Day 16 programs

Problem 1: Patient Information Management System

Description: Create a menu-driven program to manage patient information, including basic details, medical history, and current medications.

Menu Options:

Add New Patient

View Patient Details

Update Patient Information

Delete Patient Record

List All Patients

Exit

Requirements:

Use variables to store patient details.

Utilize static and const for immutable data such as hospital name.

Implement switch case for menu selection.

Employ loops for iterative tasks like listing patients.

Use pointers for dynamic memory allocation.

Implement functions for CRUD operations.

Utilize arrays for storing multiple patient records.

Use structures for organizing patient data.

Apply nested structures for detailed medical history.

Use unions for optional data fields.

Employ nested unions for multi-type data entries.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>
```

```
#define MAX_PATIENTS 100

// Structure for storing medical history
typedef struct {
    char condition[100];
    char treatment[100];
} MedicalHistory;

// Union for optional data fields
typedef union {
    char phone[15];
    char email[50];
} ContactInfo;

// Nested union for multi-type data entries
typedef union {
    int age;
    float weight;
    char bloodGroup[5];
} OptionalDetails;

// Structure for patient data
typedef struct {
    int id;
    char name[50];
    char gender[10];
    MedicalHistory history[5]; // Array to hold multiple medical history records
    char medications[200];
    ContactInfo contact;
```

```
    OptionalDetails details;
} Patient;

// Array to store patients
Patient *patients[MAX_PATIENTS];
int patientCount = 0;

// Function prototypes
void addNewPatient();
void viewPatientDetails();
void updatePatientInformation();
void deletePatientRecord();
void listAllPatients();

int main() {
    int choice;

    printf("----------Patient Information Management System----------\n");

    do {
        printf("\nMenu:\n");
        printf("1. Add New 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 choice: ");
        scanf("%d", &choice);
```

```
        switch (choice) {
            case 1:
                addNewPatient();
                break;
            case 2:
                viewPatientDetails();
                break;
            case 3:
                updatePatientInformation();
                break;
            case 4:
                deletePatientRecord();
                break;
            case 5:
                listAllPatients();
                break;
            case 6:
                printf("Exiting the program. Goodbye!\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 6);

    return 0;
}

void addNewPatient() {
```

```
    if (patientCount >= MAX_PATIENTS) {
        printf("Patient capacity reached!\n");
        return;
    }

    Patient *newPatient = (Patient *)malloc(sizeof(Patient));

    printf("Enter Patient ID: ");
    scanf("%d", &newPatient->id);
    printf("Enter Patient Name: ");
    scanf("%s", newPatient->name);
    printf("Enter Gender: ");
    scanf("%s", newPatient->gender);
    printf("Enter Medications: ");
    scanf("%s", newPatient->medications);

    printf("Enter Contact Phone: ");
    scanf("%s", newPatient->contact.phone);

    printf("Enter Age: ");
    scanf("%d", &newPatient->details.age);

    patients[patientCount++] = newPatient;
    printf("Patient added successfully!\n");
}

void viewPatientDetails() {
    int id;
    printf("Enter Patient ID to view details: ");
```

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

    for (int i = 0; i < patientCount; i++) {
        if (patients[i]->id == id) {
            printf("Patient ID: %d\n", patients[i]->id);
            printf("Name: %s\n", patients[i]->name);
            printf("Gender: %s\n", patients[i]->gender);
            printf("Medications: %s\n", patients[i]->medications);
            printf("Contact Phone: %s\n", patients[i]->contact.phone);
            printf("Age: %d\n", patients[i]->details.age);
            return;
        }
    }

    printf("Patient not found!\n");
}

void updatePatientInformation() {
    int id;
    printf("Enter Patient ID to update: ");
    scanf("%d", &id);

    for (int i = 0; i < patientCount; i++) {
        if (patients[i]->id == id) {
            printf("Enter New Name: ");
            scanf("%s", patients[i]->name);
            printf("Enter New Medications: ");
            scanf("%s", patients[i]->medications);
            printf("Enter New Contact Phone: ");
```

```
            scanf("%s", patients[i]->contact.phone);
            printf("Patient information updated successfully!\n");
            return;
        }
    }

    printf("Patient not found!\n");
}

void deletePatientRecord() {
    int id;
    printf("Enter Patient ID to delete: ");
    scanf("%d", &id);

    for (int i = 0; i < patientCount; i++) {
        if (patients[i]->id == id) {
            free(patients[i]);
            for (int j = i; j < patientCount - 1; j++) {
                patients[j] = patients[j + 1];
            }
            patientCount--;
            printf("Patient record deleted successfully!\n");
            return;
        }
    }

    printf("Patient not found!\n");
}
```

```
void listAllPatients() {
    if (patientCount == 0) {
        printf("No patients found!\n");
        return;
    }

    printf("\nList of Patients:\n");
    for (int i = 0; i < patientCount; i++) {
        printf("ID: %d, Name: %s, Gender: %s, Medications: %s\n",
               patients[i]->id, patients[i]->name, patients[i]->gender, patients[i]->medications);
    }
}
```

Problem 2: Hospital Inventory Management

Description: Design a system to manage the inventory of medical supplies.

Menu Options:

Add Inventory Item

View Inventory Item

Update Inventory Item

Delete Inventory Item

List All Inventory Items

Exit

Requirements:

Declare variables for inventory details.

Use static and const for fixed supply details.

Implement switch case for different operations like adding, deleting, and viewing inventory.

Utilize loops for repetitive inventory checks.

Use pointers to handle inventory records.

Create functions for managing inventory.

Use arrays to store inventory items.

Define structures for each supply item.

Use nested structures for detailed item specifications.

Employ unions for variable item attributes.

Implement nested unions for complex item data types.

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

#define MAX_ITEMS 100



// Structure for detailed item specifications
typedef struct {
    char manufacturer[50];
    char expiryDate[15];
} ItemSpecifications;

// Union for variable item attributes
typedef union {
    int quantity;
    float weight;
} ItemAttributes;

// Nested union for complex item data types
typedef union {
    char serialNumber[20];
    char lotNumber[20];
```

```
} ComplexItemData;

// Structure for inventory item
typedef struct {
    int id;
    char name[50];
    char category[30];
    ItemSpecifications specs;
    ItemAttributes attributes;
    ComplexItemData complexData;
} InventoryItem;

// Array to store inventory items
InventoryItem *inventory[MAX_ITEMS];
int itemCount = 0;

// Function prototypes
void addInventoryItem();
void viewInventoryItem();
void updateInventoryItem();
void deleteInventoryItem();
void listAllInventoryItems();

int main() {
    int choice;

    printf("----------Hospital Inventory Management----------\n");

    do {
```

```
printf("\nMenu:\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 choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        addInventoryItem();
        break;
    case 2:
        viewInventoryItem();
        break;
    case 3:
        updateInventoryItem();
        break;
    case 4:
        deleteInventoryItem();
        break;
    case 5:
        listAllInventoryItems();
        break;
    case 6:
        printf("Exiting the program. Goodbye!\n");
        break;
```

```
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 6);

    return 0;
}

void addInventoryItem() {
    if (itemCount >= MAX_ITEMS) {
        printf("Inventory capacity reached! Cannot add more items.\n");
        return;
    }

    InventoryItem *newItem = (InventoryItem *)malloc(sizeof(InventoryItem));

    printf("Enter Item ID: ");
    scanf("%d", &newItem->id);
    printf("Enter Item Name: ");
    scanf("%s", newItem->name);
    printf("Enter Item Category: ");
    scanf("%s", newItem->category);
    printf("Enter Manufacturer: ");
    scanf("%s", newItem->specs.manufacturer);
    printf("Enter Expiry Date: ");
    scanf("%s", newItem->specs.expiryDate);
    printf("Enter Quantity: ");
    scanf("%d", &newItem->attributes.quantity);
    printf("Enter Serial Number: ");
```

```
    scanf("%s", newItem->complexData.serialNumber);

    inventory[itemCount++] = newItem;
    printf("Inventory item added successfully!\n");
}

void viewInventoryItem() {
    int id;
    printf("Enter Item ID to view details: ");
    scanf("%d", &id);

    for (int i = 0; i < itemCount; i++) {
        if (inventory[i]->id == id) {
            printf("Item ID: %d\n", inventory[i]->id);
            printf("Name: %s\n", inventory[i]->name);
            printf("Category: %s\n", inventory[i]->category);
            printf("Manufacturer: %s\n", inventory[i]->specs.manufacturer);
            printf("Expiry Date: %s\n", inventory[i]->specs.expiryDate);
            printf("Quantity: %d\n", inventory[i]->attributes.quantity);
            printf("Serial Number: %s\n", inventory[i]->complexData.serialNumber);
            return;
        }
    }

    printf("Item not found!\n");
}

void updateInventoryItem() {
    int id;
```

```
    printf("Enter Item ID to update: ");
    scanf("%d", &id);

    for (int i = 0; i < itemCount; i++) {
        if (inventory[i]->id == id) {
            printf("Enter New Name: ");
            scanf("%s", inventory[i]->name);
            printf("Enter New Category: ");
            scanf("%s", inventory[i]->category);
            printf("Enter New Quantity: ");
            scanf("%d", &inventory[i]->attributes.quantity);
            printf("Enter New Serial Number: ");
            scanf("%s", inventory[i]->complexData.serialNumber);
            printf("Inventory item updated successfully!\n");
            return;
        }
    }

    printf("Item not found!\n");
}

void deleteInventoryItem() {
    int id;
    printf("Enter Item ID to delete: ");
    scanf("%d", &id);

    for (int i = 0; i < itemCount; i++) {
        if (inventory[i]->id == id) {
            free(inventory[i]);
```

