### ****Mini Project Report****

**Title**: Hospital Patient Management System using queues for patient records and scheduling.

**Course**: BCS304 – Data Structures and Applications  
**Submitted By**: Group No: 1

|  |  |  |
| --- | --- | --- |
| SL. NO. | USN | NAME |
| 1 | 4MW23CS026 | C Vignesh Kumar |
| 2 | 4MW23CS030 | Chirag S Poojary |
| 3 | 4MW23CS035 | Deepak |
| 4 | 4MW23CS042 | Goutham Shetty |
| 5 | 4MW23CS053 | K Vijeth V Naik |

**Date**: 19-12-24

### ****1. Abstract****

### The Hospital Patient Management System is a mini-project designed to efficiently handle patient records in a hospital environment. This system uses a circular queue to store and manage patient data, providing features such as adding new patients, processing the next patient, deleting specific records, and displaying all current patients. By incorporating the system ensures flexibility and efficient memory usage as the number of patients fluctuates. The user interacts with the system through a menu-driven interface, making it easy to manage and update patient information. This project demonstrates the practical application of circular queues and dynamic memory management in solving real-world problems in healthcare.

### ****2. Introduction****

### Hospitals manage a large amount of data, including patient details, medical issues. The dynamic nature of hospital workflows makes circular queues with dynamic memory allocation an ideal data structure for this purpose. Unlike static arrays, circular queues efficiently utilize memory and allow seamless additions and deletions, ensuring optimal performance even with fluctuating patient numbers. This project simulates a hospital patient management system using a circular queue, focusing on user-friendly functionality, efficient memory usage and real-time data management.

### ****3. Problem Statement****

Develop a **Hospital Patient Management System** that:

1. Maintains a **list of patient data** with details such as book ID, name, age, Health issues etc.
2. Maintains a **list of members** with details such as member ID, name, and contact information etc.
3. Allows operations like adding patient ID and searching patient details.

### ****4. Objectives****

1. Implement a **menu-driven program** to manage Patient details in a library.
2. Demonstrate operations like addition, deletion and searching using **circular queue.**
3. **Patient addition and removal maintain the first-in-first-out (FIFO) principle.**
4. Ensure efficient memory utilization and error handling.
5. Provide an interactive interface for ease of use.

### ****5. System Design****

**Data Structures Used**:

* **Circular Queue**
  + One structure for collecting Patient details.
  + One structure for assigning the Staff Username and Password.

