Hospital Management System

Abstract

The Hospital Management System is a console-based application developed in the C programming language.

It uses Data Structures and File Handling to manage hospital data such as patient details, doctor records, appointments, and billing information.

Each data type is stored in a suitable data structure:-

- Patients and Doctors are stored using Linked Lists
- Appointments are stored using an Array,
- Billing records are maintained using a Binary Search Tree (BST).

The system also uses File Handling to store and retrieve data permanently, ensuring that all records are preserved between sessions.

Objectives

- Manage patient, doctor, appointment, and billing information using C data structures.

- Demonstrate use of dynamic memory allocation, linked lists, arrays, and file handling.

Data Structures Used:

Component	Data Structure	Purpose
Patient Records	Singly Linked List	Dynamic addition and deletion of patients
Doctor Records	Singly Linked List	Easy insertion of new doctor records
Appointments	Array	Fixed maximum appointments and easy sequential access
Billing	BST	Sorted storage and efficient retrieval of billing data
File Handling	Binary Files	Permanent data storage (patients.dat, doctors.dat, etc.

Below are screenshots of the actual source file showing structs, functions, and main menu logic.

Source Code

#structure :-

```c
typedef struct Patient {
    int id;
    char name[50];
    int age;
    char gender[10];
    char disease[50];
    struct Patient *next;
} Patient;

typedef struct Doctor {
    int id;
    char name[50];
    char specialization[50];
    struct Doctor *next;
} Doctor;

typedef struct {
    int patientID;
    int doctorID;
    char date[15];
} Appointment;

typedef struct Bill {
    int billID;
    int patientID;
    float amount;
    struct Bill *left, *right;
} Bill;
```

Functional Modules:-

- Patient Management
- Add new patient record
- Display all patients
- Store/retrieve patient data from files

#Add Patient

```c
// patient
Patient* addPatient(Patient *head) {
    Patient *newNode = (Patient*)malloc(sizeof(Patient));
    printf("\nEnter Patient ID: ");
    scanf("%d", &newNode->id);
    printf("\nEnter Name: ");
    scanf("%s", newNode->name);
    printf("\nEnter Age: ");
    scanf("%d", &newNode->age);
    printf("\nEnter Gender: ");
    scanf("%s", newNode->gender);
    printf("\nEnter Disease: ");
    scanf("%s", newNode->disease);
    newNode->next = head;
    printf("\nPatient added successfully!\n");
    return newNode;
}
```

#Display Patient

```c
void displayPatients(Patient *head) {
    if (!head) {
        printf("No patients found.\n");
        return;
    }
    printf("\nPatient List\n");
    while (head) {
        printf("ID: %d | Name: %s | Age: %d | Gender: %s | Disease: %s\n",
               head->id, head->name, head->age, head->gender, head->disease);
        head = head->next;
    }
}
```

#store and retrieve data:

```c
void savePatients(Patient *head) {
    FILE *fp = fopen("patients.dat", "wb");
    if (!fp) return;
    while (head) {
        fwrite(head, sizeof(Patient) - sizeof(Patient*), 1, fp);
        head = head->next;
    }
    fclose(fp);
}

Patient* loadPatients() {
    FILE *fp = fopen("patients.dat", "rb");
    if (!fp) return NULL;
    Patient temp, *head = NULL;
    while (fread(&temp, sizeof(Patient) - sizeof(Patient*), 1, fp)) {
        Patient *newNode = (Patient*)malloc(sizeof(Patient));
        *newNode = temp;
        newNode->next = head;
        head = newNode;
    }
    fclose(fp);
    return head;
}
```

Doctor Management:-

- Add doctor record
- Display all doctors
- Store/retrieve doctor data from file

#Add Doctor:-

```c
// doctor
Doctor* addDoctor(Doctor *head) {
    Doctor *newNode = (Doctor*)malloc(sizeof(Doctor));
    printf("\nEnter Doctor ID: ");
    scanf("%d", &newNode->id);
    printf("\nEnter Name: ");
    scanf("%s", newNode->name);
    printf("\nEnter Specialization: ");
    scanf("%s", newNode->specialization);
    newNode->next = head;
    printf("\nDoctor added successfully\n");
    return newNode;
}
```

#Display doctors:-

```c
void displayDoctors(Doctor *head) {
    if (!head) {
        printf("No doctors found.\n");
        return;
    }
    printf("\nDoctor List \n");
    while (head) {
        printf("ID: %d | Name: %s | Specialization: %s\n",
               head->id, head->name, head->specialization);
        head = head->next;
    }
}
```

#Store/Retrive data:-

```c
void saveDoctors(Doctor *head) {
    FILE *fp = fopen("doctors.dat", "wb");
    if (!fp) return;
    while (head) {
        fwrite(head, sizeof(Doctor) - sizeof(Doctor*), 1, fp);
        head = head->next;
    }
    fclose(fp);
}
```

```c
Doctor* loadDoctors() {
    FILE *fp = fopen("doctors.dat", "rb");
    if (!fp) return NULL;
    Doctor temp, *head = NULL;
    while (fread(&temp, sizeof(Doctor) - sizeof(Doctor*), 1, fp)) {
        Doctor *newNode = (Doctor*)malloc(sizeof(Doctor));
        *newNode = temp;
        newNode->next = head;
        head = newNode;
    }
    fclose(fp);
    return head;
}
```

Appointment Management:

- Add appointments
- Display appointment list
- Save appointments to file

#Add appointment:-

```c
// appointment
void addAppointment(Appointment *appointments, int *count) {
    if (*count >= MAX_APPOINTMENTS) {
        printf("Appointment list full!\n");
        return;
    }
    printf("\nEnter Patient ID: ");
    scanf("%d", &appointments[*count].patientID);
    printf("Enter Doctor ID: ");
    scanf("%d", &appointments[*count].doctorID);
    printf("Enter Date (DD-MM-YYYY): ");
    scanf("%s", appointments[*count].date);
    (*count)++;
    printf("Appointment added successfully!\n");
}
```

#Display appointment:-

```c
void displayAppointments(Appointment *appointments, int count) {
    if (count == 0) {
        printf("No appointments found.\n");
        return;
    }
    printf("\n--- Appointments ---\n");
    for (int i = 0; i < count; i++) {
        printf("Patient ID: %d | Doctor ID: %d | Date: %s\n",
               appointments[i].patientID, appointments[i].doctorID, appointments[i].date);
    }
}
```

#Save appointment:-

```c
void saveAppointments(Appointment *appointments, int count) {
    FILE *fp = fopen("appointments.dat", "wb");
    if (!fp) return;
    fwrite(&count, sizeof(int), 1, fp);
    fwrite(appointments, sizeof(Appointment), count, fp);
    fclose(fp);
}

int loadAppointments(Appointment *appointments) {
    FILE *fp = fopen("appointments.dat", "rb");
    if (!fp) return 0;
    int count = 0;
    fread(&count, sizeof(int), 1, fp);
    fread(appointments, sizeof(Appointment), count, fp);
    fclose(fp);
    return count;
}
```

Billing System:-

- Create and view bills
- Store bills
- Save bills to file

#create and view Bills:-

```c
// bill
Bill* insertBill(Bill *root, int billID, int patientID, float amount) {
    if (!root) {
        Bill *newNode = (Bill*)malloc(sizeof(Bill));
        newNode->billID = billID;
        newNode->patientID = patientID;
        newNode->amount = amount;
        newNode->left = newNode->right = NULL;
        return newNode;
    }
    if (billID < root->billID)
        root->left = insertBill(root->left, billID, patientID, amount);
    else
        root->right = insertBill(root->right, billID, patientID, amount);
    return root;
}
```

```c
void inorderBills(Bill *root) {
    if (root) {
        inorderBills(root->left);
        printf("Bill ID: %d | Patient ID: %d | Amount: %.2f\n",
               root->billID, root->patientID, root->amount);
        inorderBills(root->right);
    }
}
```

#store and save bills

```c
void freeBills(Bill *root) {
    if (!root) return;
    freeBills(root->left);
    freeBills(root->right);
    free(root);
}


void write_node(Bill *n, FILE *fp) {
    if (!n) return;
    fwrite(&n->billID, sizeof(int), 1, fp);
    fwrite(&n->patientID, sizeof(int), 1, fp);
    fwrite(&n->amount, sizeof(float), 1, fp);
    write_node(n->left, fp);
    write_node(n->right, fp);
}

void saveBills(Bill *root) {
    FILE *fp = fopen("bills.dat", "wb");
    if (!fp) return;
    write_node(root, fp);
    fclose(fp);
}
```

```c
Bill* loadBills() {
    FILE *fp = fopen("bills.dat", "rb");
    if (!fp) return NULL;
    Bill *root = NULL;
    int billID, patientID;
    float amount;
    while (fread(&billID, sizeof(int), 1, fp) == 1) {
        if (fread(&patientID, sizeof(int), 1, fp) != 1) break;
        if (fread(&amount, sizeof(float), 1, fp) != 1) break;
        root = insertBill(root, billID, patientID, amount);
    }
    fclose(fp);
    return root;
}
```

6. Algorithms :-

(a) Add Patient

Algorithm AddPatient():

1. Create a new Patient node.

2. Input patient details (ID, Name, Age, Gender, Disease).

3. Insert the node at the beginning of the linked list.

4. Return updated head pointer.

(b) Add Doctor

Algorithm AddDoctor():

1. Create a new Doctor node.

2. Input doctor details (ID, Name, Specialization).

3. Insert the node at the beginning of the linked list.

4. Return updated head pointer.

(c) Add Appointment

Algorithm AddAppointment():

1. If count >= MAX_APPOINTMENTS:

   Display “List Full” and return.

2. Input patient ID, doctor ID, and date.

3. Store data in the next array position.

4. Increment appointment count.

(d) Insert Bill (Binary Search Tree)

Algorithm InsertBill(root, billID, patientID, amount):

1. If root == NULL:

   Create a new bill node and return it.

2. If billID < root->billID:

   root->left = InsertBill(root->left, billID, patientID, amount)

3. Else:

   root->right = InsertBill(root->right, billID, patientID, amount)

4. Return root.

8. File Handling Overview:-

File Name	Purpose
patients.dat	Stores patient linked list data
doctors.dat	Stores doctor linked list data
appointments.dat	Stores appointment array
bills.dat	Stores BST billing data

Sample Input / Output

```
---HOSPITAL MANAGEMENT SYSTEM---
1. Add Patient
2. View Patients
3. Add Doctor
4. View Doctors
5. Add Appointment
6. View Appointments
7. Add Bill
8. View Bills
9. Exit
Enter choice: 1

Enter Patient ID: 101

Enter Name: Rahul


Enter Age: 30

Enter Gender: Male

Enter Disease: Fever
```

```
Patient added successfully!
```

Conclusion

The Hospital Management System successfully demonstrates the use of Data Structures (Linked List, Array, BST) and File Handling in C programming for building a real-world style application.

It provides a structured and efficient approach to manage hospital data digitally, making it a valuable academic project.

References

- Let Us C by Yashavant Kanetkar
- GeeksforGeeks – Data Structures in C
- TutorialsPoint – File Handling in C