```
            for (int j = i; j < itemCount - 1; j++) {
                inventory[j] = inventory[j + 1];
            }
            itemCount--;
            printf("Inventory item deleted successfully!\n");
            return;
        }
    }

    printf("Item not found!\n");
}

void listAllInventoryItems() {
    if (itemCount == 0) {
        printf("No inventory items found!\n");
        return;
    }

    printf("\nList of Inventory Items:\n");
    for (int i = 0; i < itemCount; i++) {
        printf("ID: %d, Name: %s, Category: %s, Manufacturer: %s\n",
               inventory[i]->id, inventory[i]->name, inventory[i]->category, inventory[i]->specs.manufacturer);
    }
}
```

Problem 3: Medical Appointment Scheduling System

Description: Develop a system to manage patient appointments.

Menu Options:

Schedule Appointment

View Appointment

Update Appointment

Cancel Appointment

List All Appointments

Exit

Requirements:

Use variables for appointment details.

Apply static and const for non-changing data like clinic hours.

Implement switch case for appointment operations.

Utilize loops for scheduling.

Use pointers for dynamic data manipulation.

Create functions for appointment handling.

Use arrays for storing appointments.

Define structures for appointment details.

Employ nested structures for detailed doctor and patient information.

Utilize unions for optional appointment data.

Apply nested unions for complex appointment data.

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

#define MAX_APPOINTMENTS 100
#define CLINIC_HOURS "9:00 AM - 5:00 PM"

// Structure for doctor and patient details
typedef struct {
    char doctorName[50];
    char patientName[50];
```

```
    char patientContact[15];
} AppointmentDetails;

// Union for optional appointment data
typedef union {
    char notes[100];
    char followUpDate[15];
} OptionalData;

// Nested union for complex appointment data
typedef union {
    int roomNumber;
    char virtualLink[100];
} ComplexData;

// Structure for appointment
typedef struct {
    int id;
    char date[15];
    char time[10];
    AppointmentDetails details;
    OptionalData optional;
    ComplexData complex;
} Appointment;

// Array to store appointments
Appointment *appointments[MAX_APPOINTMENTS];
int appointmentCount = 0;
```

```
// Function prototypes
void scheduleAppointment();
void viewAppointment();
void updateAppointment();
void cancelAppointment();
void listAllAppointments();

int main() {
    int choice;

    printf("----------Medical Appointment Scheduling System----------\n");
    printf("Clinic Hours: %s\n", CLINIC_HOURS);

    do {
        printf("\nMenu:\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 choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                scheduleAppointment();
                break;
            case 2:
```

```
            viewAppointment();
            break;
        case 3:
            updateAppointment();
            break;
        case 4:
            cancelAppointment();
            break;
        case 5:
            listAllAppointments();
            break;
        case 6:
            printf("Exiting the program. Goodbye!\n");
            break;
        default:
            printf("Invalid choice! Please try again.\n");
      }
   } while (choice != 6);

   return 0;
}

void scheduleAppointment() {
   if (appointmentCount >= MAX_APPOINTMENTS) {
      printf("Appointment capacity reached\n");
      return;
   }

   Appointment *newAppointment = (Appointment *)malloc(sizeof(Appointment));
```

```
    printf("Enter Appointment ID: ");
    scanf("%d", &newAppointment->id);
    printf("Enter Appointment Date (DD/MM/YYYY): ");
    scanf("%s", newAppointment->date);
    printf("Enter Appointment Time (HH:MM): ");
    scanf("%s", newAppointment->time);
    printf("Enter Doctor Name: ");
    scanf("%s", newAppointment->details.doctorName);
    printf("Enter Patient Name: ");
    scanf("%s", newAppointment->details.patientName);
    printf("Enter Patient Contact: ");
    scanf("%s", newAppointment->details.patientContact);
    printf("Enter Notes (optional): ");
    scanf("%s", newAppointment->optional.notes);
    printf("Enter Room Number: ");
    scanf("%d", &newAppointment->complex.roomNumber);

    appointments[appointmentCount++] = newAppointment;
    printf("Appointment scheduled successfully!\n");
}

void viewAppointment() {
    int id;
    printf("Enter Appointment ID to view: ");
    scanf("%d", &id);

    for (int i = 0; i < appointmentCount; i++) {
        if (appointments[i]->id == id) {
```

```
            printf("Appointment ID: %d\n", appointments[i]->id);
            printf("Date: %s\n", appointments[i]->date);
            printf("Time: %s\n", appointments[i]->time);
            printf("Doctor: %s\n", appointments[i]->details.doctorName);
            printf("Patient: %s\n", appointments[i]->details.patientName);
            printf("Contact: %s\n", appointments[i]->details.patientContact);
            printf("Notes: %s\n", appointments[i]->optional.notes);
            printf("Room Number: %d\n", appointments[i]->complex.roomNumber);
            return;
        }
    }

    printf("Appointment not found!\n");
}

void updateAppointment() {
    int id;
    printf("Enter Appointment ID to update: ");
    scanf("%d", &id);

    for (int i = 0; i < appointmentCount; i++) {
        if (appointments[i]->id == id) {
            printf("Enter New Date: ");
            scanf("%s", appointments[i]->date);
            printf("Enter New Time: ");
            scanf("%s", appointments[i]->time);
            printf("Enter New Notes: ");
            scanf("%s", appointments[i]->optional.notes);
            printf("Enter New Room Number: ");
```

```
            scanf("%d", &appointments[i]->complex.roomNumber);
            printf("Appointment updated successfully!\n");
            return;
        }
    }

    printf("Appointment not found!\n");
}

void cancelAppointment() {
    int id;
    printf("Enter Appointment ID to cancel: ");
    scanf("%d", &id);

    for (int i = 0; i < appointmentCount; i++) {
        if (appointments[i]->id == id) {
            free(appointments[i]);
            for (int j = i; j < appointmentCount - 1; j++) {
                appointments[j] = appointments[j + 1];
            }
            appointmentCount--;
            printf("Appointment canceled successfully!\n");
            return;
        }
    }

    printf("Appointment not found!\n");
}
```

```
void listAllAppointments() {
    if (appointmentCount == 0) {
        printf("No appointments found!\n");
        return;
    }

    printf("\nList of Appointments:\n");
    for (int i = 0; i < appointmentCount; i++) {
        printf("ID: %d, Date: %s, Time: %s, Doctor: %s, Patient: %s\n",
               appointments[i]->id, appointments[i]->date, appointments[i]->time,
               appointments[i]->details.doctorName, appointments[i]-
>details.patientName);
    }
}
```

Problem 4: Patient Billing System

Description: Create a billing system for patients.

Menu Options:

Generate Bill

View Bill

Update Bill

Delete Bill

List All Bills

Exit

Requirements:

Declare variables for billing information.

Use static and const for fixed billing rates.

Implement switch case for billing operations.

Utilize loops for generating bills.

Use pointers for bill calculations.

Create functions for billing processes.

Use arrays for storing billing records.

Define structures for billing components.

Employ nested structures for detailed billing breakdown.

Use unions for variable billing elements.

Apply nested unions for complex billing scenarios.

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

#define MAX_BILLS 100

// Structure for patient and billing details
typedef struct {
    char patientName[50];
    char patientContact[15];
    float consultationFee;
    float roomCharges;
    float medicationCost;
    float surgeryCost;
} BillingDetails;

// Union for optional billing data
typedef union {
    float discount;
    char specialNotes[100];
} OptionalBillingData;
```

```
// Nested union for complex billing data
typedef union {
    int roomNumber;
    char billingLink[100];
} ComplexBillingData;

// Structure for billing information
typedef struct {
    int id;
    char date[15];
    BillingDetails details;
    OptionalBillingData optional;
    ComplexBillingData complex;
    float totalBill;
} Bill;

// Array to store all bills
Bill *bills[MAX_BILLS];
int billCount = 0;

// Function prototypes
void generateBill();
void viewBill();
void updateBill();
void deleteBill();
void listAllBills();

int main() {
    int choice;
```

```
printf("----------Medical Billing System----------\n");

do {
    printf("\nMenu:\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 choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            generateBill();
            break;
        case 2:
            viewBill();
            break;
        case 3:
            updateBill();
            break;
        case 4:
            deleteBill();
            break;
        case 5:
            listAllBills();
```

```
            break;
        case 6:
            printf("Exiting the program. Goodbye!\n");
            break;
        default:
            printf("Invalid choice! Please try again.\n");
      }
   } while (choice != 6);

   return 0;
}

void generateBill() {
   if (billCount >= MAX_BILLS) {
      printf("Billing capacity reached\n");
      return;
   }

   Bill *newBill = (Bill *)malloc(sizeof(Bill));

   printf("Enter Bill ID: ");
   scanf("%d", &newBill->id);
   printf("Enter Date (DD/MM/YYYY): ");
   scanf("%s", newBill->date);
   printf("Enter Patient Name: ");
   scanf("%s", newBill->details.patientName);
   printf("Enter Patient Contact: ");
   scanf("%s", newBill->details.patientContact);
   printf("Enter Consultation Fee: ");
```

```
    scanf("%f", &newBill->details.consultationFee);
    printf("Enter Room Charges: ");
    scanf("%f", &newBill->details.roomCharges);
    printf("Enter Medication Cost: ");
    scanf("%f", &newBill->details.medicationCost);
    printf("Enter Surgery Cost: ");
    scanf("%f", &newBill->details.surgeryCost);

    // Optional billing data
    printf("Enter Discount (optional, 0 if none): ");
    scanf("%f", &newBill->optional.discount);



    // Calculating total bill
    newBill->totalBill = newBill->details.consultationFee + newBill-
>details.roomCharges + newBill->details.medicationCost + newBill-
>details.surgeryCost - newBill->optional.discount;

    bills[billCount++] = newBill;
    printf("Bill generated successfully!\n");
}

void viewBill() {
    int id;
    printf("Enter Bill ID to view: ");
    scanf("%d", &id);

    for (int i = 0; i < billCount; i++) {
        if (bills[i]->id == id) {
            printf("Bill ID: %d\n", bills[i]->id);
```

```
            printf("Date: %s\n", bills[i]->date);
            printf("Patient: %s\n", bills[i]->details.patientName);
            printf("Contact: %s\n", bills[i]->details.patientContact);
            printf("Consultation Fee: $%.2f\n", bills[i]->details.consultationFee);
            printf("Room Charges: $%.2f\n", bills[i]->details.roomCharges);
            printf("Medication Cost: $%.2f\n", bills[i]->details.medicationCost);
            printf("Surgery Cost: $%.2f\n", bills[i]->details.surgeryCost);
            printf("Discount: $%.2f\n", bills[i]->optional.discount);
            if (bills[i]->complex.roomNumber != 0) {
                printf("Room Number: %d\n", bills[i]->complex.roomNumber);
            } else {
                printf("Virtual Billing Link: %s\n", bills[i]->complex.billingLink);
            }
            printf("Total Bill: $%.2f\n", bills[i]->totalBill);
            return;
        }
    }

    printf("Bill not found!\n");
}

void updateBill() {
    int id;
    printf("Enter Bill ID to update: ");
    scanf("%d", &id);

    for (int i = 0; i < billCount; i++) {
        if (bills[i]->id == id) {
            printf("Enter New Date: ");
```

```
            scanf("%s", bills[i]->date);
            printf("Enter New Consultation Fee: ");
            scanf("%f", &bills[i]->details.consultationFee);
            printf("Enter New Room Charges: ");
            scanf("%f", &bills[i]->details.roomCharges);
            printf("Enter New Medication Cost: ");
            scanf("%f", &bills[i]->details.medicationCost);
            printf("Enter New Surgery Cost: ");
            scanf("%f", &bills[i]->details.surgeryCost);
            printf("Enter New Discount: ");
            scanf("%f", &bills[i]->optional.discount);

            // Update complex data (room number or billing link)
            printf("Enter New Room Number (or 0 if virtual): ");
            scanf("%d", &bills[i]->complex.roomNumber);
            if (bills[i]->complex.roomNumber == 0) {
                printf("Enter New Virtual Billing Link: ");
                scanf("%s", bills[i]->complex.billingLink);
            }

            // Recalculate total bill
            bills[i]->totalBill = bills[i]->details.consultationFee + bills[i]-
>details.roomCharges + bills[i]->details.medicationCost + bills[i]->details.surgeryCost
- bills[i]->optional.discount;

            printf("Bill updated successfully!\n");
            return;
        }
    }
```

```
    printf("Bill not found!\n");
}

void deleteBill() {
    int id;
    printf("Enter Bill ID to delete: ");
    scanf("%d", &id);

    for (int i = 0; i < billCount; i++) {
        if (bills[i]->id == id) {
            free(bills[i]);
            for (int j = i; j < billCount - 1; j++) {
                bills[j] = bills[j + 1];
            }
            billCount--;
            printf("Bill deleted successfully!\n");
            return;
        }
    }

    printf("Bill not found!\n");
}

void listAllBills() {
    if (billCount == 0) {
        printf("No bills found!\n");
        return;
    }
```

