```c
#include <stdlib.h>
#include <stdio.h>

#define NAME_LEN 100

enum status { waiting, underTreatment, discharged, inCharge, notAvailable };

struct node { // For patients and docs
    int age;
    int ID;
    char *name;
    enum status stat;
    struct node *left;
    struct node *right;
};

struct hospital { // For hospital
    struct hospital *lef;
    struct hospital *rig;
    char *rray;
};

//string length function
int size(char *s) {
    int q = 0;
    while (s[q] != '\0') {
        q++;
    }
    return q;
}
//string copy function
void copy(char *t, char *s) {
    int i = 0;
    while (s[i] != '\0') {
        t[i] = s[i];
        i++;
    }
    t[i]='\0';
}

// Search in Binary Search Tree
struct node *searchTree(struct node *root, int id) {
    if (root == NULL) return NULL;
    if (id == root->ID) return root;
```

```c
    else if (id < root->ID) return searchTree(root->left, id);
    else return searchTree(root->right, id);
}

// Search in Stack (linked list)
struct node *searchStack(struct node **top, int id) {
    struct node *temp = *top;
    while (temp != NULL) {
        if (temp->ID == id) return temp;
        temp = temp->left;
    }
    return NULL;
}

// Search in Queue (linked list)
struct node *searchQueue(struct node **front, struct node **rear, int id) {
    struct node *temp = *front;
    while (temp != NULL) {
        if (temp->ID == id) return temp;
        temp = temp->left;
    }
    return NULL;
}

void display(struct node *p) {
    if (p == NULL) {
        printf("No patient to display.\n");
        return;
    }
    printf("Name: %s\n", p->name);
    printf("ID: %d\n", p->ID);
    printf("Age: %d\n", p->age);
    printf("Status: ");
    switch (p->stat) {
        case 0: // waiting
            printf("Waiting");
            break;
        case 1: // underTreatment
            printf("Under Treatment");
            break;
        case 2: // discharged
            printf("Discharged");
            break;
        case 3: // inCharge
            printf("In Charge");
```

```
            break;
        case 4: // notAvailable
            printf("Not Available");
            break;
        default:
            printf("Unknown");
            break;
    }
    printf("\n");
}

// Push onto Stack
struct node *push(struct node **top, struct node *p) {
    if (p == NULL) return *top;
    p->left = *top;
    *top = p;
    return p;
}

// Enqueue in Queue
struct node *enqueue(struct node **front, struct node **rear, struct node *p) {
    if (p == NULL) return NULL;
    p->left = NULL;
    if (*rear == NULL) {
        *front = p;
        *rear = p;
    } else {
        (*rear)->left = p;
        *rear = p;
    }
    return p;
}

// Dequeue from Queue
struct node *dequeue(struct node **front, struct node **rear) {
    if (*front == NULL) return NULL;
    struct node *p = *front;
    *front = (*front)->left;
    if (*front == NULL) *rear = NULL;
    p->left = NULL;
    return p;
}

// Find min in BST
struct node* findMin(struct node* node) {
```

```
    while (node !=NULL && node->left != NULL){
        node = node->left;}
    return node;
}

// Delete from BST
struct node *deleteFromTree(struct node *root, int id) {
    if (root == NULL) return root;

    if (id < root->ID) {
        root->left = deleteFromTree(root->left, id);
    } else if (id > root->ID) {
        root->right = deleteFromTree(root->right, id);
    } else {
        if (root->left == NULL) {
            struct node* temp = root->right;
            free(root->name);
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct node* temp = root->left;
            free(root->name);
            free(root);
            return temp;
        }
        struct node* temp = findMin(root->right);
        root->ID = temp->ID;
        root->age = temp->age;
        root->stat = temp->stat;

        free(root->name);
        root->name = (char*)malloc(sizeof(char)*(size(temp->name) + 1));
        copy(root->name, temp->name);

        root->right = deleteFromTree(root->right, temp->ID);
    }
    return root;
}

// Insert into Linked List
void insert(struct node **head, char *nam, int g, int d, enum status s) {
    struct node *new = (struct node*)malloc(sizeof(struct node));
    if (new == NULL) {
        printf("Memory allocation failed!\n");
        return;
```

```
    }

    new->age = g;
    new->ID = d;
    new->stat = s;

    int length = size(nam);
    new->name = (char*)malloc((length + 1) * sizeof(char));
    if (new->name == NULL) {
        printf("Memory allocation failed!\n");
        free(new);
        return;
    }
    copy(new->name, nam);

    new->left = *head;
    new->right = NULL;
    *head = new;
}

// Search in LLD
struct node *searchLLD(struct node **head, int Id) {
    struct node *p = *head;
    while (p != NULL) {
        if (p->ID == Id) return p;
        p = p->left;
    }
    return NULL;
}

struct node *create(char *nam, int ag, int id) {
    struct node *new = (struct node *)malloc(sizeof(struct node));
    if (new == NULL) {
        printf("Memory allocation failed!\n");
        return NULL;
