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:

1. Add New Patient
2. View Patient Details
3. Update Patient Information
4. Delete Patient Record
5. List All Patients
6. Exit

Requirements:

7. Use variables to store patient details.
8. Utilize static and const for immutable data such as hospital name.
9. Implement switch case for menu selection.
10. Employ loops for iterative tasks like listing patients.
11. Use pointers for dynamic memory allocation.
12. Implement functions for CRUD operations.
13. Utilize arrays for storing multiple patient records.
14. Use structures for organizing patient data.
15. Apply nested structures for detailed medical history.
16. Use unions for optional data fields.
17. Employ nested unions for multi-type data entries.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>
```

```
// Define constants
static const char HOSPITAL_NAME[] = "HealthCare Hospital";

// Structure definitions
typedef struct {
    char illness[100];
    char treatment[100];
} MedicalHistory;

typedef union {
    char allergies[100];
    char surgeries[100];
} OptionalData;

typedef struct {
    char name[100];
    int age;
    char gender[10];
    MedicalHistory medicalHistory;
    OptionalData optionalData;
    char medications[100];
} Patient;

// Function prototypes
void addPatient(Patient patients[], int *count);
void viewPatient(Patient patients[], int count);
void updatePatient(Patient patients[], int count);
void deletePatient(Patient patients[], int *count);
```

```
void listPatients(Patient patients[], int count);


int main() {
    Patient patients[100];
    int count = 0;
    int choice;


    while (1) {
        printf("\n%s - Patient Information Management System\n", HOSPITAL_NAME);
        printf("1. Add New Patient\n2. View Patient Details\n3. Update Patient Information\n4. Delete Patient Record\n5. List All Patients\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);


        switch (choice) {
            case 1:
                addPatient(patients, &count);
                break;
            case 2:
                viewPatient(patients, count);
                break;
            case 3:
                updatePatient(patients, count);
                break;
            case 4:
                deletePatient(patients, &count);
                break;
            case 5:
```

```
            listPatients(patients, count);
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

void addPatient(Patient patients[], int *count) {
   if (*count >= 100) {
      printf("Patient list is full!\n");
      return;
   }

   printf("Enter patient name: ");
   scanf("%s", patients[*count].name);

   printf("Enter patient age: ");
   scanf("%d", &patients[*count].age);

   printf("Enter patient gender: ");
   scanf("%s", patients[*count].gender);
```

```
    printf("Enter medical history (illness): ");
    scanf("%s", patients[*count].medicalHistory.illness);

    printf("Enter medical history (treatment): ");
    scanf("%s", patients[*count].medicalHistory.treatment);

    printf("Enter optional data (allergies or surgeries): ");
    scanf("%s", patients[*count].optionalData.allergies);

    printf("Enter current medications: ");
    scanf("%s", patients[*count].medications);

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

void viewPatient(Patient patients[], int count) {
    if (count == 0) {
        printf("No patients to display!\n");
        return;
    }

    int index;
    printf("Enter patient index to view: ");
    scanf("%d", &index);

    if (index < 0 || index >= count) {
        printf("Invalid index!\n");
```

```
        return;
    }

    printf("Patient Name: %s\n", patients[index].name);
    printf("Age: %d\n", patients[index].age);
    printf("Gender: %s\n", patients[index].gender);
    printf("Medical History - Illness: %s\n", patients[index].medicalHistory.illness);
    printf("Medical History - Treatment: %s\n", patients[index].medicalHistory.treatment);
    printf("Optional Data: %s\n", patients[index].optionalData.allergies);
    printf("Current Medications: %s\n", patients[index].medications);
}

void updatePatient(Patient patients[], int count) {
    if (count == 0) {
        printf("No patients to update!\n");
        return;
    }

    int index;
    printf("Enter patient index to update: ");
    scanf("%d", &index);

    if (index < 0 || index >= count) {
        printf("Invalid index!\n");
        return;
    }

    printf("Update patient name (current: %s): ", patients[index].name);
```

```
    scanf("%s", patients[index].name);

    printf("Update patient age (current: %d): ", patients[index].age);
    scanf("%d", &patients[index].age);

    printf("Update patient gender (current: %s): ", patients[index].gender);
    scanf("%s", patients[index].gender);

    printf("Update medical history (illness) (current: %s): ",
patients[index].medicalHistory.illness);
    scanf("%s", patients[index].medicalHistory.illness);

    printf("Update medical history (treatment) (current: %s): ",
patients[index].medicalHistory.treatment);
    scanf("%s", patients[index].medicalHistory.treatment);

    printf("Update optional data (current: %s): ", patients[index].optionalData.allergies);
    scanf("%s", patients[index].optionalData.allergies);

    printf("Update current medications (current: %s): ", patients[index].medications);
    scanf("%s", patients[index].medications);

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

void deletePatient(Patient patients[], int *count) {
    if (*count == 0) {
        printf("No patients to delete!\n");
        return;
```

```
    }

    int index;
    printf("Enter patient index to delete: ");
    scanf("%d", &index);

    if (index < 0 || index >= *count) {
        printf("Invalid index!\n");
        return;
    }

    for (int i = index; i < *count - 1; i++) {
        patients[i] = patients[i + 1];
    }

    (*count)--;
    printf("Patient record deleted successfully!\n");
}

void listPatients(Patient patients[], int count) {
    if (count == 0) {
        printf("No patients to list!\n");
        return;
    }

    for (int i = 0; i < count; i++) {
        printf("%d. %s\n", i, patients[i].name);
    }
```

}

Problem 2: Hospital Inventory Management

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

Menu Options:

1. Add Inventory Item
2. View Inventory Item
3. Update Inventory Item
4. Delete Inventory Item
5. List All Inventory Items
6. Exit

Requirements:

7. Declare variables for inventory details.
8. Use static and const for fixed supply details.
9. Implement switch case for different operations like adding, deleting, and viewing inventory.
10. Utilize loops for repetitive inventory checks.
11. Use pointers to handle inventory records.
12. Create functions for managing inventory.
13. Use arrays to store inventory items.
14. Define structures for each supply item.
15. Use nested structures for detailed item specifications.
16. Employ unions for variable item attributes.
17. Implement nested unions for complex item data types.

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

```
#include <string.h>

// Define constants
static const char HOSPITAL_NAME[] = "HealthCare Hospital";

// Structure definitions
typedef struct {
    char manufacturer[100];
    char expiryDate[20];
} ItemDetails;

typedef union {
    char batchNumber[20];
    char serialNumber[20];
} OptionalAttributes;

typedef struct {
    char name[100];
    int quantity;
    ItemDetails details;
    OptionalAttributes attributes;
} InventoryItem;

// Function prototypes
void addItem(InventoryItem items[], int *count);
void viewItem(InventoryItem items[], int count);
void updateItem(InventoryItem items[], int count);
void deleteItem(InventoryItem items[], int *count);
```

```
void listItems(InventoryItem items[], int count);

int main() {
    InventoryItem items[100];
    int count = 0;
    int choice;

    while (1) {
        printf("\n%s - Hospital Inventory Management System\n", HOSPITAL_NAME);
        printf("1. Add Inventory Item\n2. View Inventory Item\n3. Update Inventory Item\n4. Delete Inventory Item\n5. List All Inventory Items\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addItem(items, &count);
                break;
            case 2:
                viewItem(items, count);
                break;
            case 3:
                updateItem(items, count);
                break;
            case 4:
                deleteItem(items, &count);
                break;
            case 5:
```

```
            listItems(items, count);
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

void addItem(InventoryItem items[], int *count) {
   if (*count >= 100) {
      printf("Inventory list is full!\n");
      return;
   }

   printf("Enter item name: ");
   scanf("%s", items[*count].name);

   printf("Enter item quantity: ");
   scanf("%d", &items[*count].quantity);

   printf("Enter manufacturer: ");
   scanf("%s", items[*count].details.manufacturer);
```

```
    printf("Enter expiry date: ");
    scanf("%s", items[*count].details.expiryDate);

    printf("Enter optional attribute (batch number or serial number): ");
    scanf("%s", items[*count].attributes.batchNumber);

    (*count)++;
    printf("Item added successfully!\n");
}

void viewItem(InventoryItem items[], int count) {
    if (count == 0) {
        printf("No items to display!\n");
        return;
    }

    int index;
    printf("Enter item index to view: ");
    scanf("%d", &index);

    if (index < 0 || index >= count) {
        printf("Invalid index!\n");
        return;
    }

    printf("Item Name: %s\n", items[index].name);
    printf("Quantity: %d\n", items[index].quantity);
    printf("Manufacturer: %s\n", items[index].details.manufacturer);
```

```
    printf("Expiry Date: %s\n", items[index].details.expiryDate);
    printf("Optional Attribute: %s\n", items[index].attributes.batchNumber);
}

void updateItem(InventoryItem items[], int count) {
    if (count == 0) {
        printf("No items to update!\n");
        return;
    }

    int index;
    printf("Enter item index to update: ");
    scanf("%d", &index);

    if (index < 0 || index >= count) {
        printf("Invalid index!\n");
        return;
    }

    printf("Update item name (current: %s): ", items[index].name);
    scanf("%s", items[index].name);

    printf("Update item quantity (current: %d): ", items[index].quantity);
    scanf("%d", &items[index].quantity);

    printf("Update manufacturer (current: %s): ", items[index].details.manufacturer);
    scanf("%s", items[index].details.manufacturer);
```

```
    printf("Update expiry date (current: %s): ", items[index].details.expiryDate);
    scanf("%s", items[index].details.expiryDate);

    printf("Update optional attribute (current: %s): ", items[index].attributes.batchNumber);
    scanf("%s", items[index].attributes.batchNumber);

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

void deleteItem(InventoryItem items[], int *count) {
    if (*count == 0) {
        printf("No items to delete!\n");
        return;
    }

    int index;
    printf("Enter item index to delete: ");
    scanf("%d", &index);

    if (index < 0 || index >= *count) {
        printf("Invalid index!\n");
        return;
    }

    for (int i = index; i < *count - 1; i++) {
        items[i] = items[i + 1];
    }
```

```
    (*count)--;
    printf("Item record deleted successfully!\n");
}

void listItems(InventoryItem items[], int count) {
    if (count == 0) {
        printf("No items to list!\n");
        return;
    }

