----------tasks----------

1.

```
#include<stdio.h>
#include<string.h>
#include<stdbool.h>
#define MAX_PATIENTS 30

struct medical_history{
    char disease[100];
    int year_diagnosed;
};

union optional_data{
    char allergies[100];
    char other_notes[100];
};

struct patient{
    char name[100];
    int age;
    char gender[10];
    struct medical_history history;
    union optional_data extra_info;
    bool has_allergies;
};

struct patient patients[MAX_PATIENTS];
const char hospital_name[] = "XYZ Hospital";
int patient_count = 0;
```

```
void add_patient(){
    if(patient_count < MAX_PATIENTS){
        struct patient *new_patient = &patients[patient_count];

        printf("Enter the patient name: ");
        getchar();
        fgets(new_patient->name, sizeof(new_patient->name), stdin);
        new_patient->name[strcspn(new_patient->name, "\n")] = '\0';

        printf("Enter the age: ");
        scanf("%d", &new_patient->age);

        printf("Enter the gender: ");
        getchar();
        fgets(new_patient->gender, sizeof(new_patient->gender), stdin);
        new_patient->gender[strcspn(new_patient->gender, "\n")] = '\0';

        printf("Enter the disease name: ");
        fgets(new_patient->history.disease, sizeof(new_patient->history.disease), stdin);
        new_patient->history.disease[strcspn(new_patient->history.disease, "\n")] = '\0';

        printf("Enter the year diagnosed: ");
        scanf("%d", &new_patient->history.year_diagnosed);

        printf("Does the patient have allergies? (1 for yes, 0 for no): ");
        scanf("%d", &new_patient->has_allergies);
        if(new_patient->has_allergies){
            printf("Enter allergy details: ");
```

```
            getchar();
            fgets(new_patient->extra_info.allergies, sizeof(new_patient->extra_info.allergies), stdin);
            new_patient->extra_info.allergies[strcspn(new_patient->extra_info.allergies, "\n")] = '\0';
        }

        patient_count++;
        printf("Patient added successfully!\n");
    }else{
        printf("Patient limit reached!\n");
    }
}

void view_patient_details(){
    int patient_id;
    printf("Enter patient ID to view details (0 to %d): ", patient_count - 1);
    scanf("%d", &patient_id);

    if(patient_id >= 0 && patient_id < patient_count){
        struct patient *patient = &patients[patient_id];

        printf("\nPatient Name: %s\n", patient->name);
        printf("Age: %d\n", patient->age);
        printf("Gender: %s\n", patient->gender);
        printf("Medical History:\n");
        printf("Disease: %s, Year Diagnosed: %d\n", patient->history.disease, patient->history.year_diagnosed);

        if(patient->has_allergies){
            printf("Allergies: %s\n", patient->extra_info.allergies);
```

```
        }else{
            printf("No allergies recorded.\n");
        }
    }else{
        printf("Invalid patient ID!\n");
    }
}

void update_patient_info(){
    int patient_id;
    printf("Enter patient ID to update: ");
    scanf("%d", &patient_id);

    if(patient_id >= 0 && patient_id < patient_count){
        struct patient *patient = &patients[patient_id];

        printf("Updating details for %s:\n", patient->name);

        printf("Enter new age: ");
        scanf("%d", &patient->age);

        printf("Enter new gender: ");
        getchar();
        fgets(patient->gender, sizeof(patient->gender), stdin);
        patient->gender[strcspn(patient->gender, "\n")] = '\0';

        printf("Enter new disease name: ");
        fgets(patient->history.disease, sizeof(patient->history.disease), stdin);
        patient->history.disease[strcspn(patient->history.disease, "\n")] = '\0';
```

```
        printf("Enter new year diagnosed: ");
        scanf("%d", &patient->history.year_diagnosed);

        printf("Does the patient have allergies? (1 for yes, 0 for no): ");
        scanf("%d", &patient->has_allergies);

        if(patient->has_allergies){
            printf("Enter new allergy details: ");
            getchar();
            fgets(patient->extra_info.allergies, sizeof(patient->extra_info.allergies), stdin);
            patient->extra_info.allergies[strcspn(patient->extra_info.allergies, "\n")] = '\0';
        }

        printf("Patient updated successfully!\n");
    }else{
        printf("Invalid patient ID.\n");
    }
}

void delete_patient_record(){
    int patient_id;
    printf("Enter patient ID to delete (0 to %d): ", patient_count - 1);
    scanf("%d", &patient_id);

    if(patient_id >= 0 && patient_id < patient_count){
        for(int i = patient_id; i < patient_count - 1; i++){
            patients[i] = patients[i + 1];
        }
```

```
        patient_count--;
        printf("Patient record deleted successfully.\n");
    }else{
        printf("Invalid patient ID.\n");
    }
}

void list_all_patients(){
    printf("List of all patients:\n");
    for(int i = 0; i < patient_count; i++){
        printf("Patient ID: %d, Name: %s, Age: %d, Gender: %s\n", i, patients[i].name, patients[i].age,
patients[i].gender);
    }
}

int main(){
    bool is_on = true;

    while(is_on){
        int user_option;
        printf("\nMenu Options:\n");
        printf("1. Add Patient\n");
        printf("2. View Patient Details\n");
        printf("3. Update Patient Information\n");
        printf("4. Delete Patient Record\n");
        printf("5. List All Patients\n");
        printf("6. Exit\n");
        printf("Enter your option: ");
        scanf("%d", &user_option);
```

```
        switch(user_option){
            case 1:
                add_patient();
                break;
            case 2:
                view_patient_details();
                break;
            case 3:
                update_patient_info();
                break;
            case 4:
                delete_patient_record();
                break;
            case 5:
                list_all_patients();
                break;
            case 6:
                printf("Exiting program...\n");
                is_on = false;
                break;
            default:
                printf("Invalid option, please try again.\n");
                break;
        }
    }

    return 0;
}
```

2.

```
#include<stdio.h>
#include<string.h>
#include<stdbool.h>

struct item_specifications {
    char manufacturer[100];
    char expiry_date[20];
};

union item_details {
    float weight;
    int quantity;
};

struct inventory_item {
    char item_name[100];
    int item_id;
    char category[50];
    struct item_specifications specs;
    union item_details details;
    bool is_weight;
};

struct inventory_item inventory[50];
int inventory_count = 0;
```

```
void add_inventory_item() {
    if (inventory_count < 50) {
        struct inventory_item *new_item = &inventory[inventory_count];
        printf("Enter the name: ");
        getchar();
        fgets(new_item->item_name, 100, stdin);
        new_item->item_name[strcspn(new_item->item_name, "\n")] = '\0';
        new_item->item_id = inventory_count;
        printf("Enter item category: ");
        fgets(new_item->category, sizeof(new_item->category), stdin);
        new_item->category[strcspn(new_item->category, "\n")] = '\0';
        printf("Enter manufacturer name: ");
        fgets(new_item->specs.manufacturer, sizeof(new_item->specs.manufacturer), stdin);
        new_item->specs.manufacturer[strcspn(new_item->specs.manufacturer, "\n")] = '\0';
        printf("Enter expiry date (DD/MM/YYYY): ");
        fgets(new_item->specs.expiry_date, sizeof(new_item->specs.expiry_date), stdin);
        new_item->specs.expiry_date[strcspn(new_item->specs.expiry_date, "\n")] = '\0';
        printf("Is the item measured by weight or quantity? (1 for weight, 0 for quantity): ");
        scanf("%d", &new_item->is_weight);

        if (new_item->is_weight) {
            printf("Enter weight of the item (in grams): ");
            scanf("%f", &new_item->details.weight);
        } else {
            printf("Enter quantity of the item: ");
            scanf("%d", &new_item->details.quantity);
        }
        inventory_count += 1;
        printf("Item added successfully!\n");
```

```
    } else {
        printf("Inventory full!\n");
    }
}

void view_inventory_item() {
    int item_id;
    printf("Enter item ID to view details (0 to %d): ", inventory_count - 1);
    scanf("%d", &item_id);
    if (item_id >= 0 && item_id < 50) {
        struct inventory_item *item = &inventory[item_id];
        printf("\nItem Name: %s\n", item->item_name);
        printf("Item ID: %d\n", item->item_id);
        printf("Category: %s\n", item->category);
        printf("Manufacturer: %s\n", item->specs.manufacturer);
        printf("Expiry Date: %s\n", item->specs.expiry_date);
        if (item->is_weight) {
            printf("Weight: %.2f grams\n", item->details.weight);
        } else {
            printf("Quantity: %d\n", item->details.quantity);
        }
    } else {
        printf("Invalid item ID!\n");
    }
}

void update_inventory_item() {
    int item_id;
```

```
printf("Enter the ID to update: ");
scanf("%d", &item_id);
if (item_id >= 0 && item_id < 50) {
    struct inventory_item *item = &inventory[item_id];
    printf("Updating details for %s:\n", item->item_name);

    printf("Enter new item name: ");
    getchar();
    fgets(item->item_name, sizeof(item->item_name), stdin);
    item->item_name[strcspn(item->item_name, "\n")] = '\0';

    printf("Enter new category: ");
    fgets(item->category, sizeof(item->category), stdin);
    item->category[strcspn(item->category, "\n")] = '\0';

    printf("Enter new manufacturer name: ");
    fgets(item->specs.manufacturer, sizeof(item->specs.manufacturer), stdin);
    item->specs.manufacturer[strcspn(item->specs.manufacturer, "\n")] = '\0';

    printf("Enter new expiry date (DD/MM/YYYY): ");
    fgets(item->specs.expiry_date, sizeof(item->specs.expiry_date), stdin);
    item->specs.expiry_date[strcspn(item->specs.expiry_date, "\n")] = '\0';

    printf("Is the item measured by weight or quantity? (1 for weight, 0 for quantity): ");
    scanf("%d", &item->is_weight);

    if (item->is_weight) {
        printf("Enter new weight of the item (in grams): ");
        scanf("%f", &item->details.weight);
```

```
        } else {
            printf("Enter new quantity of the item: ");
            scanf("%d", &item->details.quantity);
        }

        printf("Item updated successfully!\n");
    } else {
        printf("Invalid item ID.\n");
    }
}

void delete_inventory_item() {
    int item_id;
    printf("Enter item ID to delete (0 to %d): ", inventory_count - 1);
    scanf("%d", &item_id);
    if (item_id >= 0 && item_id < 50) {
        for (int i = item_id; i < inventory_count - 1; i++) {
            inventory[i] = inventory[i + 1];
        }
        inventory_count -= 1;
        printf("Item deleted successfully.\n");
    } else {
        printf("Invalid item ID.\n");
    }
}

void list_all_inventory_items() {
    printf("List of all inventory items:\n");
    for (int i = 0; i < inventory_count; i++) {
```

```
        printf("Item ID: %d, Item Name: %s, Category: %s, Manufacturer: %s\n",
               inventory[i].item_id, inventory[i].item_name, inventory[i].category,
inventory[i].specs.manufacturer);
    }
}

int main() {
    bool is_on = true;

    while (is_on) {
        int user_option;
        printf("\nMenu Options:\n");
        printf("1. Add Inventory Item\n");
        printf("2. View Inventory Item\n");
        printf("3. Update Inventory Item\n");
        printf("4. Delete Inventory Item\n");
        printf("5. List All Inventory Items\n");
        printf("6. Exit\n");
        printf("Enter your option: ");
        scanf("%d", &user_option);

        switch (user_option) {
            case 1:
                add_inventory_item();
                break;
            case 2:
                view_inventory_item();
                break;
            case 3:
```

```
            update_inventory_item();
            break;
        case 4:
            delete_inventory_item();
            break;
        case 5:
            list_all_inventory_items();
            break;
        case 6:
            printf("Exiting program...\n");
            is_on = false;
            break;
        default:
            printf("Invalid option, please try again.\n");
            break;
      }
   }

   return 0;
}
```

3.

