SET OF PROBLEMS
==========

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>

// Defining constants for the hospital name
#define HOSPITAL_NAME "City General Hospital"

// Defining maximum size for the name, medical condition, and medication
#define MAX_NAME_LENGTH 50
#define MAX_CONDITION_LENGTH 100
#define MAX_MEDICATION_LENGTH 100
#define MAX_PATIENTS 100

// Structure for storing patient information
struct Medication {
    char name[MAX_MEDICATION_LENGTH];
    int dosage; // In mg
};

struct MedicalHistory {
    char condition[MAX_CONDITION_LENGTH];
    int yearDiagnosed;
    struct Medication currentMedication;
};

struct Patient {
    int patientID;
    char name[MAX_NAME_LENGTH];
```

```
    int age;
    struct MedicalHistory medicalHistory;
    char contactNumber[15];
    union {
        char emergencyContactName[MAX_NAME_LENGTH];
        char additionalNotes[MAX_CONDITION_LENGTH];
    };
};

// Global variables
static const char* hospitalName = HOSPITAL_NAME;
struct Patient *patients[MAX_PATIENTS]; // Array of pointers to store patient records
int patientCount = 0;

// Function Prototypes
void addNewPatient();
void viewPatientDetails(int patientID);
void updatePatientInformation(int patientID);
void deletePatientRecord(int patientID);
void listAllPatients();

// Function Definitions
void addNewPatient() {
    if (patientCount >= MAX_PATIENTS) {
        printf("Error: Patient limit reached. Cannot add more patients.\n");
        return;
    }

    struct Patient* newPatient = (struct Patient*)malloc(sizeof(struct Patient));
    if (newPatient == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }

    printf("Enter Patient ID: ");
    scanf("%d", &newPatient->patientID);

    printf("Enter Patient Name: ");
    scanf("%s", newPatient->name);

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

    printf("Enter Contact Number: ");
    scanf("%s", newPatient->contactNumber);

    // Collect Medical History
    printf("Enter Medical Condition: ");
    scanf("%s", newPatient->medicalHistory.condition);

    printf("Enter Year Diagnosed: ");
    scanf("%d", &newPatient->medicalHistory.yearDiagnosed);

    printf("Enter Medication Name: ");
    scanf("%s", newPatient->medicalHistory.currentMedication.name);
```

```
    printf("Enter Medication Dosage (in mg): ");
    scanf("%d", &newPatient->medicalHistory.currentMedication.dosage);

    // Adding Emergency Contact/Notes (Union)
    printf("Enter Emergency Contact Name or Additional Notes: ");
    scanf("%s", newPatient->emergencyContactName);

    patients[patientCount++] = newPatient;

    printf("Patient added successfully!\n");
}

void viewPatientDetails(int patientID) {
    int found = 0;
    for (int i = 0; i < patientCount; i++) {
        if (patients[i]->patientID == patientID) {
            found = 1;
            struct Patient* p = patients[i];
            printf("Patient ID: %d\n", p->patientID);
            printf("Name: %s\n", p->name);
            printf("Age: %d\n", p->age);
            printf("Contact: %s\n", p->contactNumber);
            printf("Medical Condition: %s\n", p->medicalHistory.condition);
            printf("Year Diagnosed: %d\n", p->medicalHistory.yearDiagnosed);
            printf("Medication: %s (Dosage: %d mg)\n", p->medicalHistory.currentMedication.name,
p->medicalHistory.currentMedication.dosage);
            printf("Emergency Contact/Notes: %s\n", p->emergencyContactName);
            break;
        }
    }

    if (!found) {
        printf("Patient ID not found.\n");
    }
}

void updatePatientInformation(int patientID) {
    int found = 0;
    for (int i = 0; i < patientCount; i++) {
        if (patients[i]->patientID == patientID) {
            found = 1;
            struct Patient* p = patients[i];

            printf("Updating details for Patient ID: %d\n", p->patientID);

            printf("Enter new Patient Name: ");
            scanf("%s", p->name);

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

            printf("Enter new Contact Number: ");
            scanf("%s", p->contactNumber);
```

```
            // Update Medical History
            printf("Enter new Medical Condition: ");
            scanf("%s", p->medicalHistory.condition);

            printf("Enter new Year Diagnosed: ");
            scanf("%d", &p->medicalHistory.yearDiagnosed);

            printf("Enter new Medication Name: ");
            scanf("%s", p->medicalHistory.currentMedication.name);

            printf("Enter new Medication Dosage (in mg): ");
            scanf("%d", &p->medicalHistory.currentMedication.dosage);

            printf("Enter new Emergency Contact Name or Additional Notes: ");
            scanf("%s", p->emergencyContactName);

            printf("Patient information updated successfully!\n");
            break;
        }
    }

    if (!found) {
        printf("Patient ID not found.\n");
    }
}

void deletePatientRecord(int patientID) {
    int found = 0;
    for (int i = 0; i < patientCount; i++) {
        if (patients[i]->patientID == patientID) {
            found = 1;
            free(patients[i]);
            patients[i] = patients[patientCount - 1]; // Replace with last patient
            patientCount--;
            printf("Patient record deleted successfully!\n");
            break;
        }
    }

    if (!found) {
        printf("Patient ID not found.\n");
    }
}

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

    for (int i = 0; i < patientCount; i++) {
        printf("Patient ID: %d, Name: %s\n", patients[i]->patientID, patients[i]->name);
    }
}
```

```
int main() {
    int choice;
    int patientID;

    while (1) {
        printf("\nPatient Information Management System\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:
                printf("Enter Patient ID to view details: ");
                scanf("%d", &patientID);
                viewPatientDetails(patientID);
                break;
            case 3:
                printf("Enter Patient ID to update: ");
                scanf("%d", &patientID);
                updatePatientInformation(patientID);
                break;
            case 4:
                printf("Enter Patient ID to delete: ");
                scanf("%d", &patientID);
                deletePatientRecord(patientID);
                break;
            case 5:
                listAllPatients();
                break;
            case 6:
                printf("Exiting program...\n");
                for (int i = 0; i < patientCount; i++) {
                    free(patients[i]); // Free memory for each patient
                }
                return;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

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 <string.h>

// Define a constant for the maximum number of items in the inventory
#define MAX_ITEMS 100

// Define a structure for item specifications
typedef struct {
    char name[50];
    int quantity;
    float price;
} ItemSpecification;

// Define a union for different attributes of an item (price or discount)
typedef union {
    float price;
    float discount;
} ItemAttribute;

// Define a structure for each inventory item
typedef struct {
    int id;
    ItemSpecification specification;
    ItemAttribute attribute;
    int isDiscounted;  // Flag to check if discount is applied
} InventoryItem;

// Function prototypes
void addInventoryItem(InventoryItem *inventory, int *itemCount);
void viewInventoryItem(InventoryItem *inventory, int itemCount);
void updateInventoryItem(InventoryItem *inventory, int itemCount);
void deleteInventoryItem(InventoryItem *inventory, int *itemCount);
void listAllInventoryItems(InventoryItem *inventory, int itemCount);

// Main function to handle menu and operations
int main() {
    InventoryItem inventory[MAX_ITEMS];
```

```
    int itemCount = 0;
    int choice;

    do {
        printf("\nHospital Inventory Management System\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(inventory, &itemCount);
                break;
            case 2:
                viewInventoryItem(inventory, itemCount);
                break;
            case 3:
                updateInventoryItem(inventory, itemCount);
                break;
            case 4:
                deleteInventoryItem(inventory, &itemCount);
                break;
            case 5:
                listAllInventoryItems(inventory, itemCount);
                break;
            case 6:
                printf("Exiting the system.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while(choice != 6);

    return 0;
}

// Function to add an inventory item
void addInventoryItem(InventoryItem *inventory, int *itemCount) {
    if (*itemCount >= MAX_ITEMS) {
        printf("Inventory is full.\n");
        return;
    }

    InventoryItem *item = &inventory[*itemCount];

    printf("Enter item ID: ");
    scanf("%d", &item->id);

    printf("Enter item name: ");
    // Read the name using scanf with %49s (to avoid buffer overflow)
```

```
    scanf("%49s", item->specification.name);

    printf("Enter item quantity: ");
    scanf("%d", &item->specification.quantity);

    printf("Enter item price: ");
    scanf("%f", &item->specification.price);

    // Ask if the item has a discount
    printf("Does this item have a discount? (1 for Yes, 0 for No): ");
    scanf("%d", &item->isDiscounted);
    if (item->isDiscounted) {
        printf("Enter discount amount: ");
        scanf("%f", &item->attribute.discount);
    } else {
        item->attribute.price = item->specification.price;
    }

    (*itemCount)++;
    printf("Inventory item added successfully.\n");
}

// Function to view an inventory item
void viewInventoryItem(InventoryItem *inventory, int itemCount) {
    int id;
    printf("Enter item ID to view: ");
    scanf("%d", &id);

    for (int i = 0; i < itemCount; i++) {
        if (inventory[i].id == id) {
            printf("Item ID: %d\n", inventory[i].id);
            printf("Item Name: %s\n", inventory[i].specification.name);
            printf("Quantity: %d\n", inventory[i].specification.quantity);
            printf("Price: %.2f\n", inventory[i].attribute.price);
            if (inventory[i].isDiscounted) {
                printf("Discount: %.2f\n", inventory[i].attribute.discount);
            }
            return;
        }
    }
    printf("Item with ID %d not found.\n", id);
}