    }
    new->left = NULL;
    new->right = NULL;
    new->age = ag;
    new->ID = id;
    new->stat = waiting;  // Default status

    int d = size(nam);
    new->name = (char *)malloc((d + 1)* sizeof(char));
    if (new->name == NULL) {
```

```
        printf("Memory allocation failed!\n");
        free(new);
        return NULL;
    }
    copy(new->name, nam);

    return new;
}

struct node *add(struct node *root, struct node *p) {
    if (root == NULL) {
        return p;
    }
    if (p->ID < root->ID) {
        root->left = add(root->left, p);
    } else if (p->ID > root->ID) {
        root->right = add(root->right, p);
    }
    return root;
}

void displayInorder(struct node* root) {
    if (root == NULL) {
        printf("Tree is empty.\n");
        return;
    }
    if (root->left != NULL) displayInorder(root->left);

    printf("ID: %d\n", root->ID);
    printf("Age: %d\n", root->age);
    printf("Name: %s\n", root->name);
    printf("Status: ");
    switch (root->stat) {
        case 0:
            printf("Waiting\n");
            break;
        case 1:
            printf("Under Treatment\n");
            break;
        case 2:
            printf("Discharged\n");
            break;
        default:
            printf("Unknown\n");
            break;
```

```
    }
    printf("\n");

    if (root->right != NULL) displayInorder(root->right);
}

void treePreorder(struct hospital* rot) {
    if (rot != NULL) {
        printf("%s\n", rot->rray);
        treePreorder(rot->lef);
        treePreorder(rot->rig);
    }
}

struct node *pop(struct node **top) {
    struct node *p;
    if (*top == NULL) {
        printf("\nStack is empty.\n");
        return NULL;
    } else {
        p = *top;
        *top = (*top)->left;
        p->left = NULL;
        return p;
    }
}

void deleteFromQueue(struct node **front, struct node **rear, struct node *p) {
    if (*front == NULL || p == NULL) return;
    if (*front == p) {
        *front = p->left;
        if (*rear == p) *rear = NULL;
        p->left = NULL;
        return;
    }
    struct node *temp = *front;
    while (temp != NULL && temp->left != p) {
        temp = temp->left;
    }
    if (temp == NULL) return;
    temp->left = p->left;
    if (*rear == p) *rear = temp;
    p->left = NULL;
}
```

```c
void displayStack(struct node *top) {
    if (top == NULL) {
        printf("Stack is empty.\n");
        return;
    }
    printf("Discharged Patients:\n");
    struct node *temp = top;
    while (temp != NULL) {
        display(temp);
        temp = temp->left;
    }
}

void deleteLLD(struct node **head, int id) {
    struct node *p = *head;
    struct node *d = NULL;
    while (p != NULL) {
        if (p->ID == id) {
            if (d == NULL) {//d is previous node
                *head = p->left;
            } else {
                d->left = p->left;
            }
            free(p->name);
            free(p);
            printf("Doctor with ID %d deleted successfully.\n", id);
            return;
        }
        d = p;
        p = p->left;
    }
    printf("Doctor with ID %d not found.\n", id);
}

void deleteFromStack(struct node **top, struct node *p) {
    if (*top == NULL || p == NULL) return;
    struct node *temp = *top;
    struct node *d = NULL; //d the previous node
    while (temp != NULL) {
        if (temp == p) {
            if (d == NULL) *top = temp->left;
            else d->left = temp->left;
            temp->left = NULL;
            return;
        }
```

```
        d = temp;
        temp = temp->left;
    }
}

// Save tree to file function
void saveTree(struct node *root, FILE *fp) {
    if (root != NULL) {
        fprintf(fp, "%d %d %s %d\n", root->ID, root->age, root->name, root-
>stat);
        saveTree(root->left, fp);
        saveTree(root->right, fp);
    }
}

// Save queue to file
void saveQueue(struct node *front, FILE *fp) {
    struct node *temp = front;
    while (temp != NULL) {
        fprintf(fp, "%d %d %s %d\n", temp->ID, temp->age, temp->name, temp-
>stat);
        temp = temp->left;
    }
}

// Save stack to file
void saveStack(struct node *top, FILE *fp) {
    struct node *temp = top;
    while (temp != NULL) {
        fprintf(fp, "%d %d %s %d\n", temp->ID, temp->age, temp->name, temp-
>stat);
        temp = temp->left;
    }
}

// Save doctors to file
void saveDoctors(struct node *head, FILE *fp) {
    struct node *temp = head;
    while (temp != NULL) {
        fprintf(fp, "%d %d %s %d\n", temp->ID, temp->age, temp->name, temp-
>stat);
        temp = temp->left;
    }
}
```

```c
// Load data from file
void loadFromFile(struct node **root, struct node **top, struct node **front,
struct node **rear, struct node **head) {
    FILE *fp = fopen("hospital_data.txt", "r");
    if (fp == NULL) {
        printf("No saved data found or error opening file.\n");
        return;
    }

    int id, age, status;
    char name[100];

    printf("Loading data from file...\n");
    while (fscanf(fp, "%d %d %s %d", &id, &age, name, &status) == 4) {
        struct node *p = create(name, age, id);
        if (p != NULL) {
