Structures

**Branch**: ELECTRONICS COMMUNICATION

**Section**: ECE-C

**Year&SEM**: 1y-2sem

**Submitted to**: SATHISH APPANA

**SUBMITTED BY:**

**1)** CH.NAVYA SRI-24KB1A0455

**2)** A.BHUVANESWARI-24KB1A0402

**3)** T.JHANSI-24KB1A04AF

**4)** I.SIREESHA-24KB1A04C4

**5)** B.SRAVANTHI-24KB1A0430

**6)** C.GAGANA SRI-24KB1A0471

**ACKNOWLEDGEMENT:**

I would like to express my deep gratitude to [Guide/Professor Name], my project guide, for their valuable support, guidance, and encouragement throughout this project. I am also thankful to my family and friends who have helped me throughout the development of this project.

**ABSTRACT:**

This Hospital Management System utilizes a queue data structure to efficiently manage patient data, including details such as name, age, disease, and admission date. The system provides a menu-driven interface for users to add, serve, and display patients, while also saving and loading patient data to and from a file. Designed to streamline hospital operations, this system demonstrates a practical application of data structures and file management in a real-world context.

**INTRODUCTION:**

In today's healthcare landscape, efficient management of patient data and hospital operations is crucial. This Hospital Management System aims to streamline these processes using a queue data structure, providing a simple and intuitive interface for managing patient information. By leveraging technology, this system seeks to improve the delivery of healthcare services and enhance patient care.

**OBJECTIVES:**

The objectives of this Hospital Management System are:

1. To design and implement a queue-based system for managing patient data.

2. To provide a user-friendly interface for adding, serving, and displaying patients.

3. To ensure data persistence through file storage and retrieval.

4. To streamline hospital operations and improve patient care

These objectives aim to create an efficient and effective system for managing hospital data and operations.

**SYSTEM REQUIREMENTS:**

Hardware Requirements:

1. A computer with a compatible operating system (Windows, Linux, etc.)

2. Sufficient storage space for patient data

Software Requirements:

1. C compiler (e.g., GCC)

2. Text editor or IDE (e.g., Visual Studio Code)

Functional Requirements:

1. Patient data management (add, serve, display)

2. Queue operations (enqueue, dequeue)

3. File operations (save, load)

**METHODOLOGY**:

Methodology

The Hospital Management System was developed using:

Programming Language:

1. C

Data Structure:

1. Queue (linked list implementation)

Development Approach:

1. Modular design

2. Step-by-step implementation

3. Testing and debugging

Key Features:

1. Patient data management

2. Queue operations

3. File persistence

This methodology ensured a structured and efficient development process.

**PROJECT DESCRIPTION:**

The Hospital Management System is a software application designed to manage patient data and streamline hospital operations. Key features include:

1. Patient data management (add, serve, display)

2. Queue operations (enqueue, dequeuer)

3. File persistence (save, load)

The system provides a simple and intuitive menu-driven interface for users to interact with, making it easy to manage patient information and hospital operations

**FLOWCHART OR ALGORITHM:**

**1. Load Existing Data**

* Open patients\_data.txt for reading.
* For each line, read patient details (name, age, disease, admission date).
* Enqueue each patient into the queue

**2. Main Menu Loop**

* Display options:
  1. Add Patient
  2. Serve Patient
  3. Display Queue
  4. Save Data
  5. Exit

**3. Add Patient (Option 1)**

* Prompt user for patient details:
  + Name
  + Age
  + Disease
  + Admission Date
* Create a new Patient structure.
* Enqueue the new patient.​

**4. Serve Patient (Option 2)**

* If the queue is not empty:
  + Dequeue the front patient.
  + Display patient details.
* If the queue is empty:
  + Display "No patients in the queue."​

**5. Display Queue (Option 3)**

* If the queue is not empty:
  + Traverse the queue and display each patient's details.
* If the queue is empty:
  + Display "No patients in the queue."​

**6. Save Data (Option 4)**

* Open patients\_data.txt for writing.
* For each patient in the queue:
  + Write patient details to the file.
* Close the file

**7. Exit (Option 5)**

* Save data to the file.
* Display "Exiting..."
* Terminate the program.​

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Define the structure for Patient

typedef struct Patient {

char name[50];

int age;