// Function to update an inventory item
void updateInventoryItem(InventoryItem *inventory, int itemCount) {
    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 quantity: ");
            scanf("%d", &inventory[i].specification.quantity);

            printf("Enter new price: ");
```

```
            scanf("%f", &inventory[i].specification.price);

            printf("Does this item have a discount? (1 for Yes, 0 for No): ");
            scanf("%d", &inventory[i].isDiscounted);
            if (inventory[i].isDiscounted) {
                printf("Enter new discount amount: ");
                scanf("%f", &inventory[i].attribute.discount);
            } else {
                inventory[i].attribute.price = inventory[i].specification.price;
            }

            printf("Item updated successfully.\n");
            return;
        }
    }
    printf("Item with ID %d not found.\n", id);
}

// Function to delete an inventory item
void deleteInventoryItem(InventoryItem *inventory, int *itemCount) {
    int id;
    printf("Enter item ID to delete: ");
    scanf("%d", &id);

    for (int i = 0; i < *itemCount; i++) {
        if (inventory[i].id == id) {
            // Shift the remaining items to fill the gap
            for (int j = i; j < *itemCount - 1; j++) {
                inventory[j] = inventory[j + 1];
            }
            (*itemCount)--;
            printf("Item with ID %d deleted successfully.\n", id);
            return;
        }
    }
    printf("Item with ID %d not found.\n", id);
}

// Function to list all inventory items
void listAllInventoryItems(InventoryItem *inventory, int itemCount) {
    if (itemCount == 0) {
        printf("No items in inventory.\n");
        return;
    }

    printf("Inventory List:\n");
    for (int i = 0; i < itemCount; i++) {
        printf("ID: %d, Name: %s, Quantity: %d, Price: %.2f", inventory[i].id, inventory[i].specification.name,
inventory[i].specification.quantity, inventory[i].attribute.price);
        if (inventory[i].isDiscounted) {
            printf(", Discount: %.2f", inventory[i].attribute.discount);
        }
        printf("\n");
    }
}
```

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 <string.h>

#define MAX_APPOINTMENTS 100

// Structure to store appointment details
typedef struct {
    int appointmentID;
    char patientName[50];
    char doctorName[50];
    int day, month, year, hour, minute;
    int isEmergency;  // 1 if emergency, 0 if not
} Appointment;

// Function prototypes
void scheduleAppointment(Appointment appointments[], int *appointmentCount);
void viewAppointment(Appointment appointments[], int appointmentCount);
void updateAppointment(Appointment appointments[], int appointmentCount);
void cancelAppointment(Appointment appointments[], int *appointmentCount);
void listAllAppointments(Appointment appointments[], int appointmentCount);

int main() {
    Appointment appointments[MAX_APPOINTMENTS];
    int appointmentCount = 0;
    int choice;

    do {
        printf("\nMedical Appointment Scheduling System\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(appointments, &appointmentCount);
                break;
            case 2:
                viewAppointment(appointments, appointmentCount);
                break;
            case 3:
                updateAppointment(appointments, appointmentCount);
                break;
            case 4:
                cancelAppointment(appointments, &appointmentCount);
                break;
            case 5:
                listAllAppointments(appointments, appointmentCount);
                break;
            case 6:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while(choice != 6);

    return 0;
}

// Function to schedule an appointment
void scheduleAppointment(Appointment appointments[], int *appointmentCount) {
    if (*appointmentCount >= MAX_APPOINTMENTS) {
        printf("Appointment slots are full.\n");
        return;
    }

    Appointment newAppointment;
    newAppointment.appointmentID = *appointmentCount + 1;

    printf("\nEnter patient name: ");
    scanf("%s", newAppointment.patientName);
    printf("Enter doctor name: ");
    scanf("%s", newAppointment.doctorName);
    printf("Enter appointment date (day month year): ");
    scanf("%d %d %d", &newAppointment.day, &newAppointment.month, &newAppointment.year);
    printf("Enter appointment time (hour minute): ");
    scanf("%d %d", &newAppointment.hour, &newAppointment.minute);
    printf("Is this an emergency appointment? (1 for Yes, 0 for No): ");
    scanf("%d", &newAppointment.isEmergency);

    appointments[*appointmentCount] = newAppointment;
```

```
    (*appointmentCount)++;
    printf("Appointment scheduled successfully.\n");
}

// Function to view an appointment
void viewAppointment(Appointment appointments[], int appointmentCount) {
    int appointmentID;
    printf("\nEnter appointment ID to view: ");
    scanf("%d", &appointmentID);

    if (appointmentID <= 0 || appointmentID > appointmentCount) {
        printf("Appointment not found.\n");
        return;
    }

    Appointment appointment = appointments[appointmentID - 1];
    printf("\nAppointment ID: %d\n", appointment.appointmentID);
    printf("Patient Name: %s\n", appointment.patientName);
    printf("Doctor Name: %s\n", appointment.doctorName);
    printf("Appointment Date: %02d/%02d/%04d %02d:%02d\n", appointment.day, appointment.month,
appointment.year, appointment.hour, appointment.minute);
    printf("Emergency: %s\n", appointment.isEmergency ? "Yes" : "No");
}

// Function to update an appointment
void updateAppointment(Appointment appointments[], int appointmentCount) {
    int appointmentID;
    printf("\nEnter appointment ID to update: ");
    scanf("%d", &appointmentID);

    if (appointmentID <= 0 || appointmentID > appointmentCount) {
        printf("Appointment not found.\n");
        return;
    }

    Appointment* appointment = &appointments[appointmentID - 1];

    printf("\nUpdating appointment ID %d\n", appointment->appointmentID);
    printf("Enter new patient name: ");
    scanf("%s", appointment->patientName);
    printf("Enter new doctor name: ");
    scanf("%s", appointment->doctorName);
    printf("Enter new appointment date (day month year): ");
    scanf("%d %d %d", &appointment->day, &appointment->month, &appointment->year);
    printf("Enter new appointment time (hour minute): ");
    scanf("%d %d", &appointment->hour, &appointment->minute);
    printf("Is this an emergency appointment? (1 for Yes, 0 for No): ");
    scanf("%d", &appointment->isEmergency);

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

// Function to cancel an appointment
void cancelAppointment(Appointment appointments[], int *appointmentCount) {
    int appointmentID;
```

```
    printf("\nEnter appointment ID to cancel: ");
    scanf("%d", &appointmentID);

    if (appointmentID <= 0 || appointmentID > *appointmentCount) {
        printf("Appointment not found.\n");
        return;
    }

    // Shift remaining appointments to cancel the one
    for (int i = appointmentID - 1; i < *appointmentCount - 1; i++) {
        appointments[i] = appointments[i + 1];
    }

    (*appointmentCount)--;
    printf("Appointment canceled successfully.\n");
}

// Function to list all appointments
void listAllAppointments(Appointment appointments[], int appointmentCount) {
    if (appointmentCount == 0) {
        printf("No appointments scheduled.\n");
        return;
    }

    printf("\nListing all appointments:\n");
    for (int i = 0; i < appointmentCount; i++) {
        Appointment appointment = appointments[i];
        printf("\nAppointment ID: %d\n", appointment.appointmentID);
        printf("Patient Name: %s\n", appointment.patientName);
        printf("Doctor Name: %s\n", appointment.doctorName);
        printf("Appointment Date: %02d/%02d/%04d %02d:%02d\n", appointment.day, appointment.month,
appointment.year, appointment.hour, appointment.minute);
        printf("Emergency: %s\n", appointment.isEmergency ? "Yes" : "No");
    }
}
```

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>

// Fixed billing rates (static const)
static const float CONSULTATION_FEE = 50.0;
static const float ROOM_CHARGE_PER_DAY = 100.0;
static const float MEDICATION_FEE = 30.0;

// Define a union for variable billing elements (for simplicity, we'll assume these could change)
typedef union {
    float dailyRoomCharge;
    float extraMedicationFee;
} BillingDetailsUnion;

// Define a structure for detailed billing breakdown
typedef struct {
    float consultationFee;
    float roomCharge;
    float medicationFee;
    BillingDetailsUnion billingDetails;
} BillingBreakdown;

// Define a structure for patient billing record
typedef struct {
    int patientID;
    char patientName[50];
    BillingBreakdown bill;
} PatientBill;

// Global array to store bills for all patients
PatientBill patientBills[100];
int billCount = 0;

// Function to generate a bill
void generateBill() {
    PatientBill newBill;

    printf("Enter Patient ID: ");
    scanf("%d", &newBill.patientID);
    getchar();  // To consume the newline character after entering patient ID

    printf("Enter Patient Name: ");
    scanf("%49[^\n]", newBill.patientName); // Limiting the input to 49 characters to avoid overflow

    printf("Enter Number of Days in Hospital: ");
    int daysInHospital;
    scanf("%d", &daysInHospital);

    // Calculate the charges
    newBill.bill.consultationFee = CONSULTATION_FEE;
    newBill.bill.roomCharge = daysInHospital * ROOM_CHARGE_PER_DAY;
    newBill.bill.medicationFee = MEDICATION_FEE;
```

```
    // Optionally, you can add extra room charges or medication fees using the union
    printf("Enter Extra Medication Fee (or 0 if none): ");
    scanf("%f", &newBill.bill.billingDetails.extraMedicationFee);