            p->stat = (enum status)status;

            if (status == waiting) {
                enqueue(front, rear, p);
            } else if (status == underTreatment) {
                *root = add(*root, p);
            } else if (status == discharged) {
                push(top, p);
            } else if (status == inCharge || status == notAvailable) {
                insert(head, name, age, id, (enum status)status);
                free(p->name);
                free(p);
            }
        }
    }

    fclose(fp);
    printf("Data loaded successfully.\n");
}

int main() {
    struct hospital *rot = (struct hospital*)malloc(sizeof(struct hospital));
    rot->rray ="Hospital";
    rot->lef = (struct hospital*)malloc(sizeof(struct hospital));
    rot->rig = (struct hospital*)malloc(sizeof(struct hospital));

    rot->lef->rray = "Cardiology Department";

    rot->rig->rray =  "Pediatrics Department";
```

```
    // Left subtree (Cardiology)
    rot->lef->lef = (struct hospital*)malloc(sizeof(struct hospital));
    rot->lef->rig = (struct hospital*)malloc(sizeof(struct hospital));
    rot->lef->lef->rray = "Outpatient Service";
    rot->lef->rig->rray =  "Emergency Service";

    // Right subtree (Pediatrics)
    rot->rig->lef = (struct hospital*)malloc(sizeof(struct hospital));
    rot->rig->rig = (struct hospital*)malloc(sizeof(struct hospital));
    rot->rig->lef->rray =  "Inpatient Services";
    rot->rig->rig->rray = "NICU Team";
//leaves right and left are set to NULL
    rot->lef->lef->lef = rot->lef->lef->rig = NULL;
    rot->lef->rig->lef = rot->lef->rig->rig = NULL;
    rot->rig->lef->lef = rot->rig->lef->rig = NULL;
    rot->rig->rig->lef = rot->rig->rig->rig = NULL;

    struct node *root = NULL;
    struct node *top = NULL;
    struct node *front = NULL;
    struct node *rear = NULL;
    struct node *head = NULL;
    int choice;

    do {
        printf("\n===== Hospital Management =====\n");
        printf("1. Manage Patients\n");
        printf("2. Manage Doctors\n");
        printf("3. Discharge Patient\n");
        printf("4. View Waiting Queue\n");
        printf("5. Add Patient to Queue\n");
        printf("6. Undo Last Discharge\n");
        printf("7. Search Patient in Directory Tree\n");
        printf("8. View Hospital Structure Tree\n");
        printf("9. Save Data to File\n");
        printf("10. Load Data from File\n");
        printf("11. Exit\n");
        printf("Choose an option: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: { // manage patients
                int ch1;
                do {
```

```
                    printf("\n-- Manage Patients --\n");
                    printf("1. Add Patient\n");
                    printf("2. Edit Patient\n");
                    printf("3. View Patient\n");
                    printf("4. Delete Patient\n");
                    printf("5. View All Patients\n");
                    printf("6. Back\n");
                    printf("Choose an option: ");
                    scanf("%d", &ch1);

                    if (ch1 == 6) break;
                    switch (ch1) {
                        case 1: { // Add patient
                            char h[100];
                            int g, d, x;
                            printf("Enter patient name: ");
                            scanf("%99s", h);
                            printf("Enter patient ID: ");
                            scanf("%d", &d);
                            printf("Enter patient age: ");
                            scanf("%d", &g);
                            struct node *p = create(h, g, d);
                            if (p == NULL) break;
                            printf("Enter new status:\n0 - Waiting\n1 - Under
Treatment\n2 - Discharged\n");
                            scanf("%d", &x);
                            if (x < 0 || x > 2) {
                                printf("Invalid status.\n");
                                free(p->name);
                                free(p);
                                break;
                            }
                            p->stat = (enum status)x;
                            if (p->stat == waiting) {
                                printf("Adding to queue...\n");
                                enqueue(&front, &rear, p);
                            } else if (p->stat == discharged) {
                                printf("Pushing to stack...\n");
                                push(&top, p);
                            } else {  // underTreatment
                                printf("Adding to BST...\n");
                                root = add(root, p);
                            }
                            printf("Patient added successfully.\n");
                            break;
```

```
                        }
                        case 2: { // edit patient
                            int ed;
                            printf("Enter patient ID: ");
                            scanf("%d", &ed);
                            struct node *p = searchTree(root, ed);