```
    printf("\nList of All Bills:\n");
    for (int i = 0; i < billCount; i++) {
        printf("ID: %d, Patient: %s, Total Bill: $%.2f\n",
               bills[i]->id, bills[i]->details.patientName, bills[i]->totalBill);
    }
}
```

Problem 5: Medical Test Result Management

Description: Develop a system to manage and store patient test results

Menu Options:

Add Test Result

View Test Result

Update Test Result

Delete Test Result

List All Test Results

Exit

Requirements:

Declare variables for test results.

Use static and const for standard test ranges.

Implement switch case for result operations.

Utilize loops for result input and output.

Use pointers for handling result data.

Create functions for result management.

Use arrays for storing test results.

Define structures for test result details.

Employ nested structures for detailed test parameters.

Utilize unions for optional test data.

Apply nested unions for complex test result data.

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

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

#define MAX_TESTS 100

// Constants for standard test ranges
const float NORMAL_BLOOD_PRESSURE_MIN = 90.0;
const float NORMAL_BLOOD_PRESSURE_MAX = 120.0;
const float NORMAL_HEART_RATE_MIN = 60.0;
const float NORMAL_HEART_RATE_MAX = 100.0;

// Structure for individual test result details
typedef struct {
    char testName[50];
    float testValue;
    char testDate[15];
    char patientName[50];
    char patientContact[15];
} TestResultDetails;

// Union for optional test data
typedef union {
    char testNotes[100];
    float testTemperature;
} OptionalTestData;

// Nested union for complex test result data
typedef union {
    struct {
```

```
        float systolic;
        float diastolic;
    } bloodPressure;
    struct {
        float heartRate;
        float temperature;
    } heartRateData;
} ComplexTestResultData;

// Structure for test result
typedef struct {
    int id;
    TestResultDetails details;
    OptionalTestData optional;
    ComplexTestResultData complex;
} TestResult;

// Array to store test results
TestResult *testResults[MAX_TESTS];
int testCount = 0;

// Function prototypes
void addTestResult();
void viewTestResult();
void updateTestResult();
void deleteTestResult();
void listAllTestResults();

int main() {
```

```
int choice;

printf("----------Medical Test Result Management System----------\n");

do {
    printf("\nMenu:\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 choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            addTestResult();
            break;
        case 2:
            viewTestResult();
            break;
        case 3:
            updateTestResult();
            break;
        case 4:
            deleteTestResult();
            break;
        case 5:
```

```
            listAllTestResults();
            break;
        case 6:
            printf("Exiting the program. Goodbye!\n");
            break;
        default:
            printf("Invalid choice! Please try again.\n");
    }
} while (choice != 6);

return 0;
}

// Function to add test result
void addTestResult() {
    if (testCount >= MAX_TESTS) {
        printf("Test result capacity reached\n");
        return;
    }

    TestResult *newTest = (TestResult *)malloc(sizeof(TestResult));

    printf("Enter Test Result ID: ");
    scanf("%d", &newTest->id);
    printf("Enter Test Name: ");
    scanf("%s", newTest->details.testName);
    printf("Enter Patient Name: ");
    scanf("%s", newTest->details.patientName);
    printf("Enter Patient Contact: ");
```

```
    scanf("%s", newTest->details.patientContact);
    printf("Enter Test Value: ");
    scanf("%f", &newTest->details.testValue);
    printf("Enter Test Date (DD/MM/YYYY): ");
    scanf("%s", newTest->details.testDate);

    // Optional data input
    printf("Enter Test Notes (optional): ");
    scanf("%s", newTest->optional.testNotes);



    testResults[testCount++] = newTest;
    printf("Test result added successfully!\n");
}

// Function to view test result
void viewTestResult() {
    int id;
    printf("Enter Test Result ID to view: ");
    scanf("%d", &id);

    for (int i = 0; i < testCount; i++) {
        if (testResults[i]->id == id) {
            printf("Test Result ID: %d\n", testResults[i]->id);
            printf("Test Name: %s\n", testResults[i]->details.testName);
            printf("Patient: %s\n", testResults[i]->details.patientName);
            printf("Contact: %s\n", testResults[i]->details.patientContact);
            printf("Test Value: %.2f\n", testResults[i]->details.testValue);
            printf("Test Date: %s\n", testResults[i]->details.testDate);
```

```
            printf("Test Notes: %s\n", testResults[i]->optional.testNotes);

            return;
        }
    }

    printf("Test result not found!\n");
}

// Function to update a test result
void updateTestResult() {
    int id;
    printf("Enter Test Result ID to update: ");
    scanf("%d", &id);

    for (int i = 0; i < testCount; i++) {
        if (testResults[i]->id == id) {
            printf("Enter New Test Name: ");
            scanf("%s", testResults[i]->details.testName);
            printf("Enter New Test Value: ");
            scanf("%f", &testResults[i]->details.testValue);
            printf("Enter New Test Date (DD/MM/YYYY): ");
            scanf("%s", testResults[i]->details.testDate);

            // Optional data update
            printf("Enter New Test Notes (optional): ");
            scanf("%s", testResults[i]->optional.testNotes);

            printf("Test result updated successfully!\n");
```

```
            return;
        }
    }

    printf("Test result not found!\n");
}

// Function to delete a test result
void deleteTestResult() {
    int id;
    printf("Enter Test Result ID to delete: ");
    scanf("%d", &id);

    for (int i = 0; i < testCount; i++) {
        if (testResults[i]->id == id) {
            free(testResults[i]);
            for (int j = i; j < testCount - 1; j++) {
                testResults[j] = testResults[j + 1];
            }
            testCount--;
            printf("Test result deleted successfully!\n");
            return;
        }
    }

    printf("Test result not found!\n");
}

// Function to list all test results
```

```
void listAllTestResults() {
    if (testCount == 0) {
        printf("No test results found!\n");
        return;
    }

    printf("\nList of All Test Results:\n");
    for (int i = 0; i < testCount; i++) {
        printf("ID: %d, Test: %s, Patient: %s, Test Value: %.2f\n",
               testResults[i]->id, testResults[i]->details.testName,
               testResults[i]->details.patientName, testResults[i]->details.testValue);
    }
}
```

Problem 6: Staff Duty Roster Management

Description: Create a system to manage hospital staff duty rosters

Menu Options:

Add Duty Roster

View Duty Roster

Update Duty Roster

Delete Duty Roster

List All Duty Rosters

Exit

Requirements:

Use variables for staff details.

Apply static and const for fixed shift timings.

Implement switch case for roster operations.

Utilize loops for roster generation.

Use pointers for dynamic staff data.

Create functions for roster management.

Use arrays for storing staff schedules.

Define structures for duty details.

Employ nested structures for detailed duty breakdowns.

Use unions for optional duty attributes.

Apply nested unions for complex duty data.

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

#define MAX_STAFF 100

// Constants for shift timings
const char *SHIFT_1 = "08:00 AM - 04:00 PM";
const char *SHIFT_2 = "04:00 PM - 12:00 AM";
const char *SHIFT_3 = "12:00 AM - 08:00 AM";

// Structure for staff details
typedef struct {
    char name[50];
    char position[50];
    char contact[15];
} StaffDetails;

// Union for optional duty attributes
typedef union {
    char specialNotes[100];
    int overtimeHours;
} OptionalDutyAttributes;
```

```
// Nested union for complex duty data
typedef union {
    char shift[100];
    struct {
        char morningShift[50];
        char eveningShift[50];
    } splitShifts;
} ComplexDutyData;

// Structure for duty roster
typedef struct {
    int id;
    StaffDetails staff;
    ComplexDutyData duty;
    OptionalDutyAttributes optional;
    char dutyDate[15];
} DutyRoster;

// Array to store staff duty rosters
DutyRoster *rosters[MAX_STAFF];
int rosterCount = 0;

// Function prototypes
void addDutyRoster();
void viewDutyRoster();
void updateDutyRoster();
void deleteDutyRoster();
void listAllDutyRosters();
```

```
int main() {
    int choice;

    printf("----------Staff Duty Roster Management System----------\n");

    do {
        printf("\nMenu:\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 choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addDutyRoster();
                break;
            case 2:
                viewDutyRoster();
                break;
            case 3:
                updateDutyRoster();
                break;
            case 4:
                deleteDutyRoster();
```

```
            break;
        case 5:
            listAllDutyRosters();
            break;
        case 6:
            printf("Exiting the program. Goodbye!\n");
            break;
        default:
            printf("Invalid choice! Please try again.\n");
      }
   } while (choice != 6);

   return 0;
}

// Function to add duty roster
void addDutyRoster() {
   if (rosterCount >= MAX_STAFF) {
      printf("Roster capacity reached\n");
      return;
   }

   DutyRoster *newRoster = (DutyRoster *)malloc(sizeof(DutyRoster));

   printf("Enter Duty Roster ID: ");
   scanf("%d", &newRoster->id);
   printf("Enter Staff Name: ");
   scanf("%s", newRoster->staff.name);
   printf("Enter Staff Position: ");
```

```
scanf("%s", newRoster->staff.position);
printf("Enter Staff Contact: ");
scanf("%s", newRoster->staff.contact);
printf("Enter Duty Date (DD/MM/YYYY): ");
scanf("%s", newRoster->dutyDate);

// Assigning shift
printf("Enter Shift (1: Morning, 2: Evening, 3: Night, 4: Split): ");
int shiftChoice;
scanf("%d", &shiftChoice);

if (shiftChoice == 1) {
    strcpy(newRoster->duty.shift, SHIFT_1);
} else if (shiftChoice == 2) {
    strcpy(newRoster->duty.shift, SHIFT_2);
} else if (shiftChoice == 3) {
    strcpy(newRoster->duty.shift, SHIFT_3);
} else if (shiftChoice == 4) {
    printf("Enter Morning Shift Timings: ");
    scanf("%s", newRoster->duty.splitShifts.morningShift);
    printf("Enter Evening Shift Timings: ");
    scanf("%s", newRoster->duty.splitShifts.eveningShift);
} else {
    printf("Invalid choice for shift!\n");
    free(newRoster);
    return;
}

// Optional duty attributes
```

```
    printf("Enter Special Notes (optional): ");
    scanf("%s", newRoster->optional.specialNotes);

    // Storing the roster
    rosters[rosterCount++] = newRoster;
    printf("Duty roster added successfully!\n");
}