    // Store the bill in the array
    patientBills[billCount++] = newBill;

    printf("Bill Generated Successfully!\n\n");
}

// Function to view a bill
void viewBill() {
    int patientID;
    printf("Enter Patient ID to View Bill: ");
    scanf("%d", &patientID);

    for (int i = 0; i < billCount; i++) {
        if (patientBills[i].patientID == patientID) {
            printf("\nPatient Bill Details:\n");
            printf("Patient ID: %d\n", patientBills[i].patientID);
            printf("Patient Name: %s\n", patientBills[i].patientName);
            printf("Consultation Fee: $%.2f\n", patientBills[i].bill.consultationFee);
            printf("Room Charge: $%.2f\n", patientBills[i].bill.roomCharge);
            printf("Medication Fee: $%.2f\n", patientBills[i].bill.medicationFee);
            printf("Extra Medication Fee: $%.2f\n", patientBills[i].bill.billingDetails.extraMedicationFee);
            printf("Total Bill: $%.2f\n\n", patientBills[i].bill.consultationFee + patientBills[i].bill.roomCharge +
patientBills[i].bill.medicationFee + patientBills[i].bill.billingDetails.extraMedicationFee);
            return;
        }
    }
    printf("Bill for Patient ID %d not found!\n\n", patientID);
}

// Function to update a bill
void updateBill() {
    int patientID;
    printf("Enter Patient ID to Update Bill: ");
    scanf("%d", &patientID);

    for (int i = 0; i < billCount; i++) {
        if (patientBills[i].patientID == patientID) {
            printf("\nUpdate Bill for Patient ID: %d\n", patientID);

            printf("Enter Number of Additional Days in Hospital: ");
            int extraDays;
            scanf("%d", &extraDays);
            patientBills[i].bill.roomCharge += extraDays * ROOM_CHARGE_PER_DAY;

            printf("Enter Extra Medication Fee: ");
            float extraMedication;
            scanf("%f", &extraMedication);
            patientBills[i].bill.billingDetails.extraMedicationFee += extraMedication;

            printf("Bill Updated Successfully!\n\n");
```

```
            return;
        }
    }
    printf("Bill for Patient ID %d not found!\n\n", patientID);
}

// Function to delete a bill
void deleteBill() {
    int patientID;
    printf("Enter Patient ID to Delete Bill: ");
    scanf("%d", &patientID);

    for (int i = 0; i < billCount; i++) {
        if (patientBills[i].patientID == patientID) {
            // Shift the elements to delete the bill
            for (int j = i; j < billCount - 1; j++) {
                patientBills[j] = patientBills[j + 1];
            }
            billCount--;
            printf("Bill Deleted Successfully!\n\n");
            return;
        }
    }
    printf("Bill for Patient ID %d not found!\n\n", patientID);
}

// Function to list all bills
void listAllBills() {
    if (billCount == 0) {
        printf("No bills available.\n\n");
        return;
    }

    printf("\nAll Patient Bills:\n");
    for (int i = 0; i < billCount; i++) {
        printf("Patient ID: %d\n", patientBills[i].patientID);
        printf("Patient Name: %s\n", patientBills[i].patientName);
        printf("Consultation Fee: $%.2f\n", patientBills[i].bill.consultationFee);
        printf("Room Charge: $%.2f\n", patientBills[i].bill.roomCharge);
        printf("Medication Fee: $%.2f\n", patientBills[i].bill.medicationFee);
        printf("Extra Medication Fee: $%.2f\n", patientBills[i].bill.billingDetails.extraMedicationFee);
        printf("Total Bill: $%.2f\n\n", patientBills[i].bill.consultationFee + patientBills[i].bill.roomCharge +
patientBills[i].bill.medicationFee + patientBills[i].bill.billingDetails.extraMedicationFee);
    }
}

int main() {
    int choice;

    while (1) {
        // Menu
        printf("Patient Billing System\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...\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

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 <string.h>

// Define constants for standard test ranges
#define MIN_TEST_RESULT 0
#define MAX_TEST_RESULT 1000

// Define structure for test result details
typedef struct {
    int testID;
    char testName[50];
    float testValue;
    char testDate[20]; // The date of the test (YYYY-MM-DD format)
} TestResult;

// Define structure for the test range
typedef struct {
    float minValue;
    float maxValue;
} TestRange;

// Define a union for optional test data
typedef union {
    int intData;       // For integer data (e.g., blood count)
    float floatData;   // For floating point data (e.g., cholesterol level)
} TestData;

// Define a structure for the test details with optional data
typedef struct {
    TestResult result;
    TestData optionalData;
    int isOptionalDataAvailable;  // Flag to indicate if optional data is available
} TestDetails;

// Define the test range for some tests (these are just sample ranges)
const TestRange testRanges[] = {
    {0, 150},      // Test ID 1 range: example (Cholesterol range)
    {3.5, 7.5},    // Test ID 2 range: example (Blood sugar range)
};

// Maximum number of tests in the system
#define MAX_TESTS 100

// Array to store test results
TestDetails testResults[MAX_TESTS];
int numTests = 0;

// Function to add a new test result
void addTestResult() {
    if (numTests >= MAX_TESTS) {
        printf("Test result storage is full.\n");
        return;
    }

    TestDetails newTest;
```

```
    printf("Enter test ID: ");
    scanf("%d", &newTest.result.testID);

    printf("Enter test name: ");
    scanf("%s", newTest.result.testName);  // We use scanf to take input without fgets

    printf("Enter test value: ");
    scanf("%f", &newTest.result.testValue);

    printf("Enter test date (YYYY-MM-DD): ");
    scanf("%s", newTest.result.testDate);

    // Ask if there's optional data
    printf("Is there any optional data (1 for Yes, 0 for No)? ");
    scanf("%d", &newTest.isOptionalDataAvailable);

    if (newTest.isOptionalDataAvailable) {
        int option;
        printf("Enter 1 for integer data, 2 for floating-point data: ");
        scanf("%d", &option);

        if (option == 1) {
            printf("Enter integer optional data: ");
            scanf("%d", &newTest.optionalData.intData);
        } else if (option == 2) {
            printf("Enter float optional data: ");
            scanf("%f", &newTest.optionalData.floatData);
        } else {
            printf("Invalid option. Skipping optional data.\n");
        }
    }

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

// Function to view a test result by ID
void viewTestResult() {
    int testID;
    printf("Enter test ID to view: ");
    scanf("%d", &testID);

    for (int i = 0; i < numTests; i++) {
        if (testResults[i].result.testID == testID) {
            printf("Test ID: %d\n", testResults[i].result.testID);
            printf("Test Name: %s\n", testResults[i].result.testName);
            printf("Test Value: %.2f\n", testResults[i].result.testValue);
            printf("Test Date: %s\n", testResults[i].result.testDate);

            if (testResults[i].isOptionalDataAvailable) {
                printf("Optional Data: ");
                if (testResults[i].optionalData.intData) {
                    printf("%d (Integer Data)\n", testResults[i].optionalData.intData);
                } else {
```

```
                printf("%.2f (Float Data)\n", testResults[i].optionalData.floatData);
            }
        } else {
            printf("No optional data available.\n");
        }
        return;
    }
  }
  printf("Test result not found.\n");
}

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

  for (int i = 0; i < numTests; i++) {
    if (testResults[i].result.testID == testID) {
        printf("Updating Test ID: %d\n", testResults[i].result.testID);
        printf("Enter new test name: ");
        scanf("%s", testResults[i].result.testName);

        printf("Enter new test value: ");
        scanf("%f", &testResults[i].result.testValue);

        printf("Enter new test date (YYYY-MM-DD): ");
        scanf("%s", testResults[i].result.testDate);

        // Optional data update
        printf("Is there any optional data (1 for Yes, 0 for No)? ");
        scanf("%d", &testResults[i].isOptionalDataAvailable);

        if (testResults[i].isOptionalDataAvailable) {
            int option;
            printf("Enter 1 for integer data, 2 for floating-point data: ");
            scanf("%d", &option);

            if (option == 1) {
                printf("Enter new integer optional data: ");
                scanf("%d", &testResults[i].optionalData.intData);
            } else if (option == 2) {
                printf("Enter new float optional data: ");
                scanf("%f", &testResults[i].optionalData.floatData);
            } else {
                printf("Invalid option. Skipping optional data.\n");
            }
        }

        printf("Test result updated successfully.\n");
        return;
    }
  }
  printf("Test result not found.\n");
}
```

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

    for (int i = 0; i < numTests; i++) {
        if (testResults[i].result.testID == testID) {
            for (int j = i; j < numTests - 1; j++) {
                testResults[j] = testResults[j + 1];  // Shift elements left
            }
            numTests--;
            printf("Test result deleted successfully.\n");
            return;
        }
    }
    printf("Test result not found.\n");
}

// Function to list all test results
void listAllTestResults() {
    if (numTests == 0) {
        printf("No test results available.\n");
        return;
    }

    printf("Listing all test results:\n");
    for (int i = 0; i < numTests; i++) {
        printf("Test ID: %d, Test Name: %s, Test Value: %.2f, Date: %s\n",
               testResults[i].result.testID, testResults[i].result.testName,
               testResults[i].result.testValue, testResults[i].result.testDate);
    }
}

// Main function
int main() {
    int choice;

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

  return 0;
}
```

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 <string.h>

#define MAX_STAFF 10
#define SHIFT_START 8
#define SHIFT_END 16

// Structure for detailed duty breakdown
typedef struct {
  int start_time;    // Time when the duty starts
  int end_time;      // Time when the duty ends
```

```
    char task[100];    // Task assigned during the shift
} DutyDetails;

// Union for optional duty attributes
typedef union {
    int overtime_hours;   // Optional attribute for overtime hours
    int shift_category;   // Optional attribute for shift category (e.g., Day/Night)
} OptionalDutyAttributes;

// Nested structure combining DutyDetails and OptionalDutyAttributes
typedef struct {
    DutyDetails duty;
    OptionalDutyAttributes optional;
} DutyRoster;

// Array to store staff duty rosters
DutyRoster roster[MAX_STAFF];
int current_roster_count = 0;

// Structure to store staff details
typedef struct {
    char name[50];
    int id;
    DutyRoster duty;
} Staff;

Staff staff_list[MAX_STAFF];