                            if (!p) p = searchQueue(&front, &rear, ed);
                            if (!p) p = searchStack(&top, ed);
                            if (!p) {
                                printf("Patient with ID %d not found.\n", ed);
                                break;
                            }

                            int ch2;
                            do {
                                printf("\nEdit Patient Menu:\n");
                                printf("1. Name\n2. Age\n3. ID\n4. Status\n5.
Back\n");

                                printf("Choose an option: ");
                                scanf("%d", &ch2);
                                if (ch2 == 5) break;

                                switch (ch2) {
                                    case 1: { // name
                                        char nam[100];
                                        printf("Enter new name: ");
                                        scanf("%99s", nam);
                                        free(p->name);//free previous name
                                        int length = size(nam);
                                        p->name = (char*)malloc((length+1) *
sizeof(char));

                                        if (p->name == NULL) {
                                            printf("Memory allocation
failed.\n");

                                            break;
                                        }
                                        copy(p->name, nam);
                                        printf("Name updated.\n");
                                        break;
                                    }
                                    case 2: { // age
                                        int ag;
                                        printf("Enter new age: ");
                                        scanf("%d", &ag);
                                        p->age = ag;
```

```
                                    printf("Age updated.\n");
                                    break;
                                }
                                case 3: { // id
                                    int ID2;
                                    printf("Enter new ID: ");
                                    scanf("%d", &ID2);
                                    p->ID = ID2;
                                    printf("ID updated.\n");
                                    break;
                                }
                                case 4: { // status
                                    int newstat;
                                    printf("Enter new status:\n0 - Waiting\n1
- Under Treatment\n2 - Discharged\n");
                                    scanf("%d", &newstat);
                                    if (newstat < 0 || newstat > 2) {
                                        printf("Invalid status.\n");
                                        break;
                                    }
                                    if (p->stat != newstat) {
                                        // Remove from old structure
                                        if (p->stat == waiting) {
                                            deleteFromQueue(&front, &rear,
p);
                                        } else if (p->stat == underTreatment)
{
                                            root = deleteFromTree(root, p-
>ID);
                                        } else if (p->stat == discharged) {
                                            deleteFromStack(&top, p);
                                        }//Theres no free so p is saved

                                        p->stat = newstat;//update state
                                        // Add to new structure
                                        if (newstat == waiting) {
                                            enqueue(&front, &rear, p);
                                        } else if (newstat == underTreatment)
{
                                            root = add(root, p);
                                        } else {
                                            push(&top, p);
                                        }
                                        printf("Status updated and patient
moved.\n");
```

```
                                    } else {
                                        printf("Status is already set to this
value.\n");
                                    }
                                    break;
                                }
                                default:
                                    printf("Invalid choice.\n");
                                    break;
                            }
                        } while (ch2 != 5);
                        break;
                    }
                    case 3: { // view patient
                        int vId;
                        printf("Enter ID: ");
                        scanf("%d", &vId);
                        struct node *p = searchTree(root, vId);