// Function to view duty roster
void viewDutyRoster() {
    int id;
    printf("Enter Duty Roster ID to view: ");
    scanf("%d", &id);

    for (int i = 0; i < rosterCount; i++) {
        if (rosters[i]->id == id) {
            printf("Duty Roster ID: %d\n", rosters[i]->id);
            printf("Staff Name: %s\n", rosters[i]->staff.name);
            printf("Staff Position: %s\n", rosters[i]->staff.position);
            printf("Staff Contact: %s\n", rosters[i]->staff.contact);
            printf("Duty Date: %s\n", rosters[i]->dutyDate);
            if (strcmp(rosters[i]->duty.shift, SHIFT_1) == 0 || strcmp(rosters[i]->duty.shift,
SHIFT_2) == 0 || strcmp(rosters[i]->duty.shift, SHIFT_3) == 0) {
                printf("Shift: %s\n", rosters[i]->duty.shift);
            } else {
                printf("Morning Shift: %s\n", rosters[i]->duty.splitShifts.morningShift);
                printf("Evening Shift: %s\n", rosters[i]->duty.splitShifts.eveningShift);
            }
            printf("Special Notes: %s\n", rosters[i]->optional.specialNotes);
```

```
            return;
        }
    }

    printf("Duty roster not found!\n");
}

// Function to update a duty roster
void updateDutyRoster() {
    int id;
    printf("Enter Duty Roster ID to update: ");
    scanf("%d", &id);

    for (int i = 0; i < rosterCount; i++) {
        if (rosters[i]->id == id) {
            printf("Enter New Staff Name: ");
            scanf("%s", rosters[i]->staff.name);
            printf("Enter New Staff Position: ");
            scanf("%s", rosters[i]->staff.position);
            printf("Enter New Staff Contact: ");
            scanf("%s", rosters[i]->staff.contact);
            printf("Enter New Duty Date (DD/MM/YYYY): ");
            scanf("%s", rosters[i]->dutyDate);

            // Update shift
            printf("Enter New Shift (1: Morning, 2: Evening, 3: Night, 4: Split): ");
            int shiftChoice;
            scanf("%d", &shiftChoice);
```

```
            if (shiftChoice == 1) {
                strcpy(rosters[i]->duty.shift, SHIFT_1);
            } else if (shiftChoice == 2) {
                strcpy(rosters[i]->duty.shift, SHIFT_2);
            } else if (shiftChoice == 3) {
                strcpy(rosters[i]->duty.shift, SHIFT_3);
            } else if (shiftChoice == 4) {
                printf("Enter Morning Shift Timings: ");
                scanf("%s", rosters[i]->duty.splitShifts.morningShift);
                printf("Enter Evening Shift Timings: ");
                scanf("%s", rosters[i]->duty.splitShifts.eveningShift);
            }

            // Update optional duty attributes
            printf("Enter New Special Notes (optional): ");
            scanf("%s", rosters[i]->optional.specialNotes);

            printf("Duty roster updated successfully!\n");
            return;
        }
    }

    printf("Duty roster not found!\n");
}

// Function to delete a duty roster
void deleteDutyRoster() {
    int id;
    printf("Enter Duty Roster ID to delete: ");
```

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

    for (int i = 0; i < rosterCount; i++) {
        if (rosters[i]->id == id) {
            free(rosters[i]);
            for (int j = i; j < rosterCount - 1; j++) {
                rosters[j] = rosters[j + 1];
            }
            rosterCount--;
            printf("Duty roster deleted successfully!\n");
            return;
        }
    }

    printf("Duty roster not found!\n");
}

// Function to list all duty rosters
void listAllDutyRosters() {
    if (rosterCount == 0) {
        printf("No duty rosters found!\n");
        return;
    }

    printf("\nList of All Duty Rosters:\n");
    for (int i = 0; i < rosterCount; i++) {
        printf("ID: %d, Staff: %s, Duty Date: %s, Shift: %s\n",
               rosters[i]->id, rosters[i]->staff.name, rosters[i]->dutyDate, rosters[i]->duty.shift);
```

```
    }
}
```

Problem 7: Emergency Contact Management System

Description: Design a system to manage emergency contacts for patients.

Menu Options:

Add Emergency Contact

View Emergency Contact

Update Emergency Contact

Delete Emergency Contact

List All Emergency Contacts

Exit

Requirements:

Declare variables for contact details.

Use static and const for non-changing contact data.

Implement switch case for contact operations.

Utilize loops for contact handling.

Use pointers for dynamic memory allocation.

Create functions for managing contacts.

Use arrays for storing contacts.

Define structures for contact details.

Employ nested structures for detailed contact information.

Utilize unions for optional contact data.

Apply nested unions for complex contact entries.

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

```
#define MAX_CONTACTS 100

// Structure for patient details
typedef struct {
    char name[50];
    int age;
    char relationship[30];
} PatientDetails;

// Union for optional contact data
typedef union {
    char secondaryPhone[15];
    char email[50];
} OptionalContactData;

// Structure for emergency contact
typedef struct {
    int id;
    PatientDetails patient;
    char primaryPhone[15];
    OptionalContactData optional;
    char address[100];
} EmergencyContact;

// Array to store emergency contacts
EmergencyContact *contacts[MAX_CONTACTS];
int contactCount = 0;

// Function prototypes
```

```
void addEmergencyContact();
void viewEmergencyContact();
void updateEmergencyContact();
void deleteEmergencyContact();
void listAllEmergencyContacts();

int main() {
    int choice;

    do {
        printf("\n1. 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 choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: addEmergencyContact(); break;
            case 2: viewEmergencyContact(); break;
            case 3: updateEmergencyContact(); break;
            case 4: deleteEmergencyContact(); break;
            case 5: listAllEmergencyContacts(); break;
            case 6: printf("Exiting.\n"); break;
            default: printf("Invalid choice.\n");
        }
    } while (choice != 6);
```

```
    return 0;
}

void addEmergencyContact() {
    if (contactCount >= MAX_CONTACTS) {
        printf("No space for more contacts.\n");
        return;
    }

    EmergencyContact *newContact = (EmergencyContact
*)malloc(sizeof(EmergencyContact));

    printf("Enter Contact ID: ");
    scanf("%d", &newContact->id);
    printf("Enter Patient Name: ");
    scanf("%s", newContact->patient.name);
    printf("Enter Patient Age: ");
    scanf("%d", &newContact->patient.age);
    printf("Enter Relationship: ");
    scanf("%s", newContact->patient.relationship);
    printf("Enter Address: ");
    scanf(" %[^\n]%*c", newContact->address);
    printf("Enter Primary Phone: ");
    scanf("%s", newContact->primaryPhone);

    printf("Enter Secondary Contact Info (Phone/Email): ");
    char choice;
    scanf(" %c", &choice);
```

```
    if (choice == 'p' || choice == 'P') {
        printf("Enter Secondary Phone: ");
        scanf("%s", newContact->optional.secondaryPhone);
    } else {
        printf("Enter Email: ");
        scanf("%s", newContact->optional.email);
    }

    contacts[contactCount++] = newContact;
    printf("Emergency contact added.\n");
}

void viewEmergencyContact() {
    int id;
    printf("Enter Emergency Contact ID to view: ");
    scanf("%d", &id);

    for (int i = 0; i < contactCount; i++) {
        if (contacts[i]->id == id) {
            printf("ID: %d, Name: %s, Age: %d, Relationship: %s, Address: %s\n",
                   contacts[i]->id, contacts[i]->patient.name, contacts[i]->patient.age,
                   contacts[i]->patient.relationship, contacts[i]->address);
            printf("Primary Phone: %s\n", contacts[i]->primaryPhone);
            printf("Secondary Contact: ");
            if (contacts[i]->optional.secondaryPhone[0] != '\0') {
                printf("Phone: %s\n", contacts[i]->optional.secondaryPhone);
            } else {
                printf("Email: %s\n", contacts[i]->optional.email);
```

```
            }
            return;
        }
    }

    printf("Contact not found.\n");
}

void updateEmergencyContact() {
    int id;
    printf("Enter Emergency Contact ID to update: ");
    scanf("%d", &id);

    for (int i = 0; i < contactCount; i++) {
        if (contacts[i]->id == id) {
            printf("Enter New Name: ");
            scanf("%s", contacts[i]->patient.name);
            printf("Enter New Age: ");
            scanf("%d", &contacts[i]->patient.age);
            printf("Enter New Relationship: ");
            scanf("%s", contacts[i]->patient.relationship);
            printf("Enter New Address: ");
            scanf(" %[^\n]%*c", contacts[i]->address);
            printf("Enter New Primary Phone: ");
            scanf("%s", contacts[i]->primaryPhone);

            printf("Enter New Secondary Contact Info (Phone/Email): ");
            char choice;
            scanf(" %c", &choice);
```

```
            if (choice == 'p' || choice == 'P') {
                printf("Enter New Secondary Phone: ");
                scanf("%s", contacts[i]->optional.secondaryPhone);
            } else {
                printf("Enter New Email: ");
                scanf("%s", contacts[i]->optional.email);
            }

            printf("Emergency contact updated.\n");
            return;
        }
    }

    printf("Contact not found.\n");
}

void deleteEmergencyContact() {
    int id;
    printf("Enter Emergency Contact ID to delete: ");
    scanf("%d", &id);

    for (int i = 0; i < contactCount; i++) {
        if (contacts[i]->id == id) {
            free(contacts[i]);
            for (int j = i; j < contactCount - 1; j++) {
                contacts[j] = contacts[j + 1];
            }
            contactCount--;
```

```
            printf("Emergency contact deleted.\n");
            return;
        }
    }

    printf("Contact not found.\n");
}

void listAllEmergencyContacts() {
    if (contactCount == 0) {
        printf("No contacts available.\n");
        return;
    }

    printf("\nList of Emergency Contacts:\n");
    for (int i = 0; i < contactCount; i++) {
        printf("ID: %d, Name: %s, Relationship: %s, Primary Phone: %s\n",
               contacts[i]->id, contacts[i]->patient.name, contacts[i]->patient.relationship,
contacts[i]->primaryPhone);
    }
}
```

Problem 8: Medical Record Update System

Description: Create a system for updating patient medical records.

Menu Options:

Add Medical Record

View Medical Record

Update Medical Record

Delete Medical Record

List All Medical Records

Exit

Requirements:

Use variables for record details.

Apply static and const for immutable data like record ID.

Implement switch case for update operations.

Utilize loops for record updating.

Use pointers for handling records.

Create functions for record management.

Use arrays for storing records.

Define structures for record details.

Employ nested structures for detailed medical history.

Utilize unions for optional record fields.

Apply nested unions for complex record data.