// Static and const for fixed shift timings
const int FIXED_SHIFT_START = SHIFT_START;
const int FIXED_SHIFT_END = SHIFT_END;

// Function to add a new duty roster
void addDutyRoster() {
    if (current_roster_count < MAX_STAFF) {
        Staff new_staff;
        printf("Enter Staff Name: ");
        scanf("%s", new_staff.name);
        printf("Enter Staff ID: ");
        scanf("%d", &new_staff.id);

        // Assigning fixed shift timings
        new_staff.duty.duty.start_time = FIXED_SHIFT_START;
        new_staff.duty.duty.end_time = FIXED_SHIFT_END;
        printf("Enter Task for the Shift: ");
        scanf(" %[^\n]%*c", new_staff.duty.duty.task);  // To handle spaces in task

        // Ask for optional duty attributes (Overtime or Shift Category)
        char choice;
        printf("Does this duty include optional attributes? (y/n): ");
        scanf(" %c", &choice);
        if (choice == 'y' || choice == 'Y') {
            int option;
            printf("Enter 1 for Overtime or 2 for Shift Category: ");
            scanf("%d", &option);
```

```
            if (option == 1) {
                printf("Enter Overtime Hours: ");
                scanf("%d", &new_staff.duty.optional.overtime_hours);
            } else if (option == 2) {
                printf("Enter Shift Category (1 for Day, 2 for Night): ");
                scanf("%d", &new_staff.duty.optional.shift_category);
            }
        }

        // Save to staff list
        staff_list[current_roster_count] = new_staff;
        current_roster_count++;
        printf("Staff duty roster added successfully!\n");
    } else {
        printf("Roster is full! Cannot add more staff.\n");
    }
}

// Function to view a specific duty roster
void viewDutyRoster() {
    int staff_id;
    printf("Enter Staff ID to view duty roster: ");
    scanf("%d", &staff_id);

    for (int i = 0; i < current_roster_count; i++) {
        if (staff_list[i].id == staff_id) {
            printf("Staff Name: %s\n", staff_list[i].name);
            printf("Shift Time: %d - %d\n", staff_list[i].duty.duty.start_time, staff_list[i].duty.duty.end_time);
            printf("Task Assigned: %s\n", staff_list[i].duty.duty.task);

            // Display optional duty attributes
            if (staff_list[i].duty.optional.overtime_hours > 0) {
                printf("Overtime Hours: %d\n", staff_list[i].duty.optional.overtime_hours);
            } else if (staff_list[i].duty.optional.shift_category != 0) {
                printf("Shift Category: %s\n", (staff_list[i].duty.optional.shift_category == 1) ? "Day" : "Night");
            }
            return;
        }
    }

    printf("Staff with ID %d not found.\n", staff_id);
}

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

    for (int i = 0; i < current_roster_count; i++) {
        if (staff_list[i].id == staff_id) {
            printf("Updating Duty for Staff: %s\n", staff_list[i].name);
            printf("Enter new Task for the Shift: ");
            scanf(" %[^\n]%*c", staff_list[i].duty.duty.task);
```

```
            // Optionally update overtime or shift category
            char choice;
            printf("Would you like to update optional attributes? (y/n): ");
            scanf(" %c", &choice);
            if (choice == 'y' || choice == 'Y') {
                int option;
                printf("Enter 1 to update Overtime or 2 to update Shift Category: ");
                scanf("%d", &option);
                if (option == 1) {
                    printf("Enter new Overtime Hours: ");
                    scanf("%d", &staff_list[i].duty.optional.overtime_hours);
                } else if (option == 2) {
                    printf("Enter new Shift Category (1 for Day, 2 for Night): ");
                    scanf("%d", &staff_list[i].duty.optional.shift_category);
                }
            }
            printf("Duty roster updated successfully!\n");
            return;
        }
    }

    printf("Staff with ID %d not found.\n", staff_id);
}

// Function to delete a duty roster
void deleteDutyRoster() {
    int staff_id;
    printf("Enter Staff ID to delete duty roster: ");
    scanf("%d", &staff_id);

    for (int i = 0; i < current_roster_count; i++) {
        if (staff_list[i].id == staff_id) {
            // Shift subsequent records to delete the entry
            for (int j = i; j < current_roster_count - 1; j++) {
                staff_list[j] = staff_list[j + 1];
            }
            current_roster_count--;
            printf("Staff duty roster deleted successfully!\n");
            return;
        }
    }

    printf("Staff with ID %d not found.\n", staff_id);
}

// Function to list all duty rosters
void listAllDutyRosters() {
    if (current_roster_count == 0) {
        printf("No duty rosters available.\n");
        return;
    }

    for (int i = 0; i < current_roster_count; i++) {
        printf("\nStaff Name: %s\n", staff_list[i].name);
        printf("Staff ID: %d\n", staff_list[i].id);
```

```
        printf("Shift Time: %d - %d\n", staff_list[i].duty.duty.start_time, staff_list[i].duty.duty.end_time);
        printf("Task: %s\n", staff_list[i].duty.duty.task);

        // Display optional duty attributes
        if (staff_list[i].duty.optional.overtime_hours > 0) {
            printf("Overtime Hours: %d\n", staff_list[i].duty.optional.overtime_hours);
        } else if (staff_list[i].duty.optional.shift_category != 0) {
            printf("Shift Category: %s\n", (staff_list[i].duty.optional.shift_category == 1) ? "Day" : "Night");
        }
    }
}

// Main function with menu and switch case for operations
int main() {
    int choice;
    while (1) {
        printf("\n--- Staff Duty Roster Management ---\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 system.\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
}
```

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
#define MAX_NAME_LEN 100
#define MAX_PHONE_LEN 15
#define MAX_ADDRESS_LEN 200

// Structure for the emergency contact details
typedef struct {
    char name[MAX_NAME_LEN];
    char phone[MAX_PHONE_LEN];
    char address[MAX_ADDRESS_LEN];
    int isPrimary; // Flag to indicate whether this is the primary emergency contact
} ContactInfo;

// Union to store optional contact data
typedef union {
    char alternatePhone[MAX_PHONE_LEN];
    char email[MAX_NAME_LEN];
} OptionalContactData;

// Nested structure for detailed emergency contact info
typedef struct {
    ContactInfo contact;
    OptionalContactData optionalData;
    int hasAlternateContact; // Flag to indicate if the contact has alternate contact data
} EmergencyContact;

// Array to store all contacts
EmergencyContact* contacts[MAX_CONTACTS];

// Function to add a new emergency contact
void addEmergencyContact(int* contactCount) {
    if (*contactCount >= MAX_CONTACTS) {
        printf("Contact list is full. Cannot add more contacts.\n");
```

```
        return;
    }

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

    printf("Enter Name: ");
    scanf("%s", newContact->contact.name);

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

    printf("Enter Address: ");
    scanf(" %[^\n]s", newContact->contact.address);

    newContact->hasAlternateContact = 0; // Default, no alternate contact

    printf("Is this the primary contact? (1 for Yes, 0 for No): ");
    scanf("%d", &newContact->contact.isPrimary);

    // Optionally, ask for alternate contact data (phone or email)
    printf("Does this contact have an alternate phone or email? (1 for Yes, 0 for No): ");
    scanf("%d", &newContact->hasAlternateContact);

    if (newContact->hasAlternateContact) {
        printf("Enter alternate contact (Phone or Email): ");
        scanf("%s", newContact->optionalData.alternatePhone);
    }

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

// Function to view an emergency contact
void viewEmergencyContact(int contactCount) {
    if (contactCount == 0) {
        printf("No contacts available.\n");
        return;
    }

    for (int i = 0; i < contactCount; i++) {
        EmergencyContact* contact = contacts[i];
        printf("\nContact %d:\n", i + 1);
        printf("Name: %s\n", contact->contact.name);
        printf("Phone: %s\n", contact->contact.phone);
        printf("Address: %s\n", contact->contact.address);
        printf("Primary Contact: %s\n", contact->contact.isPrimary ? "Yes" : "No");

        if (contact->hasAlternateContact) {
            printf("Alternate Contact: %s\n", contact->optionalData.alternatePhone);
        }
    }
}

// Function to update an emergency contact
```

```
void updateEmergencyContact(int contactCount) {
    int contactId;
    printf("Enter the contact number to update (1 to %d): ", contactCount);
    scanf("%d", &contactId);
    contactId--; // Adjust for 0-based index

    if (contactId < 0 || contactId >= contactCount) {
        printf("Invalid contact number.\n");
        return;
    }

    EmergencyContact* contact = contacts[contactId];
    printf("Updating contact details for %s:\n", contact->contact.name);

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

    printf("Enter New Address: ");
    scanf(" %[^\n]s", contact->contact.address);

    printf("Is this the primary contact? (1 for Yes, 0 for No): ");
    scanf("%d", &contact->contact.isPrimary);

    // Optionally, update alternate contact data
    printf("Does this contact have an alternate phone or email? (1 for Yes, 0 for No): ");
    scanf("%d", &contact->hasAlternateContact);

    if (contact->hasAlternateContact) {
        printf("Enter alternate contact (Phone or Email): ");
        scanf("%s", contact->optionalData.alternatePhone);
    }

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

// Function to delete an emergency contact
void deleteEmergencyContact(int* contactCount) {
    int contactId;
    printf("Enter the contact number to delete (1 to %d): ", *contactCount);
    scanf("%d", &contactId);
    contactId--; // Adjust for 0-based index

    if (contactId < 0 || contactId >= *contactCount) {
        printf("Invalid contact number.\n");
        return;
    }

    free(contacts[contactId]);

    for (int i = contactId; i < *contactCount - 1; i++) {
        contacts[i] = contacts[i + 1];
    }

    (*contactCount)--;
```

```
    printf("Contact deleted successfully.\n");
}

// Function to list all emergency contacts
void listAllEmergencyContacts(int contactCount) {
    if (contactCount == 0) {
        printf("No contacts available.\n");
        return;
    }

    printf("\nAll Emergency Contacts:\n");
    for (int i = 0; i < contactCount; i++) {
        printf("\nContact %d:\n", i + 1);
        printf("Name: %s\n", contacts[i]->contact.name);
        printf("Phone: %s\n", contacts[i]->contact.phone);
        printf("Address: %s\n", contacts[i]->contact.address);
        printf("Primary Contact: %s\n", contacts[i]->contact.isPrimary ? "Yes" : "No");

        if (contacts[i]->hasAlternateContact) {
            printf("Alternate Contact: %s\n", contacts[i]->optionalData.alternatePhone);
        }
    }
}

int main() {
    int contactCount = 0;
    int choice;

    while (1) {
        // Display the menu options
        printf("\nEmergency Contact Management System\n");
        printf("1. Add Emergency Contact\n");
        printf("2. View Emergency Contact\n");
        printf("3. Update Emergency Contact\n");
        printf("4. Delete Emergency Contact\n");
        printf("5. List All Emergency Contacts\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addEmergencyContact(&contactCount);
                break;
            case 2:
                viewEmergencyContact(contactCount);
                break;
            case 3:
                updateEmergencyContact(contactCount);
                break;
            case 4:
                deleteEmergencyContact(&contactCount);
                break;
            case 5:
                listAllEmergencyContacts(contactCount);
```

```
            break;
        case 6:
            // Free all allocated memory before exit
            for (int i = 0; i < contactCount; i++) {
                free(contacts[i]);
            }
            printf("Exiting the system...\n");
            return 0;
        default:
            printf("Invalid choice, please try again.\n");
    }
  }

  return 0;
}
```