    for (int i = 0; i < count; i++) {
        printf("%d. %s - Quantity: %d\n", i, items[i].name, items[i].quantity);
    }
}
```

Problem 3: Medical Appointment Scheduling System

Description: Develop a system to manage patient appointments.

Menu Options:

1. Schedule Appointment
2. View Appointment
3. Update Appointment
4. Cancel Appointment
5. List All Appointments
6. Exit

Requirements:

7. Use variables for appointment details.
8. Apply static and const for non-changing data like clinic hours.
9. Implement switch case for appointment operations.
10. Utilize loops for scheduling.
11. Use pointers for dynamic data manipulation.
12. Create functions for appointment handling.
13. Use arrays for storing appointments.
14. Define structures for appointment details.
15. Employ nested structures for detailed doctor and patient information.
16. Utilize unions for optional appointment data.
17. Apply nested unions for complex appointment data.

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

// Define constants
static const char CLINIC_HOURS[] = "9 AM to 5 PM";
```

```
// Structure definitions
typedef struct {
    char doctorName[100];
    char specialization[100];
} Doctor;

typedef struct {
    char patientName[100];
    int age;
    char gender[10];
} Patient;

typedef struct {
    char date[20];
    char time[10];
} AppointmentDetails;

typedef union {
    char notes[200];
    char additionalInfo[200];
} OptionalAppointmentData;

typedef struct {
    Doctor doctor;
    Patient patient;
    AppointmentDetails details;
    OptionalAppointmentData optionalData;
} Appointment;
```

```
// Function prototypes
void scheduleAppointment(Appointment appointments[], int *count);
void viewAppointment(Appointment appointments[], int count);
void updateAppointment(Appointment appointments[], int count);
void cancelAppointment(Appointment appointments[], int *count);
void listAppointments(Appointment appointments[], int count);

int main() {
    Appointment appointments[100];
    int count = 0;
    int choice;

    while (1) {
        printf("\nClinic Hours: %s - Medical Appointment Scheduling System\n",
CLINIC_HOURS);
        printf("1. Schedule Appointment\n2. View Appointment\n3. Update Appointment\n4.
Cancel Appointment\n5. List All Appointments\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                scheduleAppointment(appointments, &count);
                break;
            case 2:
                viewAppointment(appointments, count);
                break;
            case 3:
```

```
            updateAppointment(appointments, count);
            break;
        case 4:
            cancelAppointment(appointments, &count);
            break;
        case 5:
            listAppointments(appointments, count);
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

void scheduleAppointment(Appointment appointments[], int *count) {
   if (*count >= 100) {
      printf("Appointment list is full!\n");
      return;
   }

   printf("Enter doctor's name: ");
   scanf("%s", appointments[*count].doctor.doctorName);
```

```
    printf("Enter doctor's specialization: ");
    scanf("%s", appointments[*count].doctor.specialization);

    printf("Enter patient's name: ");
    scanf("%s", appointments[*count].patient.patientName);

    printf("Enter patient's age: ");
    scanf("%d", &appointments[*count].patient.age);

    printf("Enter patient's gender: ");
    scanf("%s", appointments[*count].patient.gender);

    printf("Enter appointment date (DD/MM/YYYY): ");
    scanf("%s", appointments[*count].details.date);

    printf("Enter appointment time (HH:MM): ");
    scanf("%s", appointments[*count].details.time);

    printf("Enter optional data (notes or additional info): ");
    scanf("%s", appointments[*count].optionalData.notes);

    (*count)++;
    printf("Appointment scheduled successfully!\n");
}

void viewAppointment(Appointment appointments[], int count) {
    if (count == 0) {
        printf("No appointments to display!\n");
```

```
        return;
    }

    int index;
    printf("Enter appointment index to view: ");
    scanf("%d", &index);

    if (index < 0 || index >= count) {
        printf("Invalid index!\n");
        return;
    }

    printf("Doctor's Name: %s\n", appointments[index].doctor.doctorName);
    printf("Specialization: %s\n", appointments[index].doctor.specialization);
    printf("Patient's Name: %s\n", appointments[index].patient.patientName);
    printf("Age: %d\n", appointments[index].patient.age);
    printf("Gender: %s\n", appointments[index].patient.gender);
    printf("Appointment Date: %s\n", appointments[index].details.date);
    printf("Appointment Time: %s\n", appointments[index].details.time);
    printf("Optional Data: %s\n", appointments[index].optionalData.notes);
}

void updateAppointment(Appointment appointments[], int count) {
    if (count == 0) {
        printf("No appointments to update!\n");
        return;
    }
```

```
    int index;
    printf("Enter appointment index to update: ");
    scanf("%d", &index);

    if (index < 0 || index >= count) {
        printf("Invalid index!\n");
        return;
    }

    printf("Update doctor's name (current: %s): ", appointments[index].doctor.doctorName);
    scanf("%s", appointments[index].doctor.doctorName);

    printf("Update doctor's specialization (current: %s): ",
appointments[index].doctor.specialization);
    scanf("%s", appointments[index].doctor.specialization);

    printf("Update patient's name (current: %s): ",
appointments[index].patient.patientName);
    scanf("%s", appointments[index].patient.patientName);

    printf("Update patient's age (current: %d): ", appointments[index].patient.age);
    scanf("%d", &appointments[index].patient.age);

    printf("Update patient's gender (current: %s): ", appointments[index].patient.gender);
    scanf("%s", appointments[index].patient.gender);

    printf("Update appointment date (current: %s): ", appointments[index].details.date);
    scanf("%s", appointments[index].details.date);
```

```
    printf("Update appointment time (current: %s): ", appointments[index].details.time);
    scanf("%s", appointments[index].details.time);

    printf("Update optional data (current: %s): ", appointments[index].optionalData.notes);
    scanf("%s", appointments[index].optionalData.notes);

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

void cancelAppointment(Appointment appointments[], int *count) {
    if (*count == 0) {
        printf("No appointments to cancel!\n");
        return;
    }

    int index;
    printf("Enter appointment index to cancel: ");
    scanf("%d", &index);

    if (index < 0 || index >= *count) {
        printf("Invalid index!\n");
        return;
    }

    for (int i = index; i < *count - 1; i++) {
        appointments[i] = appointments[i + 1];
    }
```

```
    (*count)--;
    printf("Appointment cancelled successfully!\n");
}

void listAppointments(Appointment appointments[], int count) {
    if (count == 0) {
        printf("No appointments to list!\n");
        return;
    }

    for (int i = 0; i < count; i++) {
        printf("%d. Dr. %s with %s on %s at %s\n", i, appointments[i].doctor.doctorName,
appointments[i].patient.patientName, appointments[i].details.date,
appointments[i].details.time);
    }
}
```

Problem 4: Patient Billing System

Description: Create a billing system for patients.

Menu Options:

1. Generate Bill
2. View Bill
3. Update Bill
4. Delete Bill
5. List All Bills
6. Exit

Requirements:

7. Declare variables for billing information.
8. Use static and const for fixed billing rates.
9. Implement switch case for billing operations.
10. Utilize loops for generating bills.
11. Use pointers for bill calculations.
12. Create functions for billing processes.
13. Use arrays for storing billing records.
14. Define structures for billing components.
15. Employ nested structures for detailed billing breakdown.
16. Use unions for variable billing elements.
17. Apply nested unions for complex billing scenarios.

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

// Define constants for billing rates
```

```
static const double CONSULTATION_FEE = 50.0;
static const double LAB_TEST_FEE = 30.0;
static const double MEDICATION_FEE = 20.0;

// Structure definitions
typedef struct {
    double consultationFee;
    double labTestFee;
    double medicationFee;
} BillingRates;

typedef struct {
    char description[100];
    double amount;
} BillingComponent;

typedef union {
    char insuranceProvider[100];
    char paymentMethod[100];
} OptionalBillingInfo;

typedef struct {
    int billID;
    char patientName[100];
    BillingComponent components[10];
    int componentCount;
    OptionalBillingInfo optionalInfo;
    double totalAmount;
```

```
} Bill;

// Function prototypes
void generateBill(Bill bills[], int *count);
void viewBill(Bill bills[], int count);
void updateBill(Bill bills[], int count);
void deleteBill(Bill bills[], int *count);
void listBills(Bill bills[], int count);
double calculateTotal(BillingComponent components[], int componentCount);

int main() {
    Bill bills[100];
    int count = 0;
    int choice;

    while (1) {
        printf("\nPatient Billing System\n");
        printf("1. Generate Bill\n2. View Bill\n3. Update Bill\n4. Delete Bill\n5. List All Bills\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                generateBill(bills, &count);
                break;
            case 2:
                viewBill(bills, count);
```

```
            break;
        case 3:
            updateBill(bills, count);
            break;
        case 4:
            deleteBill(bills, &count);
            break;
        case 5:
            listBills(bills, count);
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

void generateBill(Bill bills[], int *count) {
   if (*count >= 100) {
      printf("Bill list is full!\n");
      return;
   }

   bills[*count].billID = *count + 1;
```

```
    printf("Enter patient name: ");
    scanf("%s", bills[*count].patientName);

    printf("Enter number of billing components: ");
    scanf("%d", &bills[*count].componentCount);

    for (int i = 0; i < bills[*count].componentCount; i++) {
        printf("Enter description for component %d: ", i + 1);
        scanf("%s", bills[*count].components[i].description);
        printf("Enter amount for component %d: ", i + 1);
        scanf("%lf", &bills[*count].components[i].amount);
    }

    printf("Enter optional billing information (insurance provider or payment method): ");
    scanf("%s", bills[*count].optionalInfo.insuranceProvider);

    bills[*count].totalAmount = calculateTotal(bills[*count].components,
bills[*count].componentCount);

    (*count)++;
    printf("Bill generated successfully!\n");
}

void viewBill(Bill bills[], int count) {
    if (count == 0) {
        printf("No bills to display!\n");
        return;
    }
```

```
    int billID;
    printf("Enter bill ID to view: ");
    scanf("%d", &billID);

    if (billID <= 0 || billID > count) {
        printf("Invalid bill ID!\n");
        return;
    }

    int index = billID - 1;
    printf("Bill ID: %d\n", bills[index].billID);
    printf("Patient Name: %s\n", bills[index].patientName);
    printf("Components:\n");
    for (int i = 0; i < bills[index].componentCount; i++) {
        printf("%d. %s: %.2lf\n", i + 1, bills[index].components[i].description,
bills[index].components[i].amount);
    }
    printf("Optional Info: %s\n", bills[index].optionalInfo.insuranceProvider);
    printf("Total Amount: %.2lf\n", bills[index].totalAmount);
}

void updateBill(Bill bills[], int count) {
    if (count == 0) {
        printf("No bills to update!\n");
        return;
    }
```

```
    int billID;
    printf("Enter bill ID to update: ");
    scanf("%d", &billID);

    if (billID <= 0 || billID > count) {
        printf("Invalid bill ID!\n");
        return;
    }

    int index = billID - 1;
    printf("Update patient name (current: %s): ", bills[index].patientName);
    scanf("%s", bills[index].patientName);

    printf("Enter number of billing components (current: %d): ",
bills[index].componentCount);
    scanf("%d", &bills[index].componentCount);

    for (int i = 0; i < bills[index].componentCount; i++) {
        printf("Update description for component %d (current: %s): ", i + 1,
bills[index].components[i].description);
        scanf("%s", bills[index].components[i].description);
        printf("Update amount for component %d (current: %.2lf): ", i + 1,
bills[index].components[i].amount);
        scanf("%lf", &bills[index].components[i].amount);
    }

    printf("Update optional billing information (current: %s): ",
bills[index].optionalInfo.insuranceProvider);
    scanf("%s", bills[index].optionalInfo.insuranceProvider);
```

```
    bills[index].totalAmount = calculateTotal(bills[index].components,
bills[index].componentCount);

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

void deleteBill(Bill bills[], int *count) {
    if (*count == 0) {
        printf("No bills to delete!\n");
        return;
    }

    int billID;
    printf("Enter bill ID to delete: ");
    scanf("%d", &billID);

    if (billID <= 0 || billID > *count) {
        printf("Invalid bill ID!\n");
        return;
    }

    int index = billID - 1;
    for (int i = index; i < *count - 1; i++) {
        bills[i] = bills[i + 1];
    }

    (*count)--;
```

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

void listBills(Bill bills[], int count) {
    if (count == 0) {
        printf("No bills to list!\n");
        return;
    }

    for (int i = 0; i < count; i++) {
        printf("%d. Bill ID: %d, Patient Name: %s, Total Amount: %.2lf\n", i + 1, bills[i].billID,
bills[i].patientName, bills[i].totalAmount);
    }
}

double calculateTotal(BillingComponent components[], int componentCount) {
    double total = 0.0;
    for (int i = 0; i < componentCount; i++) {
        total += components[i].amount;
    }
    return total;
}
```

Problem 5: Medical Test Result Management

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

Menu Options:

1. Add Test Result
2. View Test Result
3. Update Test Result
4. Delete Test Result
5. List All Test Results
6. Exit

Requirements:

7. Declare variables for test results.
8. Use static and const for standard test ranges.
9. Implement switch case for result operations.
10. Utilize loops for result input and output.
11. Use pointers for handling result data.
12. Create functions for result management.
13. Use arrays for storing test results.
14. Define structures for test result details.
15. Employ nested structures for detailed test parameters.
16. Utilize unions for optional test data.
17. Apply nested unions for complex test result data.

