Open-Ended Lab Report On: Hospital Management System

Course code: CSE 1302

Course title: Data structure lab

Section:01

Submitted to,

Sakib mahmud Dipto

Lecturer,

University of Liberal Arts Bangladesh

Submitted By ,

Shakhar Das Rony

ID:223014078

Section:01

Department of Computer Science and Engineering

University of Liberal Arts Bangladesh

Theory and Analytical:

Array:

An array is a fixed-size, contiguous block of memory that stores a collection of elements of the same data type. Elements in an array are accessed using their indices.

Advantages: Arrays provide constant-time access to elements using their index, making them efficient for read operations.

Disadvantages: Resizing an array can be inefficient, especially if the array is full. Insertion and deletion of elements may require shifting of elements, leading to time-consuming operations.

Linked List:

A linked list is a linear data structure that consists of a sequence of elements, where each element points to the next element in the sequence. Each element in a linked list is called a node and has two components: the data it holds and a reference (or pointer) to the next node. The last node usually points to a null value to indicate the end of the list. There are different types of linked lists, such as singly linked lists, doubly linked lists, and circular linked lists.

Advantages: Linked lists allow for dynamic memory allocation, efficient insertion and deletion operations (if you have a reference to the node), and are well-suited for situations where data needs to be frequently added or removed.

Disadvantages: Accessing an element at a particular index takes O(n) time in the worst case, and they use more memory due to the storage of pointers.

Stack:

A stack is a linear data structure that follows the Last-In-First-Out (LIFO) principle. It's like a stack of plates – the last plate added is the first one that can be removed. A stack can be implemented using arrays or linked lists.

Operations: Stacks typically support two main operations: push (add an element to the top of the stack) and pop (remove and return the top element). Additionally, there's peek to look at the top element without removing it.

Applications: Stacks are used in algorithms like function call management (call stack), expression evaluation, undo operations in software, and more.

Queue:

A queue is a linear data structure that follows the First-In-First-Out (FIFO) principle. It's like people waiting in a queue – the first person who joins is the first one to leave. A queue can be implemented using arrays or linked lists.

Operations: Queues typically support two main operations: enqueue (add an element to the back of the queue) and dequeue (remove and return the front element). Some queues also support peek to view the front element.

Applications: Queues are used in scenarios like task scheduling, breadth-first search algorithms, print spooling, and more.

Each of these data structures has its own advantages and disadvantages, making them suitable for different scenarios. The choice of data structure depends on the specific requirements of the problem you're trying to solve.

Switch-Case Statements:

A switch-case statement is a control structure in programming that allows you to make decisions based on the value of an expression. It's often used to replace multiple if-else statements when you need to test a single expression against multiple possible values. From a data structure theory perspective, you can think of a switch-case statement as a way to efficiently map different cases (values) to corresponding actions or code blocks. This can be seen as a simplified form of data mapping.

Functions:

Functions, also known as procedures or subroutines, are blocks of code that can be called multiple times within a program to perform a specific task. They are integral to structured programming and help modularize code, making it more readable, reusable, and maintainable. While functions themselves aren't data structures, you can think of them as organizational units that encapsulate a set of instructions and can be thought of as a way to structure code logic.

While Loops:

A while loop is a control structure that repeatedly executes a block of code as long as a given condition remains true. In terms of data structure theory, a while loop can be seen as a mechanism for processing or iterating over a collection of data until a certain condition is met or some processing goal is achieved. The loop condition often involves checking data-related conditions to decide whether to continue looping or not.

Structures (Structs or Records):

Structures, also known as structs or records, are composite data types that allow you to group related data fields together. They are used to create user-defined data structures that represent complex entities with multiple attributes. While not exactly data structures in the sense of abstract data types (e.g., linked lists, trees), structures provide a way to organize and manage data in a program, resembling a simplified form of aggregating data.