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 <string.h>

#define MAX_RECORDS 100
#define MAX_NAME_LENGTH 50
#define MAX_DISEASE_LENGTH 100

// Define a union for optional record fields
union OptionalField {
    char disease[MAX_DISEASE_LENGTH];
    int age;
};

// Define a structure for the medical history (nested structure)
struct MedicalHistory {
    char disease[MAX_DISEASE_LENGTH];
    int yearDiagnosed;
```

```
    char doctorName[MAX_NAME_LENGTH];
};

// Define the structure for a medical record
struct MedicalRecord {
    const int recordID;  // record ID is immutable (const)
    char patientName[MAX_NAME_LENGTH];
    int age;
    union OptionalField optionalField; // Optional field for disease or age
    int medicalHistoryCount;
    struct MedicalHistory history[10]; // Up to 10 medical histories
};

// Declare an array of medical records
struct MedicalRecord records[MAX_RECORDS];
int currentRecordCount = 0;  // Tracks number of records added

// Function to add a medical record
void addMedicalRecord() {
    if (currentRecordCount < MAX_RECORDS) {
        struct MedicalRecord *newRecord = &records[currentRecordCount];
        newRecord->recordID = currentRecordCount + 1; // Auto-generate record ID
        printf("Enter patient's name: ");
        scanf("%s", newRecord->patientName);
        printf("Enter patient's age: ");
        scanf("%d", &newRecord->age);

        // Optionally, enter disease or age in the union
        printf("Enter '1' for disease or '2' for age to store in record: ");
        int choice;
        scanf("%d", &choice);

        if (choice == 1) {
            printf("Enter disease: ");
            scanf("%s", newRecord->optionalField.disease);
        } else if (choice == 2) {
            newRecord->optionalField.age = newRecord->age;
        }

        printf("How many medical histories to add (max 10): ");
        scanf("%d", &newRecord->medicalHistoryCount);

        for (int i = 0; i < newRecord->medicalHistoryCount; i++) {
            printf("Enter disease %d: ", i + 1);
            scanf("%s", newRecord->history[i].disease);
            printf("Enter year diagnosed: ");
            scanf("%d", &newRecord->history[i].yearDiagnosed);
            printf("Enter doctor's name: ");
            scanf("%s", newRecord->history[i].doctorName);
        }

        currentRecordCount++;
        printf("Medical record added successfully!\n");
    } else {
        printf("Cannot add more records, storage full!\n");
```

```
    }
}

// Function to view a medical record
void viewMedicalRecord() {
    int recordID;
    printf("Enter record ID to view: ");
    scanf("%d", &recordID);

    if (recordID > 0 && recordID <= currentRecordCount) {
        struct MedicalRecord *record = &records[recordID - 1];
        printf("\nRecord ID: %d\n", record->recordID);
        printf("Patient Name: %s\n", record->patientName);
        printf("Age: %d\n", record->age);
        printf("Optional Field (Disease or Age): ");
        if (strlen(record->optionalField.disease) > 0) {
            printf("Disease: %s\n", record->optionalField.disease);
        } else {
            printf("Age: %d\n", record->optionalField.age);
        }

        printf("Medical History:\n");
        for (int i = 0; i < record->medicalHistoryCount; i++) {
            printf("Disease: %s\n", record->history[i].disease);
            printf("Year Diagnosed: %d\n", record->history[i].yearDiagnosed);
            printf("Doctor: %s\n", record->history[i].doctorName);
        }
    } else {
        printf("Record not found!\n");
    }
}

// Function to update a medical record
void updateMedicalRecord() {
    int recordID;
    printf("Enter record ID to update: ");
    scanf("%d", &recordID);

    if (recordID > 0 && recordID <= currentRecordCount) {
        struct MedicalRecord *record = &records[recordID - 1];
        int choice;

        printf("What would you like to update?\n");
        printf("1. Update Patient Name\n");
        printf("2. Update Age\n");
        printf("3. Update Disease (Optional)\n");
        printf("4. Update Medical History\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter new patient name: ");
                scanf("%s", record->patientName);
                break;
```

```
            case 2:
                printf("Enter new age: ");
                scanf("%d", &record->age);
                break;
            case 3:
                printf("Enter '1' to update disease or '2' for age in the optional field: ");
                int optChoice;
                scanf("%d", &optChoice);

                if (optChoice == 1) {
                    printf("Enter new disease: ");
                    scanf("%s", record->optionalField.disease);
                } else if (optChoice == 2) {
                    printf("Enter new age: ");
                    scanf("%d", &record->optionalField.age);
                }
                break;
            case 4:
                printf("How many medical histories to update: ");
                scanf("%d", &record->medicalHistoryCount);
                for (int i = 0; i < record->medicalHistoryCount; i++) {
                    printf("Enter new disease for history %d: ", i + 1);
                    scanf("%s", record->history[i].disease);
                    printf("Enter new year diagnosed: ");
                    scanf("%d", &record->history[i].yearDiagnosed);
                    printf("Enter new doctor's name: ");
                    scanf("%s", record->history[i].doctorName);
                }
                break;
            default:
                printf("Invalid choice!\n");
                break;
        }

        printf("Record updated successfully!\n");
    } else {
        printf("Record not found!\n");
    }
}

// Function to delete a medical record
void deleteMedicalRecord() {
    int recordID;
    printf("Enter record ID to delete: ");
    scanf("%d", &recordID);

    if (recordID > 0 && recordID <= currentRecordCount) {
        for (int i = recordID - 1; i < currentRecordCount - 1; i++) {
            records[i] = records[i + 1];
        }
        currentRecordCount--;
        printf("Record deleted successfully!\n");
    } else {
        printf("Record not found!\n");
    }
```

```
}

// Function to list all medical records
void listAllMedicalRecords() {
    if (currentRecordCount == 0) {
        printf("No records available.\n");
    } else {
        for (int i = 0; i < currentRecordCount; i++) {
            printf("\nRecord ID: %d\n", records[i].recordID);
            printf("Patient Name: %s\n", records[i].patientName);
            printf("Age: %d\n", records[i].age);
            printf("Medical History Count: %d\n", records[i].medicalHistoryCount);
        }
    }
}

// Main menu function
void menu() {
    int choice;
    do {
        printf("\n--- Medical Record Update System ---\n");
        printf("1. Add Medical Record\n");
        printf("2. View Medical Record\n");
        printf("3. Update Medical Record\n");
        printf("4. Delete Medical Record\n");
        printf("5. List All Medical Records\n");
        printf("6. Exit\n");
        printf("Enter your 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 the system.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 6);
}
```