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

// Define constants for standard test ranges
static const double MIN_BLOOD_SUGAR = 70.0;
```

```
static const double MAX_BLOOD_SUGAR = 140.0;
static const double MIN_BLOOD_PRESSURE = 90.0;
static const double MAX_BLOOD_PRESSURE = 120.0;

// Structure definitions
typedef struct {
    double bloodSugar;
    double bloodPressure;
} TestParameters;

typedef union {
    char notes[200];
    char additionalInfo[200];
} OptionalTestData;

typedef struct {
    int testID;
    char patientName[100];
    TestParameters parameters;
    OptionalTestData optionalData;
} TestResult;

// Function prototypes
void addTestResult(TestResult results[], int *count);
void viewTestResult(TestResult results[], int count);
void updateTestResult(TestResult results[], int count);
void deleteTestResult(TestResult results[], int *count);
void listTestResults(TestResult results[], int count);
```

```
int main() {
    TestResult results[100];
    int count = 0;
    int choice;

    while (1) {
        printf("\nMedical Test Result Management System\n");
        printf("1. Add Test Result\n2. View Test Result\n3. Update Test Result\n4. Delete Test Result\n5. List All Test Results\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addTestResult(results, &count);
                break;
            case 2:
                viewTestResult(results, count);
                break;
            case 3:
                updateTestResult(results, count);
                break;
            case 4:
                deleteTestResult(results, &count);
                break;
            case 5:
                listTestResults(results, count);
```

```
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

void addTestResult(TestResult results[], int *count) {
   if (*count >= 100) {
      printf("Test result list is full!\n");
      return;
   }

   results[*count].testID = *count + 1;
   printf("Enter patient name: ");
   scanf("%s", results[*count].patientName);

   printf("Enter blood sugar level: ");
   scanf("%lf", &results[*count].parameters.bloodSugar);

   printf("Enter blood pressure level: ");
   scanf("%lf", &results[*count].parameters.bloodPressure);
```

```
    printf("Enter optional test data (notes or additional info): ");
    scanf("%s", results[*count].optionalData.notes);

    (*count)++;
    printf("Test result added successfully!\n");
}

void viewTestResult(TestResult results[], int count) {
    if (count == 0) {
        printf("No test results to display!\n");
        return;
    }

    int testID;
    printf("Enter test ID to view: ");
    scanf("%d", &testID);

    if (testID <= 0 || testID > count) {
        printf("Invalid test ID!\n");
        return;
    }

    int index = testID - 1;
    printf("Test ID: %d\n", results[index].testID);
    printf("Patient Name: %s\n", results[index].patientName);
    printf("Blood Sugar Level: %.2lf\n", results[index].parameters.bloodSugar);
    printf("Blood Pressure Level: %.2lf\n", results[index].parameters.bloodPressure);
    printf("Optional Data: %s\n", results[index].optionalData.notes);
```

```
}

void updateTestResult(TestResult results[], int count) {
    if (count == 0) {
        printf("No test results to update!\n");
        return;
    }

    int testID;
    printf("Enter test ID to update: ");
    scanf("%d", &testID);

    if (testID <= 0 || testID > count) {
        printf("Invalid test ID!\n");
        return;
    }

    int index = testID - 1;
    printf("Update patient name (current: %s): ", results[index].patientName);
    scanf("%s", results[index].patientName);

    printf("Update blood sugar level (current: %.2lf): ", results[index].parameters.bloodSugar);
    scanf("%lf", &results[index].parameters.bloodSugar);

    printf("Update blood pressure level (current: %.2lf): ",
results[index].parameters.bloodPressure);
    scanf("%lf", &results[index].parameters.bloodPressure);
```

```
    printf("Update optional test data (current: %s): ", results[index].optionalData.notes);
    scanf("%s", results[index].optionalData.notes);

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

void deleteTestResult(TestResult results[], int *count) {
    if (*count == 0) {
        printf("No test results to delete!\n");
        return;
    }

    int testID;
    printf("Enter test ID to delete: ");
    scanf("%d", &testID);

    if (testID <= 0 || testID > *count) {
        printf("Invalid test ID!\n");
        return;
    }

    int index = testID - 1;
    for (int i = index; i < *count - 1; i++) {
        results[i] = results[i + 1];
    }

    (*count)--;
    printf("Test result deleted successfully!\n");
```

```
}

void listTestResults(TestResult results[], int count) {
    if (count == 0) {
        printf("No test results to list!\n");
        return;
    }

    for (int i = 0; i < count; i++) {
        printf("%d. Test ID: %d, Patient Name: %s, Blood Sugar Level: %.2lf, Blood Pressure Level: %.2lf\n", i + 1, results[i].testID, results[i].patientName, results[i].parameters.bloodSugar, results[i].parameters.bloodPressure);
    }
}
```

Problem 6: Staff Duty Roster Management

Description: Create a system to manage hospital staff duty rosters

Menu Options:

1. Add Duty Roster
2. View Duty Roster
3. Update Duty Roster
4. Delete Duty Roster
5. List All Duty Rosters
6. Exit

Requirements:

7. Use variables for staff details.
8. Apply static and const for fixed shift timings.
9. Implement switch case for roster operations.
10. Utilize loops for roster generation.
11. Use pointers for dynamic staff data.
12. Create functions for roster management.
13. Use arrays for storing staff schedules.
14. Define structures for duty details.
15. Employ nested structures for detailed duty breakdowns.
16. Use unions for optional duty attributes.
17. Apply nested unions for complex duty data.

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

```
static const char MORNING_SHIFT[] = "6 AM to 2 PM";
static const char AFTERNOON_SHIFT[] = "2 PM to 10 PM";
static const char NIGHT_SHIFT[] = "10 PM to 6 AM";


typedef struct {
    char name[100];
    char position[100];
} StaffMember;

typedef struct {
    char date[20];
    char shift[20];
} DutyDetails;

typedef union {
    char notes[200];
    char additionalInfo[200];
} OptionalDutyAttributes;

typedef struct {
    StaffMember staff;
    DutyDetails details;
    OptionalDutyAttributes optionalAttributes;
} DutyRoster;


void addDutyRoster(DutyRoster rosters[], int *count);
```

```
void viewDutyRoster(DutyRoster rosters[], int count);
void updateDutyRoster(DutyRoster rosters[], int count);
void deleteDutyRoster(DutyRoster rosters[], int *count);
void listDutyRosters(DutyRoster rosters[], int count);

int main() {
    DutyRoster rosters[100];
    int count = 0;
    int choice;

    while (1) {
        printf("\nStaff Duty Roster Management System\n");
        printf("1. Add Duty Roster\n2. View Duty Roster\n3. Update Duty Roster\n4. Delete Duty Roster\n5. List All Duty Rosters\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addDutyRoster(rosters, &count);
                break;
            case 2:
                viewDutyRoster(rosters, count);
                break;
            case 3:
                updateDutyRoster(rosters, count);
                break;
            case 4:
```

```
            deleteDutyRoster(rosters, &count);
            break;
        case 5:
            listDutyRosters(rosters, count);
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

void addDutyRoster(DutyRoster rosters[], int *count) {
   if (*count >= 100) {
      printf("Duty roster list is full!\n");
      return;
   }

   printf("Enter staff member's name: ");
   scanf("%s", rosters[*count].staff.name);

   printf("Enter staff member's position: ");
   scanf("%s", rosters[*count].staff.position);
```

```
    printf("Enter duty date (DD/MM/YYYY): ");
    scanf("%s", rosters[*count].details.date);

    printf("Enter shift (1 for Morning, 2 for Afternoon, 3 for Night): ");
    int shiftChoice;
    scanf("%d", &shiftChoice);
    switch (shiftChoice) {
        case 1:
            strcpy(rosters[*count].details.shift, MORNING_SHIFT);
            break;
        case 2:
            strcpy(rosters[*count].details.shift, AFTERNOON_SHIFT);
            break;
        case 3:
            strcpy(rosters[*count].details.shift, NIGHT_SHIFT);
            break;
        default:
            printf("Invalid shift choice!\n");
            return;
    }

    printf("Enter optional duty attributes (notes or additional info): ");
    scanf("%s", rosters[*count].optionalAttributes.notes);

    (*count)++;
    printf("Duty roster added successfully!\n");
}
```

```
void viewDutyRoster(DutyRoster rosters[], int count) {
    if (count == 0) {
        printf("No duty rosters to display!\n");
        return;
    }

    int index;
    printf("Enter duty roster index to view: ");
    scanf("%d", &index);

    if (index < 0 || index >= count) {
        printf("Invalid index!\n");
        return;
    }

    printf("Staff Member Name: %s\n", rosters[index].staff.name);
    printf("Position: %s\n", rosters[index].staff.position);
    printf("Duty Date: %s\n", rosters[index].details.date);
    printf("Shift: %s\n", rosters[index].details.shift);
    printf("Optional Attributes: %s\n", rosters[index].optionalAttributes.notes);
}

void updateDutyRoster(DutyRoster rosters[], int count) {
    if (count == 0) {
        printf("No duty rosters to update!\n");
        return;
    }
```

```
    int index;
    printf("Enter duty roster index to update: ");
    scanf("%d", &index);

    if (index < 0 || index >= count) {
        printf("Invalid index!\n");
        return;
    }

    printf("Update staff member's name (current: %s): ", rosters[index].staff.name);
    scanf("%s", rosters[index].staff.name);

    printf("Update staff member's position (current: %s): ", rosters[index].staff.position);
    scanf("%s", rosters[index].staff.position);

    printf("Update duty date (current: %s): ", rosters[index].details.date);
    scanf("%s", rosters[index].details.date);

    printf("Update shift (1 for Morning, 2 for Afternoon, 3 for Night) (current: %s): ",
rosters[index].details.shift);
    int shiftChoice;
    scanf("%d", &shiftChoice);
    switch (shiftChoice) {
        case 1:
            strcpy(rosters[index].details.shift, MORNING_SHIFT);
            break;
        case 2:
            strcpy(rosters[index].details.shift, AFTERNOON_SHIFT);
```

```
            break;
        case 3:
            strcpy(rosters[index].details.shift, NIGHT_SHIFT);
            break;
        default:
            printf("Invalid shift choice!\n");
            return;
    }

    printf("Update optional duty attributes (current: %s): ",
rosters[index].optionalAttributes.notes);
    scanf("%s", rosters[index].optionalAttributes.notes);

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

void deleteDutyRoster(DutyRoster rosters[], int *count) {
    if (*count == 0) {
        printf("No duty rosters to delete!\n");
        return;
    }

    int index;
    printf("Enter duty roster index to delete: ");
    scanf("%d", &index);

    if (index < 0 || index >= *count) {
        printf("Invalid index!\n");
```

```
        return;
    }

    for (int i = index; i < *count - 1; i++) {
        rosters[i] = rosters[i + 1];
    }

    (*count)--;
    printf("Duty roster deleted successfully!\n");
}

void listDutyRosters(DutyRoster rosters[], int count) {
    if (count == 0) {
        printf("No duty rosters to list!\n");
        return;
    }