Problem:02

The hospital administration wants to maintain separate queues for female and children patients. To achieve this, they need a dedicated repository that stores data exclusively for female and children patients. Now, your task is to create and develop this system to fulfill their requirement.

Problem Understanding and algorithm:

Include Necessary Libraries: The code includes necessary standard C libraries such as <stdio.h>, <stdlib.h>, and <string.h>.

Define the Patient Structure: The code defines a structure named Patient that holds information about a patient, including their name, age, and a pointer to the next patient in the list.

Implement the addPatient Function: The addPatient function takes the current head of the patient list, a patient's name, and age as arguments. It allocates memory for a new patient using malloc. If memory allocation fails, it prints an error message and returns the current head.

Otherwise, it copies the name and age to the new patient node, sets the next pointer to the current head, and returns the new patient node.

Implement the displayPatients Function: The displayPatients function takes the head of a patient list as an argument.It iterates through the list using a while loop and prints each patient's name and age.

Implement the main Function: In the main function, two pointers femalePatients and childrenPatients are initialized as NULL. These will be the heads of the linked lists for female and children patients, respectively. A do-while loop presents a menu of options to the user.

System Features for user :

- Add a female patient by entering their name and age.
- Add a children patient by entering their name and age.
- Display the list of female patients.
- Display the list of children patients.
- Exit the program.
- The user's choice is processed using a switch statement.
- Depending on the choice, appropriate functions are called or messages are printed.

Free Memory and Exit: After the user decides to exit the program (choice 0), the code enters a cleanup phase. It iterates through both the female and children patient lists using while loops and frees the memory allocated for each patient node. This is important to prevent memory leaks.

Return from main Function: The main function ends with a return 0 statement.

Code: Input

```
// Shakhar Das Rony ID:223014078//

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Define a structure for patient information
struct Patient {
    char name[50];
    int age;
    struct Patient *next;
};
// Function to add a new patient to the list
struct Patient* addPatient(struct Patient *head, const char *name, int age) {
    struct Patient *newPatient = (struct Patient*)malloc(sizeof(struct Patient));
    if (newPatient == NULL) {
        printf("Memory allocation failed.\n");
        return head;
    }
    strcpy(newPatient->name, name);
    newPatient->age = age;
    newPatient->next = head;
    return newPatient;
}
// Function to display the list of patients
void displayPatients(struct Patient *head) {
    printf("Patient List:\n");
    while (head != NULL) {
        printf("Name: %s, Age: %d\n", head->name, head->age);
        head = head->next;
    }
}
int main() {
    struct Patient *femalePatients = NULL;
    struct Patient *childrenPatients = NULL;

    int choice;
    do {
        printf("1. Add Female Patient\n");
        printf("2. Add Children Patient\n");
        printf("3. Display Female Patients\n");
        printf("4. Display Children Patients\n");
        printf("0. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: {
                char name[50];
                int age;
                printf("Enter patient name: ");
                scanf("%s", name);
                printf("Enter patient age: ");
                scanf("%d", &age);
                femalePatients = addPatient(femalePatients, name, age);
                printf("Female patient added.\n");
                break;
            }
            case 2: {
```