Char disease[50];

char admission\_date[20];

} Patient;

// Queue node structure

typedef struct Node {

Patient patient;

struct Node\* next;

} Node;

Node\* front = NULL;

Node\* rear = NULL;

// Function to enqueue a patient

Void enqueue(Patient patient) {

Node\* new node = (Node\*)malloc(sizeof(Node));

new node->patient = patient;

new\_node->next = NULL;

if (rear == NULL) {

front = rear = new\_node;

} else {

rear->next = new\_node;

rear = new\_node;

}

printf("Patient added to the queue.\n");

}

// Function to dequeue a patient

void dequeue() {

if (front == NULL) {

printf("No patients in the queue.\n");

return;

}

Node\* temp = front;

printf("Serving Patient:\nName: %s\nAge: %d\nDisease: %s\nAdmission Date: %s\n",

temp->patient.name, temp->patient.age, temp->patient.disease, temp->patient.admission\_date);

front = front->next;

if (front == NULL) {

rear = NULL;

}

free(temp);

}

// Function to display all patients in the queue

void displayQueue() {

if (front == NULL) {

printf("No patients in the queue.\n");

return;

}

Node\* temp = front;

printf("Patients in the queue:\n");

while (temp != NULL) {

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

temp->patient.name, temp->patient.age, temp->patient.disease, temp->patient.admission\_date);

temp = temp->next;

}

}

// Save queue to file

void saveToFile() {

FILE\* file = fopen("patients\_data.txt", "w");

if (file == NULL) {

printf("Error saving to file.\n");

return;

}

Node\* temp = front;

while (temp != NULL) {

fprintf(file, "%s %d %s %s\n", temp->patient.name, temp->patient.age, temp->patient.disease, temp->patient.admission\_date);

temp = temp->next;

}

fclose(file);

printf("Data saved to file.\n");

}

// Load queue from file

void loadFromFile() {

FILE\* file = fopen("patients\_data.txt", "r");

if (file == NULL) {

printf("No previous data found.\n");

return;

}

Patient patient;

while (fscanf(file, "%s %d %s %s", patient.name, &patient.age, patient.disease, patient.admission\_date) != EOF) {

enqueue(patient);

}

fclose(file);

printf("Data loaded from file.\n");

}

int main() {

loadFromFile(); // Load data from file if available

int choice;

do {

printf("\nHospital Management System\n");

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

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

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

printf("4. Save Data\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: {

Patient patient;

printf("Enter name: ");

scanf("%s", patient.name);

printf("Enter age: ");

scanf("%d", &patient.age);

printf("Enter disease: ");

scanf("%s", patient.disease);

printf("Enter admission date: ");

scanf("%s", patient.admission\_date);

enqueue(patient);

break;

}

case 2:

dequeue();

break;

case 3:

displayQueue();

break;

case 4:

saveToFile();

break;

case 5:

saveToFile(); // Save data before exiting

printf("Exiting...\n");

break;

default:

printf("Invalid choice. Try again.\n");

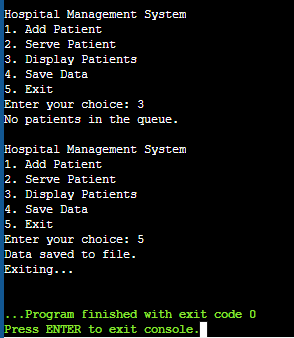
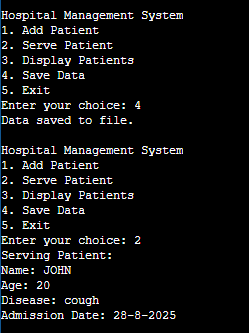
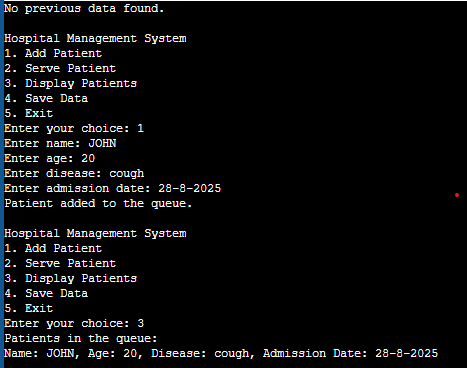
}