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX_APPOINTMENTS 50
#define CLINIC_OPEN_HOURS "9:00 AM - 6:00 PM"
```

```
struct patient_details {
    char name[100];
    int age;
    char gender;
};

struct doctor_details {
    char name[100];
    char specialization[50];
};

union optional_appointment_data {
    char reason_for_visit[200];
    char additional_notes[200];
};

struct appointment {
    int appointment_id;
    struct patient_details patient;
    struct doctor_details doctor;
    char appointment_date[20];
    char appointment_time[10];
    union optional_appointment_data extra_data;
    bool is_reason_for_visit;
};

struct appointment appointments[MAX_APPOINTMENTS];
int appointment_count = 0;
```

```
void schedule_appointment() {
    if (appointment_count < MAX_APPOINTMENTS) {
        struct appointment *new_appointment = &appointments[appointment_count];
        new_appointment->appointment_id = appointment_count;

        printf("Enter patient name: ");
        getchar();
        fgets(new_appointment->patient.name, sizeof(new_appointment->patient.name), stdin);
        new_appointment->patient.name[strcspn(new_appointment->patient.name, "\n")] = '\0';

        printf("Enter patient age: ");
        scanf("%d", &new_appointment->patient.age);

        printf("Enter patient gender (M/F): ");
        getchar();
        scanf("%c", &new_appointment->patient.gender);

        printf("Enter doctor name: ");
        getchar();
        fgets(new_appointment->doctor.name, sizeof(new_appointment->doctor.name), stdin);
        new_appointment->doctor.name[strcspn(new_appointment->doctor.name, "\n")] = '\0';

        printf("Enter doctor specialization: ");
        fgets(new_appointment->doctor.specialization, sizeof(new_appointment->doctor.specialization),
stdin);
        new_appointment->doctor.specialization[strcspn(new_appointment->doctor.specialization, "\n")] =
'\0';
```

```
    printf("Enter appointment date (DD/MM/YYYY): ");
    fgets(new_appointment->appointment_date, sizeof(new_appointment->appointment_date), stdin);
    new_appointment->appointment_date[strcspn(new_appointment->appointment_date, "\n")] = '\0';

    printf("Enter appointment time (HH:MM): ");
    fgets(new_appointment->appointment_time, sizeof(new_appointment->appointment_time), stdin);
    new_appointment->appointment_time[strcspn(new_appointment->appointment_time, "\n")] = '\0';

    int has_reason;
    printf("Is there a reason for visit? (1 for yes, 0 for no): ");
    scanf("%d", &has_reason);
    new_appointment->is_reason_for_visit = has_reason;

    if (new_appointment->is_reason_for_visit) {
        printf("Enter reason for visit: ");
        getchar();
        fgets(new_appointment->extra_data.reason_for_visit, sizeof(new_appointment->extra_data.reason_for_visit), stdin);
        new_appointment->extra_data.reason_for_visit[strcspn(new_appointment->extra_data.reason_for_visit, "\n")] = '\0';
    } else {
        printf("Enter additional notes: ");
        getchar();
        fgets(new_appointment->extra_data.additional_notes, sizeof(new_appointment->extra_data.additional_notes), stdin);
        new_appointment->extra_data.additional_notes[strcspn(new_appointment->extra_data.additional_notes, "\n")] = '\0';
    }

    appointment_count++;
```

```
        printf("Appointment scheduled successfully!\n");
    } else {
        printf("Maximum number of appointments reached!\n");
    }
}

void view_appointment() {
    int appointment_id;
    printf("Enter appointment ID to view (0 to %d): ", appointment_count - 1);
    scanf("%d", &appointment_id);

    if (appointment_id >= 0 && appointment_id < appointment_count) {
        struct appointment *appt = &appointments[appointment_id];
        printf("\nAppointment ID: %d\n", appt->appointment_id);
        printf("Patient Name: %s\n", appt->patient.name);
        printf("Age: %d\n", appt->patient.age);
        printf("Gender: %c\n", appt->patient.gender);
        printf("Doctor: Dr. %s\n", appt->doctor.name);
        printf("Specialization: %s\n", appt->doctor.specialization);
        printf("Appointment Date: %s\n", appt->appointment_date);
        printf("Appointment Time: %s\n", appt->appointment_time);

        if (appt->is_reason_for_visit) {
            printf("Reason for Visit: %s\n", appt->extra_data.reason_for_visit);
        } else {
            printf("Additional Notes: %s\n", appt->extra_data.additional_notes);
        }
    } else {
        printf("Invalid appointment ID!\n");
```

```
    }
}

void update_appointment() {
    int appointment_id;
    printf("Enter appointment ID to update (0 to %d): ", appointment_count - 1);
    scanf("%d", &appointment_id);

    if (appointment_id >= 0 && appointment_id < appointment_count) {
        struct appointment *appt = &appointments[appointment_id];
        printf("Updating appointment details for %s:\n", appt->patient.name);

        printf("Enter new appointment date (DD/MM/YYYY): ");
        getchar();
        fgets(appt->appointment_date, sizeof(appt->appointment_date), stdin);
        appt->appointment_date[strcspn(appt->appointment_date, "\n")] = '\0';
        printf("Enter new appointment time (HH:MM): ");
        fgets(appt->appointment_time, sizeof(appt->appointment_time), stdin);
        appt->appointment_time[strcspn(appt->appointment_time, "\n")] = '\0';

        int has_reason;
        printf("Is there a reason for visit? (1 for yes, 0 for no): ");
        scanf("%d", &has_reason);
        appt->is_reason_for_visit = has_reason;

        if (appt->is_reason_for_visit) {
            printf("Enter new reason for visit: ");
            getchar();
            fgets(appt->extra_data.reason_for_visit, sizeof(appt->extra_data.reason_for_visit), stdin);
```

```
            appt->extra_data.reason_for_visit[strcspn(appt->extra_data.reason_for_visit, "\n")] = '\0';
        } else {
            printf("Enter new additional notes: ");
            getchar();
            fgets(appt->extra_data.additional_notes, sizeof(appt->extra_data.additional_notes), stdin);
            appt->extra_data.additional_notes[strcspn(appt->extra_data.additional_notes, "\n")] = '\0';
        }

        printf("Appointment updated successfully!\n");
    } else {
        printf("Invalid appointment ID!\n");
    }
}

void cancel_appointment() {
    int appointment_id;
    printf("Enter appointment ID to cancel (0 to %d): ", appointment_count - 1);
    scanf("%d", &appointment_id);

    if (appointment_id >= 0 && appointment_id < appointment_count) {
        for (int i = appointment_id; i < appointment_count - 1; i++) {
            appointments[i] = appointments[i + 1];
        }
        appointment_count--;
        printf("Appointment canceled successfully!\n");
    } else {
        printf("Invalid appointment ID!\n");
    }
}
```

```
void list_all_appointments() {
    printf("List of all appointments:\n");
    for (int i = 0; i < appointment_count; i++) {
        printf("\nAppointment ID: %d\n", appointments[i].appointment_id);
        printf("Patient Name: %s, Doctor: Dr. %s, Date: %s, Time: %s\n",
               appointments[i].patient.name, appointments[i].doctor.name,
               appointments[i].appointment_date, appointments[i].appointment_time);
    }
}

int main() {
    bool is_on = true;

    while (is_on) {
        int user_option;
        printf("\nMenu Options:\n");
        printf("1. Schedule Appointment\n");
        printf("2. View Appointment\n");
        printf("3. Update Appointment\n");
        printf("4. Cancel Appointment\n");
        printf("5. List All Appointments\n");
        printf("6. Exit\n");
        printf("Enter your option: ");
        scanf("%d", &user_option);

        switch (user_option) {
            case 1:
                schedule_appointment();
```

```
            break;
        case 2:
            view_appointment();
            break;
        case 3:
            update_appointment();
            break;
        case 4:
            cancel_appointment();
            break;
        case 5:
            list_all_appointments();
            break;
        case 6:
            printf("Exiting program...\n");
            is_on = false;
            break;
        default:
            printf("Invalid option, please try again.\n");
            break;
      }
   }

   return 0;
}
```

4.

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

```
#include <string.h>
#include <stdbool.h>

#define MAX_BILLS 50
#define CONSULTATION_FEE 100.0
#define ROOM_RENT_PER_DAY 150.0
#define MEDICINE_COST_PER_DAY 50.0

struct patient_details {
    char name[100];
    int age;
    char gender;
};

struct billing_breakdown {
    float consultation_fee;
    float room_rent;
    float medicine_cost;
};

union optional_discount {
    float discount_percentage;
    float fixed_discount;
};

struct bill {
    int bill_id;
    struct patient_details patient;
    struct billing_breakdown breakdown;
```

```
    union optional_discount discount;
    bool is_percentage_discount;
    float total_amount;
};

struct bill bills[MAX_BILLS];
int bill_count = 0;

void generate_bill() {
    if (bill_count < MAX_BILLS) {
        struct bill *new_bill = &bills[bill_count];
        new_bill->bill_id = bill_count;

        printf("Enter patient name: ");
        getchar();
        fgets(new_bill->patient.name, sizeof(new_bill->patient.name), stdin);
        new_bill->patient.name[strcspn(new_bill->patient.name, "\n")] = '\0';

        printf("Enter patient age: ");
        scanf("%d", &new_bill->patient.age);

        printf("Enter patient gender (M/F): ");
        getchar();
        scanf("%c", &new_bill->patient.gender);

        int num_of_days;
        printf("Enter number of days in the hospital: ");
        scanf("%d", &num_of_days);
```

```
        new_bill->breakdown.consultation_fee = CONSULTATION_FEE;
        new_bill->breakdown.room_rent = ROOM_RENT_PER_DAY * num_of_days;
        new_bill->breakdown.medicine_cost = MEDICINE_COST_PER_DAY * num_of_days;

        printf("Is there a discount? (1 for percentage, 0 for fixed discount): ");
        scanf("%d", &new_bill->is_percentage_discount);

        if (new_bill->is_percentage_discount) {
            printf("Enter discount percentage: ");
            scanf("%f", &new_bill->discount.discount_percentage);
            new_bill->total_amount = (new_bill->breakdown.consultation_fee + new_bill->breakdown.room_rent + new_bill->breakdown.medicine_cost) * (1 - new_bill->discount.discount_percentage / 100);
        } else {
            printf("Enter fixed discount amount: ");
            scanf("%f", &new_bill->discount.fixed_discount);
            new_bill->total_amount = (new_bill->breakdown.consultation_fee + new_bill->breakdown.room_rent + new_bill->breakdown.medicine_cost) - new_bill->discount.fixed_discount;
        }

        bill_count++;
        printf("Bill generated successfully!\n");
    } else {
        printf("Maximum number of bills reached!\n");
    }
}

void view_bill() {
    int bill_id;
    printf("Enter bill ID to view (0 to %d): ", bill_count - 1);
```