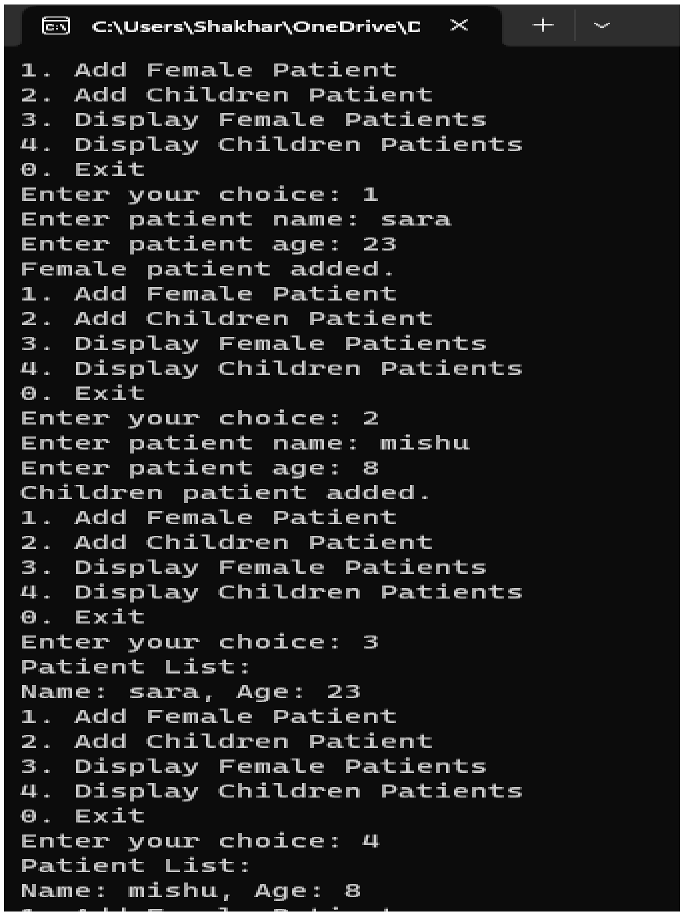
```
            char name[50];
            int age;
            printf("Enter patient name: ");
            scanf("%s", name);
            printf("Enter patient age: ");
            scanf("%d", &age);
            childrenPatients = addPatient(childrenPatients, name, age);
            printf("Children patient added.\n");
            break;
        }
        case 3:
            displayPatients(femalePatients);
            break;
        case 4:
            displayPatients(childrenPatients);
            break;
        case 0:
            printf("Exiting the program.\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
} while (choice != 0);

// Free the allocated memory before exiting
while (femalePatients != NULL) {
    struct Patient *temp = femalePatients;
    femalePatients = femalePatients->next;
    free(temp);
}
```

```
  while (childrenPatients != NULL) {
      struct Patient *temp = childrenPatients;
      childrenPatients = childrenPatients->next;
      free(temp);
  }

  return 0;
}
```

Output:



```
C:\Users\Shakhar\OneDrive\C
1. Add Female Patient
2. Add Children Patient
3. Display Female Patients
4. Display Children Patients
0. Exit
Enter your choice: 1
Enter patient name: sara
Enter patient age: 23
Female patient added.
1. Add Female Patient
2. Add Children Patient
3. Display Female Patients
4. Display Children Patients
0. Exit
Enter your choice: 2
Enter patient name: mishu
Enter patient age: 8
Children patient added.
1. Add Female Patient
2. Add Children Patient
3. Display Female Patients
4. Display Children Patients
0. Exit
Enter your choice: 3
Patient List:
Name: sara, Age: 23
1. Add Female Patient
2. Add Children Patient
3. Display Female Patients
4. Display Children Patients
0. Exit
Enter your choice: 4
Patient List:
Name: mishu, Age: 8
```

Code Explanation:

Structure Definition:

The program starts by defining a structure named Patient to hold patient information, including their name, age, and a pointer to the next patient in the linked list.

addPatient Function:

This function is responsible for adding a new patient to the linked list. It takes the current list (head), the name of the patient, and their age as input. It allocates memory for a new patient node, copies the name and age values into the node, and updates the next pointer to point to the previous head of the list. It then returns the new head of the list, which is the newly added patient.

displayPatients Function:

This function displays the list of patients. It takes the current list (head) as input and iterates through the list using a loop. For each patient, it prints out their name and age.

main Function:

In the main function, two pointers are initialized for two separate patient lists: femalePatients and childrenPatients. The program then enters a loop that displays a menu of options for the user to choose from. The options include adding a female patient, adding a children patient, displaying female patients, displaying children patients, and exiting the program.

For options 1 and 2, the user is prompted to enter the patient's name and age. The addPatient function is called with the appropriate list pointer (femalePatients or childrenPatients), and the new patient is added to the respective list.