} while (choice != 5);

return 0;

}

**OUTPUT SCREENSHOTS:**



**Testing and validation:**

## ✅**Testing the Project**

### 🔹****1. Unit Testing (Function-by-function)****

| **Function** | **Test Case** | **Expected Result** |
| --- | --- | --- |
| enqueue() | Add 1–3 patients | Patients are added to the queue in order |
| dequeue() | Remove patients from queue | Patients are served in FIFO order; message if empty |
| displayQueue() | Queue has multiple patients | Displays patients correctly in order |
| saveToFile() | Save after adding patients | File patients\_data.txt created/updated |
| loadFromFile() | Load file with valid data | Queue populated correctly on program start |

## **Validation Checklist**

| **Aspect** | **Validation Steps** | **Result** |
| --- | --- | --- |
| Input Handling | Reject empty or malformed input (e.g., age = -5) | Needs input validation improvement |
| File Operations | File must open and close properly | Check file pointer (NULL) |
| Data Integrity | Loaded data matches saved entries | Confirm by comparing queue vs file |
| Memory Leaks | Free memory on exit | Use tools like **Valgrind** (Linux) |

**LIMITATIONS:**

### 1. ****Limited Input Handling****

* **Issue**: Uses scanf() for string input, which breaks with multi-word names or diseases.
* **Impact**: Names like "John Doe" will only capture "John".
* **Solution**: Use fgets() instead of scanf() for string inputs.

### 2. ****No Input Validation****

* **Issue**: Accepts invalid inputs (e.g., negative age, empty fields).
* **Impact**: Leads to unrealistic or corrupted patient data.
* **Solution**: Add checks to validate age, non-empty names, and correctly formatted dates.

### 3. ****File I/O Format Fragility****

* **Issue**: Uses space-separated format (%s %d %s %s) in file operations.
* **Impact**: Breaks if input strings contain spaces.
* **Solution**: Use newline-separated records or structured formats like CSV or JSON (though more complex in C).

### 4. ****No Date Validation****

* **Issue**: Admission dates are not verified (e.g., "32/13/2023" is accepted).
* **Impact**: Leads to invalid or inconsistent record-keeping.
* **Solution**: Implement date parsing and validation logic.

### 5. ****Memory Leaks Possible****

* **Issue**: Nodes are not freed if the program exits unexpectedly (e.g., crash or Ctrl+C).
* **Impact**: Causes memory leaks during runtime.
* **Solution**: Add proper cleanup on exit, possibly using atexit() or signal handling.

### 6. ****No Search or Update Feature****

* **Issue**: Once a patient is enqueued, their record cannot be modified or searched.
* **Impact**: Limits usability in real scenarios.
* **Solution**: Extend the system to allow searching, updating, or deleting specific patients.

### 7. ****Single User / No Concurrency****

* **Issue**: Designed for command-line, single-user use only.
* **Impact**: Not suitable for multi-user environments like a real hospital system.
* **Solution**: Requires major redesign with threads, database, and UI if scaling up.

### 8. ****Lack of Persistence Between Runs (for served patients)****

* **Issue**: Only current queue is saved; served patients are not archived.
* **Impact**: No history or audit trail of past patients.
* **Solution**: Log served patients to a separate file or maintain a full record histoRY

**FUTURE ENHANCEMENTS:**

1. Improved Input Handling
2. Input Validation
3. Patient Search Functionality
4. Update & Delete Patient Records
5. Maintain Patient History (Served Patients)
6. Graphical User Interface (GUI)
7. Export Data to CSV or PDF
8. Database Integration (SQLite/MySQL)
9. Multi-User Support
10. Priority Queue Implementation

**CONCLUSION:**

The Hospital Management System using a queue in C provides a foundational, menu-driven application to manage patient flow efficiently using First-In-First-Out (FIFO) logic. It demonstrates essential data structures (linked lists), file handling, and modular programming practices. While the current implementation covers basic functionalities such as adding, serving, and displaying patients, there is significant scope for improvement through features like input validation, search, GUI integration, and database connectivity. With further development, this system can evolve into a more robust, real-world application suitable for small-scale hospital or clinic use.

**REFERENCES:**

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 Arum

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.