                        if (!p) p = searchQueue(&front, &rear, vId);
                        if (!p) p = searchStack(&top, vId);
                        if (!p) {
                            printf("Patient with ID %d not found.\n", vId);
                        } else {
                            display(p);
                        }
                        break;
                    }
                    case 4: { // delete patient (move to discharge)
                        int dId;
                        printf("Enter ID: ");
                        scanf("%d", &dId);
                        struct node *p = searchTree(root, dId);
                        if (!p) p = searchQueue(&front, &rear, dId);
                        if (!p) p = searchStack(&top, dId);
                        if (!p) {
                            printf("Patient with ID %d not found.\n", dId);
                        } else {
                            // Remove from current structure
                            if (p->stat == waiting) {
                                deleteFromQueue(&front, &rear, p);
                            } else if (p->stat == underTreatment) {
                                root = deleteFromTree(root, p->ID);
                            } else {
                                deleteFromStack(&top, p);
                            }
```

```
                                p->stat = discharged;
                                push(&top, p);
                                printf("Patient deleted and moved to discharged
stack.\n");
                            }
                            break;
                        }
                        case 5: { // View All Patients
                            printf("\nPatients under treatment (inorderBST):\n");
                            if (root == NULL) {
                                printf("No patients under treatment.\n");
                            } else {
                                displayInorder(root);
                            }

                            printf("\nPatients waiting (queue):\n");
                            if (front == NULL) {
                                printf("Queue is empty.\n");
                            } else {
                                struct node *temp = front;
                                while (temp != NULL) {
                                    display(temp);
                                    temp = temp->left;
                                }
                            }

                            printf("\nPatients discharged (stack):\n");
                            displayStack(top);
                            break;
                        }
                        default:
                            printf("Invalid choice.\n");
                            break;
                    }
                } while (1);
                break;
            }
            case 2: { // manage doctors
                int ch1;
                do {
                    printf("\n-- Manage Doctors --\n");
                    printf("1. Add Doctor\n2. Edit Doctor\n3. View Doctor\n4.
Delete Doctor\n5. View All Doctors\n6. Back\n");
                    printf("Choose an option: ");
                    scanf("%d", &ch1);
```

```
                    if (ch1 == 6) break;

                    switch (ch1) {
                        case 1: { // add doctor
                            char h[100];
                            int g, d, s;
                            printf("Enter Doctor name: ");
                            scanf("%99s", h);
                            printf("Enter Doctor ID: ");
                            scanf("%d", &d);
                            printf("Enter Doctor age: ");
                            scanf("%d", &g);
                            printf("Enter Doctor status:\n3 - In Charge\n4 - Not
Available\n");
                            scanf("%d", &s);
                            if (s == 3) {
                                insert(&head, h, g, d, inCharge);
                                printf("Doctor added successfully.\n");
                            } else if (s == 4) {
                                insert(&head, h, g, d, notAvailable);
                                printf("Doctor added successfully.\n");
                            } else {
                                printf("Invalid status.\n");
                            }
                            break;
                        }
                        case 2: { // edit doc
                            int ID;
                            printf("Enter Doctor ID: ");
                            scanf("%d", &ID);
                            struct node *p = searchLLD(&head, ID);
                            if (p == NULL) {
                                printf("Doctor not found.\n");
                                break;
                            }

                            int ch2;
                            do {
                                printf("What do you want to edit?\n1. Name\n2.