For options 3 and 4, the displayPatients function is called with the corresponding list pointer to display the patient information.

Option 0 exits the program.

Memory Deallocation:

Before exiting the program, the allocated memory for the linked lists is freed to prevent memory leaks. This is done using a loop that iterates through the linked list and frees each node one by one.

Function Table:

addPatient Function:	From this function we can add new patients in our code
displayPatients Function:	In this function we can display the patients details from the previous used function
main Function:	This is our main function body
Library function	Their we use many library function like(scanf,printf,etc)

Problem 03:

Develop a system for doctor’s appointment management, where a user can book an appointment for a specific day. For this system you need to implement an appropriate data structure to manage the doctor’s appointment system. In this system, female and pediatric patients will get higher priority, and they will be served on a first-come, first-served basis.

Algorithm:

Book an appointment:

Users can enter patient details (name, gender, age, and appointment date), and the details are enqueued into the appointment queue.

Serve the next appointment:

The system displays the details of the next appointment in the queue (front of the queue) and dequeues it, simulating the process of serving an appointment.

Display all appointments:

The system displays the details of all appointments currently in the queue.

Exit: The program terminates.

Code:

```
//shakhar Das Rony ID:223014078

#include <stdio.h>

#include <stdlib.h>


// Define the structure for a patient's appointment

typedef struct {

   char name[50];

   char gender;

   int age;

   char appointmentDate[20];

} Appointment;

// Define a node structure for the linked list queue

typedef struct Node {

   Appointment appointment;
```

```
    struct Node* next;
} Node;
// Define the appointment queue
typedef struct {
    Node* front;
    Node* rear;
} Queue;
// Initialize an empty queue
void initializeQueue(Queue* queue) {
    queue->front = NULL;
    queue->rear = NULL;
}
// Check if the queue is empty
int isQueueEmpty(Queue* queue) {
    return queue->front == NULL;
}
// Enqueue an appointment into the queue
void enqueue(Queue* queue, Appointment appointment) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->appointment = appointment;
    newNode->next = NULL;
    if (isQueueEmpty(queue)) {
        queue->front = newNode;
        queue->rear = newNode;
    } else {
        queue->rear->next = newNode;
        queue->rear = newNode;
    }
}
```