    for (int i = 0; i < count; i++) {
        printf("%d. %s - %s on %s (Shift: %s)\n", i, rosters[i].staff.name, rosters[i].staff.position,
rosters[i].details.date, rosters[i].details.shift);
    }
}
```

Problem 7: Emergency Contact Management System

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

Menu Options:

1. Add Emergency Contact
2. View Emergency Contact
3. Update Emergency Contact
4. Delete Emergency Contact
5. List All Emergency Contacts
6. Exit

Requirements:

7. Declare variables for contact details.
8. Use static and const for non-changing contact data.
9. Implement switch case for contact operations.
10. Utilize loops for contact handling.
11. Use pointers for dynamic memory allocation.
12. Create functions for managing contacts.
13. Use arrays for storing contacts.
14. Define structures for contact details.
15. Employ nested structures for detailed contact information.
16. Utilize unions for optional contact data.
17. Apply nested unions for complex contact entries.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>
```

```
static const char HOSPITAL_NAME[] = "HealthCare Hospital";



typedef struct {
    char name[100];
    char relationship[50];
} BasicContactInfo;

typedef struct {
    char address[200];
    char email[100];
} ContactDetails;

typedef union {
    char notes[200];
    char additionalInfo[200];
} OptionalContactData;

typedef struct {
    BasicContactInfo basicInfo;
    ContactDetails contactDetails;
    OptionalContactData optionalData;
} EmergencyContact;



void addContact(EmergencyContact contacts[], int *count);
void viewContact(EmergencyContact contacts[], int count);
void updateContact(EmergencyContact contacts[], int count);
```

```
void deleteContact(EmergencyContact contacts[], int *count);
void listContacts(EmergencyContact contacts[], int count);

int main() {
    EmergencyContact contacts[100];
    int count = 0;
    int choice;

    while (1) {
        printf("\n%s - Emergency Contact Management System\n", HOSPITAL_NAME);
        printf("1. Add Emergency Contact\n2. View Emergency Contact\n3. Update Emergency Contact\n4. Delete Emergency Contact\n5. List All Emergency Contacts\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addContact(contacts, &count);
                break;
            case 2:
                viewContact(contacts, count);
                break;
            case 3:
                updateContact(contacts, count);
                break;
            case 4:
                deleteContact(contacts, &count);
                break;
```

```
            case 5:
                listContacts(contacts, count);
                break;
            case 6:
                printf("Exiting the system...\n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}

void addContact(EmergencyContact contacts[], int *count) {
    if (*count >= 100) {
        printf("Contact list is full!\n");
        return;
    }

    printf("Enter contact name: ");
    scanf("%s", contacts[*count].basicInfo.name);

    printf("Enter relationship: ");
    scanf("%s", contacts[*count].basicInfo.relationship);

    printf("Enter address: ");
    scanf("%s", contacts[*count].contactDetails.address);
```

```
    printf("Enter email: ");
    scanf("%s", contacts[*count].contactDetails.email);

    printf("Enter optional data (notes or additional info): ");
    scanf("%s", contacts[*count].optionalData.notes);

    (*count)++;
    printf("Contact added successfully!\n");
}

void viewContact(EmergencyContact contacts[], int count) {
    if (count == 0) {
        printf("No contacts to display!\n");
        return;
    }

    int index;
    printf("Enter contact index to view: ");
    scanf("%d", &index);

    if (index < 0 || index >= count) {
        printf("Invalid index!\n");
        return;
    }

    printf("Name: %s\n", contacts[index].basicInfo.name);
    printf("Relationship: %s\n", contacts[index].basicInfo.relationship);
```

```
    printf("Address: %s\n", contacts[index].contactDetails.address);
    printf("Email: %s\n", contacts[index].contactDetails.email);
    printf("Optional Data: %s\n", contacts[index].optionalData.notes);
}

void updateContact(EmergencyContact contacts[], int count) {
    if (count == 0) {
        printf("No contacts to update!\n");
        return;
    }

    int index;
    printf("Enter contact index to update: ");
    scanf("%d", &index);

    if (index < 0 || index >= count) {
        printf("Invalid index!\n");
        return;
    }

    printf("Update contact name (current: %s): ", contacts[index].basicInfo.name);
    scanf("%s", contacts[index].basicInfo.name);

    printf("Update relationship (current: %s): ", contacts[index].basicInfo.relationship);
    scanf("%s", contacts[index].basicInfo.relationship);

    printf("Update address (current: %s): ", contacts[index].contactDetails.address);
    scanf("%s", contacts[index].contactDetails.address);
```

```
    printf("Update email (current: %s): ", contacts[index].contactDetails.email);
    scanf("%s", contacts[index].contactDetails.email);

    printf("Update optional data (current: %s): ", contacts[index].optionalData.notes);
    scanf("%s", contacts[index].optionalData.notes);

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

void deleteContact(EmergencyContact contacts[], int *count) {
    if (*count == 0) {
        printf("No contacts to delete!\n");
        return;
    }

    int index;
    printf("Enter contact index to delete: ");
    scanf("%d", &index);

    if (index < 0 || index >= *count) {
        printf("Invalid index!\n");
        return;
    }

    for (int i = index; i < *count - 1; i++) {
        contacts[i] = contacts[i + 1];
    }
```

```
    (*count)--;
    printf("Contact deleted successfully!\n");
}

void listContacts(EmergencyContact contacts[], int count) {
    if (count == 0) {
        printf("No contacts to list!\n");
        return;
    }

    for (int i = 0; i < count; i++) {
        printf("%d. %s - Relationship: %s, Address: %s, Email: %s\n", i,
contacts[i].basicInfo.name, contacts[i].basicInfo.relationship,
contacts[i].contactDetails.address, contacts[i].contactDetails.email);
    }
}
```

Problem 8: Medical Record Update System

Description: Create a system for updating patient medical records.

Menu Options:

1. Add Medical Record
2. View Medical Record
3. Update Medical Record
4. Delete Medical Record
5. List All Medical Records
6. Exit

Requirements:

7. Use variables for record details.
8. Apply static and const for immutable data like record ID.
9. Implement switch case for update operations.
10. Utilize loops for record updating.
11. Use pointers for handling records.
12. Create functions for record management.
13. Use arrays for storing records.
14. Define structures for record details.
15. Employ nested structures for detailed medical history.
16. Utilize unions for optional record fields.
17. Apply nested unions for complex record data.

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


static const int MAX_RECORDS = 100;
```

```
typedef struct {
    char illness[100];
    char treatment[100];
    char doctorName[100];
    char visitDate[20];
} MedicalHistory;

typedef union {
    char allergies[100];
    char previousSurgeries[100];
} OptionalRecordFields;

typedef struct {
    int recordID;
    char patientName[100];
    int age;
    char gender[10];
    MedicalHistory medicalHistory;
    OptionalRecordFields optionalFields;
} MedicalRecord;


void addMedicalRecord(MedicalRecord records[], int *count);
void viewMedicalRecord(MedicalRecord records[], int count);
void updateMedicalRecord(MedicalRecord records[], int count);
void deleteMedicalRecord(MedicalRecord records[], int *count);
```

```
void listMedicalRecords(MedicalRecord records[], int count);


int main() {
    MedicalRecord records[MAX_RECORDS];
    int count = 0;
    int choice;


    while (1) {
        printf("\nMedical Record Update System\n");
        printf("1. Add Medical Record\n2. View Medical Record\n3. Update Medical Record\n4.
Delete Medical Record\n5. List All Medical Records\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);


        switch (choice) {
            case 1:
                addMedicalRecord(records, &count);
                break;
            case 2:
                viewMedicalRecord(records, count);
                break;
            case 3:
                updateMedicalRecord(records, count);
                break;
            case 4:
                deleteMedicalRecord(records, &count);
                break;
            case 5:
```

```
            listMedicalRecords(records, count);
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

void addMedicalRecord(MedicalRecord records[], int *count) {
   if (*count >= MAX_RECORDS) {
      printf("Medical record list is full!\n");
      return;
   }

   records[*count].recordID = *count + 1;
   printf("Enter patient name: ");
   scanf("%s", records[*count].patientName);

   printf("Enter patient age: ");
   scanf("%d", &records[*count].age);

   printf("Enter patient gender: ");
   scanf("%s", records[*count].gender);
```

```
    printf("Enter medical history (illness): ");
    scanf("%s", records[*count].medicalHistory.illness);

    printf("Enter medical history (treatment): ");
    scanf("%s", records[*count].medicalHistory.treatment);

    printf("Enter doctor's name: ");
    scanf("%s", records[*count].medicalHistory.doctorName);

    printf("Enter visit date (DD/MM/YYYY): ");
    scanf("%s", records[*count].medicalHistory.visitDate);

    printf("Enter optional record fields (allergies or previous surgeries): ");
    scanf("%s", records[*count].optionalFields.allergies);

    (*count)++;
    printf("Medical record added successfully!\n");
}

void viewMedicalRecord(MedicalRecord records[], int count) {
    if (count == 0) {
        printf("No medical records to display!\n");
        return;
    }

    int recordID;
    printf("Enter record ID to view: ");
```

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

    if (recordID <= 0 || recordID > count) {
        printf("Invalid record ID!\n");
        return;
    }

    int index = recordID - 1;
    printf("Record ID: %d\n", records[index].recordID);
    printf("Patient Name: %s\n", records[index].patientName);
    printf("Age: %d\n", records[index].age);
    printf("Gender: %s\n", records[index].gender);
    printf("Medical History - Illness: %s\n", records[index].medicalHistory.illness);
    printf("Medical History - Treatment: %s\n", records[index].medicalHistory.treatment);
    printf("Doctor's Name: %s\n", records[index].medicalHistory.doctorName);
    printf("Visit Date: %s\n", records[index].medicalHistory.visitDate);
    printf("Optional Fields: %s\n", records[index].optionalFields.allergies);
}

void updateMedicalRecord(MedicalRecord records[], int count) {
    if (count == 0) {
        printf("No medical records to update!\n");
        return;
    }

    int recordID;
    printf("Enter record ID to update: ");
    scanf("%d", &recordID);
```

```
    if (recordID <= 0 || recordID > count) {
        printf("Invalid record ID!\n");
        return;
    }

    int index = recordID - 1;
    printf("Update patient name (current: %s): ", records[index].patientName);
    scanf("%s", records[index].patientName);

    printf("Update patient age (current: %d): ", records[index].age);
    scanf("%d", &records[index].age);

    printf("Update patient gender (current: %s): ", records[index].gender);
    scanf("%s", records[index].gender);

    printf("Update medical history (illness) (current: %s): ",
records[index].medicalHistory.illness);
    scanf("%s", records[index].medicalHistory.illness);

    printf("Update medical history (treatment) (current: %s): ",
records[index].medicalHistory.treatment);
    scanf("%s", records[index].medicalHistory.treatment);

    printf("Update doctor's name (current: %s): ",
records[index].medicalHistory.doctorName);
    scanf("%s", records[index].medicalHistory.doctorName);

    printf("Update visit date (current: %s): ", records[index].medicalHistory.visitDate);
```

```
    scanf("%s", records[index].medicalHistory.visitDate);

    printf("Update optional record fields (current: %s): ",
records[index].optionalFields.allergies);
    scanf("%s", records[index].optionalFields.allergies);

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

void deleteMedicalRecord(MedicalRecord records[], int *count) {
    if (*count == 0) {
        printf("No medical records to delete!\n");
        return;
    }

    int recordID;
    printf("Enter record ID to delete: ");
    scanf("%d", &recordID);

    if (recordID <= 0 || recordID > *count) {
        printf("Invalid record ID!\n");
        return;
    }

    int index = recordID - 1;
    for (int i = index; i < *count - 1; i++) {
        records[i] = records[i + 1];
    }
```

```
    (*count)--;
    printf("Medical record deleted successfully!\n");
}

void listMedicalRecords(MedicalRecord records[], int count) {
    if (count == 0) {
        printf("No medical records to list!\n");
        return;
    }

    for (int i = 0; i < count; i++) {
        printf("%d. Record ID: %d, Patient Name: %s, Age: %d, Gender: %s, Illness: %s,
Treatment: %s, Doctor: %s, Visit Date: %s\n",i + 1, records[i].recordID,
records[i].patientName, records[i].age, records[i].gender,
records[i].medicalHistory.illness,records[i].medicalHistory.treatment,
records[i].medicalHistory.doctorName, records[i].medicalHistory.visitDate);
    }
}
```

Problem 9: Patient Diet Plan Management

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

Menu Options:

1. Add Diet Plan
2. View Diet Plan
3. Update Diet Plan
4. Delete Diet Plan
5. List All Diet Plans
6. Exit

Requirements:

7. Declare variables for diet plan details.
8. Use static and const for fixed dietary guidelines.
9. Implement switch case for diet plan operations.
10. Utilize loops for diet plan handling.
11. Use pointers for dynamic diet data.
12. Create functions for diet plan management.
13. Use arrays for storing diet plans.
14. Define structures for diet plan details.
15. Employ nested structures for detailed dietary breakdowns.
16. Use unions for optional diet attributes.
17. Apply nested unions for complex diet plan data.