```
    scanf("%d", &bill_id);

    if (bill_id >= 0 && bill_id < bill_count) {
        struct bill *b = &bills[bill_id];
        printf("\nBill ID: %d\n", b->bill_id);
        printf("Patient Name: %s\n", b->patient.name);
        printf("Age: %d\n", b->patient.age);
        printf("Gender: %c\n", b->patient.gender);
        printf("Consultation Fee: %.2f\n", b->breakdown.consultation_fee);
        printf("Room Rent: %.2f\n", b->breakdown.room_rent);
        printf("Medicine Cost: %.2f\n", b->breakdown.medicine_cost);

        if (b->is_percentage_discount) {
            printf("Discount: %.2f%%\n", b->discount.discount_percentage);
        } else {
            printf("Discount: %.2f\n", b->discount.fixed_discount);
        }

        printf("Total Amount: %.2f\n", b->total_amount);
    } else {
        printf("Invalid bill ID!\n");
    }
}

void update_bill() {
    int bill_id;
    printf("Enter bill ID to update (0 to %d): ", bill_count - 1);
    scanf("%d", &bill_id);
```

```
    if (bill_id >= 0 && bill_id < bill_count) {
        struct bill *b = &bills[bill_id];
        printf("Updating bill for %s:\n", b->patient.name);

        int num_of_days;
        printf("Enter new number of days in the hospital: ");
        scanf("%d", &num_of_days);

        b->breakdown.room_rent = ROOM_RENT_PER_DAY * num_of_days;
        b->breakdown.medicine_cost = MEDICINE_COST_PER_DAY * num_of_days;

        printf("Is there a discount? (1 for percentage, 0 for fixed discount): ");
        scanf("%d", &b->is_percentage_discount);

        if (b->is_percentage_discount) {
            printf("Enter new discount percentage: ");
            scanf("%f", &b->discount.discount_percentage);
            b->total_amount = (b->breakdown.consultation_fee + b->breakdown.room_rent + b->breakdown.medicine_cost) * (1 - b->discount.discount_percentage / 100);
        } else {
            printf("Enter new fixed discount amount: ");
            scanf("%f", &b->discount.fixed_discount);
            b->total_amount = (b->breakdown.consultation_fee + b->breakdown.room_rent + b->breakdown.medicine_cost) - b->discount.fixed_discount;
        }

        printf("Bill updated successfully!\n");
    } else {
        printf("Invalid bill ID!\n");
```

```
    }
}

void delete_bill() {
    int bill_id;
    printf("Enter bill ID to delete (0 to %d): ", bill_count - 1);
    scanf("%d", &bill_id);

    if (bill_id >= 0 && bill_id < bill_count) {
        for (int i = bill_id; i < bill_count - 1; i++) {
            bills[i] = bills[i + 1];
        }
        bill_count--;
        printf("Bill deleted successfully!\n");
    } else {
        printf("Invalid bill ID!\n");
    }
}

void list_all_bills() {
    printf("List of all bills:\n");
    for (int i = 0; i < bill_count; i++) {
        printf("\nBill ID: %d, Patient Name: %s, Total Amount: %.2f\n",
               bills[i].bill_id, bills[i].patient.name, bills[i].total_amount);
    }
}

int main() {
    bool is_on = true;
```

```
while (is_on) {
    int user_option;
    printf("\nMenu Options:\n");
    printf("1. Generate Bill\n");
    printf("2. View Bill\n");
    printf("3. Update Bill\n");
    printf("4. Delete Bill\n");
    printf("5. List All Bills\n");
    printf("6. Exit\n");
    printf("Enter your option: ");
    scanf("%d", &user_option);

    switch (user_option) {
        case 1:
            generate_bill();
            break;
        case 2:
            view_bill();
            break;
        case 3:
            update_bill();
            break;
        case 4:
            delete_bill();
            break;
        case 5:
            list_all_bills();
            break;
```

```
            case 6:
                printf("Exiting program...\n");
                is_on = false;
                break;
            default:
                printf("Invalid option, please try again.\n");
                break;
        }
    }

    return 0;
}
```

5.

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX_TEST_RESULTS 50
#define STANDARD_HEART_RATE_MIN 60
#define STANDARD_HEART_RATE_MAX 100
#define STANDARD_BLOOD_PRESSURE_MIN 90
#define STANDARD_BLOOD_PRESSURE_MAX 120

struct test_parameters {
    float heart_rate;
    float blood_pressure;
```

```
    float temperature;
};

union optional_test_data {
    float cholesterol_level;
    float blood_sugar;
};

struct test_result {
    int test_id;
    char patient_name[100];
    struct test_parameters parameters;
    union optional_test_data optional_data;
    bool has_optional_data;
};

struct test_result test_results[MAX_TEST_RESULTS];
int test_result_count = 0;

void add_test_result() {
    if (test_result_count < MAX_TEST_RESULTS) {
        struct test_result *new_result = &test_results[test_result_count];

        new_result->test_id = test_result_count;
        printf("Enter patient name: ");
        getchar();
        fgets(new_result->patient_name, sizeof(new_result->patient_name), stdin);
        new_result->patient_name[strcspn(new_result->patient_name, "\n")] = '\0';
        printf("Enter heart rate (bpm): ");
```

```
scanf("%f", &new_result->parameters.heart_rate);
printf("Enter blood pressure (mmHg): ");
scanf("%f", &new_result->parameters.blood_pressure);
printf("Enter body temperature (°C): ");
scanf("%f", &new_result->parameters.temperature);
printf("Does the test include optional data? (1 for yes, 0 for no): ");
scanf("%d", &new_result->has_optional_data);

if (new_result->has_optional_data) {
    int optional_choice;
    printf("Enter optional test data:\n");
    printf("1. Cholesterol level\n");
    printf("2. Blood sugar level\n");
    printf("Enter your choice: ");
    scanf("%d", &optional_choice);

    if (optional_choice == 1) {
        printf("Enter cholesterol level (mg/dL): ");
        scanf("%f", &new_result->optional_data.cholesterol_level);
    } else if (optional_choice == 2) {
        printf("Enter blood sugar level (mg/dL): ");
        scanf("%f", &new_result->optional_data.blood_sugar);
    } else {
        printf("Invalid choice for optional test data!\n");
    }
}

test_result_count++;
printf("Test result added successfully!\n");
```

```
    } else {
        printf("Maximum test results reached!\n");
    }
}

void view_test_result() {
    int test_id;
    printf("Enter test ID to view (0 to %d): ", test_result_count - 1);
    scanf("%d", &test_id);

    if (test_id >= 0 && test_id < test_result_count) {
        struct test_result *result = &test_results[test_id];
        printf("\nTest ID: %d\n", result->test_id);
        printf("Patient Name: %s\n", result->patient_name);
        printf("Heart Rate: %.2f bpm\n", result->parameters.heart_rate);
        printf("Blood Pressure: %.2f mmHg\n", result->parameters.blood_pressure);
        printf("Body Temperature: %.2f °C\n", result->parameters.temperature);

        if (result->has_optional_data) {
            if (result->optional_data.cholesterol_level > 0) {
                printf("Cholesterol Level: %.2f mg/dL\n", result->optional_data.cholesterol_level);
            } else {
                printf("Blood Sugar Level: %.2f mg/dL\n", result->optional_data.blood_sugar);
            }
        }
        if (result->parameters.heart_rate < STANDARD_HEART_RATE_MIN || result->parameters.heart_rate
> STANDARD_HEART_RATE_MAX) {
            printf("Warning: Heart rate is out of normal range!\n");
        }
```

```
        if (result->parameters.blood_pressure < STANDARD_BLOOD_PRESSURE_MIN || result-
>parameters.blood_pressure > STANDARD_BLOOD_PRESSURE_MAX) {
            printf("Warning: Blood pressure is out of normal range!\n");
        }
    } else {
        printf("Invalid test ID!\n");
    }
}

void update_test_result() {
    int test_id;
    printf("Enter test ID to update (0 to %d): ", test_result_count - 1);
    scanf("%d", &test_id);

    if (test_id >= 0 && test_id < test_result_count) {
        struct test_result *result = &test_results[test_id];

        printf("Updating test result for patient %s:\n", result->patient_name);

        printf("Enter new heart rate (bpm): ");
        scanf("%f", &result->parameters.heart_rate);

        printf("Enter new blood pressure (mmHg): ");
        scanf("%f", &result->parameters.blood_pressure);

        printf("Enter new body temperature (°C): ");
        scanf("%f", &result->parameters.temperature);
```

```
        printf("Does the test include optional data? (1 for yes, 0 for no): ");
        scanf("%d", &result->has_optional_data);

        if (result->has_optional_data) {
            int optional_choice;
            printf("Enter optional test data:\n");
            printf("1. Cholesterol level\n");
            printf("2. Blood sugar level\n");
            printf("Enter your choice: ");
            scanf("%d", &optional_choice);

            if (optional_choice == 1) {
                printf("Enter cholesterol level (mg/dL): ");
                scanf("%f", &result->optional_data.cholesterol_level);
            } else if (optional_choice == 2) {
                printf("Enter blood sugar level (mg/dL): ");
                scanf("%f", &result->optional_data.blood_sugar);
            } else {
                printf("Invalid choice for optional test data!\n");
            }
        }

        printf("Test result updated successfully!\n");
    } else {
        printf("Invalid test ID!\n");
    }
}

void delete_test_result() {
```

```
    int test_id;
    printf("Enter test ID to delete (0 to %d): ", test_result_count - 1);
    scanf("%d", &test_id);

    if (test_id >= 0 && test_id < test_result_count) {
        for (int i = test_id; i < test_result_count - 1; i++) {
            test_results[i] = test_results[i + 1];
        }
        test_result_count--;
        printf("Test result deleted successfully!\n");
    } else {
        printf("Invalid test ID!\n");
    }
}

void list_all_test_results() {
    printf("List of all test results:\n");
    for (int i = 0; i < test_result_count; i++) {
        printf("Test ID: %d, Patient Name: %s\n", test_results[i].test_id, test_results[i].patient_name);
    }
}

int main() {
    bool is_on = true;

    while (is_on) {
        int user_option;
        printf("\nMenu Options:\n");
        printf("1. Add Test Result\n");
```

```
printf("2. View Test Result\n");
printf("3. Update Test Result\n");
printf("4. Delete Test Result\n");
printf("5. List All Test Results\n");
printf("6. Exit\n");
printf("Enter your option: ");
scanf("%d", &user_option);

switch (user_option) {
   case 1:
      add_test_result();
      break;
   case 2:
      view_test_result();
      break;
   case 3:
      update_test_result();
      break;
   case 4:
      delete_test_result();
      break;
   case 5:
      list_all_test_results();
      break;
   case 6:
      printf("Exiting program...\n");
      is_on = false;
      break;
   default:
```

```
            printf("Invalid option, please try again.\n");
            break;
        }
    }

    return 0;
}
```

6.