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

#define MAX_RECORDS 100

// Structure for patient details
typedef struct {
    char name[50];
    int age;
    char gender[10];
} PatientDetails;

// Union for optional medical record data
```

```
typedef union {
    char allergies[100];
    char currentMedications[100];
} OptionalMedicalData;

// Nested structure for medical history
typedef struct {
    char disease[50];
    int yearDiagnosed;
} MedicalHistory;

// Structure for medical record
typedef struct {
    const int recordID;
    PatientDetails patient;
    OptionalMedicalData optional;
    MedicalHistory history[5];
    int historyCount;
} MedicalRecord;

// Array to store medical records
MedicalRecord *records[MAX_RECORDS];
int recordCount = 0;

// Function prototypes
void addMedicalRecord();
void viewMedicalRecord();
void updateMedicalRecord();
void deleteMedicalRecord();
```

```
void listAllMedicalRecords();

int main() {
    int choice;

    do {
        printf("\n1. 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 choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: addMedicalRecord(); break;
            case 2: viewMedicalRecord(); break;
            case 3: updateMedicalRecord(); break;
            case 4: deleteMedicalRecord(); break;
            case 5: listAllMedicalRecords(); break;
            case 6: printf("Exiting.\n"); break;
            default: printf("Invalid choice.\n");
        }
    } while (choice != 6);

    return 0;
}
```

```
void addMedicalRecord() {
    if (recordCount >= MAX_RECORDS) {
        printf("No space for more records.\n");
        return;
    }

    MedicalRecord *newRecord = (MedicalRecord *)malloc(sizeof(MedicalRecord));

    printf("Enter Patient Name: ");
    scanf("%s", newRecord->patient.name);
    printf("Enter Patient Age: ");
    scanf("%d", &newRecord->patient.age);
    printf("Enter Patient Gender: ");
    scanf("%s", newRecord->patient.gender);

    printf("Enter Allergies (or press enter to skip): ");
    scanf("%s", newRecord->optional.allergies);

    newRecord->historyCount = 0;  // Initialize history count
    records[recordCount++] = newRecord;
    printf("Medical record added successfully.\n");
}

void viewMedicalRecord() {
    int id;
    printf("Enter Medical Record ID to view: ");
    scanf("%d", &id);

    if (id <= 0 || id > recordCount) {
```

```
        printf("Record not found.\n");
        return;
    }

    MedicalRecord *record = records[id - 1];
    printf("Record ID: %d\n", record->recordID);
    printf("Patient Name: %s\n", record->patient.name);
    printf("Age: %d\n", record->patient.age);
    printf("Gender: %s\n", record->patient.gender);
    printf("Allergies: %s\n", record->optional.allergies[0] ? record->optional.allergies :
"None");

    if (record->historyCount > 0) {
        printf("Medical History:\n");
        for (int i = 0; i < record->historyCount; i++) {
            printf("  Disease: %s, Year Diagnosed: %d\n", record->history[i].disease,
record->history[i].yearDiagnosed);
        }
    } else {
        printf("No medical history available.\n");
    }
}

void updateMedicalRecord() {
    int id;
    printf("Enter Medical Record ID to update: ");
    scanf("%d", &id);

    if (id <= 0 || id > recordCount) {
        printf("Record not found.\n");
```

```
        return;
    }

    MedicalRecord *record = records[id - 1];
    printf("Updating record for %s\n", record->patient.name);

    printf("Enter New Age: ");
    scanf("%d", &record->patient.age);
    printf("Enter New Gender: ");
    scanf("%s", record->patient.gender);

    printf("Enter New Allergies (or press enter to skip): ");
    scanf("%s", record->optional.allergies);

    printf("Enter a new disease history (or press enter to skip): ");
    char disease[50];
    scanf("%s", disease);
    if (disease[0] != '\0') {
        strcpy(record->history[record->historyCount].disease, disease);
        printf("Enter the year diagnosed: ");
        scanf("%d", &record->history[record->historyCount].yearDiagnosed);
        record->historyCount++;
    }

    printf("Medical record updated successfully.\n");
}

void deleteMedicalRecord() {
    int id;
```

```
    printf("Enter Medical Record ID to delete: ");
    scanf("%d", &id);

    if (id <= 0 || id > recordCount) {
        printf("Record not found.\n");
        return;
    }

    free(records[id - 1]);
    for (int i = id - 1; i < recordCount - 1; i++) {
        records[i] = records[i + 1];
    }
    recordCount--;
    printf("Medical record deleted successfully.\n");
}

void listAllMedicalRecords() {
    if (recordCount == 0) {
        printf("No medical records available.\n");
        return;
    }

    printf("\nList of Medical Records:\n");
    for (int i = 0; i < recordCount; i++) {
        printf("Record ID: %d, Name: %s, Age: %d, Gender: %s\n",
               records[i]->recordID, records[i]->patient.name,
               records[i]->patient.age, records[i]->patient.gender);
    }
}
```

Problem 9: Patient Diet Plan Management

Description: Develop a system to manage diet plans for patients.

Menu Options:

Add Diet Plan

View Diet Plan

Update Diet Plan

Delete Diet Plan

List All Diet Plans

Exit

Requirements:

Declare variables for diet plan details.

Use static and const for fixed dietary guidelines.

Implement switch case for diet plan operations.

Utilize loops for diet plan handling.

Use pointers for dynamic diet data.

Create functions for diet plan management.

Use arrays for storing diet plans.

Define structures for diet plan details.

Employ nested structures for detailed dietary breakdowns.

Use unions for optional diet attributes.

Apply nested unions for complex diet plan data.

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

#define MAX_DIET_PLANS 100
```

```
// Define structure for meal details
typedef struct {
    char mealName[50];
    int calories;
    int servings;
} MealDetails;

// Union for optional diet attributes (e.g., dietary restrictions)
typedef union {
    char dietaryRestrictions[100];  // e.g., gluten-free, vegan
    int preferredCalories;          // for those who need a specific calorie count
} OptionalDietAttributes;

// Structure for diet plan
typedef struct {
    const int planID;  // Immutable plan ID
    char patientName[50];
    MealDetails meals[7];  // Weekly meal plan, 7 days a week
    OptionalDietAttributes optionalAttributes;
} DietPlan;

// Array to store diet plans
DietPlan *dietPlans[MAX_DIET_PLANS];
int dietPlanCount = 0;

// Function prototypes
void addDietPlan();
void viewDietPlan();
void updateDietPlan();
```

```
void deleteDietPlan();
void listAllDietPlans();

int main() {
    int choice;

    do {
        printf("\n1. Add Diet Plan\n");
        printf("2. View Diet Plan\n");
        printf("3. Update Diet Plan\n");
        printf("4. Delete Diet Plan\n");
        printf("5. List All Diet Plans\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: addDietPlan(); break;
            case 2: viewDietPlan(); break;
            case 3: updateDietPlan(); break;
            case 4: deleteDietPlan(); break;
            case 5: listAllDietPlans(); break;
            case 6: printf("Exiting.\n"); break;
            default: printf("Invalid choice.\n");
        }
    } while (choice != 6);

    return 0;
}
```

```
void addDietPlan() {
    if (dietPlanCount >= MAX_DIET_PLANS) {
        printf("Diet Plan storage is full.\n");
        return;
    }

    DietPlan *newPlan = (DietPlan *)malloc(sizeof(DietPlan));

    printf("Enter Patient Name: ");
    scanf("%s", newPlan->patientName);

    for (int i = 0; i < 7; i++) {
        printf("Enter Meal for Day %d:\n", i + 1);
        printf("Meal Name: ");
        scanf("%s", newPlan->meals[i].mealName);
        printf("Calories: ");
        scanf("%d", &newPlan->meals[i].calories);
        printf("Servings: ");
        scanf("%d", &newPlan->meals[i].servings);
    }

    printf("Enter Dietary Restrictions (or None): ");
    scanf("%s", newPlan->optionalAttributes.dietaryRestrictions);

    dietPlans[dietPlanCount++] = newPlan;
    printf("Diet plan added successfully.\n");
}
```

```
void viewDietPlan() {
    int id;
    printf("Enter Diet Plan ID to view: ");
    scanf("%d", &id);

    if (id <= 0 || id > dietPlanCount) {
        printf("Diet Plan not found.\n");
        return;
    }

    DietPlan *plan = dietPlans[id - 1];
    printf("Diet Plan ID: %d\n", plan->planID);
    printf("Patient Name: %s\n", plan->patientName);

    printf("Meals for the Week:\n");
    for (int i = 0; i < 7; i++) {
        printf("Day %d: %s | Calories: %d | Servings: %d\n",
               i + 1, plan->meals[i].mealName, plan->meals[i].calories, plan-
>meals[i].servings);
    }

    printf("Dietary Restrictions: %s\n", plan->optionalAttributes.dietaryRestrictions[0] ?
plan->optionalAttributes.dietaryRestrictions : "None");
}

void updateDietPlan() {
    int id;
    printf("Enter Diet Plan ID to update: ");
    scanf("%d", &id);
```

```
    if (id <= 0 || id > dietPlanCount) {
        printf("Diet Plan not found.\n");
        return;
    }

    DietPlan *plan = dietPlans[id - 1];
    printf("Updating diet plan for %s\n", plan->patientName);

    for (int i = 0; i < 7; i++) {
        printf("Enter New Meal for Day %d:\n", i + 1);
        printf("Meal Name: ");
        scanf("%s", plan->meals[i].mealName);
        printf("Calories: ");
        scanf("%d", &plan->meals[i].calories);
        printf("Servings: ");
        scanf("%d", &plan->meals[i].servings);
    }

    printf("Enter New Dietary Restrictions (or None): ");
    scanf("%s", plan->optionalAttributes.dietaryRestrictions);

    printf("Diet plan updated successfully.\n");
}

void deleteDietPlan() {
    int id;
    printf("Enter Diet Plan ID to delete: ");
    scanf("%d", &id);
```

```
    if (id <= 0 || id > dietPlanCount) {
        printf("Diet Plan not found.\n");
        return;
    }

    free(dietPlans[id - 1]);
    for (int i = id - 1; i < dietPlanCount - 1; i++) {
        dietPlans[i] = dietPlans[i + 1];
    }
    dietPlanCount--;
    printf("Diet plan deleted successfully.\n");
}

void listAllDietPlans() {
    if (dietPlanCount == 0) {
        printf("No diet plans available.\n");
        return;
    }

    printf("\nList of Diet Plans:\n");
    for (int i = 0; i < dietPlanCount; i++) {
        printf("Diet Plan ID: %d, Patient Name: %s\n", dietPlans[i]->planID, dietPlans[i]-
>patientName);
    }
}
```

Problem 10: Surgery Scheduling System

Description: Design a system for scheduling surgeries.

Menu Options:

Schedule Surgery

View Surgery Schedule

Update Surgery Schedule

Cancel Surgery

List All Surgeries

Exit

Requirements:

Use variables for surgery details.

Apply static and const for immutable data like surgery types.

Implement switch case for scheduling operations.

Utilize loops for surgery scheduling.

Use pointers for handling surgery data.

Create functions for surgery management.

Use arrays for storing surgery schedules.

Define structures for surgery details.

Employ nested structures for detailed surgery information.

Utilize unions for optional surgery data.

Apply nested unions for complex surgery entries.