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

```
static const char STANDARD_DIET[] = "Standard Diet Plan";
static const char LOW_CARB_DIET[] = "Low Carb Diet Plan";
static const char HIGH_PROTEIN_DIET[] = "High Protein Diet Plan";

typedef struct {
    char name[100];
    int age;
    char gender[10];
} PatientDetails;

typedef struct {
    char mealType[50];
    char description[200];
} MealDetails;

typedef union {
    char notes[200];
    char specialInstructions[200];
} OptionalDietAttributes;

typedef struct {
    int dietPlanID;
    PatientDetails patient;
    MealDetails meals[3]; // Breakfast, Lunch, Dinner
    OptionalDietAttributes optionalAttributes;
} DietPlan;
```

```
void addDietPlan(DietPlan plans[], int *count);
void viewDietPlan(DietPlan plans[], int count);
void updateDietPlan(DietPlan plans[], int count);
void deleteDietPlan(DietPlan plans[], int *count);
void listDietPlans(DietPlan plans[], int count);

int main() {
    DietPlan plans[100];
    int count = 0;
    int choice;

    while (1) {
        printf("\nPatient Diet Plan Management System\n");
        printf("1. Add Diet Plan\n2. View Diet Plan\n3. Update Diet Plan\n4. Delete Diet Plan\n5. List All Diet Plans\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addDietPlan(plans, &count);
                break;
            case 2:
                viewDietPlan(plans, count);
                break;
            case 3:
                updateDietPlan(plans, count);
                break;
```

```
            case 4:
                deleteDietPlan(plans, &count);
                break;
            case 5:
                listDietPlans(plans, count);
                break;
            case 6:
                printf("Exiting the system...\n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}

void addDietPlan(DietPlan plans[], int *count) {
    if (*count >= 100) {
        printf("Diet plan list is full!\n");
        return;
    }

    plans[*count].dietPlanID = *count + 1;
    printf("Enter patient name: ");
    scanf("%s", plans[*count].patient.name);

    printf("Enter patient age: ");
```

```
    scanf("%d", &plans[*count].patient.age);

    printf("Enter patient gender: ");
    scanf("%s", plans[*count].patient.gender);

    for (int i = 0; i < 3; i++) {
        printf("Enter meal type for meal %d (e.g., Breakfast, Lunch, Dinner): ", i + 1);
        scanf("%s", plans[*count].meals[i].mealType);

        printf("Enter description for %s: ", plans[*count].meals[i].mealType);
        scanf("%s", plans[*count].meals[i].description);
    }

    printf("Enter optional diet attributes (notes or special instructions): ");
    scanf("%s", plans[*count].optionalAttributes.notes);

    (*count)++;
    printf("Diet plan added successfully!\n");
}

void viewDietPlan(DietPlan plans[], int count) {
    if (count == 0) {
        printf("No diet plans to display!\n");
        return;
    }

    int dietPlanID;
    printf("Enter diet plan ID to view: ");
```

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

    if (dietPlanID <= 0 || dietPlanID > count) {
        printf("Invalid diet plan ID!\n");
        return;
    }

    int index = dietPlanID - 1;
    printf("Diet Plan ID: %d\n", plans[index].dietPlanID);
    printf("Patient Name: %s\n", plans[index].patient.name);
    printf("Age: %d\n", plans[index].patient.age);
    printf("Gender: %s\n", plans[index].patient.gender);
    for (int i = 0; i < 3; i++) {
        printf("Meal %d: %s - %s\n", i + 1, plans[index].meals[i].mealType,
plans[index].meals[i].description);
    }
    printf("Optional Attributes: %s\n", plans[index].optionalAttributes.notes);
}

void updateDietPlan(DietPlan plans[], int count) {
    if (count == 0) {
        printf("No diet plans to update!\n");
        return;
    }

    int dietPlanID;
    printf("Enter diet plan ID to update: ");
    scanf("%d", &dietPlanID);
```

```
    if (dietPlanID <= 0 || dietPlanID > count) {
        printf("Invalid diet plan ID!\n");
        return;
    }

    int index = dietPlanID - 1;
    printf("Update patient name (current: %s): ", plans[index].patient.name);
    scanf("%s", plans[index].patient.name);

    printf("Update patient age (current: %d): ", plans[index].patient.age);
    scanf("%d", &plans[index].patient.age);

    printf("Update patient gender (current: %s): ", plans[index].patient.gender);
    scanf("%s", plans[index].patient.gender);

    for (int i = 0; i < 3; i++) {
        printf("Update meal type for meal %d (current: %s): ", i + 1,
plans[index].meals[i].mealType);
        scanf("%s", plans[index].meals[i].mealType);

        printf("Update description for %s (current: %s): ", plans[index].meals[i].mealType,
plans[index].meals[i].description);
        scanf("%s", plans[index].meals[i].description);
    }

    printf("Update optional diet attributes (current: %s): ",
plans[index].optionalAttributes.notes);
    scanf("%s", plans[index].optionalAttributes.notes);
```

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

void deleteDietPlan(DietPlan plans[], int *count) {
    if (*count == 0) {
        printf("No diet plans to delete!\n");
        return;
    }

    int dietPlanID;
    printf("Enter diet plan ID to delete: ");
    scanf("%d", &dietPlanID);

    if (dietPlanID <= 0 || dietPlanID > *count) {
        printf("Invalid diet plan ID!\n");
        return;
    }

    int index = dietPlanID - 1;
    for (int i = index; i < *count - 1; i++) {
        plans[i] = plans[i + 1];
    }

    (*count)--;
    printf("Diet plan deleted successfully!\n");
}
```

```
void listDietPlans(DietPlan plans[], int count) {
    if (count == 0) {
        printf("No diet plans to list!\n");
        return;
    }

    for (int i = 0; i < count; i++) {
        printf("%d. Diet Plan ID: %d, Patient Name: %s, Age: %d, Gender: %s\n",
            i + 1, plans[i].dietPlanID, plans[i].patient.name, plans[i].patient.age,
plans[i].patient.gender);
    }
}
```

Problem 10: Surgery Scheduling System

Description: Design a system for scheduling surgeries.

Menu Options:

1. Schedule Surgery
2. View Surgery Schedule
3. Update Surgery Schedule
4. Cancel Surgery
5. List All Surgeries
6. Exit

Requirements:

7. Use variables for surgery details.
8. Apply static and const for immutable data like surgery types.
9. Implement switch case for scheduling operations.
10. Utilize loops for surgery scheduling.
11. Use pointers for handling surgery data.
12. Create functions for surgery management.
13. Use arrays for storing surgery schedules.
14. Define structures for surgery details.
15. Employ nested structures for detailed surgery information.
16. Utilize unions for optional surgery data.
17. Apply nested unions for complex surgery entries.

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

```
static const char SURGERY_TYPES[][30] = {"Appendectomy", "Cataract Surgery", "Heart Bypass Surgery", "Knee Replacement"};

typedef struct {
    char patientName[100];
    int age;
    char gender[10];
} PatientDetails;

typedef struct {
    char date[20];
    char time[10];
    char surgeonName[100];
} SurgeryDetails;

typedef union {
    char preOpInstructions[200];
    char postOpInstructions[200];
} OptionalSurgeryData;

typedef struct {
    int surgeryID;
    PatientDetails patient;
    SurgeryDetails surgery;
    OptionalSurgeryData optionalData;
    char surgeryType[30];
} Surgery;
```

```
void scheduleSurgery(Surgery surgeries[], int *count);
void viewSurgerySchedule(Surgery surgeries[], int count);
void updateSurgerySchedule(Surgery surgeries[], int count);
void cancelSurgery(Surgery surgeries[], int *count);
void listAllSurgeries(Surgery surgeries[], int count);

int main() {
    Surgery surgeries[100];
    int count = 0;
    int choice;

    while (1) {
        printf("\nSurgery Scheduling System\n");
        printf("1. Schedule Surgery\n2. View Surgery Schedule\n3. Update Surgery
Schedule\n4. Cancel Surgery\n5. List All Surgeries\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                scheduleSurgery(surgeries, &count);
                break;
            case 2:
                viewSurgerySchedule(surgeries, count);
                break;
            case 3:
                updateSurgerySchedule(surgeries, count);
```

```
            break;
        case 4:
            cancelSurgery(surgeries, &count);
            break;
        case 5:
            listAllSurgeries(surgeries, count);
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

void scheduleSurgery(Surgery surgeries[], int *count) {
   if (*count >= 100) {
      printf("Surgery schedule list is full!\n");
      return;
   }

   surgeries[*count].surgeryID = *count + 1;
   printf("Enter patient name: ");
   scanf("%s", surgeries[*count].patient.patientName);
```

```
    printf("Enter patient age: ");
    scanf("%d", &surgeries[*count].patient.age);

    printf("Enter patient gender: ");
    scanf("%s", surgeries[*count].patient.gender);

    printf("Enter surgery date (DD/MM/YYYY): ");
    scanf("%s", surgeries[*count].surgery.date);

    printf("Enter surgery time (HH:MM): ");
    scanf("%s", surgeries[*count].surgery.time);

    printf("Enter surgeon's name: ");
    scanf("%s", surgeries[*count].surgery.surgeonName);

    printf("Enter surgery type (1 for Appendectomy, 2 for Cataract Surgery, 3 for Heart Bypass
Surgery, 4 for Knee Replacement): ");
    int surgeryChoice;
    scanf("%d", &surgeryChoice);
    if (surgeryChoice < 1 || surgeryChoice > 4) {
        printf("Invalid surgery type!\n");
        return;
    }
    strcpy(surgeries[*count].surgeryType, SURGERY_TYPES[surgeryChoice - 1]);

    printf("Enter optional surgery data (pre-op or post-op instructions): ");
    scanf("%s", surgeries[*count].optionalData.preOpInstructions);
```

```
    (*count)++;
    printf("Surgery scheduled successfully!\n");
}

void viewSurgerySchedule(Surgery surgeries[], int count) {
    if (count == 0) {
        printf("No surgeries scheduled to display!\n");
        return;
    }

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

    if (surgeryID <= 0 || surgeryID > count) {
        printf("Invalid surgery ID!\n");
        return;
    }

    int index = surgeryID - 1;
    printf("Surgery ID: %d\n", surgeries[index].surgeryID);
    printf("Patient Name: %s\n", surgeries[index].patient.patientName);
    printf("Age: %d\n", surgeries[index].patient.age);
    printf("Gender: %s\n", surgeries[index].patient.gender);
    printf("Surgery Date: %s\n", surgeries[index].surgery.date);
    printf("Surgery Time: %s\n", surgeries[index].surgery.time);
    printf("Surgeon's Name: %s\n", surgeries[index].surgery.surgeonName);
    printf("Surgery Type: %s\n", surgeries[index].surgeryType);
```

```
    printf("Optional Data: %s\n", surgeries[index].optionalData.preOpInstructions);
}

void updateSurgerySchedule(Surgery surgeries[], int count) {
    if (count == 0) {
        printf("No surgeries scheduled to update!\n");
        return;
    }

    int surgeryID;
    printf("Enter surgery ID to update: ");
    scanf("%d", &surgeryID);

    if (surgeryID <= 0 || surgeryID > count) {
        printf("Invalid surgery ID!\n");
        return;
    }

    int index = surgeryID - 1;
    printf("Update patient name (current: %s): ", surgeries[index].patient.patientName);
    scanf("%s", surgeries[index].patient.patientName);

    printf("Update patient age (current: %d): ", surgeries[index].patient.age);
    scanf("%d", &surgeries[index].patient.age);

    printf("Update patient gender (current: %s): ", surgeries[index].patient.gender);
    scanf("%s", surgeries[index].patient.gender);
```

```
    printf("Update surgery date (current: %s): ", surgeries[index].surgery.date);
    scanf("%s", surgeries[index].surgery.date);

    printf("Update surgery time (current: %s): ", surgeries[index].surgery.time);
    scanf("%s", surgeries[index].surgery.time);

    printf("Update surgeon's name (current: %s): ", surgeries[index].surgery.surgeonName);
    scanf("%s", surgeries[index].surgery.surgeonName);

    printf("Update surgery type (current: %s) (1 for Appendectomy, 2 for Cataract Surgery, 3 for Heart Bypass Surgery, 4 for Knee Replacement): ", surgeries[index].surgeryType);
    int surgeryChoice;
    scanf("%d", &surgeryChoice);
    if (surgeryChoice < 1 || surgeryChoice > 4) {
        printf("Invalid surgery type!\n");
        return;
    }
    strcpy(surgeries[index].surgeryType, SURGERY_TYPES[surgeryChoice - 1]);

    printf("Update optional surgery data (current: %s): ", surgeries[index].optionalData.preOpInstructions);
    scanf("%s", surgeries[index].optionalData.preOpInstructions);

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

void cancelSurgery(Surgery surgeries[], int *count) {
    if (*count == 0) {
        printf("No surgeries scheduled to cancel!\n");
```

```
        return;
    }

    int surgeryID;
    printf("Enter surgery ID to cancel: ");
    scanf("%d", &surgeryID);

    if (surgeryID <= 0 || surgeryID > *count) {
        printf("Invalid surgery ID!\n");
        return;
    }

    int index = surgeryID - 1;
    for (int i = index; i < *count - 1; i++) {
        surgeries[i] = surgeries[i + 1];
    }

    (*count)--;
    printf("Surgery schedule canceled successfully!\n");
}

void listAllSurgeries(Surgery surgeries[], int count) {
    if (count == 0) {
        printf("No surgeries scheduled to list!\n");
        return;
    }

    for (int i = 0; i < count; i++) {
```

```
        printf("%d. Surgery ID: %d, Patient Name: %s, Age: %d, Gender: %s, Surgery Date: %s, Surgery Time: %s, Surgeon's Name: %s, Surgery Type: %s\n",
                i + 1, surgeries[i].surgeryID, surgeries[i].patient.patientName, surgeries[i].patient.age, surgeries[i].patient.gender,
                surgeries[i].surgery.date, surgeries[i].surgery.time, surgeries[i].surgery.surgeonName, surgeries[i].surgeryType);
    }
}
```

Problem 11: Prescription Management System

Description: Develop a system to manage patient prescriptions.