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX_ROSTERS 50
#define SHIFT_START "08:00 AM"
#define SHIFT_END "04:00 PM"
#define NUM_SHIFT_TYPES 3

struct shift_details {
    char shift_type[20];
    char start_time[10];
    char end_time[10];
};

union optional_duty_info {
    char special_tasks[100];
    char notes[100];
};
```

```
struct duty_roster {
    int staff_id;
    char staff_name[100];
    char role[50];
    struct shift_details shift;
    union optional_duty_info optional_info;
    bool has_special_tasks;
};

struct duty_roster rosters[MAX_ROSTERS];
int roster_count = 0;

void add_duty_roster() {
    if (roster_count < MAX_ROSTERS) {
        struct duty_roster *new_roster = &rosters[roster_count];

        new_roster->staff_id = roster_count;
        printf("Enter staff name: ");
        getchar();
        fgets(new_roster->staff_name, sizeof(new_roster->staff_name), stdin);
        new_roster->staff_name[strcspn(new_roster->staff_name, "\n")] = '\0';
        printf("Enter staff role: ");
        fgets(new_roster->role, sizeof(new_roster->role), stdin);
        new_roster->role[strcspn(new_roster->role, "\n")] = '\0';
        printf("Enter shift type (Morning/Evening/Night): ");
        fgets(new_roster->shift.shift_type, sizeof(new_roster->shift.shift_type), stdin);
        new_roster->shift.shift_type[strcspn(new_roster->shift.shift_type, "\n")] = '\0';
```

```
        printf("Enter shift start time (HH:MM AM/PM): ");
        fgets(new_roster->shift.start_time, sizeof(new_roster->shift.start_time), stdin);
        new_roster->shift.start_time[strcspn(new_roster->shift.start_time, "\n")] = '\0';

        printf("Enter shift end time (HH:MM AM/PM): ");
        fgets(new_roster->shift.end_time, sizeof(new_roster->shift.end_time), stdin);
        new_roster->shift.end_time[strcspn(new_roster->shift.end_time, "\n")] = '\0';

        printf("Does this shift include special tasks? (1 for yes, 0 for no): ");
        scanf("%d", &new_roster->has_special_tasks);

        if (new_roster->has_special_tasks) {
            printf("Enter special task or notes: ");
            getchar();
            fgets(new_roster->optional_info.special_tasks, sizeof(new_roster->optional_info.special_tasks),
stdin);
            new_roster->optional_info.special_tasks[strcspn(new_roster->optional_info.special_tasks, "\n")]
= '\0';
        }

        roster_count++;
        printf("Duty roster added successfully!\n");
    } else {
        printf("Roster is full!\n");
    }
}

void view_duty_roster() {
    int staff_id;
```

```
    printf("Enter staff ID to view duty roster (0 to %d): ", roster_count - 1);
    scanf("%d", &staff_id);

    if (staff_id >= 0 && staff_id < roster_count) {
        struct duty_roster *roster = &rosters[staff_id];

        printf("\nStaff ID: %d\n", roster->staff_id);
        printf("Staff Name: %s\n", roster->staff_name);
        printf("Role: %s\n", roster->role);
        printf("Shift Type: %s\n", roster->shift.shift_type);
        printf("Shift Start Time: %s\n", roster->shift.start_time);
        printf("Shift End Time: %s\n", roster->shift.end_time);

        if (roster->has_special_tasks) {
            printf("Special Tasks/Notes: %s\n", roster->optional_info.special_tasks);
        }

    } else {
        printf("Invalid staff ID!\n");
    }
}

void update_duty_roster() {
    int staff_id;
    printf("Enter staff ID to update duty roster (0 to %d): ", roster_count - 1);
    scanf("%d", &staff_id);

    if (staff_id >= 0 && staff_id < roster_count) {
        struct duty_roster *roster = &rosters[staff_id];
```

```
    printf("Updating duty roster for %s:\n", roster->staff_name);
    printf("Enter new shift type (Morning/Evening/Night): ");
    getchar();
    fgets(roster->shift.shift_type, sizeof(roster->shift.shift_type), stdin);
    roster->shift.shift_type[strcspn(roster->shift.shift_type, "\n")] = '\0';

    printf("Enter new shift start time (HH:MM AM/PM): ");
    fgets(roster->shift.start_time, sizeof(roster->shift.start_time), stdin);
    roster->shift.start_time[strcspn(roster->shift.start_time, "\n")] = '\0';

    printf("Enter new shift end time (HH:MM AM/PM): ");
    fgets(roster->shift.end_time, sizeof(roster->shift.end_time), stdin);
    roster->shift.end_time[strcspn(roster->shift.end_time, "\n")] = '\0';

    printf("Does this shift include special tasks? (1 for yes, 0 for no): ");
    scanf("%d", &roster->has_special_tasks);

    if (roster->has_special_tasks) {
        printf("Enter special task or notes: ");
        getchar();
        fgets(roster->optional_info.special_tasks, sizeof(roster->optional_info.special_tasks), stdin);
        roster->optional_info.special_tasks[strcspn(roster->optional_info.special_tasks, "\n")] = '\0';
    }

    printf("Duty roster updated successfully!\n");
} else {
    printf("Invalid staff ID!\n");
}
```

```
}

void delete_duty_roster() {
    int staff_id;
    printf("Enter staff ID to delete duty roster (0 to %d): ", roster_count - 1);
    scanf("%d", &staff_id);

    if (staff_id >= 0 && staff_id < roster_count) {
        for (int i = staff_id; i < roster_count - 1; i++) {
            rosters[i] = rosters[i + 1];
        }
        roster_count--;
        printf("Duty roster deleted successfully!\n");
    } else {
        printf("Invalid staff ID!\n");
    }
}

void list_all_duty_rosters() {
    printf("List of all duty rosters:\n");
    for (int i = 0; i < roster_count; i++) {
        printf("Staff ID: %d, Staff Name: %s, Role: %s, Shift Type: %s\n",
               rosters[i].staff_id, rosters[i].staff_name, rosters[i].role, rosters[i].shift.shift_type);
    }
}

int main() {
    bool is_on = true;
```

```
while (is_on) {
    int user_option;
    printf("\nMenu Options:\n");
    printf("1. Add Duty Roster\n");
    printf("2. View Duty Roster\n");
    printf("3. Update Duty Roster\n");
    printf("4. Delete Duty Roster\n");
    printf("5. List All Duty Rosters\n");
    printf("6. Exit\n");
    printf("Enter your option: ");
    scanf("%d", &user_option);

    switch (user_option) {
        case 1:
            add_duty_roster();
            break;
        case 2:
            view_duty_roster();
            break;
        case 3:
            update_duty_roster();
            break;
        case 4:
            delete_duty_roster();
            break;
        case 5:
            list_all_duty_rosters();
            break;
        case 6:
```

```
            printf("Exiting program...\n");
            is_on = false;
            break;
        default:
            printf("Invalid option, please try again.\n");
            break;
      }
   }

   return 0;
}
```

7.

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX_CONTACTS 50

struct contact_info {
   char name[50];
   char relationship[50];
   char phone_number[15];
};

union additional_contact_data {
   char email[50];
   char address[100];
};
```

```
struct emergency_contact {
    int contact_id;
    struct contact_info info;
    union additional_contact_data extra_data;
    bool has_email;
};

struct emergency_contact contacts[MAX_CONTACTS];
int contact_count = 0;

void add_emergency_contact() {
    if (contact_count < MAX_CONTACTS) {
        struct emergency_contact *new_contact = &contacts[contact_count];
        new_contact->contact_id = contact_count + 1;

        printf("Enter contact name: ");
        getchar();
        fgets(new_contact->info.name, 50, stdin);
        new_contact->info.name[strcspn(new_contact->info.name, "\n")] = '\0';

        printf("Enter relationship: ");
        fgets(new_contact->info.relationship, 50, stdin);
        new_contact->info.relationship[strcspn(new_contact->info.relationship, "\n")] = '\0';

        printf("Enter phone number: ");
        fgets(new_contact->info.phone_number, 15, stdin);
        new_contact->info.phone_number[strcspn(new_contact->info.phone_number, "\n")] = '\0';
```

```
        int choice;
        printf("Do you want to add an email (1) or an address (0)? ");
        scanf("%d", &choice);
        new_contact->has_email = (choice == 1);

        if (new_contact->has_email) {
            printf("Enter email: ");
            getchar();
            fgets(new_contact->extra_data.email, 50, stdin);
            new_contact->extra_data.email[strcspn(new_contact->extra_data.email, "\n")] = '\0';
        } else {
            printf("Enter address: ");
            getchar();
            fgets(new_contact->extra_data.address, 100, stdin);
            new_contact->extra_data.address[strcspn(new_contact->extra_data.address, "\n")] = '\0';
        }

        contact_count++;
        printf("Emergency contact added successfully!\n");
    } else {
        printf("Emergency contact list is full!\n");
    }
}

void view_emergency_contact() {
    int contact_id;
    printf("Enter contact ID to view details: ");
    scanf("%d", &contact_id);
```

```
    if (contact_id > 0 && contact_id <= contact_count) {
        struct emergency_contact *contact = &contacts[contact_id - 1];
        printf("\nContact ID: %d\n", contact->contact_id);
        printf("Name: %s\n", contact->info.name);
        printf("Relationship: %s\n", contact->info.relationship);
        printf("Phone Number: %s\n", contact->info.phone_number);

        if (contact->has_email) {
            printf("Email: %s\n", contact->extra_data.email);
        } else {
            printf("Address: %s\n", contact->extra_data.address);
        }
    } else {
        printf("Invalid contact ID!\n");
    }
}

void update_emergency_contact() {
    int contact_id;
    printf("Enter contact ID to update: ");
    scanf("%d", &contact_id);

    if (contact_id > 0 && contact_id <= contact_count) {
        struct emergency_contact *contact = &contacts[contact_id - 1];

        printf("Updating contact ID: %d\n", contact->contact_id);

        printf("Enter new name: ");
        getchar();
```

```
fgets(contact->info.name, 50, stdin);
contact->info.name[strcspn(contact->info.name, "\n")] = '\0';

printf("Enter new relationship: ");
fgets(contact->info.relationship, 50, stdin);
contact->info.relationship[strcspn(contact->info.relationship, "\n")] = '\0';

printf("Enter new phone number: ");
fgets(contact->info.phone_number, 15, stdin);
contact->info.phone_number[strcspn(contact->info.phone_number, "\n")] = '\0';

int choice;
printf("Do you want to update email (1) or address (0)? ");
scanf("%d", &choice);
contact->has_email = (choice == 1);

if (contact->has_email) {
    printf("Enter new email: ");
    getchar();
    fgets(contact->extra_data.email, 50, stdin);
    contact->extra_data.email[strcspn(contact->extra_data.email, "\n")] = '\0';
} else {
    printf("Enter new address: ");
    getchar();
    fgets(contact->extra_data.address, 100, stdin);
    contact->extra_data.address[strcspn(contact->extra_data.address, "\n")] = '\0';
}

printf("Emergency contact updated successfully!\n");
```

```
    } else {
        printf("Invalid contact ID!\n");
    }
}

void delete_emergency_contact() {
    int contact_id;
    printf("Enter contact ID to delete: ");
    scanf("%d", &contact_id);

    if (contact_id > 0 && contact_id <= contact_count) {
        for (int i = contact_id - 1; i < contact_count - 1; i++) {
            contacts[i] = contacts[i + 1];
        }
        contact_count--;
        printf("Emergency contact deleted successfully!\n");
    } else {
        printf("Invalid contact ID!\n");
    }
}

void list_all_emergency_contacts() {
    if (contact_count == 0) {
        printf("No emergency contacts available.\n");
        return;
    }
    for (int i = 0; i < contact_count; i++) {
        printf("\nContact ID: %d\n", contacts[i].contact_id);
        printf("Name: %s\n", contacts[i].info.name);
```

```
        printf("Relationship: %s\n", contacts[i].info.relationship);
        printf("Phone Number: %s\n", contacts[i].info.phone_number);

        if (contacts[i].has_email) {
            printf("Email: %s\n", contacts[i].extra_data.email);
        } else {
            printf("Address: %s\n", contacts[i].extra_data.address);
        }
    }
}

int main() {
    bool is_on = true;

    while (is_on) {
        int user_option;
        printf("\nMenu Options:\n");
        printf("1. Add Emergency Contact\n");
        printf("2. View Emergency Contact\n");
        printf("3. Update Emergency Contact\n");
        printf("4. Delete Emergency Contact\n");
        printf("5. List All Emergency Contacts\n");
        printf("6. Exit\n");
        printf("Enter your option: ");
        scanf("%d", &user_option);

        switch (user_option) {
            case 1:
                add_emergency_contact();
```

```
            break;
        case 2:
            view_emergency_contact();
            break;
        case 3:
            update_emergency_contact();
            break;
        case 4:
            delete_emergency_contact();
            break;
        case 5:
            list_all_emergency_contacts();
            break;
        case 6:
            printf("Exiting program...\n");
            is_on = false;
            break;
        default:
            printf("Invalid option, please try again.\n");
            break;
      }
   }

   return 0;
}
```

8.