### ****6. Algorithms****

### *1. Initialization*

### *Define constants: MAX is set to 100 for queue capacity.*

### *Create structures:*

### *Patient: Holds patient details such as ID, name, issue, date of admission, age, blood group and contact number.*

### *Queue: Represents a circular queue containing Patient records with front and rear pointers.*

### *Staff: Contains staff login credentials (username and password).*

### *2. Functions*

### *create(queue): Initializes an empty queue by setting front = -1 and rear = -1.*

### *isEmpty(queue): Returns true if front == -1.*

### *isFull(queue): Returns true if (rear + 1) % MAX == front.*

### *enqueue(queue, patient):*

### *If the queue is full, displays an appropriate message.*

### *Else, add a patient record:*

### *If empty, set front = rear = 0.*

### *Else, increment rear circularly using (rear + 1) % MAX.*

### *Insert the patient at rear and display success message.*

### *dequeue(queue):*

### *If the queue is empty, display a message*

### *Otherwise, remove the patient at front:*

### *If it’s the last patient, reset the queue (front = rear = -1).*

### *Otherwise, increment front circularly using (front + 1) % MAX.*

### *Return the removed patient.*

### *display(queue):*

### *If the queue is empty, display a message.*

### *Otherwise, iterate from front to rear circularly, printing each patient's details.*

### *deletePatient(queue, id):*

### *If the queue is empty, display a message.*

### *Use a temporary queue to filter patients:*

### *Dequeue each patient.*

### *If the patient’s ID matches, mark it as deleted and skip adding it to the temporary queue.*

### *Otherwise, enqueue it into the temporary queue.*

### *Restore patients from the temporary queue to the original queue.*

### *If no patient was found, display a "not found" message.*

### *authenticateStaff(username, password):*

### *Compares input credentials with predefined staff credentials.*

### *Returns true if valid, false otherwise.*

### *3. Main Program Flow*

### *Display the main menu:*

### *Option 1: Staff Login*

### *Prompt for credentials.*

### *Authenticate using authenticateStaff.*

### *Exit if invalid.*

### *Option 2: Exit the program.*

### *Enter a loop for patient management tasks:*

### *Option 1: Add Patient*

### *Input patient details (ID, name, age, contact number, issue, date of admit).*

### *Option 2: Process the Next Patient*

### *Remove the next patient using dequeue.*

### *Display the removed patient's details if valid.*

### *Option 3: Display All Patient Records*

### *Use display to show the queue.*

### *Option 4: Delete Patient Record*

### *Input the patient ID to delete.*

### *Use deletePatient to remove the patient by ID.*

### *Option 5: Exit*

### *Exit the program*

### *Default: Display an error message for invalid choices.*

### ****7. Implementation (Code)****

### *#include <stdio.h> // For input and output operations*

### *#include <stdlib.h> // For using inbuilt library functions*

### *#include <string.h> // For using inbuilt string operations*

### *#define MAX 100 // Maximum size of the queue is 100*

### *// Structure to represent a patient*

### *typedef struct*

### *{*

### *int id;*

### *char name[50];*

### *char issue[100];*

### *char dateOfAdmit[15]; // Date of admit*

### *int age; // Patient's age*

### *char bloodGroup[5]; // Blood group*

### *char contactNo[15]; // Contact