Menu Options:

1. Add Prescription
2. View Prescription
3. Update Prescription
4. Delete Prescription
5. List All Prescriptions
6. Exit

Requirements:

7. Declare variables for prescription details.
8. Use static and const for fixed prescription guidelines.
9. Implement switch case for prescription operations.
10. Utilize loops for prescription handling.
11. Use pointers for dynamic prescription data.
12. Create functions for prescription management.
13. Use arrays for storing prescriptions.
14. Define structures for prescription details.
15. Employ nested structures for detailed prescription information.
16. Use unions for optional prescription fields.
17. Apply nested unions for complex prescription data.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>
```

```
static const char PRESCRIPTION_GUIDELINES[] = "Take medication as prescribed. Do not exceed the recommended dose.";



typedef struct {
    char medicationName[100];
    int dosage;
    char frequency[50];
} Medication;

typedef struct {
    char patientName[100];
    int age;
    char gender[10];
} PatientDetails;

typedef union {
    char notes[200];
    char additionalInstructions[200];
} OptionalPrescriptionFields;

typedef struct {
    int prescriptionID;
    PatientDetails patient;
    Medication medications[10];
    int medicationCount;
    OptionalPrescriptionFields optionalFields;
} Prescription;
```

```
void addPrescription(Prescription prescriptions[], int *count);
void viewPrescription(Prescription prescriptions[], int count);
void updatePrescription(Prescription prescriptions[], int count);
void deletePrescription(Prescription prescriptions[], int *count);
void listPrescriptions(Prescription prescriptions[], int count);

int main() {
    Prescription prescriptions[100];
    int count = 0;
    int choice;

    while (1) {
        printf("\nPrescription Management System\n");
        printf("1. Add Prescription\n2. View Prescription\n3. Update Prescription\n4. Delete
Prescription\n5. List All Prescriptions\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addPrescription(prescriptions, &count);
                break;
            case 2:
                viewPrescription(prescriptions, count);
                break;
            case 3:
                updatePrescription(prescriptions, count);
```

```
            break;
        case 4:
            deletePrescription(prescriptions, &count);
            break;
        case 5:
            listPrescriptions(prescriptions, count);
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

void addPrescription(Prescription prescriptions[], int *count) {
   if (*count >= 100) {
      printf("Prescription list is full!\n");
      return;
   }

  // prescriptions[*count].prescriptionID = *count + 1;
   printf("Enter patient name: ");
   scanf("%s", prescriptions[*count].patient.patientName);
```

```
    printf("Enter patient age: ");
    scanf("%d", &prescriptions[*count].patient.age);

    printf("Enter patient gender: ");
    scanf("%s", prescriptions[*count].patient.gender);

    printf("Enter number of medications: ");
    scanf("%d", &prescriptions[*count].medicationCount);

    for (int i = 0; i < prescriptions[*count].medicationCount; i++) {
        printf("Enter medication name for medication %d: ", i + 1);
        scanf("%s", prescriptions[*count].medications[i].medicationName);

        printf("Enter dosage for %s: ", prescriptions[*count].medications[i].medicationName);
        scanf("%d", &prescriptions[*count].medications[i].dosage);

        printf("Enter frequency for %s: ",
prescriptions[*count].medications[i].medicationName);
        scanf("%s", prescriptions[*count].medications[i].frequency);
    }

    printf("Enter optional prescription fields (notes or additional instructions): ");
    scanf("%s", prescriptions[*count].optionalFields.notes);

    (*count)++;
    printf("Prescription added successfully!\n");
}
```

```
void viewPrescription(Prescription prescriptions[], int count) {
    if (count == 0) {
        printf("No prescriptions to display!\n");
        return;
    }

    int prescriptionID;
    printf("Enter prescription ID to view: ");
    scanf("%d", &prescriptionID);

    if (prescriptionID <= 0 || prescriptionID > count) {
        printf("Invalid prescription ID!\n");
        return;
    }

    int index = prescriptionID - 1;
    printf("Prescription ID: %d\n", prescriptions[index].prescriptionID);
    printf("Patient Name: %s\n", prescriptions[index].patient.patientName);
    printf("Age: %d\n", prescriptions[index].patient.age);
    printf("Gender: %s\n", prescriptions[index].patient.gender);
    for (int i = 0; i < prescriptions[index].medicationCount; i++) {
        printf("Medication %d: %s - Dosage: %d - Frequency: %s\n", i + 1,
prescriptions[index].medications[i].medicationName,
prescriptions[index].medications[i].dosage, prescriptions[index].medications[i].frequency);
    }
    printf("Optional Fields: %s\n", prescriptions[index].optionalFields.notes);
}

void updatePrescription(Prescription prescriptions[], int count) {
```

```
    if (count == 0) {
        printf("No prescriptions to update!\n");
        return;
    }

    int prescriptionID;
    printf("Enter prescription ID to update: ");
    scanf("%d", &prescriptionID);

    if (prescriptionID <= 0 || prescriptionID > count) {
        printf("Invalid prescription ID!\n");
        return;
    }

    int index = prescriptionID - 1;
    printf("Update patient name (current: %s): ", prescriptions[index].patient.patientName);
    scanf("%s", prescriptions[index].patient.patientName);

    printf("Update patient age (current: %d): ", prescriptions[index].patient.age);
    scanf("%d", &prescriptions[index].patient.age);

    printf("Update patient gender (current: %s): ", prescriptions[index].patient.gender);
    scanf("%s", prescriptions[index].patient.gender);

    printf("Update number of medications (current: %d): ",
prescriptions[index].medicationCount);
    scanf("%d", &prescriptions[index].medicationCount);
```

```
    for (int i = 0; i < prescriptions[index].medicationCount; i++) {
        printf("Update medication name for medication %d (current: %s): ", i + 1,
prescriptions[index].medications[i].medicationName);
        scanf("%s", prescriptions[index].medications[i].medicationName);

        printf("Update dosage for %s (current: %d): ",
prescriptions[index].medications[i].medicationName,
prescriptions[index].medications[i].dosage);
        scanf("%d", &prescriptions[index].medications[i].dosage);

        printf("Update frequency for %s (current: %s): ",
prescriptions[index].medications[i].medicationName,
prescriptions[index].medications[i].frequency);
        scanf("%s", prescriptions[index].medications[i].frequency);
    }

    printf("Update optional prescription fields (current: %s): ",
prescriptions[index].optionalFields.notes);
    scanf("%s", prescriptions[index].optionalFields.notes);

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

void deletePrescription(Prescription prescriptions[], int *count) {
    if (*count == 0) {
        printf("No prescriptions to delete!\n");
        return;
    }

    int prescriptionID;
```

```
    printf("Enter prescription ID to delete: ");
    scanf("%d", &prescriptionID);

    if (prescriptionID <= 0 || prescriptionID > *count) {
        printf("Invalid prescription ID!\n");
        return;
    }

    int index = prescriptionID - 1;
    for (int i = index; i < *count - 1; i++) {
        prescriptions[i] = prescriptions[i + 1];
    }

    (*count)--;
    printf("Prescription deleted successfully!\n");
}

void listPrescriptions(Prescription prescriptions[], int count) {
    if (count == 0) {
        printf("No prescriptions to list!\n");
        return;
    }

    for (int i = 0; i < count; i++) {
        printf("%d. Prescription ID: %d, Patient Name: %s, Age: %d, Gender: %s\n",  i + 1,
prescriptions[i].prescriptionID, prescriptions[i].patient.patientName,
prescriptions[i].patient.age, prescriptions[i].patient.gender);
    }
}
```

Problem 12: Doctor Consultation Management

Description: Create a system for managing doctor consultations.

Menu Options:

1. Schedule Consultation
2. View Consultation
3. Update Consultation
4. Cancel Consultation
5. List All Consultations
6. Exit

Requirements:

7. Use variables for consultation details.
8. Apply static and const for non-changing data like consultation fees.
9. Implement

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

static const double CONSULTATION_FEE = 100.0;

typedef struct {
    char patientName[100];
    int age;
    char gender[10];
} PatientDetails;
```

```
typedef struct {
    char date[20];
    char time[10];
    char doctorName[100];
} ConsultationDetails;

typedef union {
    char notes[200];
    char additionalInfo[200];
} OptionalConsultationData;

typedef struct {
    int consultationID;
    PatientDetails patient;
    ConsultationDetails consultation;
    OptionalConsultationData optionalData;
    double fee;
} Consultation;

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

int main() {
    Consultation consultations[100];
    int count = 0;
```

```
    int choice;

    while (1) {
        printf("\nDoctor Consultation Management System\n");
        printf("1. Schedule Consultation\n2. View Consultation\n3. Update Consultation\n4. Cancel Consultation\n5. List All Consultations\n6. 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");
                exit(0);
```

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

    return 0;
}

void scheduleConsultation(Consultation consultations[], int *count) {
    if (*count >= 100) {
        printf("Consultation schedule list is full!\n");
        return;
    }

    consultations[*count].consultationID = *count + 1;
    printf("Enter patient name: ");
    scanf("%s", consultations[*count].patient.patientName);

    printf("Enter patient age: ");
    scanf("%d", &consultations[*count].patient.age);

    printf("Enter patient gender: ");
    scanf("%s", consultations[*count].patient.gender);

    printf("Enter consultation date (DD/MM/YYYY): ");
    scanf("%s", consultations[*count].consultation.date);

    printf("Enter consultation time (HH:MM): ");
```

```
    scanf("%s", consultations[*count].consultation.time);

    printf("Enter doctor's name: ");
    scanf("%s", consultations[*count].consultation.doctorName);

    printf("Enter optional consultation data (notes or additional info): ");
    scanf("%s", consultations[*count].optionalData.notes);

    consultations[*count].fee = CONSULTATION_FEE;

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

void viewConsultation(Consultation consultations[], int count) {
    if (count == 0) {
        printf("No consultations scheduled to display!\n");
        return;
    }

    int consultationID;
    printf("Enter consultation ID to view: ");
    scanf("%d", &consultationID);

    if (consultationID <= 0 || consultationID > count) {
        printf("Invalid consultation ID!\n");
        return;
    }
```

```
    int index = consultationID - 1;
    printf("Consultation ID: %d\n", consultations[index].consultationID);
    printf("Patient Name: %s\n", consultations[index].patient.patientName);
    printf("Age: %d\n", consultations[index].patient.age);
    printf("Gender: %s\n", consultations[index].patient.gender);
    printf("Consultation Date: %s\n", consultations[index].consultation.date);
    printf("Consultation Time: %s\n", consultations[index].consultation.time);
    printf("Doctor's Name: %s\n", consultations[index].consultation.doctorName);
    printf("Optional Data: %s\n", consultations[index].optionalData.notes);
    printf("Consultation Fee: %.2lf\n", consultations[index].fee);
}

void updateConsultation(Consultation consultations[], int count) {
    if (count == 0) {
        printf("No consultations scheduled to update!\n");
        return;
    }

    int consultationID;
    printf("Enter consultation ID to update: ");
    scanf("%d", &consultationID);

    if (consultationID <= 0 || consultationID > count) {
        printf("Invalid consultation ID!\n");
        return;
    }
```

```
    int index = consultationID - 1;
    printf("Update patient name (current: %s): ", consultations[index].patient.patientName);
    scanf("%s", consultations[index].patient.patientName);

    printf("Update patient age (current: %d): ", consultations[index].patient.age);
    scanf("%d", &consultations[index].patient.age);

    printf("Update patient gender (current: %s): ", consultations[index].patient.gender);
    scanf("%s", consultations[index].patient.gender);

    printf("Update consultation date (current: %s): ", consultations[index].consultation.date);
    scanf("%s", consultations[index].consultation.date);

    printf("Update consultation time (current: %s): ", consultations[index].consultation.time);
    scanf("%s", consultations[index].consultation.time);

    printf("Update doctor's name (current: %s): ",
consultations[index].consultation.doctorName);
    scanf("%s", consultations[index].consultation.doctorName);

    printf("Update optional consultation data (current: %s): ",
consultations[index].optionalData.notes);
    scanf("%s", consultations[index].optionalData.notes);