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

```
#include<stdbool.h>
#define max_size 50

struct medical_history {
   char disease[30];
   char year_diagnosed[30];
};

union optional_data {
   char allergies[100];
   char optional_notes[100];
};

struct details {
   int record_id;
   char patient_name[30];
   int age;
   char gender;
   struct medical_history history;
   union optional_data extra_info;
   bool is_extra_info;
};

struct details arr[max_size];
int record_count = 0;

void add_record() {
   if (record_count < max_size) {
      struct details *new_detail = &arr[record_count];
```

```
new_detail->record_id = record_count + 1;

printf("Enter the patient name: ");
getchar();
fgets(new_detail->patient_name, 30, stdin);
new_detail->patient_name[strcspn(new_detail->patient_name, "\n")] = '\0';

printf("Enter the age: ");
scanf("%d", &new_detail->age);

printf("Enter the gender (M/F): ");
scanf(" %c", &new_detail->gender);

printf("Enter the disease name: ");
getchar();
fgets(new_detail->history.disease, 30, stdin);
new_detail->history.disease[strcspn(new_detail->history.disease, "\n")] = '\0';

printf("Enter the year diagnosed (dd/mm/yyyy): ");
fgets(new_detail->history.year_diagnosed, 30, stdin);
new_detail->history.year_diagnosed[strcspn(new_detail->history.year_diagnosed, "\n")] = '\0';

printf("Enter 1 to enter the optional data (allergies/notes), else 0: ");
scanf("%d", &new_detail->is_extra_info);

if (new_detail->is_extra_info) {
   printf("Enter the allergy details: ");
   getchar();
   fgets(new_detail->extra_info.allergies, 100, stdin);
```

```
            new_detail->extra_info.allergies[strcspn(new_detail->extra_info.allergies, "\n")] = '\0';

            printf("Enter the optional notes: ");
            fgets(new_detail->extra_info.optional_notes, 100, stdin);
            new_detail->extra_info.optional_notes[strcspn(new_detail->extra_info.optional_notes, "\n")] =
'\0';
        }

        record_count += 1;
        printf("Medical details added successfully!\n");
    } else {
        printf("Medical record storage full!\n");
    }
}

void view_record() {
    int rec_id;
    printf("Enter the record ID: ");
    scanf("%d", &rec_id);

    if (rec_id >= 0 && rec_id < record_count) {
        struct details *detail = &arr[rec_id];
        printf("Patient Name: %s\n", detail->patient_name);
        printf("Age: %d\n", detail->age);
        printf("Gender: %c\n", detail->gender);
        printf("Disease: %s\n", detail->history.disease);
        printf("Year Diagnosed: %s\n", detail->history.year_diagnosed);

        if (detail->is_extra_info) {
```

```
            printf("Allergies: %s\n", detail->extra_info.allergies);
            printf("Optional Notes: %s\n", detail->extra_info.optional_notes);
        } else {
            printf("No allergies recorded.\n");
        }
    } else {
        printf("Invalid record ID!\n");
    }
}

void update_record() {
    int rec_id;
    printf("Enter the record ID to update: ");
    scanf("%d", &rec_id);

    if (rec_id >= 0 && rec_id < record_count) {
        struct details *detail = &arr[rec_id];

        printf("Updating record for %s:\n", detail->patient_name);
        printf("Enter new age: ");
        scanf("%d", &detail->age);

        printf("Enter new gender (M/F): ");
        scanf(" %c", &detail->gender);

        printf("Enter new disease name: ");
        getchar();
        fgets(detail->history.disease, 30, stdin);
        detail->history.disease[strcspn(detail->history.disease, "\n")] = '\0';
```

```
        printf("Enter new year diagnosed (dd/mm/yyyy): ");
        fgets(detail->history.year_diagnosed, 30, stdin);
        detail->history.year_diagnosed[strcspn(detail->history.year_diagnosed, "\n")] = '\0';

        printf("Enter 1 to update optional data (allergies/notes), else 0: ");
        scanf("%d", &detail->is_extra_info);

        if (detail->is_extra_info) {
            printf("Enter new allergy details: ");
            getchar();
            fgets(detail->extra_info.allergies, 100, stdin);
            detail->extra_info.allergies[strcspn(detail->extra_info.allergies, "\n")] = '\0';

            printf("Enter new optional notes: ");
            fgets(detail->extra_info.optional_notes, 100, stdin);
            detail->extra_info.optional_notes[strcspn(detail->extra_info.optional_notes, "\n")] = '\0';

        printf("Record updated successfully!\n");
    } else {
        printf("Invalid record ID!\n");
    }
}

void delete_record() {
    int rec_id;
    printf("Enter the record ID to delete: ");
    scanf("%d", &rec_id);
```

```
    if (rec_id >= 0 && rec_id < record_count) {
        for (int i = rec_id; i < record_count - 1; i++) {
            arr[i] = arr[i + 1];
        }
        record_count--;
        printf("Medical record deleted successfully!\n");
    } else {
        printf("Invalid record ID!\n");
    }
}

void list_record() {
    printf("List of all medical records:\n");
    for (int i = 0; i < record_count; i++) {
        printf("Record ID: %d, Name: %s, Age: %d, Gender: %c\n", arr[i].record_id, arr[i].patient_name,
arr[i].age, arr[i].gender);
    }
}

int main() {
    bool is_on = true;

    while (is_on) {
        int user_option;
        printf("Menu Options:\n");
        printf("1. Add Medical Record\n");
        printf("2. View Medical Record\n");
        printf("3. Update Medical Record\n");
        printf("4. Delete Medical Record\n");
```

```
printf("5. List All Medical Records\n");
printf("6. Exit\n");
printf("Enter your option: ");
scanf("%d", &user_option);

switch (user_option) {
   case 1:
      add_record();
      break;
   case 2:
      view_record();
      break;
   case 3:
      update_record();
      break;
   case 4:
      delete_record();
      break;
   case 5:
      list_record();
      break;
   case 6:
      is_on = false;
      printf("Exiting program...\n");
      break;
   default:
      printf("Invalid option, please try again.\n");
      break;
}
```

```
    }

    return 0;
}
```

9.

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX_SURGERIES 50

struct patient_info {
    char patient_name[50];
    int patient_age;
    char patient_gender;
};

union surgery_optional_data {
    char surgery_notes[100];
    char pre_surgery_instructions[100];
};

struct surgery_schedule {
    int surgery_id;
    struct patient_info patient;
    char surgery_type[50];
    char surgery_date[20];
    union surgery_optional_data optional_data;
```

```
    bool has_optional_data;
};

struct surgery_schedule surgeries[MAX_SURGERIES];
int surgery_count = 0;

void schedule_surgery() {
    if (surgery_count < MAX_SURGERIES) {
        struct surgery_schedule *new_surgery = &surgeries[surgery_count];
        new_surgery->surgery_id = surgery_count + 1;

        printf("Enter patient's name: ");
        getchar();
        fgets(new_surgery->patient.patient_name, 50, stdin);
        new_surgery->patient.patient_name[strcspn(new_surgery->patient.patient_name, "\n")] = '\0';

        printf("Enter patient's age: ");
        scanf("%d", &new_surgery->patient.patient_age);

        printf("Enter patient's gender (M/F): ");
        scanf(" %c", &new_surgery->patient.patient_gender);

        printf("Enter surgery type: ");
        getchar();
        fgets(new_surgery->surgery_type, 50, stdin);
        new_surgery->surgery_type[strcspn(new_surgery->surgery_type, "\n")] = '\0';

        printf("Enter surgery date (DD/MM/YYYY): ");
        fgets(new_surgery->surgery_date, 20, stdin);
```

```
        new_surgery->surgery_date[strcspn(new_surgery->surgery_date, "\n")] = '\0';

        int choice;
        printf("Do you want to add optional notes or pre-surgery instructions? (1 for Notes, 0 for None): ");
        scanf("%d", &choice);
        new_surgery->has_optional_data = (choice == 1);

        if (new_surgery->has_optional_data) {
            printf("Enter optional surgery notes: ");
            getchar();
            fgets(new_surgery->optional_data.surgery_notes, 100, stdin);
            new_surgery->optional_data.surgery_notes[strcspn(new_surgery->optional_data.surgery_notes,
"\n")] = '\0';
        } else {
            printf("Enter pre-surgery instructions: ");
            getchar();
            fgets(new_surgery->optional_data.pre_surgery_instructions, 100, stdin);
            new_surgery->optional_data.pre_surgery_instructions[strcspn(new_surgery-
>optional_data.pre_surgery_instructions, "\n")] = '\0';
        }

        surgery_count++;
        printf("Surgery scheduled successfully!\n");
    } else {
        printf("Surgery schedule is full!\n");
    }
}

void view_surgery_schedule() {
```

```
    int surgery_id;
    printf("Enter surgery ID to view details: ");
    scanf("%d", &surgery_id);

    if (surgery_id > 0 && surgery_id <= surgery_count) {
        struct surgery_schedule *surgery = &surgeries[surgery_id - 1];
        printf("\nSurgery ID: %d\n", surgery->surgery_id);
        printf("Patient Name: %s\n", surgery->patient.patient_name);
        printf("Patient Age: %d\n", surgery->patient.patient_age);
        printf("Patient Gender: %c\n", surgery->patient.patient_gender);
        printf("Surgery Type: %s\n", surgery->surgery_type);
        printf("Surgery Date: %s\n", surgery->surgery_date);

        if (surgery->has_optional_data) {
            printf("Surgery Notes: %s\n", surgery->optional_data.surgery_notes);
        } else {
            printf("Pre-Surgery Instructions: %s\n", surgery->optional_data.pre_surgery_instructions);
        }
    } else {
        printf("Invalid surgery ID!\n");
    }
}

void update_surgery_schedule() {
    int surgery_id;
    printf("Enter surgery ID to update: ");
    scanf("%d", &surgery_id);

    if (surgery_id > 0 && surgery_id <= surgery_count) {
```

```
    struct surgery_schedule *surgery = &surgeries[surgery_id - 1];

    printf("Updating surgery ID: %d\n", surgery->surgery_id);

    printf("Enter new patient's name: ");
    getchar();
    fgets(surgery->patient.patient_name, 50, stdin);
    surgery->patient.patient_name[strcspn(surgery->patient.patient_name, "\n")] = '\0';

    printf("Enter new patient's age: ");
    scanf("%d", &surgery->patient.patient_age);

    printf("Enter new patient's gender (M/F): ");
    scanf(" %c", &surgery->patient.patient_gender);

    printf("Enter new surgery type: ");
    getchar();
    fgets(surgery->surgery_type, 50, stdin);
    surgery->surgery_type[strcspn(surgery->surgery_type, "\n")] = '\0';

    printf("Enter new surgery date (DD/MM/YYYY): ");
    fgets(surgery->surgery_date, 20, stdin);
    surgery->surgery_date[strcspn(surgery->surgery_date, "\n")] = '\0';

    int choice;
    printf("Do you want to update optional notes or pre-surgery instructions? (1 for Notes, 0 for None):
");
    scanf("%d", &choice);
    surgery->has_optional_data = (choice == 1);
```