```
int main() {
    menu();
    return 0;
}
```

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 <string.h>

#define MAX_PLANS 100
#define MAX_NAME_LEN 50
#define MAX_FOOD_ITEMS 10

// Define a structure for food details
typedef struct {
    char foodName[MAX_NAME_LEN];
    int quantity; // in grams
} FoodItem;

// Define a union for optional attributes of the diet plan (e.g., vegetarian or gluten-free)
typedef union {
    int vegetarian;  // 1 for vegetarian, 0 for non-vegetarian
    int glutenFree;  // 1 for gluten-free, 0 for not gluten-free
} OptionalAttributes;

// Define a structure for a detailed diet plan
typedef struct {
    char planName[MAX_NAME_LEN]; // Name of the diet plan
    int calorieCount;            // Total calories
    FoodItem foodItems[MAX_FOOD_ITEMS]; // Array of food items
    int foodCount;               // Number of food items in the plan
    OptionalAttributes options; // Union for optional attributes
} DietPlan;
```

```
// Declare a static array to store diet plans
static DietPlan dietPlans[MAX_PLANS];
static int currentPlanCount = 0;

// Declare constant dietary guidelines
const int MAX_CALORIES = 2500;  // Max calories for a daily plan

// Function prototypes
void addDietPlan();
void viewDietPlan(int index);
void updateDietPlan(int index);
void deleteDietPlan(int index);
void listAllDietPlans();
void handleDietPlanOperations();

// Function to add a new diet plan
void addDietPlan() {
    if (currentPlanCount >= MAX_PLANS) {
        printf("Error: Cannot add more diet plans. Storage is full.\n");
        return;
    }

    DietPlan newPlan;
    printf("Enter the name of the diet plan: ");
    scanf("%s", newPlan.planName);

    printf("Enter total calories for the plan: ");
    scanf("%d", &newPlan.calorieCount);

    printf("Enter the number of food items in the diet plan: ");
    scanf("%d", &newPlan.foodCount);

    for (int i = 0; i < newPlan.foodCount; i++) {
        printf("Enter the name of food item %d: ", i + 1);
        scanf("%s", newPlan.foodItems[i].foodName);

        printf("Enter the quantity (in grams) for food item %d: ", i + 1);
        scanf("%d", &newPlan.foodItems[i].quantity);
    }

    printf("Enter 1 if the diet plan is vegetarian, 0 otherwise: ");
    scanf("%d", &newPlan.options.vegetarian);

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

// Function to view a specific diet plan
void viewDietPlan(int index) {
    if (index < 0 || index >= currentPlanCount) {
        printf("Invalid index. No such diet plan exists.\n");
        return;
    }

    DietPlan plan = dietPlans[index];
```

```
    printf("Diet Plan Name: %s\n", plan.planName);
    printf("Total Calories: %d\n", plan.calorieCount);
    printf("Food Items:\n");

    for (int i = 0; i < plan.foodCount; i++) {
        printf("  - %s (%d grams)\n", plan.foodItems[i].foodName, plan.foodItems[i].quantity);
    }

    printf("Vegetarian: %s\n", plan.options.vegetarian == 1 ? "Yes" : "No");
}

// Function to update a diet plan
void updateDietPlan(int index) {
    if (index < 0 || index >= currentPlanCount) {
        printf("Invalid index. No such diet plan exists.\n");
        return;
    }

    DietPlan* plan = &dietPlans[index];
    printf("Enter the new name of the diet plan: ");
    scanf("%s", plan->planName);

    printf("Enter the new total calories for the plan: ");
    scanf("%d", &plan->calorieCount);

    printf("Enter the new number of food items in the diet plan: ");
    scanf("%d", &plan->foodCount);

    for (int i = 0; i < plan->foodCount; i++) {
        printf("Enter the new name of food item %d: ", i + 1);
        scanf("%s", plan->foodItems[i].foodName);

        printf("Enter the new quantity (in grams) for food item %d: ", i + 1);
        scanf("%d", &plan->foodItems[i].quantity);
    }

    printf("Enter 1 if the diet plan is vegetarian, 0 otherwise: ");
    scanf("%d", &plan->options.vegetarian);

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

// Function to delete a specific diet plan
void deleteDietPlan(int index) {
    if (index < 0 || index >= currentPlanCount) {
        printf("Invalid index. No such diet plan exists.\n");
        return;
    }

    for (int i = index; i < currentPlanCount - 1; i++) {
        dietPlans[i] = dietPlans[i + 1];
    }

    currentPlanCount--;
    printf("Diet plan deleted successfully.\n");
```

```
}

// Function to list all diet plans
void listAllDietPlans() {
    if (currentPlanCount == 0) {
        printf("No diet plans available.\n");
        return;
    }

    printf("List of all diet plans:\n");
    for (int i = 0; i < currentPlanCount; i++) {
        printf("%d. %s\n", i + 1, dietPlans[i].planName);
    }
}

// Main menu for handling diet plan operations
void handleDietPlanOperations() {
    int choice, index;

    while (1) {
        printf("\nMenu Options:\n");
        printf("1. 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:
                printf("Enter the index of the diet plan to view (1 - %d): ", currentPlanCount);
                scanf("%d", &index);
                viewDietPlan(index - 1);
                break;
            case 3:
                printf("Enter the index of the diet plan to update (1 - %d): ", currentPlanCount);
                scanf("%d", &index);
                updateDietPlan(index - 1);
                break;
            case 4:
                printf("Enter the index of the diet plan to delete (1 - %d): ", currentPlanCount);
                scanf("%d", &index);
                deleteDietPlan(index - 1);
                break;
            case 5:
                listAllDietPlans();
                break;
            case 6:
                printf("Exiting the program.\n");
                return;
```

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

int main() {
    handleDietPlanOperations();
    return 0;
}
```

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 <string.h>

#define MAX_SURGERIES 100

// Define constant for surgery types
const char *SURGERY_TYPES[] = {"Cardiac Surgery", "Orthopedic Surgery", "Neurosurgery", "Plastic Surgery", "General Surgery"};

// Define structure for surgery details
struct Surgery {
    int id;
    char patientName[50];
    int surgeryType;
    char date[20];
    int duration;  // Duration in minutes
};

// Union for optional data (e.g., additional details)
union SurgeryOptional {
    char room[20];
    int doctorID;
```

```
};

// Define a structure for surgery schedule that includes optional surgery data
struct SurgerySchedule {
    struct Surgery surgeryDetails;
    union SurgeryOptional optionalDetails;
    int hasOptionalDetails;  // Flag to check if optional details are provided
};

// Global array to store the surgeries
struct SurgerySchedule surgeries[MAX_SURGERIES];
int surgeryCount = 0;  // Keep track of the number of surgeries scheduled

// Function prototypes
void scheduleSurgery();
void viewSurgerySchedule();
void updateSurgerySchedule();
void cancelSurgery();
void listAllSurgeries();
void printSurgeryDetails(int index);

int main() {
    int choice;

    while (1) {
        // Menu display
        printf("\nSurgery Scheduling System\n");
        printf("1. Schedule Surgery\n");
        printf("2. View Surgery Schedule\n");
        printf("3. Update Surgery Schedule\n");
        printf("4. Cancel Surgery\n");
        printf("5. List All Surgeries\n");
        printf("6. Exit\n");
        printf("Enter your 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 the system.\n");
                return 0;
```

```
            default:
                printf("Invalid choice, please try again.\n");
        }
    }

    return 0;
}

// Function to schedule a surgery
void scheduleSurgery() {
    if (surgeryCount >= MAX_SURGERIES) {
        printf("Cannot schedule more surgeries. Maximum limit reached.\n");
        return;
    }

    struct SurgerySchedule newSurgery;
    newSurgery.surgeryDetails.id = surgeryCount + 1;  // Assign a unique ID for the surgery

    // Input surgery details
    printf("Enter patient's name: ");
    scanf(" %[^\n]", newSurgery.surgeryDetails.patientName);

    printf("Select surgery type:\n");
    for (int i = 0; i < 5; i++) {
        printf("%d. %s\n", i + 1, SURGERY_TYPES[i]);
    }
    printf("Enter surgery type (1-5): ");
    scanf("%d", &newSurgery.surgeryDetails.surgeryType);
    newSurgery.surgeryDetails.surgeryType--;  // Adjust to 0-based index

    printf("Enter surgery date (YYYY-MM-DD): ");
    scanf(" %[^\n]", newSurgery.surgeryDetails.date);

    printf("Enter surgery duration (in minutes): ");
    scanf("%d", &newSurgery.surgeryDetails.duration);

    // Optional details (room or doctor ID)
    printf("Do you want to provide optional details? (1 for yes, 0 for no): ");
    scanf("%d", &newSurgery.hasOptionalDetails);

    if (newSurgery.hasOptionalDetails) {
        int optionalChoice;
        printf("Enter 1 for room or 2 for doctor ID: ");
        scanf("%d", &optionalChoice);

        if (optionalChoice == 1) {
            printf("Enter room number: ");
            scanf(" %[^\n]", newSurgery.optionalDetails.room);
        } else if (optionalChoice == 2) {
            printf("Enter doctor ID: ");
            scanf("%d", &newSurgery.optionalDetails.doctorID);
        } else {
            printf("Invalid choice, skipping optional details.\n");
        }
    }
```

```
    // Add the new surgery to the array
    surgeries[surgeryCount++] = newSurgery;
    printf("Surgery scheduled successfully with ID %d.\n", newSurgery.surgeryDetails.id);
}

// Function to view a specific surgery schedule
void viewSurgerySchedule() {
    int id;
    printf("Enter surgery ID to view details: ");
    scanf("%d", &id);

    if (id > 0 && id <= surgeryCount) {
        printSurgeryDetails(id - 1);
    } else {
        printf("Surgery with ID %d not found.\n", id);
    }
}

// Function to print details of a specific surgery
void printSurgeryDetails(int index) {
    struct SurgerySchedule surgery = surgeries[index];
    printf("\nSurgery ID: %d\n", surgery.surgeryDetails.id);
    printf("Patient Name: %s\n", surgery.surgeryDetails.patientName);
    printf("Surgery Type: %s\n", SURGERY_TYPES[surgery.surgeryDetails.surgeryType]);
    printf("Surgery Date: %s\n", surgery.surgeryDetails.date);
    printf("Surgery Duration: %d minutes\n", surgery.surgeryDetails.duration);

    if (surgery.hasOptionalDetails) {
        printf("Optional Details:\n");
        if (strlen(surgery.optionalDetails.room) > 0) {
            printf("Room: %s\n", surgery.optionalDetails.room);
        } else {
            printf("Doctor ID: %d\n", surgery.optionalDetails.doctorID);
        }
    }
}

// Function to update a surgery schedule
void updateSurgerySchedule() {
    int id;
    printf("Enter surgery ID to update: ");
    scanf("%d", &id);

    if (id > 0 && id <= surgeryCount) {
        struct SurgerySchedule *surgery = &surgeries[id - 1];

        printf("Updating details for Surgery ID %d\n", id);

        printf("Enter new patient's name: ");
        scanf(" %[^\n]", surgery->surgeryDetails.patientName);

        printf("Enter new surgery date (YYYY-MM-DD): ");
        scanf(" %[^\n]", surgery->surgeryDetails.date);
```

```
        printf("Enter new surgery duration (in minutes): ");
        scanf("%d", &surgery->surgeryDetails.duration);

        printf("Do you want to update optional details? (1 for yes, 0 for no): ");
        scanf("%d", &surgery->hasOptionalDetails);

        if (surgery->hasOptionalDetails) {
            int optionalChoice;
            printf("Enter 1 for room or 2 for doctor ID: ");
            scanf("%d", &optionalChoice);

            if (optionalChoice == 1) {
                printf("Enter new room number: ");
                scanf(" %[^\n]", surgery->optionalDetails.room);
            } else if (optionalChoice == 2) {
                printf("Enter new doctor ID: ");
                scanf("%d", &surgery->optionalDetails.doctorID);
            } else {
                printf("Invalid choice, skipping optional details.\n");
            }
        }
        printf("Surgery details updated successfully.\n");
    } else {
        printf("Surgery with ID %d not found.\n", id);
    }
}

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

    if (id > 0 && id <= surgeryCount) {
        for (int i = id - 1; i < surgeryCount - 1; i++) {
            surgeries[i] = surgeries[i + 1];  // Shift subsequent surgeries
        }
        surgeryCount--;  // Decrement surgery count
        printf("Surgery with ID %d has been canceled.\n", id);
    } else {
        printf("Surgery with ID %d not found.\n", id);
    }
}

// Function to list all surgeries
void listAllSurgeries() {
    if (surgeryCount == 0) {
        printf("No surgeries scheduled yet.\n");
        return;
    }

    printf("\nListing All Surgeries:\n");
    for (int i = 0; i < surgeryCount; i++) {
        printf("Surgery ID: %d\n", surgeries[i].surgeryDetails.id);
        printf("Patient Name: %s\n", surgeries[i].surgeryDetails.patientName);
```

```
        printf("Surgery Type: %s\n", SURGERY_TYPES[surgeries[i].surgeryDetails.surgeryType]);
        printf("Surgery Date: %s\n", surgeries[i].surgeryDetails.date);
        printf("Surgery Duration: %d minutes\n", surgeries[i].surgeryDetails.duration);
        printf("----------------------------\n");
    }
}
```