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

void cancelConsultation(Consultation consultations[], int *count) {
    if (*count == 0) {
```

```
        printf("No consultations scheduled to cancel!\n");
        return;
    }

    int consultationID;
    printf("Enter consultation ID to cancel: ");
    scanf("%d", &consultationID);

    if (consultationID <= 0 || consultationID > *count) {
        printf("Invalid consultation ID!\n");
        return;
    }

    int index = consultationID - 1;
    for (int i = index; i < *count - 1; i++) {
        consultations[i] = consultations[i + 1];
    }

    (*count)--;
    printf("Consultation canceled successfully!\n");
}

void listAllConsultations(Consultation consultations[], int count) {
    if (count == 0) {
        printf("No consultations scheduled to list!\n");
        return;
    }
```

```
    for (int i = 0; i < count; i++) {
        printf("%d. Consultation ID: %d, Patient Name: %s, Age: %d, Gender: %s, Consultation Date: %s, Consultation Time: %s, Doctor's Name: %s, Fee: %.2lf\n",i + 1, consultations[i].consultationID, consultations[i].patient.patientName, consultations[i].patient.age, consultations[i].patient.gender,consultations[i].consultation.date, consultations[i].consultation.time, consultations[i].consultation.doctorName, consultations[i].fee);
    }
}
```

LIKNED LIST_____

Creating and Displaying

```
#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;
    }

}
```

////inserting elemnts to the linked list inbetween

```
#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;
    }

}
```

Creating a linked list with insertion function

```
#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;
    }

}
```

```
/*************************************************************************
****

Problem 1: Patient Queue Management
Description: Implement a linked list to manage a queue of patients waiting for consultation.
Operations:
Create a new patient queue.
Insert a patient into the queue.
Display the current queue of patients.
*************************************************************************
****/

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

```
#include <string.h>


struct Patient {
   char name[100];
   struct Patient *next;
} *first = NULL;


void createQueue(char *[], int);
void displayQueue(struct Patient *);
void insertPatient(struct Patient **, char *);

int main() {
   char *patients[] = {"Ab", "cd", "ef", "gh", "hi"};
   int n = 5;
   createQueue(patients, n);
   displayQueue(first);

   insertPatient(&first, "abh");
   printf("\n");
   displayQueue(first);

   return 0;
}


void createQueue(char *names[], int n) {
```

```
    int i;
    struct Patient *temp, *last;
    first = (struct Patient*)malloc(sizeof(struct Patient));
    strcpy(first->name, names[0]);
    first->next = NULL;
    last = first;
    for (i = 1; i < n; i++) {
        temp = (struct Patient*)malloc(sizeof(struct Patient));
        strcpy(temp->name, names[i]);
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
}


void displayQueue(struct Patient *p) {
    while (p != NULL) {
        printf("%s -> ", p->name);
        p = p->next;
    }
    printf("NULL\n");
}


void insertPatient(struct Patient **p, char *name) {
    struct Patient *temp, *last;
```

```
    temp = (struct Patient *)malloc(sizeof(struct Patient));
    strcpy(temp->name, name);
    temp->next = NULL;
    if (*p == NULL) {
        *p = temp;
    } else {
        last = *p;
        while (last->next != NULL) {
            last = last->next;
        }
        last->next = temp;
    }
    printf("Patient %s added to the queue.\n", name);
}
```

```
/**********************************************************************
****
```

Problem 2: Hospital Ward Allocation

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

Operations:

Create a list of available beds.

Insert a patient into an available bed.

Display the current bed allocation.

```
**********************************************************************
****/
```

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

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


struct Bed{
   int bedNumber;
   char patientName[100];
   struct Bed *next;
}*first = NULL;

void CreateWard(int totalBeds);
void allocateBed(int bedNumber, char *patientName);
void displayWard();


int main(){

   int choice,totalBeds,bedNumber;
   char patientName[100];

   while(1){

      printf("1.Create list of available beds\n2.Allocate bed to patient\n3.Display bed allocation\n4.Exit\n");
      printf("Entr your choice :");
      scanf("%d",&choice);

      switch(choice){
```

```
            case 1:
            printf("enter the no of beds available: ");
            scanf("%d",&totalBeds);
            CreateWard(totalBeds);
            break;
            case 2:
            printf("enter the bed number : ");
            scanf("%d",&bedNumber);
            printf("enter patient name : ");
            scanf("%s",patientName);
            allocateBed(bedNumber,patientName);
            break;
            case 3:
               displayWard();
               break;
            case 4:
            printf("Exiting the system...\n");
            exit(0);
            default:
            printf("Invalid choice! Please try again.\n");

        }

    }

   return 0;
}
void CreateWard(int totalBeds){
```

```
    int i;
    struct Bed *temp, *last;
    first = (struct Bed*)malloc(sizeof(struct Bed));
    first->bedNumber=1;
    strcpy(first->patientName, "Available");
    first->next = NULL;
    last = first;
    for (i = 2; i <= totalBeds; i++) {
        temp = (struct Bed*)malloc(sizeof(struct Bed));
        temp->bedNumber = i;
        strcpy(temp->patientName, "Available");
        temp->next = NULL;
        last->next = temp;
        last = temp;
    }
    printf("List of %d available beds created.\n", totalBeds);
}


void allocateBed(int bedNumber, char *patientName) {
    struct Bed *temp = first;
    while (temp != NULL) {
        if (temp->bedNumber == bedNumber) {
            if (strcmp(temp->patientName, "Available") == 0) {
                strcpy(temp->patientName, patientName);
                printf("Bed %d allocated to patient %s.\n", bedNumber, patientName);
                return;
```

```
            } else {
                printf("Bed %d is already occupied.\n", bedNumber);
                return;
            }
        }
        temp = temp->next;
    }
    printf("Bed %d does not exist.\n", bedNumber);
}



void displayWard() {
    struct Bed *temp = first;
    if (temp == NULL) {
        printf("No beds available.\n");
        return;
    }
    printf("Current bed allocation:\n");
    while (temp != NULL) {
        printf("Bed %d: %s\n", temp->bedNumber, temp->patientName);
        temp = temp->next;
    }
}
```

```
/*************************************************************************
****

Problem 3: Medical Inventory Tracking
Description: Maintain a linked list to track inventory items in a medical store. Operations:
Create an inventory list.
Insert a new inventory item.
Display the current inventory.
*************************************************************************
****/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


struct InventoryItem {
    char name[100];
    int quantity;
    struct InventoryItem *next;
} *first = NULL;

// Function prototypes
void createInventory();
void insertInventoryItem(char *name, int quantity);
void displayInventory();

int main() {
    int choice, quantity;
    char name[100];
```

```
    while (1) {
        printf("\nMedical Inventory Tracking System\n");
        printf("1. Create Inventory List\n2. Insert New Inventory Item\n3. Display Current Inventory\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createInventory();
                break;
            case 2:
                printf("Enter item name: ");
                scanf("%s", name);
                printf("Enter item quantity: ");
                scanf("%d", &quantity);
                insertInventoryItem(name, quantity);
                break;
            case 3:
                displayInventory();
                break;
            case 4:
                printf("Exiting the system...\n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
```

```
    }

    return 0;
}

void createInventory() {
    first = NULL;
    printf("New inventory list created.\n");
}

void insertInventoryItem(char *name, int quantity) {
    struct InventoryItem *newItem = (struct InventoryItem*)malloc(sizeof(struct
InventoryItem));
    strcpy(newItem->name, name);
    newItem->quantity = quantity;
    newItem->next = NULL;

    if (first == NULL) {
        first = newItem;
    } else {
        struct InventoryItem *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newItem;
    }
```

```
    printf("Inventory item %s added with quantity %d.\n", name, quantity);
}



void displayInventory() {
    struct InventoryItem *temp = first;
    if (temp == NULL) {
        printf("The inventory is empty.\n");
        return;
    }
    printf("Current inventory:\n");
    while (temp != NULL) {
        printf("Item: %s, Quantity: %d\n", temp->name, temp->quantity);
        temp = temp->next;
    }
}



/**********************************************************************
****

Problem 4: Doctor Appointment Scheduling
Description: Develop a linked list to schedule doctor appointments. Operations:
Create an appointment list.
Insert a new appointment.
Display all scheduled appointments.
**********************************************************************
****/
#include <stdio.h>
```

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

struct Appointment {
    char patientName[100];
    char appointmentDate[20];
    char appointmentTime[10];
    struct Appointment *next;
} *first = NULL;

void createAppointmentList();
void insertAppointment(char *patientName, char *appointmentDate, char
*appointmentTime);
void displayAppointments();

int main() {
    int choice;
    char patientName[100], appointmentDate[20], appointmentTime[10];

    while (1) {
        printf("\nDoctor Appointment Scheduling System\n");
        printf("1. Create Appointment List\n2. Insert New Appointment\n3. Display All
Scheduled Appointments\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
```

```
            case 1:
                createAppointmentList();
                break;
            case 2:
                printf("Enter patient name: ");
                scanf("%s", patientName);
                printf("Enter appointment date (DD/MM/YYYY): ");
                scanf("%s", appointmentDate);
                printf("Enter appointment time (HH:MM): ");
                scanf("%s", appointmentTime);
                insertAppointment(patientName, appointmentDate, appointmentTime);
                break;
            case 3:
                displayAppointments();
                break;
            case 4:
                printf("Exiting the system...\n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}



void createAppointmentList() {
```

```
    first = NULL;
    printf("New appointment list created.\n");
}

void insertAppointment(char *patientName, char *appointmentDate, char
*appointmentTime) {
    struct Appointment *newAppointment = (struct Appointment*)malloc(sizeof(struct
Appointment));
    strcpy(newAppointment->patientName, patientName);
    strcpy(newAppointment->appointmentDate, appointmentDate);
    strcpy(newAppointment->appointmentTime, appointmentTime);
    newAppointment->next = NULL;

    if (first == NULL) {
        first = newAppointment;
    } else {
        struct Appointment *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newAppointment;
    }
    printf("Appointment for patient %s scheduled on %s at %s.\n", patientName,
appointmentDate, appointmentTime);
}

void displayAppointments() {
```

```
    struct Appointment *temp = first;
    if (temp == NULL) {
        printf("No appointments scheduled.\n");
        return;
    }
    printf("Scheduled Appointments:\n");
    while (temp != NULL) {
        printf("Patient: %s, Date: %s, Time: %s\n", temp->patientName, temp-
>appointmentDate, temp->appointmentTime);
        temp = temp->next;
    }
}
```

```
/*************************************************************************
****
```

Problem 5: Emergency Contact List

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

Create a contact list.

Insert a new contact.

Display all emergency contacts.