number*

### *} Patient;*

### *// Queue structure*

### *typedef struct*

### *{*

### *Patient patients[MAX]; // Structure variable is declared with MAX size*

### *int front;*

### *int rear;*

### *} Queue;*

### *typedef struct Staff*

### *{*

### *char username[MAX];*

### *char password[MAX];*

### *} Staff;*

### *Staff staff = {"admin", "admin123"}; // Staff ID and password are set for Login purpose*

### *// Function to initialize the queue*

### *void create(Queue \*q)*

### *{*

### *q->front = -1;*

### *q->rear = -1;*

### *}*

### *// Function to check if the queue is empty*

### *int isEmpty(Queue \*q)*

### *{*

### *return q->front == -1;*

### *}*

### *// Function to check if the queue is full*

### *int isFull(Queue \*q)*

### *{*

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

### *}*

### *// Function to add a patient to the queue*

### *void enqueue(Queue \*q, Patient p)*

### *{*

### *if (isFull(q))*

### *{*

### *printf("Queue is full! Cannot add more patients data.\n");*

### *return;*

### *}*

### *if (isEmpty(q))*

### *{*

### *q->front = q->rear = 0;*

### *}*

### *else*

### *{*

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

### *}*

### *q->patients[q->rear] = p;*

### *printf("Patient data added successfully.\n");*

### *}*

### *// Function to remove a patient from the queue*

### *Patient dequeue(Queue \*q)*

### *{*

### *Patient p;*

### *if (isEmpty(q))*

### *{*

### *printf("Queue is empty! No more patients to process.\n");*

### *p.id = -1; // Invalid ID to indicate no patient*

### *return p;*

### *}*

### *p = q->patients[q->front];*

### *if (q->front == q->rear)*

### *{*

### *q->front = q->rear = -1; // Reset queue*

### *}*

### *else*

### *{*

### *q->front = (q->front + 1) % MAX;*

### *}*

### *return p;*

### *}*

### *// Function to display the queue*

### *void display(Queue \*q)*

### *{*

### *if (isEmpty(q))*

### *{*

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

### *return;*

### *}*

### *printf("Current Patients in the Queue are:\n");*

### *int i = q->front;*

### *while (1)*

### *{*

### *Patient p = q->patients[i];*

### *printf("ID: %d\tName: %s\tIssue: %s\tDate of Admit: %s\tAge: %d\tBlood Group: %s\tContact No: %s\n",*

### *p.id, p.name, p.issue, p.dateOfAdmit, p.age, p.bloodGroup, p.contactNo);*

### *if (i == q->rear)*

### *break;*

### *i = (i + 1) % MAX;*

### *}*

### *}*

### *// Function to delete a specific patient record by ID*

### *void deletePatient(Queue \*q, int id)*

### *{*

### *if (isEmpty(q))*

### *{*

### *printf("Queue is empty! No more patients data to delete.\n");*

### *return;*

### *}*

### *int found = 0;*

### *Queue tempQueue;*

### *create(&tempQueue);*

### *// Traverse the queue and transfer patients to a temporary queue, skipping the one to be deleted*

### *while (!isEmpty(q))*

### *{*

### *Patient current = dequeue(q);*

### *if (current.id == id)*

### *{*

### *found = 1; // Mark that the patient is found*

### *printf("Patient with ID %d has been deleted.\n", id);*

### *}*

### *else*

### *{*

### *enqueue(&tempQueue, current); // Keep other patients in the temporary queue*

### *}*

### *}*

### *// Restore patients from the temporary queue back to the original queue*

### *while (!isEmpty(&tempQueue))*

### *{*

### *enqueue(q, dequeue(&tempQueue));*

### *}*

### *if (!found)*

### *{*

### *printf("Patient with ID %d not found.