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

#define MAX_SURGERIES 100

// Define structure for surgery details
typedef struct {
    char patientName[50];
    char surgeryType[30];
```

```
    char surgeryDate[15];
    char surgeonName[50];
    char operationRoom[10];
} SurgeryDetails;

// Union for optional surgery data
typedef union {
    char anesthesiaType[30];
    int surgeryDuration;
} OptionalSurgeryAttributes;

// Structure for surgery schedule
typedef struct {
    const int surgeryID;
    SurgeryDetails details;
    OptionalSurgeryAttributes optionalAttributes;
} SurgerySchedule;

// Array to store surgery schedules
SurgerySchedule *surgerySchedules[MAX_SURGERIES];
int surgeryCount = 0;

// Function prototypes
void scheduleSurgery();
void viewSurgerySchedule();
void updateSurgerySchedule();
void cancelSurgery();
void listAllSurgeries();
```

```
int main() {
    int choice;

    do {
        printf("\n1. 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 choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: scheduleSurgery(); break;
            case 2: viewSurgerySchedule(); break;
            case 3: updateSurgerySchedule(); break;
            case 4: cancelSurgery(); break;
            case 5: listAllSurgeries(); break;
            case 6: printf("Exiting.\n"); break;
            default: printf("Invalid choice.\n");
        }
    } while (choice != 6);

    return 0;
}

void scheduleSurgery() {
    if (surgeryCount >= MAX_SURGERIES) {
```

```
        printf("Surgery schedule is full.\n");
        return;
    }

    SurgerySchedule *newSurgery = (SurgerySchedule
*)malloc(sizeof(SurgerySchedule));

    printf("Enter Patient Name: ");
    scanf("%s", newSurgery->details.patientName);
    printf("Enter Surgery Type: ");
    scanf("%s", newSurgery->details.surgeryType);
    printf("Enter Surgery Date (YYYY-MM-DD): ");
    scanf("%s", newSurgery->details.surgeryDate);
    printf("Enter Surgeon Name: ");
    scanf("%s", newSurgery->details.surgeonName);
    printf("Enter Operation Room: ");
    scanf("%s", newSurgery->details.operationRoom);

    printf("Enter Anesthesia Type (General or Local): ");
    scanf("%s", newSurgery->optionalAttributes.anesthesiaType);

    surgerySchedules[surgeryCount++] = newSurgery;
    printf("Surgery scheduled successfully.\n");
}

void viewSurgerySchedule() {
    int id;
    printf("Enter Surgery ID to view details: ");
    scanf("%d", &id);
```

```
    if (id <= 0 || id > surgeryCount) {
        printf("Surgery not found.\n");
        return;
    }

    SurgerySchedule *schedule = surgerySchedules[id - 1];
    printf("Surgery ID: %d\n", schedule->surgeryID);
    printf("Patient Name: %s\n", schedule->details.patientName);
    printf("Surgery Type: %s\n", schedule->details.surgeryType);
    printf("Surgery Date: %s\n", schedule->details.surgeryDate);
    printf("Surgeon Name: %s\n", schedule->details.surgeonName);
    printf("Operation Room: %s\n", schedule->details.operationRoom);
    printf("Anesthesia Type: %s\n", schedule->optionalAttributes.anesthesiaType);
}

void updateSurgerySchedule() {
    int id;
    printf("Enter Surgery ID to update: ");
    scanf("%d", &id);

    if (id <= 0 || id > surgeryCount) {
        printf("Surgery not found.\n");
        return;
    }

    SurgerySchedule *schedule = surgerySchedules[id - 1];
    printf("Updating surgery schedule for patient: %s\n", schedule-
>details.patientName);
```

```
    printf("Enter New Surgery Type: ");
    scanf("%s", schedule->details.surgeryType);
    printf("Enter New Surgery Date: ");
    scanf("%s", schedule->details.surgeryDate);
    printf("Enter New Surgeon Name: ");
    scanf("%s", schedule->details.surgeonName);
    printf("Enter New Operation Room: ");
    scanf("%s", schedule->details.operationRoom);

    printf("Enter New Anesthesia Type: ");
    scanf("%s", schedule->optionalAttributes.anesthesiaType);

    printf("Surgery schedule updated successfully.\n");
}

void cancelSurgery() {
    int id;
    printf("Enter Surgery ID to cancel: ");
    scanf("%d", &id);

    if (id <= 0 || id > surgeryCount) {
        printf("Surgery not found.\n");
        return;
    }

    free(surgerySchedules[id - 1]);
    for (int i = id - 1; i < surgeryCount - 1; i++) {
        surgerySchedules[i] = surgerySchedules[i + 1];
```

```
    }
    surgeryCount--;
    printf("Surgery cancelled successfully.\n");
}

void listAllSurgeries() {
    if (surgeryCount == 0) {
        printf("No surgeries scheduled.\n");
        return;
    }

    printf("\nList of Scheduled Surgeries:\n");
    for (int i = 0; i < surgeryCount; i++) {
        printf("Surgery ID: %d, Patient Name: %s, Surgery Type: %s, Surgery Date:
%s\n",
               surgerySchedules[i]->surgeryID, surgerySchedules[i]->details.patientName,
               surgerySchedules[i]->details.surgeryType, surgerySchedules[i]-
>details.surgeryDate);
    }
}
```

Problem 11: Prescription Management System

Description: Develop a system to manage patient prescriptions.

Menu Options:

Add Prescription

View Prescription

Update Prescription

Delete Prescription

List All Prescriptions

Exit

Requirements:

Declare variables for prescription details.

Use static and const for fixed prescription guidelines.

Implement switch case for prescription operations.

Utilize loops for prescription handling.

Use pointers for dynamic prescription data.

Create functions for prescription management.

Use arrays for storing prescriptions.

Define structures for prescription details.

Employ nested structures for detailed prescription information.

Use unions for optional prescription fields.

Apply nested unions for complex prescription data.

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

#define MAX_PRESCRIPTIONS 100

// Structure for prescription details
typedef struct {
    char patientName[50];
    char medicineName[50];
    char dosage[20];
    char frequency[20];
    char startDate[15];
    char endDate[15];
} PrescriptionDetails;
```

```
// Union for optional prescription data
typedef union {
    char additionalNotes[100];
    int refillCount;
} OptionalPrescriptionAttributes;

// Structure for prescription schedule
typedef struct {
    const int prescriptionID;
    PrescriptionDetails details;
    OptionalPrescriptionAttributes optionalAttributes;
} PrescriptionSchedule;

// Array to store prescription schedules
PrescriptionSchedule *prescriptions[MAX_PRESCRIPTIONS];
int prescriptionCount = 0;

// Function prototypes
void addPrescription();
void viewPrescription();
void updatePrescription();
void deletePrescription();
void listAllPrescriptions();

int main() {
    int choice;

    do {
        printf("\n1. Add Prescription\n");
```

```
        printf("2. View Prescription\n");
        printf("3. Update Prescription\n");
        printf("4. Delete Prescription\n");
        printf("5. List All Prescriptions\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: addPrescription(); break;
            case 2: viewPrescription(); break;
            case 3: updatePrescription(); break;
            case 4: deletePrescription(); break;
            case 5: listAllPrescriptions(); break;
            case 6: printf("Exiting.\n"); break;
            default: printf("Invalid choice.\n");
        }
    } while (choice != 6);

    return 0;
}

void addPrescription() {
    if (prescriptionCount >= MAX_PRESCRIPTIONS) {
        printf("Prescription storage is full.\n");
        return;
    }
```

```
    PrescriptionSchedule *newPrescription = (PrescriptionSchedule
*)malloc(sizeof(PrescriptionSchedule));



    printf("Enter Patient Name: ");
    scanf("%s", newPrescription->details.patientName);
    printf("Enter Medicine Name: ");
    scanf("%s", newPrescription->details.medicineName);
    printf("Enter Dosage (in mg): ");
    scanf("%s", newPrescription->details.dosage);
    printf("Enter Frequency: ");
    scanf("%s", newPrescription->details.frequency);
    printf("Enter Start Date (YYYY-MM-DD): ");
    scanf("%s", newPrescription->details.startDate);
    printf("Enter End Date (YYYY-MM-DD): ");
    scanf("%s", newPrescription->details.endDate);

    printf("Enter Additional Notes (optional): ");
    scanf("%s", newPrescription->optionalAttributes.additionalNotes);

    prescriptions[prescriptionCount++] = newPrescription;
    printf("Prescription added successfully.\n");
}

void viewPrescription() {
    int id;
    printf("Enter Prescription ID to view details: ");
    scanf("%d", &id);
```

```
    if (id <= 0 || id > prescriptionCount) {
        printf("Prescription not found.\n");
        return;
    }

    PrescriptionSchedule *schedule = prescriptions[id - 1];
    printf("Prescription ID: %d\n", schedule->prescriptionID);
    printf("Patient Name: %s\n", schedule->details.patientName);
    printf("Medicine Name: %s\n", schedule->details.medicineName);
    printf("Dosage: %s\n", schedule->details.dosage);
    printf("Frequency: %s\n", schedule->details.frequency);
    printf("Start Date: %s\n", schedule->details.startDate);
    printf("End Date: %s\n", schedule->details.endDate);
    printf("Additional Notes: %s\n", schedule->optionalAttributes.additionalNotes);
}

void updatePrescription() {
    int id;
    printf("Enter Prescription ID to update: ");
    scanf("%d", &id);

    if (id <= 0 || id > prescriptionCount) {
        printf("Prescription not found.\n");
        return;
    }

    PrescriptionSchedule *schedule = prescriptions[id - 1];
    printf("Updating prescription for patient: %s\n", schedule->details.patientName);
```

```
    printf("Enter New Medicine Name: ");
    scanf("%s", schedule->details.medicineName);
    printf("Enter New Dosage: ");
    scanf("%s", schedule->details.dosage);
    printf("Enter New Frequency: ");
    scanf("%s", schedule->details.frequency);
    printf("Enter New Start Date: ");
    scanf("%s", schedule->details.startDate);
    printf("Enter New End Date: ");
    scanf("%s", schedule->details.endDate);

    printf("Enter New Additional Notes (optional): ");
    scanf("%s", schedule->optionalAttributes.additionalNotes);

    printf("Prescription updated successfully.\n");
}

void deletePrescription() {
    int id;
    printf("Enter Prescription ID to delete: ");
    scanf("%d", &id);

    if (id <= 0 || id > prescriptionCount) {
        printf("Prescription not found.\n");
        return;
    }

    free(prescriptions[id - 1]);
    for (int i = id - 1; i < prescriptionCount - 1; i++) {
```

```
        prescriptions[i] = prescriptions[i + 1];
    }
    prescriptionCount--;
    printf("Prescription deleted successfully.\n");
}

void listAllPrescriptions() {
    if (prescriptionCount == 0) {
        printf("No prescriptions found.\n");
        return;
    }

    printf("\nList of All Prescriptions:\n");
    for (int i = 0; i < prescriptionCount; i++) {
        printf("Prescription ID: %d, Patient Name: %s, Medicine: %s, Dosage: %s,
Frequency: %s\n",
               prescriptions[i]->prescriptionID, prescriptions[i]->details.patientName,
               prescriptions[i]->details.medicineName, prescriptions[i]->details.dosage,
prescriptions[i]->details.frequency);
    }
}
```

Problem 12: Doctor Consultation Management

Description: Create a system for managing doctor consultations.

Menu Options:

Schedule Consultation

View Consultation

Update Consultation

Cancel Consultation

List All Consultations

Exit

Requirements:

Use variables for consultation details.

Apply static and const for non-changing data like consultation fees.