```
        if (surgery->has_optional_data) {
            printf("Enter new surgery notes: ");
            getchar();
            fgets(surgery->optional_data.surgery_notes, 100, stdin);
            surgery->optional_data.surgery_notes[strcspn(surgery->optional_data.surgery_notes, "\n")] =
'\0';
        } else {
            printf("Enter new pre-surgery instructions: ");
            getchar();
            fgets(surgery->optional_data.pre_surgery_instructions, 100, stdin);
            surgery->optional_data.pre_surgery_instructions[strcspn(surgery-
>optional_data.pre_surgery_instructions, "\n")] = '\0';
        }

        printf("Surgery schedule updated successfully!\n");
    } else {
        printf("Invalid surgery ID!\n");
    }
}

void cancel_surgery() {
    int surgery_id;
    printf("Enter surgery ID to cancel: ");
    scanf("%d", &surgery_id);

    if (surgery_id > 0 && surgery_id <= surgery_count) {
        for (int i = surgery_id - 1; i < surgery_count - 1; i++) {
            surgeries[i] = surgeries[i + 1];
```

```
        }
        surgery_count--;
        printf("Surgery cancelled successfully!\n");
    } else {
        printf("Invalid surgery ID!\n");
    }
}

void list_all_surgeries() {
    if (surgery_count == 0) {
        printf("No surgeries scheduled.\n");
        return;
    }
    for (int i = 0; i < surgery_count; i++) {
        printf("\nSurgery ID: %d\n", surgeries[i].surgery_id);
        printf("Patient Name: %s\n", surgeries[i].patient.patient_name);
        printf("Surgery Type: %s\n", surgeries[i].surgery_type);
        printf("Surgery Date: %s\n", surgeries[i].surgery_date);
    }
}

int main() {
    bool is_on = true;

    while (is_on) {
        int user_option;
        printf("\nMenu Options:\n");
        printf("1. Schedule Surgery\n");
        printf("2. View Surgery Schedule\n");
```

```
printf("3. Update Surgery Schedule\n");
printf("4. Cancel Surgery\n");
printf("5. List All Surgeries\n");
printf("6. Exit\n");
printf("Enter your option: ");
scanf("%d", &user_option);

switch (user_option) {
   case 1:
      schedule_surgery();
      break;
   case 2:
      view_surgery_schedule();
      break;
   case 3:
      update_surgery_schedule();
      break;
   case 4:
      cancel_surgery();
      break;
   case 5:
      list_all_surgeries();
      break;
   case 6:
      printf("Exiting program...\n");
      is_on = false;
      break;
   default:
      printf("Invalid option, please try again.\n");
```

```
            break;
        }
    }

    return 0;
}
```

10.

```
#include<stdio.h>
#include<string.h>
#include<stdbool.h>
struct patient_history{
    char disease[50];
    char year[50];
};
struct details{
    int surgery_id;
    char patient_name[30];
    int age;
    char date[30];
    struct patient_history history;
};
struct details arr[50];
int record_count=0;
void schedule_surgery(){
    if(record_count<50){
        struct details *new_detail=&arr[record_count];
        new_detail->surgery_id=record_count+1;
```

```
        printf("enter the patient name :");
        getchar();
        fgets(new_detail->patient_name,30,stdin);
        new_detail->patient_name[strcspn(new_detail->patient_name,"\n")]='\0';
        printf("Enter the patient age :");
        scanf("%d",&new_detail->age);
        printf("enter the surgery Date :");
        getchar();
        fgets(new_detail->date,30,stdin);
        new_detail->date[strcspn(new_detail->date,"\n")]='\0';
        printf("enter the disease name :");
        getchar();
        fgets(new_detail->history.disease,30,stdin);
        new_detail->history.disease[strcspn(new_detail->history.disease,"\n")]='\0';
        printf("enter the year of diaognosis :");
        getchar();
        fgets(new_detail->history.year,30,stdin);
        new_detail->history.year[strcspn(new_detail->history.year,"\n")]='\0';
        record_count+=1;
        printf("Sucessfully Added!");
    }else{
        printf("records full!,cannot Enter more details!");
    }
}
void view_surgery_details(){
    int rec_id;
    printf("Enter the Surgery id :");
    scanf("%d",&rec_id);
    if(rec_id>=0 && rec_id<=record_count){
```

```
        struct details *detail=&arr[rec_id];
        printf("Name: %s\n",detail->patient_name);
        printf("Age :%d\n",detail->age);
        printf("Disease :%s\n",detail->history.disease);
        printf("Year :%s\n",detail->history.year);
    }else{
        printf("Enter a valid ID!");
    }
}
void Update_surgery_schedule(){
    int rec_id;
    printf("Enter the Surgery id :");
    scanf("%d",&rec_id);
    if(rec_id>=0 && rec_id<=record_count){
        struct details *detail=&arr[rec_id];
        printf("Enter the new operation schedule :");
        getchar();
        fgets(detail->date,30,stdin);
        detail->date[strcspn(detail->date,"\n")]='\0';
        printf("Sucessfully Updated!");
    }else{
        printf("enter a valid ID!");
    }
}
void cancel_surgery(){
    int rec_id;
    printf("Enter the Surgery id :");
    scanf("%d",&rec_id);
    if(rec_id>=0 && rec_id<=record_count){
```

```
        for(int i=rec_id;i<record_count;i++){
            arr[i]=arr[i+1];
        }
        record_count-=1;
        printf("Sucessfully Cancelled!");
    }else{
        printf("Enter a valid ID!");
    }
}
void list_all_surgery(){
    if(record_count==0){
        printf("No surgeries scheduled!");
        return;
    }else{
        printf("the surgeries are :\n");
        for(int i=0;i<record_count;i++){
        printf("\nSurgery ID: %d\n", arr[i].surgery_id);
        printf("Patient Name: %s\n", arr[i].patient_name);
        printf("Age: %d\n", arr[i].age);
        printf("Surgery Date: %s\n", arr[i].date);
        printf("\n");
        }
    }
}
int main(){
    bool is_on=true;
    while(is_on){
        int user_option;
```

```
    printf("\n1.Schedule Surgery\n2.View Surgery Schedule\n3.Update Surgery Schedule\n4.Cancel Surgery\n5.List All Surgeries\n6.Exit\nEnter your option :");
    scanf("%d",&user_option);
    switch(user_option){
      case 1:
        schedule_surgery();
        break;
      case 2:
        view_surgery_details();
        break;
      case 3:
        Update_surgery_schedule();
        break;
      case 4:
        cancel_surgery();
        break;
      case 5:
        list_all_surgery();
        break;
      case 6:
        printf("Exiting...");
        is_on=false;
        break;
      default:
        printf("Please Enter a valid option!");
    }
  }
  return 0;
```

```
}
```

----------Linked List----------

1.

patient queue management

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

struct node {
   int data;
   struct node *next;
} *first = NULL;

void create_queue(int a[], int size) {
   struct node *temp, *last;
   first = (struct node*) malloc(sizeof(struct node));
   first->data = a[0];
   first->next = NULL;
   last = first;

   for (int i = 1; i < size; i++) {
      temp = (struct node*) malloc(sizeof(struct node));
      temp->data = a[i];
      temp->next = NULL;
      last->next = temp;
      last = temp;
   }
}

void insert_patient(int patient) {
```

```
    struct node *temp = (struct node*) malloc(sizeof(struct node));
    temp->data = patient;
    temp->next = NULL;

    if (first == NULL) {
        first = temp;
    } else {
        struct node *last = first;
        while (last->next != NULL) {
            last = last->next;
        }
        last->next = temp;
    }
}

void display_queue() {
    struct node *temp = first;
    if (temp == NULL) {
        printf("No patients in the queue.\n");
        return;
    }
    while (temp != NULL) {
        printf("Patient ID: %d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
```

```
    int queue[] = {1, 2, 3, 4, 5};
    create_queue(queue, 5);
    display_queue();

    insert_patient(6);
    display_queue();

    return 0;
}
```

2.

hospital ward allocation

```
#include <stdio.h>
#include <stdlib.h>
#include<stdbool.h>
struct bed {
    int bed_id;
    int is_occupied;
    struct bed *next;
}*first = NULL;

void create_beds(int total_beds) {
    struct bed *last, *new_bed;
    first = (struct bed *)malloc(sizeof(struct bed));
    first->bed_id = 1;
    first->is_occupied = 0;
    first->next = NULL;
    last = first;
```

```
    for (int i = 2; i <= total_beds; i++) {
        new_bed = (struct bed *)malloc(sizeof(struct bed));
        new_bed->bed_id = i;
        new_bed->is_occupied = 0;
        new_bed->next = NULL;
        last->next = new_bed;
        last = new_bed;
    }
}

void allocate_bed() {
    struct bed *temp = first;
    int bed_found = 0;

    while (temp != NULL) {
        if (temp->is_occupied == 0) {
            temp->is_occupied = 1;
            printf("Bed %d has been allocated.\n", temp->bed_id);
            bed_found = 1;
            break;
        }
        temp = temp->next;
    }

    if (!bed_found) {
        printf("No free beds available.\n");
    }
}
```

```
void display_beds() {
    struct bed *temp = first;
    printf("Current Bed Allocation:\n");
    while (temp != NULL) {
        if (temp->is_occupied == 0) {
            printf("Bed %d: Free\n", temp->bed_id);
        } else {
            printf("Bed %d: Occupied\n", temp->bed_id);
        }
        temp = temp->next;
    }
}

int main() {
    int user_input;
    int beds;
    printf("Enter the number of beds :");
    scanf("%d",&beds);
    create_beds(beds);
    bool is_on=true;
    int user_choice;
    while (is_on) {
        printf("1. Allocate Bed\n");
        printf("2. Display Allocated beds\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &user_choice);
        switch (user_choice) {
            case 1:
```

```
            allocate_bed();
            break;
        case 2:
            display_beds();
            break;
        case 3:
            printf("Exiting...\n");
            is_on=false;
            break;
        default:
            printf("Invalid choice!\n");
      }
   }

   return 0;
}
```

3.

Doctor appointment sheduling

```
#include<stdio.h>
#include<string.h>
#include<stdbool.h>
#include<stdlib.h>
struct appoinment{
  char patient_name[50];
  char appoinment_date[50];
  char appoinment_time[50];
  struct appoinment *next;
};
```

```
void insert_appoinment(struct appoinment** head){
   struct appoinment *new_appoinment=(struct appoinment*)malloc(sizeof(struct appoinment));
   printf("enter the patient name :");
   getchar();
   fgets(new_appoinment->patient_name,50,stdin);
   new_appoinment->patient_name[strcspn(new_appoinment->patient_name, "\n")] = '\0';
   printf("Enter appointment date (YYYY-MM-DD): ");
   fgets(new_appoinment->appoinment_date, 20, stdin);
   new_appoinment->appoinment_date[strcspn(new_appoinment->appoinment_date, "\n")] = '\0';

   printf("Enter appointment time (HH:MM): ");
   fgets(new_appoinment->appoinment_time, 10, stdin);
   new_appoinment->appoinment_time[strcspn(new_appoinment->appoinment_time, "\n")] = '\0';

   new_appoinment->next=*head;
   *head=new_appoinment;
}
void display_appoinments(struct appoinment *head){
   if (head == NULL) {
      printf("No appointments scheduled.\n");
      return;
   }
   struct appoinment *temp=head;
   while(head!=NULL){
         printf("Patient: %s, Date: %s, Time: %s\n", temp->patient_name, temp->appoinment_date,
temp->appoinment_time);
      temp = temp->next;
   }
}
```

```
int main(){
   struct appoinment *head=NULL;
   int choice;
   while(1){
      printf("\nDoctor Appointment Scheduling Menu:\n");
      printf("1. Schedule a new appointment\n");
      printf("2. Display all appointments\n");
      printf("3. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);
      switch(choice){
         case 1:
            insert_appoinment(&head);
            break;
         case 2:
            display_appoinments(head);
            break;
         case 3:
            printf("Exiting...");
         default:
            printf("Invalid choice..");
      }
   }
   return 0;
}
```

4.