\n", id);*

### *}*

### *}*

### *int authenticateStaff(const char \*username, const char \*password)*

### *{*

### *return strcmp(username, staff.username) == 0 && strcmp(password, staff.password) == 0;*

### *}*

### *void staffLogin()*

### *{*

### *char username[MAX], password[MAX];*

### *int choice;*

### *printf("Enter Username: ");*

### *scanf("%s", username);*

### *printf("Enter Password: ");*

### *scanf("%s", password);*

### *if (authenticateStaff(username, password))*

### *{*

### *printf("Login successful!\n");*

### *}*

### *else*

### *{*

### *printf("Invalid ID or password. Try again!\n");*

### *exit(0);*

### *}*

### *}*

### *int main()*

### *{*

### *Queue queue;*

### *create(&queue);*

### *int choice1;*

### *printf("\n--- Hospital Patient Management System ---\n");*

### *printf("1. Staff Login\n");*

### *printf("2. Exit\n");*

### *printf("Enter your choice: ");*

### *scanf("%d", &choice1);*

### *switch (choice1)*

### *{*

### *case 1:*

### *staffLogin();*

### *break;*

### *case 2:*

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

### *exit(0);*

### *default:*

### *printf("Invalid choice. Please try again.\n");*

### *exit(0);*

### *}*

### *int choice;*

### *while (1)*

### *{*

### *printf("\n~Hospital Patient Management System~\n");*

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

### *printf("2. Process the Next Patient\n");*

### *printf("3. Display All the Patient Records\n");*

### *printf("4. Delete Patient Record\n");*

### *printf("5. Exit\n");*

### *printf("Enter your choice: ");*

### *scanf("%d", &choice);*

### *switch (choice)*

### *{*

### *case 1:*

### *{*

### *Patient p;*

### *printf("Enter Patient ID: ");*

### *scanf("%d", &p.id);*

### *printf("Enter Patient Name: ");*

### *scanf(" %[^\n]%\*c", p.name);*

### *printf("Enter Patient Age: ");*

### *scanf("%d", &p.age);*

### *printf("Enter Contact Number: ");*

### *scanf(" %[^\n]%\*c", p.contactNo);*

### *printf("Enter Patient Issue: ");*

### *scanf(" %[^\n]%\*c", p.issue);*

### *printf("Enter Date of Admit (dd/mm/yyyy): ");*

### *scanf(" %[^\n]%\*c", p.dateOfAdmit);*

### *enqueue(&queue, p);*

### *break;*

### *}*

### *case 2:*

### *Patient p = dequeue(&queue);*

### *if (p.id != -1)*

### *{*

### *printf("Processing Patient:\n");*

### *printf("ID: %d\t Name: %s\t Age: %d\t Contact No: %s\t Blood Group: %s\t Issue: %s\t Date of Admit: %s\n",*

### *p.id, p.name, p.age, p.contactNo, p.bloodGroup, p.issue, p.dateOfAdmit);*

### *}*

### *break;*

### *case 3:*

### *display(&queue);*

### *break;*

### *case 4:*

### *int id;*

### *printf("Enter the ID of the patient to delete: ");*

### *scanf("%d", &id);*

### *deletePatient(&queue, id);*

### *break;*

### *case 5:*

### *printf("Exiting the system.\n");*

### *exit(0);*

### *break;*

### *default:*

### *printf("Invalid choice! Please try again.\n");*

### *}*

### *}*

### *return 0;*

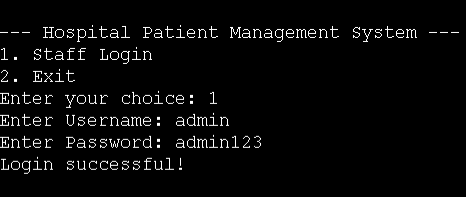
### *}*

### ****8. Test Cases****

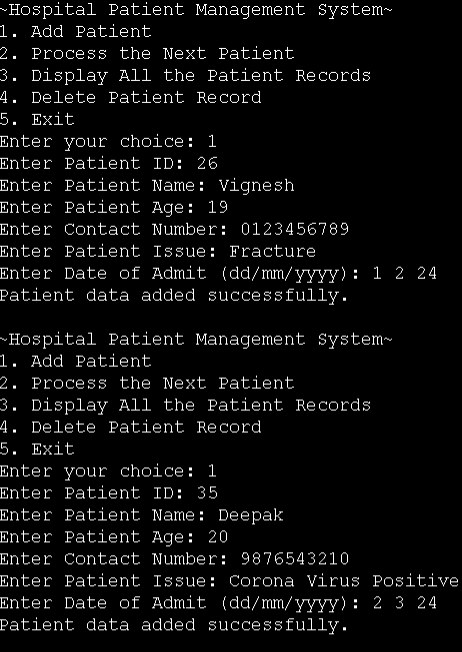
| **Operation** | **Input** | **Output** |
| --- | --- | --- |
| Staff Login | Username, password | Login successfully. |
| Add Patient Details | Name, age, Contact no., blood group, issues | Patient ID added successfully. |
| Process Patient ID | N/A | Next Patient details |
| Display Members | N/A | List of all members. |
| Delete Patient Details | Patient ID | Deletion successful |