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

#define MAX_CONSULTATIONS 100

typedef struct {
    int id;
    char patientName[50];
    char doctorName[50];
    char date[15];
    char time[10];
} Consultation;

Consultation consultations[MAX_CONSULTATIONS];
int consultationCount = 0;

void scheduleConsultation() {
    if (consultationCount >= MAX_CONSULTATIONS) {
        printf("Max consultations reached!\n");
        return;
    }

    Consultation *c = &consultations[consultationCount];
```

```
    c->id = consultationCount + 1;
    printf("Enter patient name: ");
    scanf("%s", c->patientName);
    printf("Enter doctor name: ");
    scanf("%s", c->doctorName);
    printf("Enter date (DD/MM/YYYY): ");
    scanf("%s", c->date);
    printf("Enter time (HH:MM): ");
    scanf("%s", c->time);

    consultationCount++;
    printf("Consultation scheduled successfully!\n");
}

void viewConsultation() {
    int id;
    printf("Enter consultation ID to view: ");
    scanf("%d", &id);

    for (int i = 0; i < consultationCount; i++) {
        if (consultations[i].id == id) {
            printf("ID: %d, Patient: %s, Doctor: %s, Date: %s, Time: %s\n",
                   consultations[i].id, consultations[i].patientName,
                   consultations[i].doctorName, consultations[i].date, consultations[i].time);
            return;
        }
    }
    printf("Consultation not found!\n");
}
```

```
void updateConsultation() {
    int id;
    printf("Enter consultation ID to update: ");
    scanf("%d", &id);

    for (int i = 0; i < consultationCount; i++) {
        if (consultations[i].id == id) {
            printf("Enter new patient name: ");
            scanf("%s", consultations[i].patientName);
            printf("Enter new doctor name: ");
            scanf("%s", consultations[i].doctorName);
            printf("Enter new date (DD/MM/YYYY): ");
            scanf("%s", consultations[i].date);
            printf("Enter new time (HH:MM): ");
            scanf("%s", consultations[i].time);

            printf("Consultation updated successfully!\n");
            return;
        }
    }
    printf("Consultation not found!\n");
}

void cancelConsultation() {
    int id;
    printf("Enter consultation ID to cancel: ");
    scanf("%d", &id);
```

```
    for (int i = 0; i < consultationCount; i++) {
        if (consultations[i].id == id) {
            for (int j = i; j < consultationCount - 1; j++) {
                consultations[j] = consultations[j + 1];
            }
            consultationCount--;
            printf("Consultation canceled successfully!\n");
            return;
        }
    }
    printf("Consultation not found!\n");
}

void listAllConsultations() {
    if (consultationCount == 0) {
        printf("No consultations scheduled!\n");
        return;
    }

    for (int i = 0; i < consultationCount; i++) {
        printf("ID: %d, Patient: %s, Doctor: %s, Date: %s, Time: %s\n",
               consultations[i].id, consultations[i].patientName,
               consultations[i].doctorName, consultations[i].date, consultations[i].time);
    }
}

int main() {
    int choice;
```

```
    while (1) {
        printf("\nDoctor Consultation Management System\n");
        printf("1. Schedule Consultation\n");
        printf("2. View Consultation\n");
        printf("3. Update Consultation\n");
        printf("4. Cancel Consultation\n");
        printf("5. List All Consultations\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: scheduleConsultation(); break;
            case 2: viewConsultation(); break;
            case 3: updateConsultation(); break;
            case 4: cancelConsultation(); break;
            case 5: listAllConsultations(); break;
            case 6: printf("Goodbye!\n"); return 0;
            default: printf("Invalid choice! Try again.\n");
        }
    }
}
```

Linked List Programs

Problem 1: Patient Queue Management

Description: Implement a linked list to manage a queue of patients waiting for consultation. Operations:

Create a new patient queue.

Insert a patient into the queue.

Display the current queue of patients.

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

// Structure for a patient node
typedef struct Patient {
    int id;
    char name[50];
    struct Patient *next;
} Patient;

Patient *first = NULL, *last = NULL;

// Function to add a patient to the queue
void addPatient(int id, const char *name) {
    Patient *newPatient = (Patient *)malloc(sizeof(Patient));
    newPatient->id = id;
    strcpy(newPatient->name, name);
    newPatient->next = NULL;

    if (last == NULL) {
        first = last = newPatient;
    } else {
        last->next = newPatient;
        last = newPatient;
    }
```

```
    printf("Patient %s (ID: %d) added to the queue.\n", name, id);
}

// Function to display the queue
void displayQueue() {
    if (first == NULL) {
        printf("The queue is empty.\n");
        return;
    }

    printf("Patient Queue:\n");
    Patient *current = first;
    while (current != NULL) {
        printf("ID: %d, Name: %s\n", current->id, current->name);
        current = current->next;
    }
}

int main() {
    int choice, id;
    char name[50];

    while (1) {
        printf("\n1. Add Patient\n2. View Queue\n3. Exit\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter Patient ID: ");
```

```
            scanf("%d", &id);
            printf("Enter Patient Name: ");
            scanf("%s", name);
            addPatient(id, name);
            break;
        case 2:
            displayQueue();
            break;
        case 3:
            printf("Exiting the program.\n");
            return 0;
        default:
            printf("Invalid choice! Try again.\n");
      }
   }
}
```

Problem 2: Hospital Ward Allocation

Description: Use a linked list to allocate beds in a hospital ward. Operations:

Create a list of available beds.

Insert a patient into an available bed.

Display the current bed allocation.

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

// Structure for a bed in the ward
typedef struct Bed {
    int bedNumber;
```

```
    char patientName[50];
    struct Bed *next;
} Bed;

Bed *first = NULL, *last = NULL;

// Function to add a bed to the list
void addBed(int bedNumber) {
    Bed *newBed = (Bed *)malloc(sizeof(Bed));
    newBed->bedNumber = bedNumber;
    strcpy(newBed->patientName, "Available");
    newBed->next = NULL;

    if (first == NULL) {
        first = last = newBed;
    } else {
        last->next = newBed;
        last = newBed;
    }
}

// Function to allocate a bed to a patient
void allocateBed(const char *patientName) {
    Bed *current = first;
    while (current != NULL) {
        if (strcmp(current->patientName, "Available") == 0) {
            strcpy(current->patientName, patientName);
            printf("Bed %d allocated to %s.\n", current->bedNumber, patientName);
            return;
```

```
        }
        current = current->next;
    }
    printf("No available beds to allocate.\n");
}

// Function to display the current bed allocation
void displayBeds() {
    Bed *current = first;
    printf("Current Bed Allocation:\n");
    while (current != NULL) {
        printf("Bed %d: %s\n", current->bedNumber, current->patientName);
        current = current->next;
    }
}

int main() {
    int numBeds, choice;
    char patientName[50];

    printf("Enter the number of beds: ");
    scanf("%d", &numBeds);

    for (int i = 1; i <= numBeds; i++) {
        addBed(i);
    }

    while (1) {
        printf("\nMenu:\n1. Allocate Bed\n2. View Beds\n3. Exit\nEnter your choice: ");
```

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

        if (choice == 1) {
            printf("Enter Patient Name: ");
            scanf("%s", patientName);
            allocateBed(patientName);
        } else if (choice == 2) {
            displayBeds();
        } else if (choice == 3) {
            printf("Exiting the program.\n");
            break;
        } else {
            printf("Invalid choice! Try again.\n");
        }
    }

    return 0;
}
```

Problem 3: Medical Inventory Tracking

Description: Maintain a linked list to track inventory items in a medical store.
Operations:

Create an inventory list.

Insert a new inventory item.

Display the current inventory.

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

```
// Structure for an inventory item
typedef struct InventoryItem {
    int itemID;
    char itemName[50];
    int quantity;
    struct InventoryItem *next;
} InventoryItem;

InventoryItem *first = NULL, *last = NULL;

// Function to add a new inventory item
void addItem(int itemID, const char *itemName, int quantity) {
    InventoryItem *newItem = (InventoryItem *)malloc(sizeof(InventoryItem));
    newItem->itemID = itemID;
    strcpy(newItem->itemName, itemName);
    newItem->quantity = quantity;
    newItem->next = NULL;

    if (first == NULL) {
        first = last = newItem;
    } else {
        last->next = newItem;
        last = newItem;
    }
    printf("Item %d (%s) added with quantity %d.\n", itemID, itemName, quantity);
}

// Function to display the current inventory
```

```
void displayInventory() {
    if (first == NULL) {
        printf("No items in the inventory.\n");
        return;
    }

    InventoryItem *current = first;
    printf("Current Inventory:\n");
    printf("ID\tName\t\tQuantity\n");
    while (current != NULL) {
        printf("%d\t%s\t\t%d\n", current->itemID, current->itemName, current->quantity);
        current = current->next;
    }
}

int main() {
    int choice, itemID, quantity;
    char itemName[50];

    while (1) {
        printf("\nMenu:\n");
        printf("1. Add Inventory Item\n");
        printf("2. View Inventory\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            printf("Enter Item ID: ");
```

```
            scanf("%d", &itemID);
            printf("Enter Item Name: ");
            scanf("%s", itemName);
            printf("Enter Quantity: ");
            scanf("%d", &quantity);
            addItem(itemID, itemName, quantity);
        } else if (choice == 2) {
            displayInventory();
        } else if (choice == 3) {
            printf("Exiting the program.\n");
            break;
        } else {
            printf("Invalid choice! Try again.\n");
        }
    }

    return 0;
}
```

Problem 4: Doctor Appointment Scheduling

Description: Develop a linked list to schedule doctor appointments. Operations:

Create an appointment list.

Insert a new appointment.

Display all scheduled appointments.

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

```
// Structure for an appointment
typedef struct Appointment {
    int appointmentID;
    char patientName[50];
    char doctorName[50];
    struct Appointment *next;
} Appointment;

Appointment *first = NULL, *last = NULL;

// Function to add a new appointment
void addAppointment(int appointmentID, char *patientName, char *doctorName) {
    Appointment *newAppointment = (Appointment *)malloc(sizeof(Appointment));
    newAppointment->appointmentID = appointmentID;
    strcpy(newAppointment->patientName, patientName);
    strcpy(newAppointment->doctorName, doctorName);
    newAppointment->next = NULL;

    if (first == NULL) {
        first = last = newAppointment;
    } else {
        last->next = newAppointment;
        last = newAppointment;
    }
}

// Function to display all appointments
void displayAppointments() {
    Appointment *current = first;
```

```
    if (current == NULL) {
        printf("No appointments scheduled.\n");
        return;
    }

    printf("\nScheduled Appointments:\n");
    while (current != NULL) {
        printf("Appointment ID: %d, Patient: %s, Doctor: %s\n",
               current->appointmentID, current->patientName, current->doctorName);
        current = current->next;
    }
}

int main() {
    int choice, appointmentID;
    char patientName[50], doctorName[50];

    while (1) {
        printf("\nMenu:\n");
        printf("1. Schedule Appointment\n");
        printf("2. View All Appointments\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter Appointment ID: ");
                scanf("%d", &appointmentID);
```

```
            printf("Enter Patient Name: ");
            scanf("%s", patientName);
            printf("Enter Doctor Name: ");
            scanf("%s", doctorName);
            addAppointment(appointmentID, patientName, doctorName);
            break;
        case 2:
            displayAppointments();
            break;
        case 3:
            printf("Exiting the program.\n");
            return 0;
        default:
            printf("Invalid choice. Try again.\n");
    }
  }
}
```

Problem 5: Emergency Contact List

Description: Implement a linked list to manage emergency contacts for hospital staff. Operations:

Create a contact list.