Emergency contact list

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

```
#include<stdbool.h>
#include<stdlib.h>
#include<string.h>
struct contact{
    int contact_id;
    char name[30];
    char phone_number[30];
    struct contact *next;
    int is_allocated;
}*first=NULL;

void create_contact_list(int n){
    struct contact *last,*new_contact;
    first=(struct contact*)malloc(sizeof(struct contact));
    first->contact_id=1;
    printf("Enter the name of the user :");
    getchar();
    fgets(first->name,30,stdin);
    first->name[strcspn(first->name,"\n")]='\0';
    printf("enter the phone number :");
    getchar();
    fgets(first->phone_number,30,stdin);
    first->phone_number[strcspn(first->phone_number,"\n")]='\0';
    first->is_allocated=0;
    first->next=NULL;
    last=first;
    for(int i=1;i<n;i++){
        struct contact *temp=(struct contact*)malloc(sizeof(struct contact));
        temp->contact_id=i+1;
```

```
        temp->is_allocated=0;
        printf("Enter the name of the user: ");
        getchar();
        fgets(temp->name, 30, stdin);
        temp->name[strcspn(temp->name, "\n")] = '\0';
        printf("Enter the phone number: ");
        fgets(temp->phone_number, 30, stdin);
        temp->phone_number[strcspn(temp->phone_number, "\n")] = '\0';
        temp->next=NULL;
        last->next=temp;
        last=temp;
    }
}
void allocate_contact(){
    struct contact *temp=first;
    int contact_found=0;
    while(temp!=NULL){
        if(temp->is_allocated==0){
            temp->is_allocated=1;
            printf("Contact ID %d (%s, %s) has been allocated.\n", temp->contact_id, temp->name, temp-
>phone_number);
            contact_found=1;
            break;
        }
        temp=temp->next;
    }
}
void display_contacts(){
    struct contact *temp=first;
```

```
    while(temp!=NULL){
        printf("Contact ID: %d, Name: %s, Phone: %s, Allocated: %s\n",temp->contact_id,temp->name,temp->phone_number,(temp->is_allocated==0)?"No":"Yes");
        temp=temp->next;
    }
}
int main() {
    int user_input;
    int contacts;
    printf("Enter the number of contacts: ");
    scanf("%d", &contacts);
    create_contact_list(contacts);
    bool is_on = true;
    int user_choice;
    while (is_on) {
        printf("\n1. Allocate Emergency Contact\n");
        printf("2. Display All Contacts\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &user_choice);
        switch (user_choice) {
            case 1:
                allocate_contact();
                break;
            case 2:
                display_contacts();
                break;
            case 3:
                printf("Exiting...\n");
```

```
            is_on = false;
            break;
        default:
            printf("Invalid choice!\n");
    }
  }

  return 0;
}
```

5.

surgery scheduling system

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

struct Surgery {
  int surgery_id;
  char surgery_name[50];
  char patient_name[50];
  char surgery_date[20];
  struct Surgery* next;
} *first_surgery = NULL;

void create_surgery_schedule(int n) {
  struct Surgery *last, *new_surgery;
  first_surgery = (struct Surgery *)malloc(sizeof(struct Surgery));

  first_surgery->surgery_id = 1;
```

```
    printf("Enter Surgery Name: ");
    fgets(first_surgery->surgery_name, 50, stdin);
    first_surgery->surgery_name[strcspn(first_surgery->surgery_name, "\n")] = '\0';  // Remove newline
    printf("Enter Patient Name: ");
    fgets(first_surgery->patient_name, 50, stdin);
    first_surgery->patient_name[strcspn(first_surgery->patient_name, "\n")] = '\0';  // Remove newline
    printf("Enter Surgery Date (DD/MM/YYYY): ");
    fgets(first_surgery->surgery_date, 20, stdin);
    first_surgery->surgery_date[strcspn(first_surgery->surgery_date, "\n")] = '\0';  // Remove newline
    first_surgery->next = NULL;
    last = first_surgery;

    for (int i = 1; i < n; i++) {
        new_surgery = (struct Surgery *)malloc(sizeof(struct Surgery));
        new_surgery->surgery_id = i + 1;
        printf("Enter Surgery Name: ");
        fgets(new_surgery->surgery_name, 50, stdin);
        new_surgery->surgery_name[strcspn(new_surgery->surgery_name, "\n")] = '\0';  // Remove newline
        printf("Enter Patient Name: ");
        fgets(new_surgery->patient_name, 50, stdin);
        new_surgery->patient_name[strcspn(new_surgery->patient_name, "\n")] = '\0';  // Remove newline
        printf("Enter Surgery Date (DD/MM/YYYY): ");
        fgets(new_surgery->surgery_date, 20, stdin);
        new_surgery->surgery_date[strcspn(new_surgery->surgery_date, "\n")] = '\0';  // Remove newline
        new_surgery->next = NULL;
        last->next = new_surgery;
        last = new_surgery;
    }
}
```

```
void insert_new_surgery() {
    struct Surgery *new_surgery = (struct Surgery *)malloc(sizeof(struct Surgery));
    printf("Enter Surgery Name: ");
    getchar();  // Consume newline from previous input
    fgets(new_surgery->surgery_name, 50, stdin);
    new_surgery->surgery_name[strcspn(new_surgery->surgery_name, "\n")] = '\0';
    printf("Enter Patient Name: ");
    fgets(new_surgery->patient_name, 50, stdin);
    new_surgery->patient_name[strcspn(new_surgery->patient_name, "\n")] = '\0';
    printf("Enter Surgery Date (DD/MM/YYYY): ");
    fgets(new_surgery->surgery_date, 20, stdin);
    new_surgery->surgery_date[strcspn(new_surgery->surgery_date, "\n")] = '\0';
    new_surgery->next = first_surgery;
    first_surgery = new_surgery;
}

void display_surgeries() {
    struct Surgery *temp = first_surgery;
    while (temp != NULL) {
        printf("Surgery ID: %d, Surgery: %s, Patient: %s, Date: %s\n",
               temp->surgery_id, temp->surgery_name, temp->patient_name, temp->surgery_date);
        temp = temp->next;
    }
}

int main() {
    int num_surgeries;
    printf("Enter the number of surgeries to schedule: ");
```

```
    scanf("%d", &num_surgeries);
    create_surgery_schedule(num_surgeries);

    int choice;
    while (1) {
        printf("\n1. Insert New Surgery\n");
        printf("2. Display All Surgeries\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                insert_new_surgery();
                break;
            case 2:
                display_surgeries();
                break;
            case 3:
                exit(0);
            default:
                printf("Invalid choice!\n");
        }
    }

    return 0;
}
```

6.patient history record

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

struct PatientHistory {
    int record_id;
    char patient_name[50];
    char disease[50];
    char treatment[50];
    struct PatientHistory* next;
} *first_history = NULL;

void create_patient_history(int n) {
    struct PatientHistory *last, *new_record;
    first_history = (struct PatientHistory *)malloc(sizeof(struct PatientHistory));

    first_history->record_id = 1;
    printf("Enter Patient Name: ");
    getchar();  // Consume newline from previous input
    fgets(first_history->patient_name, 50, stdin);
    first_history->patient_name[strcspn(first_history->patient_name, "\n")] = '\0'; // Remove newline
    printf("Enter Disease: ");
    fgets(first_history->disease, 50, stdin);
    first_history->disease[strcspn(first_history->disease, "\n")] = '\0';
    printf("Enter Treatment: ");
    fgets(first_history->treatment, 50, stdin);
    first_history->treatment[strcspn(first_history->treatment, "\n")] = '\0';
    first_history->next = NULL;
    last = first_history;
```

```
    for (int i = 1; i < n; i++) {
        new_record = (struct PatientHistory *)malloc(sizeof(struct PatientHistory));
        new_record->record_id = i + 1;
        printf("Enter Patient Name: ");
        fgets(new_record->patient_name, 50, stdin);
        new_record->patient_name[strcspn(new_record->patient_name, "\n")] = '\0';
        printf("Enter Disease: ");
        fgets(new_record->disease, 50, stdin);
        new_record->disease[strcspn(new_record->disease, "\n")] = '\0';
        printf("Enter Treatment: ");
        fgets(new_record->treatment, 50, stdin);
        new_record->treatment[strcspn(new_record->treatment, "\n")] = '\0';
        new_record->next = NULL;
        last->next = new_record;
        last = new_record;
    }
}

void insert_new_record() {
    struct PatientHistory *new_record = (struct PatientHistory *)malloc(sizeof(struct PatientHistory));
    printf("Enter Patient Name: ");
    getchar();  // Consume newline from previous input
    fgets(new_record->patient_name, 50, stdin);
    new_record->patient_name[strcspn(new_record->patient_name, "\n")] = '\0';
    printf("Enter Disease: ");
    fgets(new_record->disease, 50, stdin);
    new_record->disease[strcspn(new_record->disease, "\n")] = '\0';
    printf("Enter Treatment: ");
```

```
    fgets(new_record->treatment, 50, stdin);
    new_record->treatment[strcspn(new_record->treatment, "\n")] = '\0';
    new_record->next = first_history;
    first_history = new_record;
}

void display_patient_history() {
    struct PatientHistory *temp = first_history;
    while (temp != NULL) {
        printf("Record ID: %d, Patient: %s, Disease: %s, Treatment: %s\n",
               temp->record_id, temp->patient_name, temp->disease, temp->treatment);
        temp = temp->next;
    }
}

int main() {
    int num_records;
    printf("Enter the number of patient history records: ");
    scanf("%d", &num_records);
    create_patient_history(num_records);

    int choice;
    while (1) {
        printf("\n1. Insert New Record\n");
        printf("2. Display All Patient Records\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
        switch (choice) {
            case 1:
                insert_new_record();
                break;
            case 2:
                display_patient_history();
                break;
            case 3:
                exit(0);
            default:
                printf("Invalid choice!\n");
        }
    }

    return 0;
}
```

7.