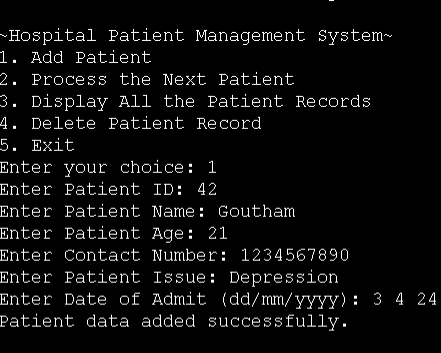
**9. Screenshots**

**1. Staff Login**

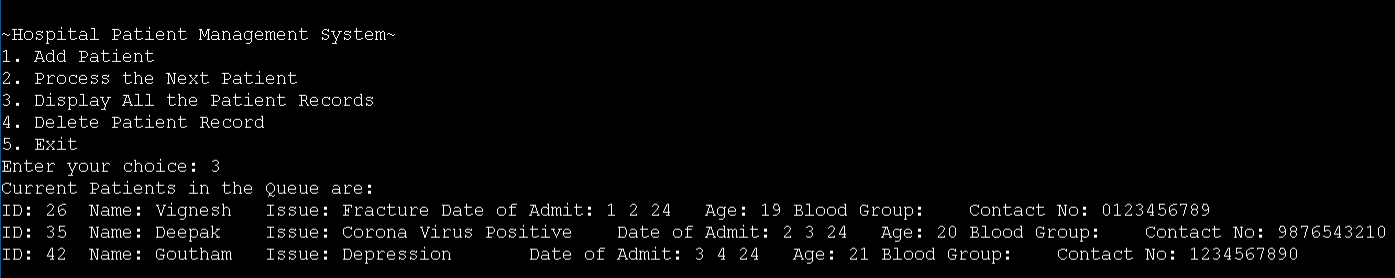
****

**2. Adding Patient Data**

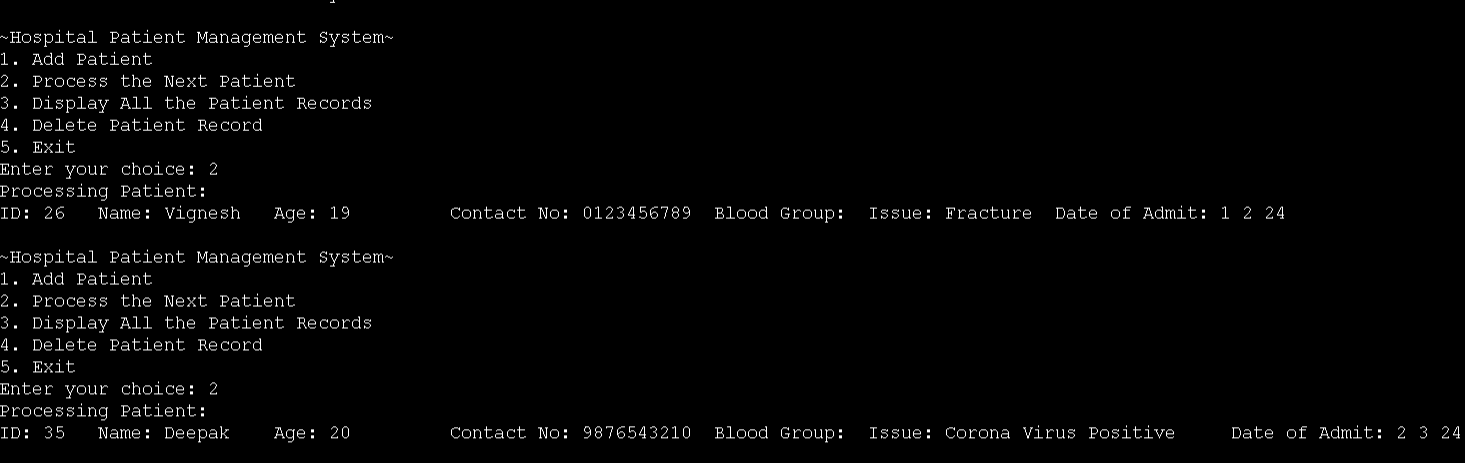
****

****

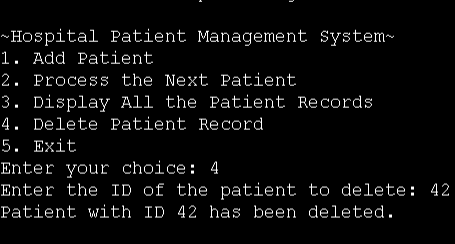
**3. Displaying all Patient Details**

****

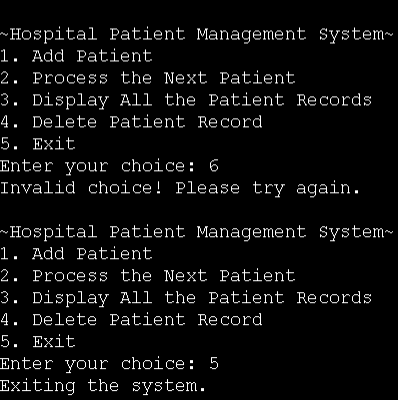
**4. Processing Patients**

****

**5.Deleting Patient Records**

****

**6.Exit**

****

### ****10. Conclusion****

This mini-project demonstrates the use of **Circular** **Queue data structure** for managing data in a hospital system. The implementation effectively handles operations like adding, deleting and displaying records, showcasing the practical utility of data structures in solving real-world problems.