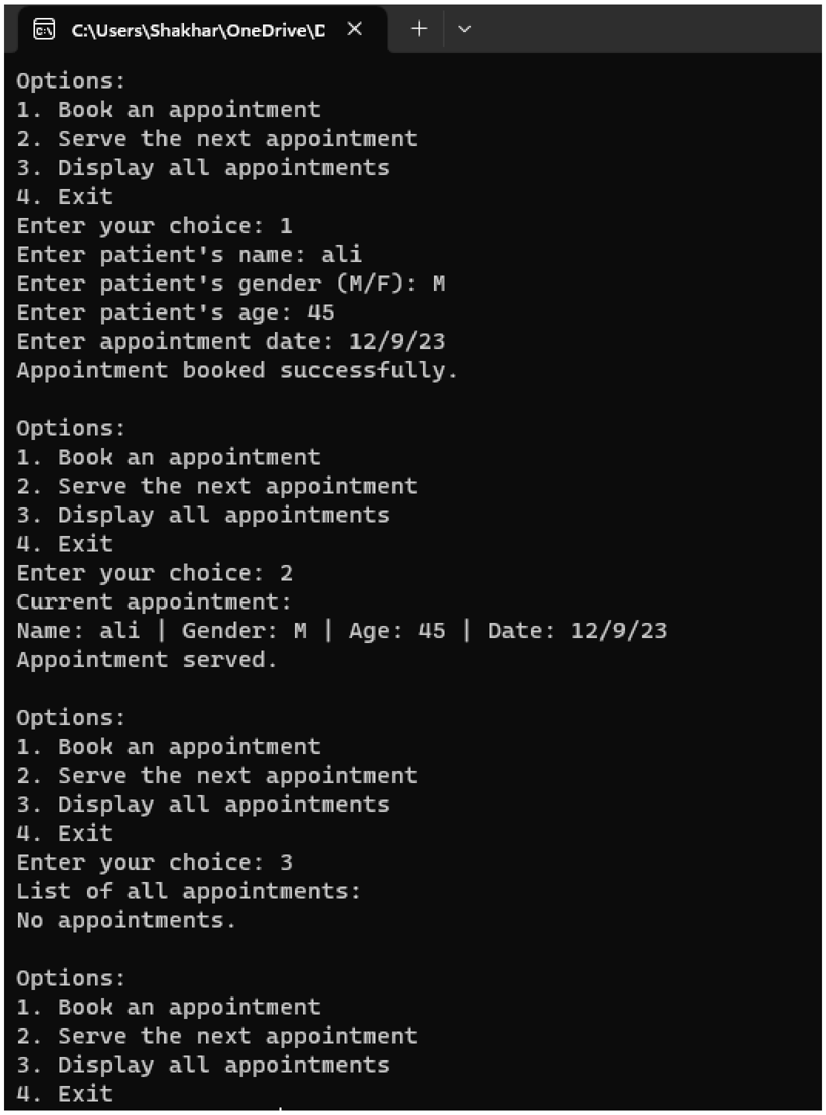
```
// Dequeue an appointment from the queue
void dequeue(Queue* queue) {
    if (isQueueEmpty(queue)) {
        printf("Queue is empty.\n");
    } else {
        Node* temp = queue->front;
        queue->front = queue->front->next;
        free(temp);
    }
}
// Display the appointments in the queue
void displayAppointments(Queue* queue) {
    if (isQueueEmpty(queue)) {
        printf("No appointments.\n");
    } else {
        Node* current = queue->front;
        while (current != NULL) {
            printf("Name: %s | Gender: %c | Age: %d | Date: %s\n",
                   current->appointment.name,
                   current->appointment.gender,
                   current->appointment.age,
                   current->appointment.appointmentDate);
            current = current->next;
        }
    }
}

int main() {
    Queue appointmentQueue;
    initializeQueue(&appointmentQueue);
```

```
while (1) {
    printf("\nOptions:\n");
    printf("1. Book an appointment\n");
    printf("2. Serve the next appointment\n");
    printf("3. Display all appointments\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    int choice;
    scanf("%d", &choice);
    switch (choice) {
        case 1: {
            Appointment newAppointment;
            printf("Enter patient's name: ");
            scanf("%s", newAppointment.name);
            printf("Enter patient's gender (M/F): ");
            scanf(" %c", &newAppointment.gender);
            printf("Enter patient's age: ");
            scanf("%d", &newAppointment.age);
            printf("Enter appointment date: ");
            scanf("%s", newAppointment.appointmentDate);
            enqueue(&appointmentQueue, newAppointment);
            printf("Appointment booked successfully.\n");
            break;
        }
        case 2: {
            if (!isQueueEmpty(&appointmentQueue)) {
                printf("Current appointment:\n");
                printf("Name: %s | Gender: %c | Age: %d | Date: %s\n",
                       appointmentQueue.front->appointment.name,
                       appointmentQueue.front->appointment.gender,
                    appointmentQueue.front->appointment.age,
                    appointmentQueue.front->appointment.appointmentDate);
                dequeue(&appointmentQueue);
                printf("Appointment served.\n");
            } else {
                printf("No appointments to serve.\n");
            }
            break;
        }
        case 3: {
            printf("List of all appointments:\n");
            displayAppointments(&appointmentQueue);
            break;
        }
        case 4: {
            printf("Exiting the program.\n");
            exit(0);
        }
        default: {
            printf("Invalid choice. Please select a valid option.\n");
        }
      }
    }

    return 0;
}
```