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

struct PatientHistory {
    int record_id;
    char patient_name[50];
    char disease[50];
    char treatment[50];
    struct PatientHistory* next;
} *first_history = NULL;
```

```
void create_patient_history(int n) {
    struct PatientHistory *last, *new_record;
    first_history = (struct PatientHistory *)malloc(sizeof(struct PatientHistory));

    first_history->record_id = 1;
    printf("Enter Patient Name: ");
    getchar();  // Consume newline from previous input
    fgets(first_history->patient_name, 50, stdin);
    first_history->patient_name[strcspn(first_history->patient_name, "\n")] = '\0'; // Remove newline
    printf("Enter Disease: ");
    fgets(first_history->disease, 50, stdin);
    first_history->disease[strcspn(first_history->disease, "\n")] = '\0';
    printf("Enter Treatment: ");
    fgets(first_history->treatment, 50, stdin);
    first_history->treatment[strcspn(first_history->treatment, "\n")] = '\0';
    first_history->next = NULL;
    last = first_history;

    for (int i = 1; i < n; i++) {
        new_record = (struct PatientHistory *)malloc(sizeof(struct PatientHistory));
        new_record->record_id = i + 1;
        printf("Enter Patient Name: ");
        fgets(new_record->patient_name, 50, stdin);
        new_record->patient_name[strcspn(new_record->patient_name, "\n")] = '\0';
        printf("Enter Disease: ");
        fgets(new_record->disease, 50, stdin);
        new_record->disease[strcspn(new_record->disease, "\n")] = '\0';
        printf("Enter Treatment: ");
```

```
        fgets(new_record->treatment, 50, stdin);
        new_record->treatment[strcspn(new_record->treatment, "\n")] = '\0';
        new_record->next = NULL;
        last->next = new_record;
        last = new_record;
    }
}

void insert_new_record() {
    struct PatientHistory *new_record = (struct PatientHistory *)malloc(sizeof(struct PatientHistory));
    printf("Enter Patient Name: ");
    getchar();  // Consume newline from previous input
    fgets(new_record->patient_name, 50, stdin);
    new_record->patient_name[strcspn(new_record->patient_name, "\n")] = '\0';
    printf("Enter Disease: ");
    fgets(new_record->disease, 50, stdin);
    new_record->disease[strcspn(new_record->disease, "\n")] = '\0';
    printf("Enter Treatment: ");
    fgets(new_record->treatment, 50, stdin);
    new_record->treatment[strcspn(new_record->treatment, "\n")] = '\0';
    new_record->next = first_history;
    first_history = new_record;
}

void display_patient_history() {
    struct PatientHistory *temp = first_history;
    while (temp != NULL) {
        printf("Record ID: %d, Patient: %s, Disease: %s, Treatment: %s\n",
               temp->record_id, temp->patient_name, temp->disease, temp->treatment);
```

```
        temp = temp->next;
    }
}

int main() {
    int num_records;
    printf("Enter the number of patient history records: ");
    scanf("%d", &num_records);
    create_patient_history(num_records);

    int choice;
    while (1) {
        printf("\n1. Insert New Record\n");
        printf("2. Display All Patient Records\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                insert_new_record();
                break;
            case 2:
                display_patient_history();
                break;
            case 3:
                exit(0);
            default:
                printf("Invalid choice!\n");
```

```
        }
    }

    return 0;
}

8.
medical test tarcking
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct MedicalTest {
    int test_id;
    char patient_name[50];
    char test_name[50];
    char test_result[50];
    struct MedicalTest* next;
} *first_test = NULL;

void create_medical_test_list(int n) {
    struct MedicalTest *last, *new_test;
    first_test = (struct MedicalTest *)malloc(sizeof(struct MedicalTest));

    first_test->test_id = 1;
    printf("Enter Patient Name: ");
    getchar();  // Consume newline from previous input
    fgets(first_test->patient_name, 50, stdin);
    first_test->patient_name[strcspn(first_test->patient_name, "\n")] = '\0'; // Remove newline
```

```
    printf("Enter Test Name: ");
    fgets(first_test->test_name, 50, stdin);
    first_test->test_name[strcspn(first_test->test_name, "\n")] = '\0';
    printf("Enter Test Result: ");
    fgets(first_test->test_result, 50, stdin);
    first_test->test_result[strcspn(first_test->test_result, "\n")] = '\0';
    first_test->next = NULL;
    last = first_test;

    for (int i = 1; i < n; i++) {
        new_test = (struct MedicalTest *)malloc(sizeof(struct MedicalTest));
        new_test->test_id = i + 1;
        printf("Enter Patient Name: ");
        fgets(new_test->patient_name, 50, stdin);
        new_test->patient_name[strcspn(new_test->patient_name, "\n")] = '\0';
        printf("Enter Test Name: ");
        fgets(new_test->test_name, 50, stdin);
        new_test->test_name[strcspn(new_test->test_name, "\n")] = '\0';
        printf("Enter Test Result: ");
        fgets(new_test->test_result, 50, stdin);
        new_test->test_result[strcspn(new_test->test_result, "\n")] = '\0';
        new_test->next = NULL;
        last->next = new_test;
        last = new_test;
    }
}

void insert_new_test_result() {
    struct MedicalTest *new_test = (struct MedicalTest *)malloc(sizeof(struct MedicalTest));
```

```
    printf("Enter Patient Name: ");
    getchar();  // Consume newline from previous input
    fgets(new_test->patient_name, 50, stdin);
    new_test->patient_name[strcspn(new_test->patient_name, "\n")] = '\0';
    printf("Enter Test Name: ");
    fgets(new_test->test_name, 50, stdin);
    new_test->test_name[strcspn(new_test->test_name, "\n")] = '\0';
    printf("Enter Test Result: ");
    fgets(new_test->test_result, 50, stdin);
    new_test->test_result[strcspn(new_test->test_result, "\n")] = '\0';
    new_test->next = first_test;
    first_test = new_test;
}

void display_test_results() {
    struct MedicalTest *temp = first_test;
    while (temp != NULL) {
        printf("Test ID: %d, Patient: %s, Test: %s, Result: %s\n",
               temp->test_id, temp->patient_name, temp->test_name, temp->test_result);
        temp = temp->next;
    }
}

int main() {
    int num_tests;
    printf("Enter the number of medical tests to track: ");
    scanf("%d", &num_tests);
    create_medical_test_list(num_tests);
```

```
    int choice;
    while (1) {
        printf("\n1. Insert New Test Result\n");
        printf("2. Display All Test Results\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                insert_new_test_result();
                break;
            case 2:
                display_test_results();
                break;
            case 3:
                exit(0);
            default:
                printf("Invalid choice!\n");
        }
    }

    return 0;
}
```

9.prescription tracking system

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

```
struct Prescription {
    int prescription_id;
    char patient_name[50];
    char medication[50];
    char dosage[50];
    struct Prescription* next;
} *first_prescription = NULL;

void create_prescription_list(int n) {
    struct Prescription *last, *new_prescription;
    first_prescription = (struct Prescription *)malloc(sizeof(struct Prescription));

    first_prescription->prescription_id = 1;
    printf("Enter Patient Name: ");
    getchar();  // Consume newline from previous input
    fgets(first_prescription->patient_name, 50, stdin);
    first_prescription->patient_name[strcspn(first_prescription->patient_name, "\n")] = '\0'; // Remove
newline
    printf("Enter Medication: ");
    fgets(first_prescription->medication, 50, stdin);
    first_prescription->medication[strcspn(first_prescription->medication, "\n")] = '\0';
    printf("Enter Dosage: ");
    fgets(first_prescription->dosage, 50, stdin);
    first_prescription->dosage[strcspn(first_prescription->dosage, "\n")] = '\0';
    first_prescription->next = NULL;
    last = first_prescription;

    for (int i = 1; i < n; i++) {
```

```
        new_prescription = (struct Prescription *)malloc(sizeof(struct Prescription));
        new_prescription->prescription_id = i + 1;
        printf("Enter Patient Name: ");
        fgets(new_prescription->patient_name, 50, stdin);
        new_prescription->patient_name[strcspn(new_prescription->patient_name, "\n")] = '\0';
        printf("Enter Medication: ");
        fgets(new_prescription->medication, 50, stdin);
        new_prescription->medication[strcspn(new_prescription->medication, "\n")] = '\0';
        printf("Enter Dosage: ");
        fgets(new_prescription->dosage, 50, stdin);
        new_prescription->dosage[strcspn(new_prescription->dosage, "\n")] = '\0';
        new_prescription->next = NULL;
        last->next = new_prescription;
        last = new_prescription;
    }
}

void insert_new_prescription() {
    struct Prescription *new_prescription = (struct Prescription *)malloc(sizeof(struct Prescription));
    printf("Enter Patient Name: ");
    getchar();  // Consume newline from previous input
    fgets(new_prescription->patient_name, 50, stdin);
    new_prescription->patient_name[strcspn(new_prescription->patient_name, "\n")] = '\0';
    printf("Enter Medication: ");
    fgets(new_prescription->medication, 50, stdin);
    new_prescription->medication[strcspn(new_prescription->medication, "\n")] = '\0';
    printf("Enter Dosage: ");
    fgets(new_prescription->dosage, 50, stdin);
    new_prescription->dosage[strcspn(new_prescription->dosage, "\n")] = '\0';
```

```
    new_prescription->next = first_prescription;
    first_prescription = new_prescription;
}

void display_prescriptions() {
    struct Prescription *temp = first_prescription;
    while (temp != NULL) {
        printf("Prescription ID: %d, Patient: %s, Medication: %s, Dosage: %s\n",
               temp->prescription_id, temp->patient_name, temp->medication, temp->dosage);
        temp = temp->next;
    }
}

int main() {
    int num_prescriptions;
    printf("Enter the number of prescriptions to manage: ");
    scanf("%d", &num_prescriptions);
    create_prescription_list(num_prescriptions);

    int choice;
    while (1) {
        printf("\n1. Insert New Prescription\n");
        printf("2. Display All Prescriptions\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
```

```
            insert_new_prescription();
            break;
        case 2:
            display_prescriptions();
            break;
        case 3:
            exit(0);
        default:
            printf("Invalid choice!\n");
      }
   }

   return 0;
}
```

10.hospital staff roster

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

struct Staff {
   int staff_id;
   char name[50];
   char role[50];
   struct Staff* next;
} *first_staff = NULL;

void create_staff_roster(int n) {
   struct Staff *last, *new_staff;
```

```
    first_staff = (struct Staff *)malloc(sizeof(struct Staff));

    first_staff->staff_id = 1;
    printf("Enter Name: ");
    getchar();  // Consume newline from previous input
    fgets(first_staff->name, 50, stdin);
    first_staff->name[strcspn(first_staff->name, "\n")] = '\0';  // Remove newline
    printf("Enter Role: ");
    fgets(first_staff->role, 50, stdin);
    first_staff->role[strcspn(first_staff->role, "\n")] = '\0';
    first_staff->next = NULL;
    last = first_staff;

    for (int i = 1; i < n; i++) {
        new_staff = (struct Staff *)malloc(sizeof(struct Staff));
        new_staff->staff_id = i + 1;
        printf("Enter Name: ");
        fgets(new_staff->name, 50, stdin);
        new_staff->name[strcspn(new_staff->name, "\n")] = '\0';
        printf("Enter Role: ");
        fgets(new_staff->role, 50, stdin);
        new_staff->role[strcspn(new_staff->role, "\n")] = '\0';
        new_staff->next = NULL;
        last->next = new_staff;
        last = new_staff;
    }
}

void insert_new_staff_member() {
```

```
    struct Staff *new_staff = (struct Staff *)malloc(sizeof(struct Staff));
    printf("Enter Name: ");
    getchar();  // Consume newline from previous input
    fgets(new_staff->name, 50, stdin);
    new_staff->name[strcspn(new_staff->name, "\n")] = '\0';
    printf("Enter Role: ");
    fgets(new_staff->role, 50, stdin);
    new_staff->role[strcspn(new_staff->role, "\n")] = '\0';
    new_staff->next = first_staff;
    first_staff = new_staff;
}

void display_staff_roster() {
    struct Staff *temp = first_staff;
    while (temp != NULL) {
        printf("Staff ID: %d, Name: %s, Role: %s\n",
               temp->staff_id, temp->name, temp->role);
        temp = temp->next;
    }
}

int main() {
    int num_staff;
    printf("Enter the number of staff members: ");
    scanf("%d", &num_staff);
    create_staff_roster(num_staff);

    int choice;
    while (1) {
```

```
        printf("\n1. Insert New Staff Member\n");
        printf("2. Display Staff Roster\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                insert_new_staff_member();
                break;
            case 2:
                display_staff_roster();
                break;
            case 3:
                exit(0);
            default:
                printf("Invalid choice!\n");
        }
    }

    return 0;
}
```