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>

// Constants and static variables
#define MAX_PRESCRIPTIONS 100
#define MAX_MEDICINE_NAME_LENGTH 50
#define MAX_DOCTOR_NAME_LENGTH 50
#define MAX_PATIENT_NAME_LENGTH 50

// Structure for Prescription Details
struct Prescription {
    int id;
    char medicineName[MAX_MEDICINE_NAME_LENGTH];
    int quantity;
    float price;
    char doctorName[MAX_DOCTOR_NAME_LENGTH];
    char patientName[MAX_PATIENT_NAME_LENGTH];

    // Optional/Complex Fields using Union and Nested Structures
    union {
        struct {
            char allergies[MAX_MEDICINE_NAME_LENGTH];
            int isEmergency;
        };
        struct {
```

```
            char instructions[MAX_MEDICINE_NAME_LENGTH];
            int refillCount;
        };
    };
};

// Array of prescriptions
struct Prescription prescriptions[MAX_PRESCRIPTIONS];
int prescriptionCount = 0;

// Function Prototypes
void addPrescription();
void viewPrescription();
void updatePrescription();
void deletePrescription();
void listAllPrescriptions();
void displayPrescription(struct Prescription *prescription);

int main() {
    int choice;

    while (1) {
        // Display menu
        printf("\nPrescription Management System\n");
        printf("1. 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-case for menu options
        if (choice == 1) {
            addPrescription();
        } else if (choice == 2) {
            viewPrescription();
        } else if (choice == 3) {
            updatePrescription();
        } else if (choice == 4) {
            deletePrescription();
        } else if (choice == 5) {
            listAllPrescriptions();
        } else if (choice == 6) {
            printf("Exiting the system...\n");
            return 0;
        } else {
            printf("Invalid choice, please try again.\n");
        }
    }

    return 0;
}
```

```
// Function to add a prescription
void addPrescription() {
    if (prescriptionCount >= MAX_PRESCRIPTIONS) {
        printf("Cannot add more prescriptions. Maximum limit reached.\n");
        return;
    }

    struct Prescription newPrescription;

    // Input prescription details
    newPrescription.id = prescriptionCount + 1;
    printf("Enter medicine name: ");
    scanf(" %[^\n]", newPrescription.medicineName);  // Using space before % to accept spaces in input
    printf("Enter quantity: ");
    scanf("%d", &newPrescription.quantity);
    printf("Enter price: ");
    scanf("%f", &newPrescription.price);
    printf("Enter doctor name: ");
    scanf(" %[^\n]", newPrescription.doctorName);
    printf("Enter patient name: ");
    scanf(" %[^\n]", newPrescription.patientName);

    // Optional information
    printf("Do you want to add allergy information? (1 for Yes, 0 for No): ");
    int addAllergyInfo;
    scanf("%d", &addAllergyInfo);

    if (addAllergyInfo) {
        printf("Enter allergy details: ");
        scanf(" %[^\n]", newPrescription.allergies);
        newPrescription.isEmergency = 1;
    } else {
        printf("Enter instructions for use: ");
        scanf(" %[^\n]", newPrescription.instructions);
        printf("Enter refill count: ");
        scanf("%d", &newPrescription.refillCount);
    }

    // Store the prescription in the array
    prescriptions[prescriptionCount++] = newPrescription;
    printf("Prescription added successfully.\n");
}

// Function to view a prescription
void viewPrescription() {
    int id;
    printf("Enter prescription ID to view: ");
    scanf("%d", &id);

    if (id > 0 && id <= prescriptionCount) {
        displayPrescription(&prescriptions[id - 1]);
    } else {
        printf("Invalid prescription ID.\n");
    }
}
```

```
// Function to update a prescription
void updatePrescription() {
    int id;
    printf("Enter prescription ID to update: ");
    scanf("%d", &id);

    if (id > 0 && id <= prescriptionCount) {
        struct Prescription *prescription = &prescriptions[id - 1];
        printf("Updating prescription ID: %d\n", id);

        // Update details
        printf("Enter new medicine name (current: %s): ", prescription->medicineName);
        scanf(" %[^\n]", prescription->medicineName);
        printf("Enter new quantity (current: %d): ", prescription->quantity);
        scanf("%d", &prescription->quantity);
        printf("Enter new price (current: %.2f): ", prescription->price);
        scanf("%f", &prescription->price);
        printf("Enter new doctor name (current: %s): ", prescription->doctorName);
        scanf(" %[^\n]", prescription->doctorName);
        printf("Enter new patient name (current: %s): ", prescription->patientName);
        scanf(" %[^\n]", prescription->patientName);

        // Optional information
        printf("Do you want to update allergy information? (1 for Yes, 0 for No): ");
        int updateAllergyInfo;
        scanf("%d", &updateAllergyInfo);

        if (updateAllergyInfo) {
            printf("Enter new allergy details (current: %s): ", prescription->allergies);
            scanf(" %[^\n]", prescription->allergies);
            prescription->isEmergency = 1;
        } else {
            printf("Enter new instructions for use (current: %s): ", prescription->instructions);
            scanf(" %[^\n]", prescription->instructions);
            printf("Enter new refill count (current: %d): ", prescription->refillCount);
            scanf("%d", &prescription->refillCount);
        }

        printf("Prescription updated successfully.\n");
    } else {
        printf("Invalid prescription ID.\n");
    }
}

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

    if (id > 0 && id <= prescriptionCount) {
        for (int i = id - 1; i < prescriptionCount - 1; i++) {
            prescriptions[i] = prescriptions[i + 1];
        }
```

```
        prescriptionCount--;
        printf("Prescription deleted successfully.\n");
    } else {
        printf("Invalid prescription ID.\n");
    }
}

// Function to list all prescriptions
void listAllPrescriptions() {
    if (prescriptionCount == 0) {
        printf("No prescriptions available.\n");
        return;
    }

    for (int i = 0; i < prescriptionCount; i++) {
        printf("\nPrescription ID: %d\n", prescriptions[i].id);
        displayPrescription(&prescriptions[i]);
    }
}

// Function to display prescription details
void displayPrescription(struct Prescription *prescription) {
    printf("Medicine Name: %s\n", prescription->medicineName);
    printf("Quantity: %d\n", prescription->quantity);
    printf("Price: %.2f\n", prescription->price);
    printf("Doctor: %s\n", prescription->doctorName);
    printf("Patient: %s\n", prescription->patientName);

    // Optional/Complex Data
    if (prescription->isEmergency) {
        printf("Allergy: %s\n", prescription->allergies);
    } else {
        printf("Instructions: %s\n", prescription->instructions);
        printf("Refill Count: %d\n", prescription->refillCount);
    }
}
```

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.
Implement

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

// Define a structure to store consultation details
```

```
struct Consultation {
    char patient_name[100];
    char doctor_name[100];
    char date[20];
    char time[10];
    int consultation_fee;
    int status; // 1 for active, 0 for canceled
};

// Constants
const int CONSULTATION_FEE = 50; // Fee for consultation

// Function prototypes
void scheduleConsultation(struct Consultation consultations[], int *count);
void viewConsultation(struct Consultation consultations[], int count);
void updateConsultation(struct Consultation consultations[], int count);
void cancelConsultation(struct Consultation consultations[], int count);
void listAllConsultations(struct Consultation consultations[], int count);

int main() {
    struct Consultation consultations[100];
    int count = 0; // Track the number of scheduled consultations
    int choice;

    // Main menu loop
    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(consultations, &count);
                break;
            case 2:
                viewConsultation(consultations, count);
                break;
            case 3:
                updateConsultation(consultations, count);
                break;
            case 4:
                cancelConsultation(consultations, count);
                break;
            case 5:
                listAllConsultations(consultations, count);
                break;
            case 6:
                printf("Exiting the system.\n");
                return 0;
```