Age\n3. ID\n4. Status\n5. Back\n");
                                scanf("%d", &ch2);
                                if (ch2 == 5) break;

                                switch (ch2) {
```

```
                                    case 1: { // name
                                        char nam[100];
                                        printf("Enter new name: ");
                                        scanf("%99s", nam);
                                        free(p->name);
                                        int lengt = size(nam);
                                        p->name = (char*)malloc((lengt +1) *
sizeof(char));
                                        if (p->name == NULL) {
                                            printf("Memory allocation
failed.\n");
                                            break;
                                        }
                                        copy(p->name, nam);
                                        printf("Name updated.\n");
                                        break;
                                    }
                                    case 2: { // age
                                        int ag;
                                        printf("Enter new age: ");
                                        scanf("%d", &ag);
                                        p->age = ag;
                                        printf("Age updated.\n");
                                        break;
                                    }
                                    case 3: { // id
                                        int ID2;
                                        printf("Enter new ID: ");
                                        scanf("%d", &ID2);
                                        p->ID = ID2;
                                        printf("ID updated.\n");
                                        break;
                                    }
                                    case 4: { // status
                                        int x;
                                        printf("Enter new status:\n3 - In
Charge\n4 - Not Available\n");
                                        scanf("%d", &x);
                                        if (x == 3 || x == 4) {
                                            p->stat = x;
                                            printf("Status updated.\n");
                                        } else {
                                            printf("Invalid status.\n");
                                        }
                                        break;
```

```
                        }
                        default:
                            printf("Invalid choice.\n");
                            break;
                    }
                } while (ch2 != 5);
                break;
            }
            case 3: { // view doc
                int Id;
                printf("Enter ID: ");
                scanf("%d", &Id);
                struct node *q = searchLLD(&head, Id);
                if (q) {
                    display(q);
                } else {
                    printf("Doctor not found.\n");
                }
                break;
            }
            case 4: { // delete doc
                int Id;
                printf("Enter ID: ");
                scanf("%d", &Id);
                deleteLLD(&head, Id);
                break;
            }
            case 5: { // Display All Doctors
                if (head == NULL) {
                    printf("No doctors in the system.\n");
                } else {
                    printf("All Doctors:\n");
                    struct node *temp = head;
                    while (temp) {
                        display(temp);
                        temp = temp->left;
                    }
                }
                break;
            }
            default:
                printf("Invalid choice.\n");
                break;
        }
    } while (1);
```

```
            break;
        }
        case 3: { // discharge patient
            int dId;
            printf("Enter ID: ");
            scanf("%d", &dId);
            struct node *p = searchTree(root, dId);// search patient
            if (!p) p = searchQueue(&front, &rear, dId);
            if (!p) p = searchStack(&top, dId);
            if (!p) {
                printf("Patient with ID %d not found.\n", dId);
            } else {
                // Remove from current structure
                if (p->stat == waiting) {
                    deleteFromQueue(&front, &rear, p);
                } else if (p->stat == underTreatment) {
                    root = deleteFromTree(root, p->ID);
                } else {
                    deleteFromStack(&top, p);
                }
                p->stat = discharged;
                push(&top, p);
                printf("Patient discharged successfully.\n");
            }
            break;
        }
        case 4: { // View Waiting Queue
            printf("\n-- Waiting Queue --\n");
            if (front == NULL) {
                printf("Queue is empty.\n");
            } else {
                printf("Patients waiting for treatment:\n");
                struct node *temp = front;
                int position = 1; //How long the patient will wait
                while (temp != NULL) {
                    printf("Position %d:\n", position);
                    display(temp);
                    temp = temp->left;
                    position++;
                }
            }
            break;
        }
        case 5: { // Add Patient to Queue
            int z;
```

```
            printf("Already existing patient?\n");
            printf("1 - Yes\t0 - No\n");
            scanf("%d", &z);
            if (z ==1) {
                int patId;
                printf("Enter patient ID: ");
                scanf("%d", &patId);
                struct node *p = searchTree(root, patId);
                if (!p) p = searchStack(&top, patId);
                if (p) {
                    // Remove from current structure
                    if (p->stat == underTreatment) {
                        root = deleteFromTree(root, p->ID);
                    } else if (p->stat == discharged) {
                        deleteFromStack(&top, p);
                    }
                    p->stat = waiting;
                    enqueue(&front, &rear, p);
                    printf("Patient added to waiting queue.\n");
                } else {