Output:



```
Options:
1. Book an appointment
2. Serve the next appointment
3. Display all appointments
4. Exit
Enter your choice: 1
Enter patient's name: ali
Enter patient's gender (M/F): M
Enter patient's age: 45
Enter appointment date: 12/9/23
Appointment booked successfully.

Options:
1. Book an appointment
2. Serve the next appointment
3. Display all appointments
4. Exit
Enter your choice: 2
Current appointment:
Name: ali | Gender: M | Age: 45 | Date: 12/9/23
Appointment served.

Options:
1. Book an appointment
2. Serve the next appointment
3. Display all appointments
4. Exit
Enter your choice: 3
List of all appointments:
No appointments.

Options:
1. Book an appointment
2. Serve the next appointment
3. Display all appointments
4. Exit
```

Code expalnation:

- Header Inclusions:

The code starts by including the necessary header files: stdio.h for input/output functions and stdlib.h for memory allocation.

- Appointment Structure:

The Appointment structure defines the information related to a patient's appointment, including their name, gender, age, and appointment date.

- Node Structure:

The Node structure defines a linked list node containing an Appointment and a pointer to the next node.

- Queue Structure:

The Queue structure holds pointers to the front and rear nodes of the queue.

- Function: initializeQueue

This function initializes an empty queue by setting both the front and rear pointers to NULL.

- Function: isQueueEmpty

This function checks whether the queue is empty by verifying if the front pointer is NULL.

- Function: enqueue

This function adds an appointment to the end of the queue. It allocates memory for a new node, populates it with the provided appointment data, and updates the front and rear pointers accordingly.

- Function: dequeue

This function removes the front appointment from the queue and deallocates the memory used by the associated node. It also updates the front pointer.

- Function: displayAppointments

This function displays all appointments in the queue. It iterates through the linked list and prints the details of each appointment.

- Main Function:

The main function contains the main logic of the program. It starts by initializing an empty appointment queue. It then enters an infinite loop where users can choose from several options:

Option 1: Book an appointment. Users enter patient details, and the appointment is added to the queue.

Option 2: Serve the next appointment. The details of the next appointment are displayed, and it is removed from the queue.

Option 3: Display all appointments. Displays the details of all appointments in the queue.

Option 4: Exit the program.

- Switch Case:

The program uses a switch-case statement to handle the user's choice. For each case, appropriate actions are taken based on the selected option.

- Exiting the Program:

The program exits when the user selects option 4. It displays a farewell message and uses the exit function to terminate the program.

This code demonstrates a basic implementation of a queue-based appointment management system using a linked list. Users can interact with the program to book appointments, serve them in the order they were booked, and view all appointments. Keep in mind that this is a simplified example, and real-world applications might require additional features and error handling.

conclusion:

The provided C code implements an appointment booking and management system using a linked list queue data structure. The program allows users to interactively book appointments, serve the next appointment, display all appointments, and exit the program. The code demonstrates fundamental concepts in programming such as data structures (queues), memory allocation, user input handling, and basic control flow.

The structure of the code is as follows:

It defines a structure Appointment to hold information about a patient's appointment.

It defines a structure Node to represent a node in the linked list queue.

It defines a structure Queue to hold the front and rear pointers of the queue.

The code initializes an empty queue using the initializeQueue function.

The program offers a menu-driven interface within a while loop, where users can choose from several options: booking an appointment, serving the next appointment, displaying all appointments, and exiting the program.

Depending on the user's choice, the program calls appropriate functions like enqueue, dequeue, and displayAppointments to perform the desired actions.

The program handles user input for appointment details, and after processing each operation, it provides appropriate feedback to the user.

****References:****

https://www.wikipedia.org/

https://www.google.co.uk/

class work

https://www.geeksforgeeks.org/

https://www.programiz.com/

https://www.w3schools.com/

https://www.tutorialspoint.com/

Etc.

----Thank you----