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

    return 0;
}

// Function to schedule a new consultation
void scheduleConsultation(struct Consultation consultations[], int *count) {
    struct Consultation new_consultation;

    if (*count >= 100) {
        printf("Maximum number of consultations reached.\n");
        return;
    }

    printf("Enter patient’s name: ");
    scanf(" %[^\n]s", new_consultation.patient_name); // Read full name
    printf("Enter doctor’s name: ");
    scanf(" %[^\n]s", new_consultation.doctor_name);
    printf("Enter date (DD-MM-YYYY): ");
    scanf(" %[^\n]s", new_consultation.date);
    printf("Enter time (HH:MM): ");
    scanf(" %[^\n]s", new_consultation.time);

    new_consultation.consultation_fee = CONSULTATION_FEE;
    new_consultation.status = 1; // Mark consultation as active

    consultations[*count] = new_consultation; // Store the new consultation
    (*count)++;

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

// Function to view a consultation
void viewConsultation(struct Consultation consultations[], int count) {
    int i;
    char patient_name[100];

    printf("Enter patient’s name to view consultation: ");
    scanf(" %[^\n]s", patient_name);

    for (i = 0; i < count; i++) {
        if (strcmp(consultations[i].patient_name, patient_name) == 0) {
            printf("\nConsultation Details:\n");
            printf("Patient Name: %s\n", consultations[i].patient_name);
            printf("Doctor Name: %s\n", consultations[i].doctor_name);
            printf("Date: %s\n", consultations[i].date);
            printf("Time: %s\n", consultations[i].time);
            printf("Consultation Fee: $%d\n", consultations[i].consultation_fee);
            printf("Status: %s\n", consultations[i].status ? "Active" : "Canceled");
            return;
        }
    }
```

```
    printf("No consultation found for the given patient.\n");
}

// Function to update a consultation
void updateConsultation(struct Consultation consultations[], int count) {
    int i;
    char patient_name[100];

    printf("Enter patient's name to update consultation: ");
    scanf(" %[^\n]s", patient_name);

    for (i = 0; i < count; i++) {
        if (strcmp(consultations[i].patient_name, patient_name) == 0) {
            printf("Enter new doctor's name: ");
            scanf(" %[^\n]s", consultations[i].doctor_name);
            printf("Enter new date (DD-MM-YYYY): ");
            scanf(" %[^\n]s", consultations[i].date);
            printf("Enter new time (HH:MM): ");
            scanf(" %[^\n]s", consultations[i].time);
            printf("Consultation updated successfully!\n");
            return;
        }
    }

    printf("No consultation found for the given patient.\n");
}

// Function to cancel a consultation
void cancelConsultation(struct Consultation consultations[], int count) {
    int i;
    char patient_name[100];

    printf("Enter patient's name to cancel consultation: ");
    scanf(" %[^\n]s", patient_name);

    for (i = 0; i < count; i++) {
        if (strcmp(consultations[i].patient_name, patient_name) == 0) {
            consultations[i].status = 0; // Mark consultation as canceled
            printf("Consultation canceled successfully!\n");
            return;
        }
    }

    printf("No consultation found for the given patient.\n");
}

// Function to list all consultations
void listAllConsultations(struct Consultation consultations[], int count) {
    int i;

    if (count == 0) {
        printf("No consultations scheduled.\n");
        return;
    }
```

```
    printf("\nList of All Consultations:\n");
    for (i = 0; i < count; i++) {
        printf("\nPatient Name: %s\n", consultations[i].patient_name);
        printf("Doctor Name: %s\n", consultations[i].doctor_name);
        printf("Date: %s\n", consultations[i].date);
        printf("Time: %s\n", consultations[i].time);
        printf("Consultation Fee: $%d\n", consultations[i].consultation_fee);
        printf("Status: %s\n", consultations[i].status ? "Active" : "Canceled");
    }
}
```

CREATING A LINKED LIST

==========

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

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

void create(int [], int);
void display(struct Node *);

int main()
{
    int A[] = {1,2,3,4,5};
    create(A,5);
    display(first);

    return 0;
}

void create(int A[], int n){
    int i;
    struct Node *temp, *last;
    first = (struct Node*)malloc(sizeof(struct Node));
    first->data = A[0];
    first->next = NULL;
    last = first;
    for(i = 1;i<n;i++){
        temp = (struct Node*)malloc(sizeof(struct Node));
        temp->data = A[i];
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }

}

void display(struct Node *p){
    while(p!=NULL){
        printf("%d -> ",p->data);
        p = p->next;
```

```
    }

}
```

INSERTION

==========

1.VALUE AT BEGINING

==========

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

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

void create(int [], int);
void display(struct Node *);
void Insert(struct Node*,int,int);

int main()
{
    int A[] = {1,2,3,4,5};
    create(A,5);
    display(first);
    Insert(first,0,6);
    printf("\n");
    display(first);

    return 0;
}

void create(int A[], int n){
    int i;
    struct Node *temp, *last;
    first = (struct Node*)malloc(sizeof(struct Node));
    first->data = A[0];
    first->next = NULL;
    last = first;
    for(i = 1;i<n;i++){
        temp = (struct Node*)malloc(sizeof(struct Node));
        temp->data = A[i];
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }

}

void display(struct Node *p){
    while(p!=NULL){
        printf("%d -> ",p->data);
        p = p->next;
    }
```

```
}
void Insert(struct Node *p,int index,int x){
    struct Node *temp;

    int i;
    temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data=x;
    if(index==0){
        temp->next=first;
        first=temp;
    }
    else{
        for(i=0;i<(index-1);i++){
            p=p->next;
        }
        temp->next=p->next;
        p->next=temp;
    }
}
```

2.INSERTING BETWEEN

==========

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

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

void create(int [], int);
void display(struct Node *);
void Insert(struct Node*,int,int);

int main()
{
    int A[] = {1,2,3,4,5};
    create(A,5);
    display(first);
    Insert(first,4,6);
    printf("\n");
    display(first);

    return 0;
}

void create(int A[], int n){
    int i;
    struct Node *temp, *last;
    first = (struct Node*)malloc(sizeof(struct Node));
    first->data = A[0];
    first->next = NULL;
    last = first;
    for(i = 1;i<n;i++){
```

```
        temp = (struct Node*)malloc(sizeof(struct Node));
        temp->data = A[i];
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }

}

void display(struct Node *p){
    while(p!=NULL){
        printf("%d -> ",p->data);
        p = p->next;
    }

}
void Insert(struct Node *p,int index,int x){
    struct Node *temp;

    int i;
    temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data=x;
    if(index==0){
        temp->next=first;
        first=temp;
    }
    else{
        for(i=0;i<(index-1);i++){
            p=p->next;
        }
        temp->next=p->next;
        p->next=temp;
    }
}
```

3.INSERTING AT LAST

==========

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

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

void create(int [], int);
void display(struct Node *);
void Insert(struct Node*,int,int);

int main()
{
    int A[] = {1,2,3,4,5};
    create(A,5);
    display(first);
```

```
    Insert(first,5,6);
    printf("\n");
    display(first);

    return 0;
}

void create(int A[], int n){
    int i;
    struct Node *temp, *last;
    first = (struct Node*)malloc(sizeof(struct Node));
    first->data = A[0];
    first->next = NULL;
    last = first;
    for(i = 1;i<n;i++){
        temp = (struct Node*)malloc(sizeof(struct Node));
        temp->data = A[i];
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }

}

void display(struct Node *p){
    while(p!=NULL){
        printf("%d -> ",p->data);
        p = p->next;
    }

}
void Insert(struct Node *p,int index,int x){
    struct Node *temp;

    int i;
    temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data=x;
    if(index==0){
        temp->next=first;
        first=temp;
    }
    else{
        for(i=0;i<(index-1);i++){
            p=p->next;
        }
        temp->next=p->next;
        p->next=temp;
    }
}
```

USING INSERT FUNCTION ONLY

==========

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

```
struct Node{
    int data;
    struct Node *next;
}*first = NULL;

void create(int [], int);
void display(struct Node *);
void Insert(struct Node*,int,int);

int main()
{
    int A[] = {1,2,3,4,5};
    //create(A,5);
    //display(first);
    Insert(first,0,1);
    Insert(first,1,2);
    Insert(first,2,3);

    printf("\n");
    display(first);

    return 0;
}

void create(int A[], int n){
    int i;
    struct Node *temp, *last;
    first = (struct Node*)malloc(sizeof(struct Node));
    first->data = A[0];
    first->next = NULL;
    last = first;
    for(i = 1;i<n;i++){
        temp = (struct Node*)malloc(sizeof(struct Node));
        temp->data = A[i];
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }

}

void display(struct Node *p){
    while(p!=NULL){
        printf("%d -> ",p->data);
        p = p->next;
    }

}
void Insert(struct Node *p,int index,int x){
    struct Node *temp;

    int i;
    temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data=x;
```

```
    if(index==0){
        temp->next=first;
        first=temp;
    }
    else{
        for(i=0;i<(index-1);i++){
            p=p->next;
        }
        temp->next=p->next;
        p->next=temp;
    }
}
```