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

```
struct Contact {
    char name[100];
    char phoneNumber[20];
    struct Contact *next;
} *first = NULL;



void createContactList();
void insertContact(char *name, char *phoneNumber);
void displayContacts();

int main() {
    int choice;
    char name[100], phoneNumber[20];

    while (1) {
        printf("\nEmergency Contact List Management System\n");
        printf("1. Create Contact List\n2. Insert New Contact\n3. Display All Emergency
Contacts\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

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

```
            printf("Enter contact name: ");
            scanf("%s", name);
            printf("Enter phone number: ");
            scanf("%s", phoneNumber);
            insertContact(name, phoneNumber);
            break;
        case 3:
            displayContacts();
            break;
        case 4:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}



void createContactList() {
   first = NULL;
   printf("New contact list created.\n");
}



void insertContact(char *name, char *phoneNumber) {
```

```
    struct Contact *newContact = (struct Contact*)malloc(sizeof(struct Contact));
    strcpy(newContact->name, name);
    strcpy(newContact->phoneNumber, phoneNumber);
    newContact->next = NULL;

    if (first == NULL) {
        first = newContact;
    } else {
        struct Contact *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newContact;
    }
    printf("Contact %s with phone number %s added.\n", name, phoneNumber);
}



void displayContacts() {
    struct Contact *temp = first;
    if (temp == NULL) {
        printf("The contact list is empty.\n");
        return;
    }
    printf("Emergency Contacts:\n");
    while (temp != NULL) {
        printf("Name: %s, Phone Number: %s\n", temp->name, temp->phoneNumber);
        temp = temp->next;
```

```
    }
}



/*************************************************************************
****
Problem 6: Surgery Scheduling System
Description: Use a linked list to manage surgery schedules.
Operations:
Create a surgery schedule.
Insert a new surgery into the schedule.
Display all scheduled surgeries.
*************************************************************************
****/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>



struct Surgery {
    char patientName[100];
    char surgeryDate[20];
    char surgeryTime[10];
    char surgeonName[100];
    struct Surgery *next;
} *first = NULL;



void createSurgerySchedule();
```

```
void insertSurgery(char *patientName, char *surgeryDate, char *surgeryTime, char
*surgeonName);

void displaySurgeries();


int main() {

   int choice;

   char patientName[100], surgeryDate[20], surgeryTime[10], surgeonName[100];


   while (1) {

      printf("\nSurgery Scheduling System\n");

      printf("1. Create Surgery Schedule\n2. Insert New Surgery\n3. Display All Scheduled
Surgeries\n4. Exit\n");

      printf("Enter your choice: ");

      scanf("%d", &choice);


      switch (choice) {

         case 1:

            createSurgerySchedule();

            break;

         case 2:

            printf("Enter patient name: ");

            scanf("%s", patientName);

            printf("Enter surgery date (DD/MM/YYYY): ");

            scanf("%s", surgeryDate);

            printf("Enter surgery time (HH:MM): ");

            scanf("%s", surgeryTime);

            printf("Enter surgeon's name: ");

            scanf("%s", surgeonName);

            insertSurgery(patientName, surgeryDate, surgeryTime, surgeonName);
```

```
            break;
        case 3:
            displaySurgeries();
            break;
        case 4:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}



void createSurgerySchedule() {
   first = NULL;
   printf("New surgery schedule created.\n");
}



void insertSurgery(char *patientName, char *surgeryDate, char *surgeryTime, char
*surgeonName) {
   struct Surgery *newSurgery = (struct Surgery*)malloc(sizeof(struct Surgery));
   strcpy(newSurgery->patientName, patientName);
   strcpy(newSurgery->surgeryDate, surgeryDate);
   strcpy(newSurgery->surgeryTime, surgeryTime);
```

```
    strcpy(newSurgery->surgeonName, surgeonName);
    newSurgery->next = NULL;

    if (first == NULL) {
        first = newSurgery;
    } else {
        struct Surgery *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newSurgery;
    }
    printf("Surgery for patient %s scheduled on %s at %s with surgeon %s.\n", patientName,
surgeryDate, surgeryTime, surgeonName);
}

void displaySurgeries() {
    struct Surgery *temp = first;
    if (temp == NULL) {
        printf("No surgeries scheduled.\n");
        return;
    }
    printf("Scheduled Surgeries:\n");
    while (temp != NULL) {
        printf("Patient: %s, Date: %s, Time: %s, Surgeon: %s\n", temp->patientName, temp-
>surgeryDate, temp->surgeryTime, temp->surgeonName);
        temp = temp->next;
    }
}
```

```
/*************************************************************************
****
Problem 7: Patient History Record
Description: Maintain a linked list to keep track of patient history records.
Operations:
Create a history record list.
Insert a new record.
Display all patient history records.
*************************************************************************
****/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


struct HistoryRecord {
   char patientName[100];
   char illness[100];
   char treatment[100];
   char visitDate[20];
   struct HistoryRecord *next;
} *first = NULL;


void createHistoryRecordList();
void insertHistoryRecord(char *patientName, char *illness, char *treatment, char
*visitDate);
void displayHistoryRecords();
```

```
int main() {
    int choice;
    char patientName[100], illness[100], treatment[100], visitDate[20];

    while (1) {

        printf("1. Create History Record List\n2. Insert New Record\n3. Display All Patient History Records\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createHistoryRecordList();
                break;
            case 2:
                printf("Enter patient name: ");
                scanf("%s", patientName);
                printf("Enter illness: ");
                scanf("%s", illness);
                printf("Enter treatment: ");
                scanf("%s", treatment);
                printf("Enter visit date (DD/MM/YYYY): ");
                scanf("%s", visitDate);
                insertHistoryRecord(patientName, illness, treatment, visitDate);
                break;
            case 3:
                displayHistoryRecords();
```

```
            break;
        case 4:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}



void createHistoryRecordList() {
   first = NULL;
   printf("New history record list created.\n");
}



void insertHistoryRecord(char *patientName, char *illness, char *treatment, char *visitDate)
{
   struct HistoryRecord *newRecord = (struct HistoryRecord*)malloc(sizeof(struct
HistoryRecord));
   strcpy(newRecord->patientName, patientName);
   strcpy(newRecord->illness, illness);
   strcpy(newRecord->treatment, treatment);
   strcpy(newRecord->visitDate, visitDate);
   newRecord->next = NULL;
```

```
    if (first == NULL) {
        first = newRecord;
    } else {
        struct HistoryRecord *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newRecord;
    }
    printf("History record for patient %s added.\n", patientName);
}



void displayHistoryRecords() {
    struct HistoryRecord *temp = first;
    if (temp == NULL) {
        printf("No history records available.\n");
        return;
    }
    printf("Patient History Records:\n");
    while (temp != NULL) {
        printf("Patient: %s, Illness: %s, Treatment: %s, Visit Date: %s\n", temp->patientName,
temp->illness, temp->treatment, temp->visitDate);
        temp = temp->next;
    }
}
```

```
/*************************************************************************
****
Problem 8: Medical Test Tracking
Description: Implement a linked list to track medical tests for patients. Operations:
Create a list of medical tests.
Insert a new test result.
Display all test results.
*************************************************************************
****/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


struct MedicalTest {
   char patientName[100];
   char testName[100];
   char testDate[20];
   char result[100];
   struct MedicalTest *next;
} *first = NULL;


void createTestList();
void insertTestResult(char *patientName, char *testName, char *testDate, char *result);
void displayTestResults();

int main() {
   int choice;
```

```
    char patientName[100], testName[100], testDate[20], result[100];


    while (1) {
        printf("\nMedical Test Tracking\n");
        printf("1. Create Test List\n2. Insert New Test Result\n3. Display All Test Results\n4.
Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);


        switch (choice) {
            case 1:
                createTestList();
                break;
            case 2:
                printf("Enter patient name: ");
                scanf("%s", patientName);
                printf("Enter test name: ");
                scanf("%s", testName);
                printf("Enter test date (DD/MM/YYYY): ");
                scanf("%s", testDate);
                printf("Enter test result: ");
                scanf("%s", result);
                insertTestResult(patientName, testName, testDate, result);
                break;
            case 3:
                displayTestResults();
                break;
            case 4:
```

```
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}



void createTestList() {
   first = NULL;
   printf("New test list created.\n");
}



void insertTestResult(char *patientName, char *testName, char *testDate, char *result) {
   struct MedicalTest *newTest = (struct MedicalTest*)malloc(sizeof(struct MedicalTest));
   strcpy(newTest->patientName, patientName);
   strcpy(newTest->testName, testName);
   strcpy(newTest->testDate, testDate);
   strcpy(newTest->result, result);
   newTest->next = NULL;

   if (first == NULL) {
      first = newTest;
   } else {
```

```
        struct MedicalTest *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newTest;
    }
    printf("Test result for patient %s added.\n", patientName);
}



void displayTestResults() {
    struct MedicalTest *temp = first;
    if (temp == NULL) {
        printf("No test results available.\n");
        return;
    }
    printf("Medical Test Results:\n");
    while (temp != NULL) {
        printf("Patient: %s, Test: %s, Date: %s, Result: %s\n", temp->patientName, temp-
>testName, temp->testDate, temp->result);
        temp = temp->next;
    }
}
```

```
/*************************************************************************
****
Problem 9: Prescription Management System
Description: Use a linked list to manage patient prescriptions. Operations:
Create a prescription list.
Insert a new prescription.
Display all prescriptions.
*************************************************************************
****/

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


struct Prescription {
   char patientName[100];
   char medication[100];
   char dosage[100];
   struct Prescription *next;
};

struct Prescription* createPrescription(char* patientName, char* medication, char* dosage)
{
   struct Prescription* newPrescription = (struct Prescription*)malloc(sizeof(struct
Prescription));
   strcpy(newPrescription->patientName, patientName);
   strcpy(newPrescription->medication, medication);
   strcpy(newPrescription->dosage, dosage);
```

```
    newPrescription->next = NULL;
    return newPrescription;
}

void insertPrescription(struct Prescription** head, char* patientName, char* medication,
char* dosage) {
    struct Prescription* newPrescription = createPrescription(patientName, medication,
dosage);
    if (*head == NULL) {
        *head = newPrescription;
    } else {
        struct Prescription* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newPrescription;
    }
}

void displayPrescriptions(struct Prescription* head) {
    if (head == NULL) {
        printf("No prescriptions found.\n");
        return;
    }

    struct Prescription* temp = head;
    while (temp != NULL) {
```

```
        printf("Patient Name: %s\n", temp->patientName);
        printf("Medication: %s\n", temp->medication);
        printf("Dosage: %s\n", temp->dosage);
        printf("----------------------------\n");
        temp = temp->next;
    }
}

int main() {
    struct Prescription* head = NULL;
    int choice;
    char patientName[100], medication[100], dosage[100];

    while (1) {
        printf("\nPrescription Management System\n");
        printf("1. Insert new prescription\n");
        printf("2. Display all prescriptions\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter patient name: ");
                scanf("%s", patientName);

                printf("Enter medication: ");
                scanf("%s", medication);
```

```
            printf("Enter dosage: ");
            scanf("%s", dosage);

            insertPrescription(&head, patientName, medication, dosage);
            printf("Prescription added successfully.\n");
            break;

        case 2:
            displayPrescriptions(head);
            break;

        case 3:
            printf("Exiting the program.\n");
            exit(0);

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

   return 0;
}
```

```
/**************************************************************************
****
Problem 10: Hospital Staff Roster
Description: Develop a linked list to manage the hospital staff roster.
Operations:
Create a staff roster.
Insert a new staff member into the roster.
Display the current staff roster.
**************************************************************************
****/

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


struct StaffMember {
    char name[100];
    char position[100];
    char shift[50];
    struct StaffMember *next;
} *first = NULL;


void createStaffRoster();
void insertStaffMember(char *name, char *position, char *shift);
void displayStaffRoster();

int main() {
```

```
    int choice;
    char name[100], position[100], shift[50];

    while (1) {
        printf("\nHospital Staff Roster Management System\n");
        printf("1. Create Staff Roster\n2. Insert New Staff Member\n3. Display Current Staff Roster\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createStaffRoster();
                break;
            case 2:
                printf("Enter staff member name: ");
                scanf("%s", name);
                printf("Enter position: ");
                scanf("%s", position);
                printf("Enter shift: ");
                scanf("%s", shift);
                insertStaffMember(name, position, shift);
                break;
            case 3:
                displayStaffRoster();
                break;
            case 4:
                printf("Exiting the system...\n");
```

```
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
    }
  }

  return 0;
}



void createStaffRoster() {
  first = NULL;
  printf("New staff roster created.\n");
}



void insertStaffMember(char *name, char *position, char *shift) {
  struct StaffMember *newStaff = (struct StaffMember*)malloc(sizeof(struct StaffMember));
  strcpy(newStaff->name, name);
  strcpy(newStaff->position, position);
  strcpy(newStaff->shift, shift);
  newStaff->next = NULL;

  if (first == NULL) {
    first = newStaff;
  } else {
    struct StaffMember *temp = first;
    while (temp->next != NULL) {
```

```
            temp = temp->next;
        }
        temp->next = newStaff;
    }
    printf("Staff member %s added.\n", name);
}



void displayStaffRoster() {
    struct StaffMember *temp = first;
    if (temp == NULL) {
        printf("The staff roster is empty.\n");
        return;
    }
    printf("Current Staff Roster:\n");
    while (temp != NULL) {
        printf("Name: %s, Position: %s, Shift: %s\n", temp->name, temp->position, temp->shift);
        temp = temp->next;
    }
}
```