                    printf("Patient not found or already in queue.\n");
                }
            } else if (z==0){
                char h[100];
                int g, d;
                printf("Enter patient name: ");
                scanf("%99s", h);
                printf("Enter patient ID: ");
                scanf("%d", &d);
                printf("Enter patient age: ");
                scanf("%d", &g);
                struct node *p = create(h, g, d);
                if (p != NULL) {
                    p->stat = waiting;
                    enqueue(&front, &rear, p);
                    printf("New patient added to waiting queue.\n");
                }
            }
            break;
        }
        case 6: { // Undo Last Discharge
            struct node *par = pop(&top);
            if (par == NULL) {
                printf("No discharged patients to undo.\n");
            } else {
```

```
                    printf("Undoing discharge for patient: %s with ID: %d\n",
par->name, par->ID);
                    printf("Where would you like to move the patient?\n");
                    printf("0 - Waiting Queue\n1 - Under Treatment\n");
                    int choice;
                    scanf("%d", &choice);

                    if (choice == 0) {
                        par->stat = waiting;
                        enqueue(&front, &rear, par);
                        printf("Patient moved to waiting queue.\n");
                    } else if (choice == 1) {
                        par->stat = underTreatment;
                        root = add(root, par);
                        printf("Patient moved to under treatment.\n");
                    } else {
                        printf("Invalid choice. patient is not removed.\n");
                        push(&top, par);
                    }
                }
                break;
            }
            case 7: { // Search Patient in Directory Tree
                int searchId;
                printf("Enter patient ID to search: ");
                scanf("%d", &searchId);

                printf("Searching in all patient records...\n");

                // Search in BST (under treatment)
                struct node *p = searchTree(root, searchId);
                if (p) {
                    printf("Patient found in treatment directory (BST):\n");
                    display(p);
                    break;
                }
                if (!p){// Search in Queue (waiting)
                    p = searchQueue(&front, &rear, searchId);
                    if (p) {
                        printf("Patient found in waiting Queue:\n");
                        display(p);
                        break;
                    }
                    if (!p){
                        // Search in Stack (discharged)
```

```
                        p = searchStack(&top, searchId);
                        if (p) {
                            printf("Patient found in Discharged list:\n");
                            display(p);
                            break;
                        }
                    }
                }
                printf("Patient with ID %d not found in any records.\n",
searchId);
                break;
            }
            case 8: { // View Hospital Structure Tree
                printf("\n-- Hospital Structure Tree --\n");
                printf("Hospital Organization (Preorder Traversal):\n");
                treePreorder(rot);
                break;
            }
            case 9: { // Save Data to File
                FILE *fp = fopen("hospital_data.txt", "w");
                if (fp == NULL) {
                    printf("Error opening file for writing.\n");
                    break;
                }

                printf("Saving hospital data...\n");

                // Save patients under treatment (BST)
                if (root != NULL) {
                    saveTree(root, fp);
                }
                // Save waiting patients (Queue)
                if (front != NULL) {
                    saveQueue(front, fp);
                }
                // Save discharged patients (Stack)
                if (top != NULL) {
                    saveStack(top, fp);
                }

                // Save doctors (Linked List)
                if (head != NULL) {
                    saveDoctors(head, fp);
                }
```

```
                fclose(fp);
                printf("Data saved successfully to hospital_data.txt\n");
                break;
            }
            case 10: { // Load Data from File
                printf("Loading data will replace current data. Continue? (1-Yes,
0-No): ");
                int c;
                scanf("%d", &c);
                if (c == 1) {
                    // Clear current data structures
                    root = NULL;
                    top = NULL;
                    front = NULL;
                    rear = NULL;
                    head = NULL;

                    loadFromFile(&root, &top, &front, &rear, &head);
                } else {
                    printf("Load operation cancelled.\n");
                }
                break;
            }
            case 11: { // Exit
                printf("Do you want to save data before exiting? (1-Yes, 0-No):
");
                int choice;
                scanf("%d", &choice);
                if (choice == 1) {
                    FILE *fp = fopen("hospital_data.txt", "w");
                    if (fp != NULL) {
                        if (root != NULL) saveTree(root, fp);
                        if (front != NULL) saveQueue(front, fp);
                        if (top != NULL) saveStack(top, fp);
                        if (head != NULL) saveDoctors(head, fp);
                        fclose(fp);
                        printf("Data saved successfully.\n");
                    }
                }
                printf("Thanks !\n");
                break;
            }
            default:
                printf("Invalid choice ,try again.\n");
                break;
```

```
        }
    } while (choice != 11);

    // Free allocated memory for hospital structure
    free(rot->lef->lef->rray);
    free(rot->lef->lef);
    free(rot->lef->rig->rray);
    free(rot->lef->rig);
    free(rot->rig->lef->rray);
    free(rot->rig->lef);
    free(rot->rig->rig->rray);
    free(rot->rig->rig);
    free(rot->lef->rray);
    free(rot->lef);
    free(rot->rig->rray);
    free(rot->rig);
    free(rot->rray);
    free(rot);

    return 0;
}
```