Insert a new contact.

Display all emergency contacts.

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

```
// Structure for an emergency contact
typedef struct EmergencyContact {
    int contactID;
    char name[50];
    char relationship[50];
    char phoneNumber[20];
    struct EmergencyContact *next;
} EmergencyContact;

EmergencyContact *first = NULL, *last = NULL;

// Function to add a new emergency contact
void addEmergencyContact(int contactID, char *name, char *relationship, char
*phoneNumber) {
    EmergencyContact *newContact = (EmergencyContact
*)malloc(sizeof(EmergencyContact));
    newContact->contactID = contactID;
    strcpy(newContact->name, name);
    strcpy(newContact->relationship, relationship);
    strcpy(newContact->phoneNumber, phoneNumber);
    newContact->next = NULL;

    if (first == NULL) {
        first = last = newContact;
    } else {
        last->next = newContact;
        last = newContact;
    }
}
```

```
// Function to display all emergency contacts
void displayEmergencyContacts() {
    EmergencyContact *current = first;
    if (current == NULL) {
        printf("No emergency contacts found.\n");
        return;
    }

    printf("\nEmergency Contact List:\n");
    while (current != NULL) {
        printf("Contact ID: %d, Name: %s, Relationship: %s, Phone: %s\n",
               current->contactID, current->name, current->relationship, current->phoneNumber);
        current = current->next;
    }
}

int main() {
    int choice, contactID;
    char name[50], relationship[50], phoneNumber[20];

    while (1) {
        printf("\nMenu:\n");
        printf("1. Add Emergency Contact\n");
        printf("2. View All Emergency Contacts\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
        switch (choice) {
            case 1:
                printf("Enter Contact ID: ");
                scanf("%d", &contactID);
                printf("Enter Contact Name: ");
                scanf("%s", name);
                printf("Enter Relationship: ");
                scanf("%s", relationship);
                printf("Enter Phone Number: ");
                scanf("%s", phoneNumber);
                addEmergencyContact(contactID, name, relationship, phoneNumber);
                break;
            case 2:
                displayEmergencyContacts();
                break;
            case 3:
                printf("Exiting the program.\n");
                return 0;
            default:
                printf("Invalid choice. Try again.\n");
        }
    }
}
```

Problem 6: Surgery Scheduling System

Description: Use a linked list to manage surgery schedules. Operations:

Create a surgery schedule.

Insert a new surgery into the schedule.

Display all scheduled surgeries.

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

// Structure for a surgery
typedef struct Surgery {
    int surgeryID;
    char patientName[50];
    struct Surgery *next;
} Surgery;

Surgery *first = NULL, *last = NULL;

// Function to add a new surgery to the schedule
void addSurgery(int surgeryID, char *patientName) {
    Surgery *newSurgery = (Surgery *)malloc(sizeof(Surgery));
    newSurgery->surgeryID = surgeryID;
    strcpy(newSurgery->patientName, patientName);
    newSurgery->next = NULL;

    if (first == NULL) {
        first = last = newSurgery;
    } else {
        last->next = newSurgery;
        last = newSurgery;
    }
}
```

```
// Function to display all scheduled surgeries
void displaySurgeries() {
    Surgery *current = first;
    if (current == NULL) {
        printf("No surgeries scheduled.\n");
        return;
    }

    printf("\nScheduled Surgeries:\n");
    while (current != NULL) {
        printf("Surgery ID: %d, Patient Name: %s\n", current->surgeryID, current->patientName);
        current = current->next;
    }
}

int main() {
    int choice, surgeryID;
    char patientName[50];

    while (1) {
        printf("\nMenu:\n");
        printf("1. Schedule Surgery\n");
        printf("2. View All Scheduled Surgeries\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
```

```
            case 1:
                printf("Enter Surgery ID: ");
                scanf("%d", &surgeryID);
                printf("Enter Patient Name: ");
                scanf("%s", patientName);
                addSurgery(surgeryID, patientName);
                break;
            case 2:
                displaySurgeries();
                break;
            case 3:
                printf("Exiting the program.\n");
                return 0;
            default:
                printf("Invalid choice. Try again.\n");
        }
    }
}
```

Problem 7: Patient History Record

Description: Maintain a linked list to keep track of patient history records. Operations:

Create a history record list.

Insert a new record.

Display all patient history records.

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

```
// Structure for patient history record
typedef struct PatientHistory {
    int recordID;
    char patientName[50];
    struct PatientHistory *next;
} PatientHistory;

PatientHistory *first = NULL, *last = NULL;

// Function to add a new patient history record
void addHistoryRecord(int recordID, char *patientName) {
    PatientHistory *newRecord = (PatientHistory *)malloc(sizeof(PatientHistory));
    newRecord->recordID = recordID;
    strcpy(newRecord->patientName, patientName);
    newRecord->next = NULL;

    // If the list is empty, initialize both first and last
    if (first == NULL) {
        first = last = newRecord;
    } else {
        last->next = newRecord;
        last = newRecord;
    }
}

// Function to display all patient history records
void displayHistoryRecords() {
    PatientHistory *current = first;
    if (current == NULL) {
```

```
        printf("No patient history records found.\n");
        return;
    }

    printf("\nPatient History Records:\n");
    while (current != NULL) {
        printf("Record ID: %d, Patient Name: %s\n", current->recordID, current-
>patientName);
        current = current->next;
    }
}

int main() {
    int choice, recordID;
    char patientName[50];

    while (1) {
        printf("\nMenu:\n");
        printf("1. Add Patient History Record\n");
        printf("2. View All Patient History Records\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter Record ID: ");
                scanf("%d", &recordID);
                printf("Enter Patient Name: ");
```

```
            scanf("%s", patientName);
            addHistoryRecord(recordID, patientName);
            break;
        case 2:
            displayHistoryRecords();
            break;
        case 3:
            printf("Exiting the program.\n");
            return 0;
        default:
            printf("Invalid choice. Try again.\n");
    }
  }
}
```

Problem 8: Medical Test Tracking

Description: Implement a linked list to track medical tests for patients. Operations:

Create a list of medical tests.

Insert a new test result.

Display all test results.

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

// Structure for medical test result
typedef struct MedicalTest {
    int testID;
    char testName[50];
```

```
    struct MedicalTest *next;
} MedicalTest;

// Pointers to the first and last test in the list
MedicalTest *first = NULL, *last = NULL;

// Function to add a new test result
void addTestResult(int testID, char *testName) {
    MedicalTest *newTest = (MedicalTest *)malloc(sizeof(MedicalTest));
    newTest->testID = testID;
    strcpy(newTest->testName, testName);
    newTest->next = NULL;

    // If the list is empty, initialize both first and last
    if (first == NULL) {
        first = last = newTest;
    } else {
        last->next = newTest;
        last = newTest;
    }
}

// Function to display all test results
void displayTestResults() {
    MedicalTest *current = first;
    if (current == NULL) {
        printf("No test results found.\n");
        return;
    }
```

```
    printf("\nMedical Test Results:\n");
    while (current != NULL) {
        printf("Test ID: %d, Test Name: %s\n", current->testID, current->testName);
        current = current->next;
    }
}

int main() {
    int choice, testID;
    char testName[50];

    while (1) {
        printf("\nMenu:\n");
        printf("1. Add Medical Test Result\n");
        printf("2. View All Medical Test Results\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter Test ID: ");
                scanf("%d", &testID);
                printf("Enter Test Name: ");
                scanf("%s", testName);
                addTestResult(testID, testName);
                break;
            case 2:
```

```
            displayTestResults();
            break;
        case 3:
            printf("Exiting the program.\n");
            return 0;
        default:
            printf("Invalid choice. Try again.\n");
      }
   }
}
```

Problem 9: Prescription Management System

Description: Use a linked list to manage patient prescriptions. Operations:

Create a prescription list.

Insert a new prescription.

Display all prescriptions.

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

// Structure for prescription details
typedef struct Prescription {
    int prescriptionID;
    char medicineName[50];
    struct Prescription *next;
} Prescription;

// Pointers to the first and last prescription
```

```
Prescription *first = NULL, *last = NULL;

// Function to add a prescription
void addPrescription() {
    Prescription *newPrescription = (Prescription *)malloc(sizeof(Prescription));

    // Getting input from the user
    printf("Enter Prescription ID: ");
    scanf("%d", &newPrescription->prescriptionID);
    printf("Enter Medicine Name: ");
    scanf("%s", newPrescription->medicineName);

    newPrescription->next = NULL;

    // If the list is empty, make both first and last point to the new node
    if (first == NULL) {
        first = last = newPrescription;
    } else {
        last->next = newPrescription;
        last = newPrescription;
    }
}

// Function to display all prescriptions
void displayPrescriptions() {
    Prescription *current = first;
    if (current == NULL) {
        printf("No prescriptions to display.\n");
        return;
```

```
    }

    printf("\nPrescription List:\n");
    while (current != NULL) {
        printf("Prescription ID: %d, Medicine: %s\n", current->prescriptionID, current->medicineName);
        current = current->next;
    }
}

int main() {
    int choice;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Add Prescription\n");
        printf("2. View All Prescriptions\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            addPrescription();
        } else if (choice == 2) {
            displayPrescriptions();
        } else if (choice == 3) {
            printf("Exiting the program.\n");
            break;
        } else {
```

```
            printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

Problem 10: Hospital Staff Roster

Description: Develop a linked list to manage the hospital staff roster. Operations:

Create a staff roster.

Insert a new staff member into the roster.

Display the current staff roster.

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

// Structure for staff details
typedef struct Staff {
    int staffID;
    char name[50];
    char role[50];
    struct Staff *next;
} Staff;

// Pointers to the first and last staff member
Staff *first = NULL, *last = NULL;

// Function to insert a new staff member into the roster
```

```
void addStaff() {
    Staff *newStaff = (Staff *)malloc(sizeof(Staff));

    // Getting input from the user
    printf("Enter Staff ID: ");
    scanf("%d", &newStaff->staffID);
    printf("Enter Staff Name: ");
    scanf("%s", newStaff->name);
    printf("Enter Staff Role: ");
    scanf("%s", newStaff->role);

    newStaff->next = NULL;

    // If the roster is empty, make both first and last point to the new node
    if (first == NULL) {
        first = last = newStaff;
    } else {
        last->next = newStaff;
        last = newStaff;
    }
}

// Function to display the current staff roster
void displayRoster() {
    Staff *current = first;
    if (current == NULL) {
        printf("No staff members in the roster.\n");
        return;
    }
```

```
    printf("\nStaff Roster:\n");
    while (current != NULL) {
        printf("Staff ID: %d, Name: %s, Role: %s\n", current->staffID, current->name,
current->role);
        current = current->next;
    }
}

int main() {
    int choice;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Add Staff Member\n");
        printf("2. View Staff Roster\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            addStaff();
        } else if (choice == 2) {
            displayRoster();
        } else if (choice == 3) {
            printf("Exiting the program.\n");
            break;
        } else {
            printf("Invalid choice. Please try again.\n");
```

```
        }
    }

    return 0;
}
```